Stereo Matching for Martian Surface Depth Estimation on a Single-Board Computer

C. Lau

TU Delft

Stereo Matching for Martian Surface Depth Estimation on a Single-Board Computer

by

C. Lau

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday August 31, 2021 at 09:00 AM.

Student number: 4155955

Thesis committee:	Dr. S. Speretta,	TU Delft, supervisor
	Dr. J. Guo,	TU Delft, chair
	Dr.ir. B.C. Root	TU Delft, examiner

TU Delft

Abstract

Stereo matching, the process of inferring depth maps from stereo images, is one of the most heavily investigated topics in computer vision. It is part of the first module of navigation systems of planetary rovers, e.g., NASA's Mars Exploration Rover (MER) missions, NASA's Mars Science Laboratory (MSL) mission, and ESA's ExoMars mission. Many stereo matching algorithms, traditional and deep learning based, are available and their performance is typically evaluated on indoor objects or outdoor city scenes.

In this thesis, our goal is to improve insight into the performance of stereo matching algorithms for the Martian surface on a single-board computer. First, we search for and compare stereo matching algorithms ranked on stereo vision datasets to obtain a manageable set of algorithms and verify their implementations. Second, we derive performance metrics from the requirements and validate our set of verified algorithms on the Katwijk Beach Planetary Rover dataset. Through experimental evaluation, we gain insight into the performance of four stereo matching algorithms.

Preface

This master thesis is the peak of many years of education at the Delft University of Technology. It has been a long and memorable journey, filled with ups and down, and I am happy to have completed it. As every student graduating in this period will say; a down is definitely COVID-19. It has affected the lives of everyone, and created changes in society that are unimaginable to us just a few years ago, before the pandemic started.

First, I want to thank my supervisor, Stefano Speretta, for supporting me and giving me great insights. I am still amazed how you can simultaneously listen to me and read my report during the (online) Zoom meetings, what a skill to have! The meetings were so entertaining, because of your kids making noise in the background and you shushing them.

Second, I want to thank Tara, for all your love and support. You have been an up during this difficult period. I learned so much about emotional development from you, and I still have a long way to go.

Finally, I want to thank my family for their love, support, and patience. You have always supported me from behind the scenes. I am ready for the next step: developing my own identity and creating my own plan.

C. Lau
Ridderkerk, August 2021

Contents

1

Introduction

Stereo matching, the process of inferring depth maps from stereo images, is one of the most heavily investigated topics in computer vision [1, 2]. It is part of the first module of navigation systems of planetary rovers, e.g., from NASA's Mars Exploration Rover missions [3, 4], NASA's Mars Science Laboratory (MSL) mission [5], ESA's ExoMars mission [6], and Chang'e-3 (CE-3) and Chang'e-4 (CE-4) missions [7, 8]. Stereo matching also has applications in object detection [9], remote sensing [10] and autonomous driving [11].

This thesis focuses on stereo matching for the Martian surface on a single-board computer. A single-board computer is a complete computer build on a single circuit board. It is used in the testing phase of the development of a rover, in particular, in the Lunar Zebro project [12]. The Lunar Zebro is developed at the Delft University of Technology, the Delft Robotics Institute, to study swarm behaviour on the Lunar surface [13]. For their testing phase, the Raspberry Pi 4 Model B [14] is used. This work could facilitate the choice of algorithm(s) on the Lunar Zebro's future missions.

The Martian surface is rocky, with canyons, volcanoes, dry lake beds and craters all over it [15]. Not only the exploration of the Martian surface is of interest, but also the exploration of the Martian subsurface, i.e., lava tubes and caves, has been a growing interest [16–20]. However, in this thesis we focus on the Martian surface, as our work is dependent on available datasets with stereo images. For the Martian surface the Katwijk Beach Planetary Rover dataset [21] is available, which is a Mars analog of the surface. However, for the Martian subsurface, there is no dataset available with stereo images, making evaluating stereo matching algorithms a difficult task.

Many algorithms have been developed for depth estimation (see Section 1.1). The list of algorithms has been increasing, due to stereo matching algorithms becoming a deep learning task [22] with the development of convolutional neural networks (CNN) [23]. Each stereo matching algorithm can be categorized into one of the two groups: *traditional* stereo matching algorithms, and *deep learning* stereo matching algorithms. Although the algorithms are tested on existing benchmarks to show their performances, the performance comparison of the Martian surface on a single-board computer has not been studied. This is of importance, as the performance of algorithms on a single-board computer serves as an initial indicator of which algorithm(s) to focus on. In [24], the performance (matching error and matching speed) of a deep learning algorithm (DispNet [25]) is compared with traditional algorithms (semi-global matching (SGM) [26], bidirectional matching (BM) [27] and sum of absolute difference (SAD) [28]) for the Lunar surface. It was demonstrated that DispNet is able to robustly reconstruct the lunar surface under different illumination conditions, with a lower matching error and matching speed than traditional algorithms.

We focus on understanding the performance of traditional and deep learning stereo matching algorithms on a single-board computer in a Martian scene. Stereo matching algorithms are typically evaluated on datasets for autonomous driving, e.g., KITTI 2015 stereo vision dataset [29], based on the run time and error rate of the algorithms. For the Martian surface scene, the performance of stereo matching algorithms have not yet been studied.

1.1. What is Stereo Matching?

Stereo matching is generating a depth map from a pair of stereo images. Just like how humans infer depth from the combination of the left and right eye images, computers are able to infer depth from left and right images from a stereo camera. In essence, the stereo matching process finds for each pixel in the left camera image, the corresponding pixel in the right camera image. Figure 1.1 depicts the left and right image of a tree. Stereo matching is finding the corresponding pixel of the top of the tree in the left image, with the top of the tree in the right image. We observe that the tree in the right image is shifted to the left side. The number of pixels it has shifted in the horizontal direction is the disparity. In this case, the disparity is 3, as the tree in the right image has shifted 3 pixels with respect to the left image. Performing this operation for all pixels, results in a disparity map, where each pixel in the disparity map contains the shifted amount.

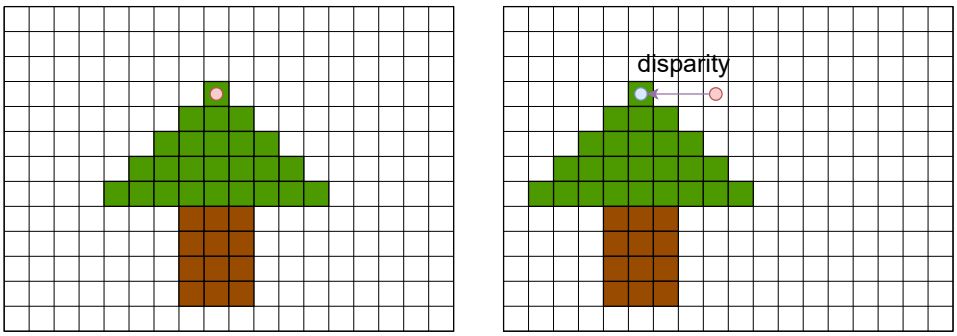


Figure 1.1: Stereo matching example

Stereo matching algorithms attempt to find the corresponding pixels and produce a predicted disparity map. To give an illustration, we use a popular stereo matching algorithm, the semi-global matching (SGM) [26], to generate a predicted disparity map from the Tsukuba stereo image. The Tsukuba stereo image is part of the Middlebury stereo vision dataset [30]. The stereo image is one of several stereo images to evaluate the performance of stereo matching algorithms in the works of [1]. In Figure 1.2, the left image, right image, predicted disparity map, and ground-truth (true disparity map) of the Tsukuba dataset is depicted. The disparity map is produced by the SGM algorithm, where the lighter values are corresponding to closer points, and darker values points that are farther away. We observe that the shapes in the predicted disparity map correspond to that of the ground-truth disparity map. However, details are missing, e.g., the rods of the lamp on the right of the scene.

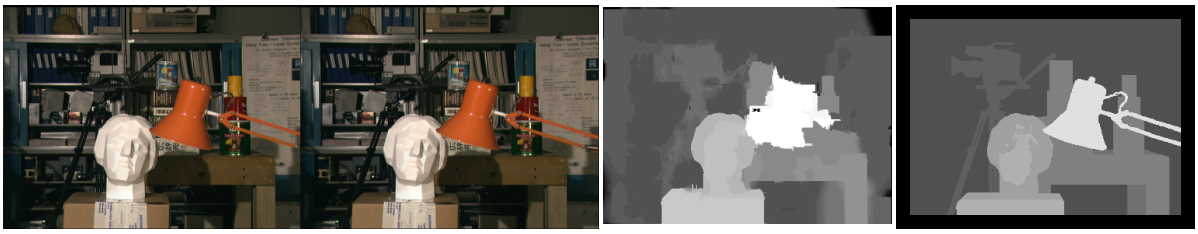


Figure 1.2: Tsukuba left image, right image, predicted disparity map, and ground-truth disparity map

1.2. Problem Statement

The goal is to improve insight into the performance of dense stereo matching algorithms, for the Martian surface on a single-board computer. Dense stereo matching algorithms produce a disparity estimate at each pixel of the stereo images [1]. Whereas sparse stereo matching algorithms (feature-based algorithms [31]), produce disparity estimates at certain pixels, which would be insufficient for reconstructing detailed 3D terrains [32]. Many dense stereo matching algorithms, traditional and deep learning based, are available and there is a need for comparing the performance of a manageable selection of algorithms. We use the Katwijk Beach Planetary Rover dataset, representative of the ExoMars rover [21]. Such that the stereo matching algorithms will perform specific to the ExoMars requirements

(see Chapter 3.4). As (new) algorithms are being designed, their performance are typically evaluated on the Middlebury Stereo Vision benchmark [1], containing indoor scenes with objects, or the KITTI Vision Benchmark Suite [29], containing outdoor city scenes with roads and buildings. However, the results of their performance are not comparable for the Martian surface, as texture, colour, and lighting of indoor objects and outdoor city scenes differ from the Martian surface. The work of [24] provides a glimpse of the performance on the Lunar surface using both traditional and deep learning stereo matching algorithms. However, as mentioned before, the performance of stereo matching algorithms is dependent the surrounding scene, i.e., the performance of an algorithm in the Lunar scene is different from the Martian scene, as the texture, colour, and lighting differs. In addition, running the algorithms, in particular deep learning algorithms, on a single-board computer, has not been heavily studied. The evaluation on benchmarks is typically performed using GPUs (Nvidia GTX 1080Ti, Nvidia Titan X) [22]. In terms of single-board computer performance, in [33], the run time of CNNs on CPUs and GPUs are presented and in [34] the performance of CNNs on a Raspberry Pi 3 Model B. However, for both works the performance of stereo matching algorithms are not studied.

1.3. Main Research Questions

To facilitate the comparison and analysis of the performance of stereo matching algorithms for the Martian surface on a single-board computer, we address in this work the following research questions:

- **RQ1: How to obtain a manageable selection of stereo matching algorithms for comparison?**
  Many dense stereo matching algorithms are available (see Section 1.1) to choose from. Thus, a selection of algorithms, spread across various datasets and using different environments, is of particular importance. Performance metrics, e.g., run time, matching error and memory usage (and potentially more), are used for the selection of algorithms, based on the set of requirements. The requirement for a manageable selection is explicitly included, i.e., the number of selected algorithms is sufficient for the scope of the thesis.

- **RQ2: How to efficiently compare the performance of stereo matching algorithms?**
  The performance metrics are used to evaluate the algorithms, based on the set of requirements. In addition, the dataset used for validation is of importance for performance comparison, as it should be representative for the real-world environment in which the ExoMars rover operates. The requirement for an approach that is efficient is explicitly included, i.e., it aims to minimize the time and effort required to compare a set of stereo matching algorithms.

- **RQ3: What algorithm best applies to the requirements?**
  A set of requirements is generated, that are the constraints. An algorithm can be selected based on these constraints. The requirement for an algorithm that best applies is explicitly included, i.e., a single algorithm is selected, instead multiple that can be within the constraints.

1.4. Approach

Guided by the research questions, we develop an approach to develop the answers to the questions. Addressing RQ1 (see Chapter 4 and Chapter 5), we search for stereo vision algorithms ranked on the popular stereo vision datasets, Middlebury [30] and KITTI 2015 [29], based on indoor and outdoor scenes. We obtain a list of algorithms based on their error rate. The error rate of the ranked algorithms does not directly indicate how they would performs on a Martian environment, since they are evaluated in different scenes. However, to this date, there are no stereo vision algorithms ranked on a Martian-like stereo vision dataset. To facilitate the search for algorithms, we therefore use the Middlebury and KITTI 2015 datasets. Next, the selection of algorithms is guided by the set of selection criteria derived from literature, ExoMars requirements (see Chapter 3.4), and the availability of existing code. The set of selected algorithms are then verified to check if their implementation is accurate, based on their error rate and run time, compared with the values provided in literature. Any changes in the set of selected algorithms appear at this stage.

Addressing RQ2 (see Chapter 6 and Chapter 7), we validate our set of verified algorithms. The testing environment is setup using the Katwijk Beach Planetary Rover (Katwijk) dataset [21] as the input for the set of algorithms. The Katwijk dataset uses a rover with similar specifications as the

ExoMars rover. The performance metrics to compare the predicted disparity maps are derived from the ExoMars requirements.

Addressing RQ3 (see Chapter 7), we select a final stereo matching algorithm that best applies to our constraints, i.e., the ExoMars requirements (see Chapter 3.4).

1.5. Main Contributions

The main contributions of this thesis are:

1. The comparison of (public) implementations of traditional stereo vision algorithms. Many implementations of the same traditional algorithms are available, however their performance with respect to each other has not been studied. To select our algorithms for validation, we perform a detailed comparison (RQ1).
2. The evaluation of stereo matching algorithms on the Katwijk dataset. Most algorithms are evaluated on the Middlebury and KITTI 2015 datasets, however the Katwijk dataset has not been used to evaluate algorithms (RQ2).
3. The development of an error rate metric for the evaluation of a disparity map. We argue that the standard bad matched pixels (BMP) percentage error rate metric is not a suitable metric in the context of navigation. We develop a metric which transforms the disparity map to a distance map and uses the distance threshold as its error (RQ2).
4. Execution of a deep learning stereo vision algorithm on a single-board computer. Rovers to this date use traditional stereo vision algorithms for the prediction of disparity maps. In this thesis we analyze the performance of a deep learning algorithm on a single-board computer.

1.6. Guidelines for Reading

The thesis is structured as follows:

- In Chapter 2, we discuss the background information, i.e., the epipolar geometry, the stereo matching pipelines, and the convolutional neural network (CNN).
- In Chapter 3, we describe our guiding vehicle and develop the requirements.
- In Chapter 4, we describe the selection process, selection criteria, datasets, and draft selection.
- In Chapter 5, we verify the draft selection of algorithms and obtain a final set of selected algorithms.
- In Chapter 6, we describe setup of our experiments, along with the metrics.
- In Chapter 7, we analyze and discuss the results.
- In Chapter 8, we conclude the thesis and provide items for future work.

2

Background

In this chapter we present background information to understand the basics of stereo matching algorithms. In Section 2.1 we describe the epipolar geometry. For deep learning methods, a basic understanding of neural networks is required, in particular the convolutional neural network (CNN) [23], described in Section 2.2. In Section 2.3 we describe the stereo matching pipelines for traditional and deep learning methods.

2.1. Epipolar Geometry

The epipolar geometry is the geometry of stereo matching, i.e., how to compute for a given pixel in one image the range of possible locations the pixel might appear at in the other image, i.e., its *epipolar line* [35]. Figure 2.1(a) depicts a stereo configuration, where two cameras are modeled as pinholes with the image planes $\vec{I}_0$ and $\vec{I}_1$ in front of the lenses [36]. For each point $\vec{p}$ in the scene, there is a plane, the *epipolar plane*, that passes through the point and the line joining the two lens centers $\vec{c}_0$ and $\vec{c}_1$ [36]. In addition, the *epipoles* $\vec{e}_0$ and $\vec{e}_1$ are the intersection of the image planes with the line joining the lens centers [36]. All the points in an epipolar plane are projected onto one epipolar line in the first image and onto the corresponding epipolar line in the second image [36], where the point $\vec{x}_0$ corresponds to the point $\vec{x}_1$. The range of possible locations the pixel might appear at the other image is reduced from two dimensions to one [35, 36].

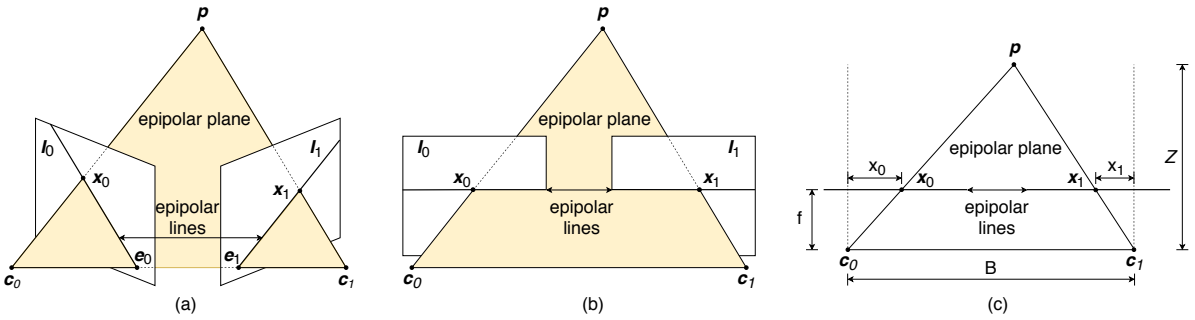


Figure 2.1: Stereo configuration: (a) corresponding set of epipolar lines and their epipolar plane; (b) rectified epipolar lines and their epipolar plane; (c) top view of the rectified configuration (adapted from [35, 36]).

2.1.1. Rectification

To find a match for a point along an epipolar line in one image, all that is necessary is to search along the corresponding epipolar line in the second image; this geometric relationship is the *epipolar constraint* [36]. This search can be further simplified by making the epipolar lines in the images horizontal, such that they correspond to the rows of the images. Figure 2.1(b) depicts the horizontal epipolar lines. The transformation to horizontal epipolar lines is obtained by *rectifying*, i.e., warping the images [35]. In general, rectification algorithms can be categorized into two types: non-calibrated and calibrated cases [37]. For non-calibrated rectification, the camera parameters are not required [37], whereas for

calibrated rectification, the cameras internal parameters (i.e., the focal lengths, mutual position and orientation) should be known [38]. Figure 2.1(c) depicts the top view of the stereo configuration after rectification. This resulting *standard rectified geometry* leads to an inverse relationship between the 3D depth Z and the disparity d [35],

$$d = f\frac{B}{Z}; \quad d = x_0 - x_1 \tag{2.1}$$

where f is the focal length (measured in pixels) and B is the baseline. The correspondence between a point, i.e., pixel (x_0, y_0) in left image and a pixel (x_1, y_1) in the right image is given by [1],

$$x_1 = x_0 + sd(x_0, y_0), \quad y_1 = y_0 \tag{2.2}$$

where $s = \pm 1$ is a sign chosen to produce positive disparities. Extracting depth from a set of images becomes a task of estimating the *disparity map* $d(x, y)$ [35].

2.2. Convolutional Neural Network (CNN)

In this chapter we describe the basics of neural networks (Section 2.2.1) to develop an understanding of the convolutional neural network (CNN) [23, 39, 40] (Section 2.2.2). The CNN is the primary building block of the deep learning stereo matching pipeline, described in Section 2.3.2. The CNN has been popularized by AlexNet [41], one of the most influential networks in computer vision, it has spurred many more networks employing a CNN.

2.2.1. A Primer on Neural Networks

In principle, a neural network has the power of a universal approximator, i.e., it can realise an arbitrary mapping of one vector space onto another vector space [42]. E.g, a set images as the inputs to a neural network can be mapped to a set of categories, i.e., an image classifier. A neural network is able to learn an arbitrary mapping by *training* the neural network, i.e., to adjust the weight coefficients in such a way that the mapping is fulfilled. A simple neural network, the *multi-layer feed-forward* (MLF) [42] neural network, is depicted in Figure 2.3. In general, a MLF network consists of neurons, that are ordered into layers: input layer, hidden layer(s) and output layer. Each neuron in a particular layer is connected to all neurons in the next layer. The connection between jth (previous) neuron and the ith (current) is characterized by the weight coefficient ω_{ij}. The output of a neuron y_i is described by [42, 43],

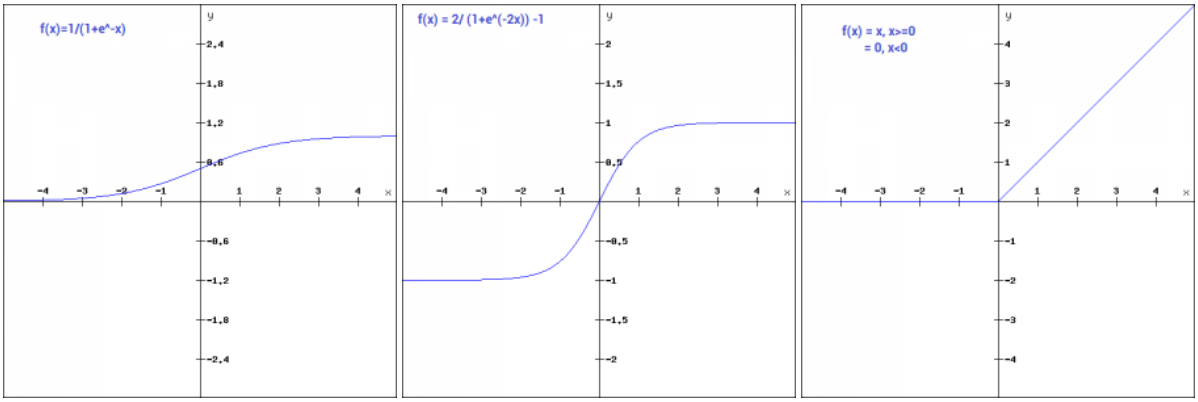
$$y_i = f(\xi_i) \tag{2.3}$$

$$\xi_i = \sum_{j=1}^{N} \omega_{ij} x_j + b_i \tag{2.4}$$

where ξ_i is the pre-activation function, x_j the neuron output of the previous layer, b_i the bias for the current layer, and f the activation function. The bias consists of a constant output, $x_b = 1$, and a weight coefficient ω_b, such that,

$$b_i = \omega_{bi} x_{bi} = \omega_{bi} \tag{2.5}$$

For the activation function, common functions include [44]: Sigmoid, Tanh, and Rectified Linear Unit (ReLU). These activation functions are depicted in Figure 2.2. These non-linear functions allow the network to approximate non-linear input to output mappings.



(a) Sigmoid activation function. (b) Tanh activation function. (c) ReLU activation function.

Figure 2.2: Common activation functions (adopted from [44]).

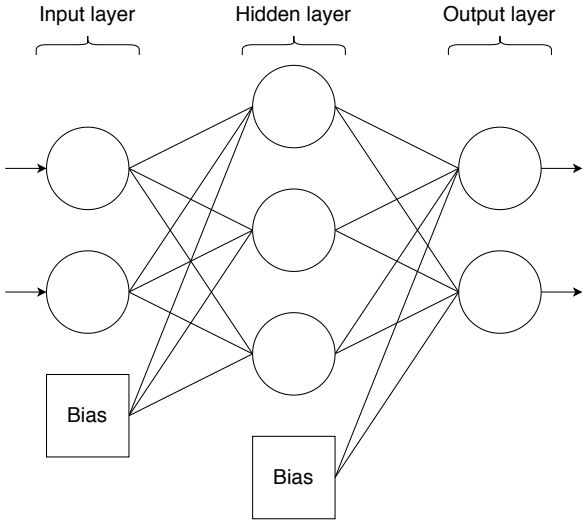


Figure 2.3: Typical feed-forward neural network composed of three layers.

Training a neural network

To train the neural network, i.e, to adjust the weight coefficients to produce the correct output, the *back-propagation* algorithm [45] is used. The basic idea of the backpropagation algorithm is the repeated application of the chain rule, to compute the influence of each weight coefficient in the network with respect to an error function [45], i.e.,

$$\frac{\delta E}{\delta \omega_{ij}} = \frac{\delta E}{\delta y_i} \frac{\delta y_i}{\delta \xi_i} \frac{\delta \xi_i}{\delta \omega_{ij}} \tag{2.6}$$

where E is the error function, e.g., squared error, ω_{ij} the weight coefficient from neuron j to neuron i, y_i is the output, and ξ_i the pre-activation function. The repeated application of the chain rule is depicted in Figure 2.4.

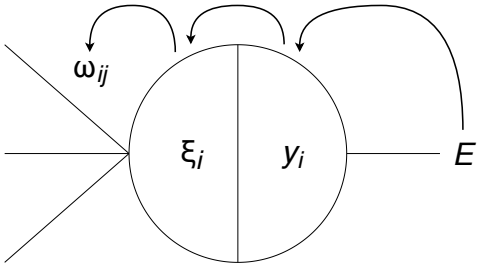


Figure 2.4: Repeated application of the chain rule (adapted from [46]).

The aim is to minimize the error function and is achieved by performing a gradient descent [45], defined by,

$$\omega_{ij}(t+1) = \omega_{ij}(t) - \epsilon \frac{\delta E}{\delta \omega_{ij}}(t) \tag{2.7}$$

where $\omega_{ij}(t+1)$ is the updated weight coefficient, t the iteration, and ϵ the learning rate. The learning rate belongs to a set of *hyperparameters*, i.e., parameters that are set before training the network. Other hyperparameters include [47]: number of hidden layers and neurons, number of iterations, momentum, batch size, and error threshold. If the learning rate is set too small, too many iterations are required to reach an acceptable error. On the contrary, a large learning rate could lead to oscillations, preventing the error to fall within an certain value [45].

Training the network is performed in two passes, the *forward pass* and *backward pass*. In the forward pass, the outputs of all neurons are computed given the input. In the backward pass, the backpropagation algorithm is used, along with gradient descent, to update the weight coefficients in the network. The training stops when either the set number of iterations is reached or the error falls below a set threshold.

2.2.2. CNN

In the work of [23, 39, 40] the convolutional neural network is proposed. The CNN combines three architectural ideas to some degree of shift, scale, and distortion invariance: *local receptive fields*, *shared weights*, and *sub-sampling* [40].

The *local receptive field* is a small area, e.g., a 5 by 5 pixel area of the input image, connected to a neuron, such that each neuron has 25 inputs [40]. The receptive fields may overlap, to cover the entire input image, resulting in a layer of neurons whose inputs are its corresponding area. The set of outputs of the neuron, i.e., pre-activation followed by the activation function, is a *feature map* [40]. In addition, the set of neurons have an identical set of weight coefficients, i.e., *shared weights*. With the local receptive fields, neurons can extract visual *features*, e.g., oriented edges, end-points and corners, and store them in the feature map [40]. Other feature maps are generated, using a different set of weight coefficients, extracting different types of features. In Figure 2.5(a) the architectural ideas of local receptive fields and shared weights is depicted. In practice, to generate a feature map, the input image is scanned using the local receptive field, and the pre-activation is computed. This operation is equivalent to a convolution, followed by adding a bias and activation function, defined by [23],

$$\vec{y}_i = f(\vec{\xi}_i) \tag{2.8}$$

$$\vec{\xi}_i = b_i + \sum_j \vec{k}_{ij} \circledast \vec{x}_j \tag{2.9}$$

where $\vec{y}_i$ is the set of neuron outputs (feature map), f the activation function, $\vec{\xi}_i$ the set of neuron pre-activations, b_i the bias, $\vec{k}_{ij}$ the kernel (set of weight coefficients), and $\vec{x}_j$ the input.

The absolute location of a detected feature is of less importance, only the approximate position relative to other features [40]. E.g., if the features of an image contain an end point of a horizontal segment in the upper left area, a corner in the upper right area, and an end point of a vertical segment in the lower area, the image represents the number 7 (see Figure 2.5(a)). Reducing the dependence on

the location of features, is to reduce the spacial resolution of the feature map, using *sub-sampling* [40] or *pooling* [23]. The sub-sampling/pooling operation is depicted in Figure 2.5(b). The feature map is scanned using a defined area and a stride (number of steps to the next area). In this case, the area is 2×2 pixels and the stride is 2. This operation can be performed by taking the average of the area, i.e., average pooling, or taking the maximum value of the area, i.e., max pooling [23].

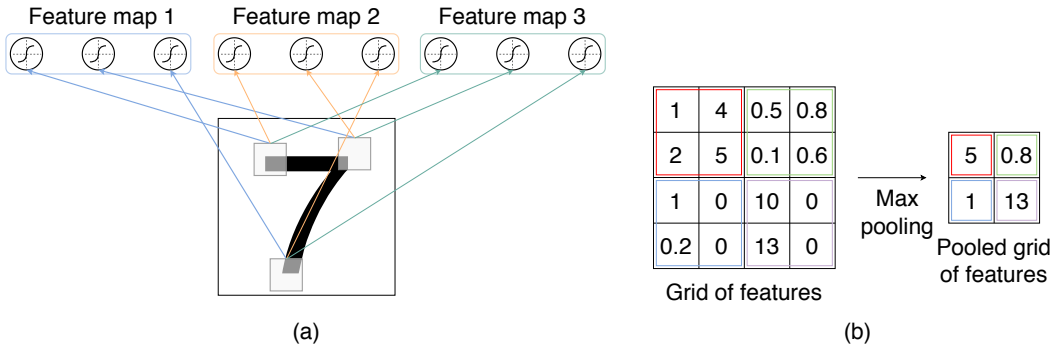


Figure 2.5: Three architectural ideas: (a) local receptive fields and shared weights; (b) max sub-sampling/pooling.

A typical convolutional network is depicted in Figure 2.6. The network comprises of 6 layers, excluding the input, all containing trainable parameters, i.e., weight coefficients. The convolutional layers are labeled C1 and C2, the sub-sampling layers are labeled S1 and S2, and the fully-connected layer is labeled F1.

C1 is a convolutional layer with 8 feature maps. Each neuron in each feature map is connected to a local receptive field in the input image. The resolution of each feature map is smaller than the input image resolution.

S1 is a sub-sampling layer with 8 feature maps with half the resolution of the C1 feature maps. E.g., an sub-sampling/pooling area of 2×2 with stride 2, results in non-overlapping sub-sampling, reducing the number of rows and column of the feature maps by half.

C2 is a convolutional layer with 16 feature maps. Each neuron in each feature map is connected to a local receptive field in the S1 feature maps.

S2 is a sub-sampling layer with 16 feature maps with half the resolution of the C2 feature maps. The sub-sampling/pooling operation can be performed again using an area of 2×2 with stride 2, reducing the resolution by half.

F1 is a fully-connected layer, similar to the MLF neural network described in Section 2.2.1.

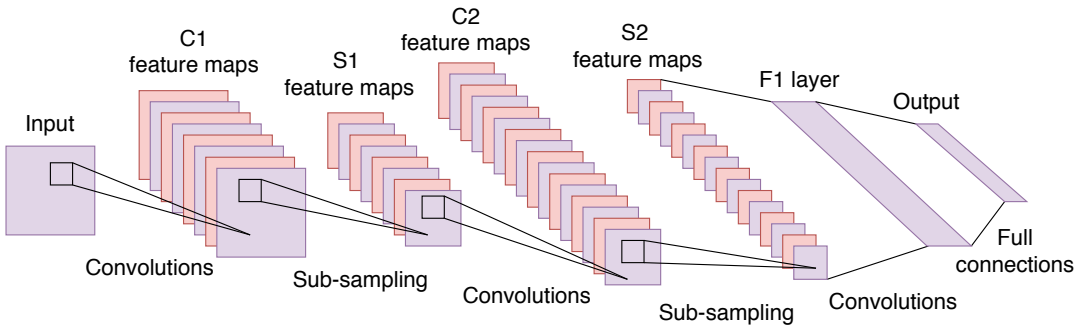


Figure 2.6: Typical convolutional neural network with two feature stages (adapted from [23]).

Upconvolution

Upconvolution, also called transposed convolution [48] or deconvolution [49], is a transformation going in the opposite direction of a normal convolution, e.g., to increase the resolution of a feature map. The upconvolution operation is depicted in Figure 2.7. The kernel is convolved over a zero padded input, obtain a higher resolution output.

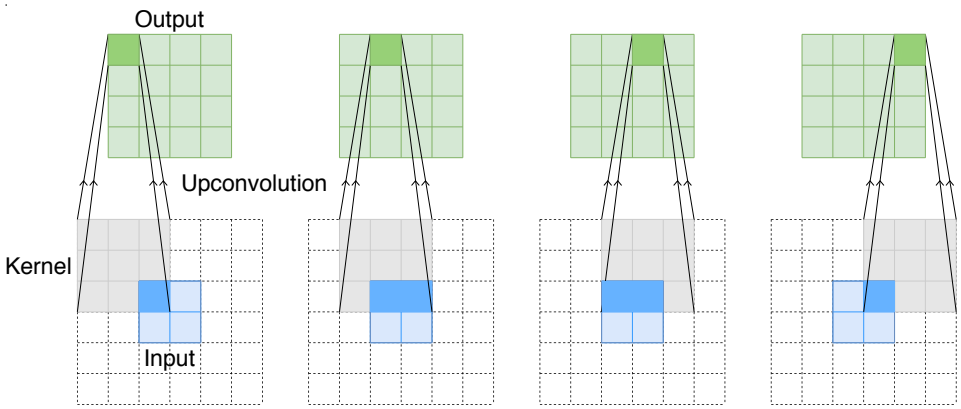


Figure 2.7: Upconvolution operation: convolution of a 3 × 3 kernel over a 2 × 2 input with a 2 × 2 border of zeros (adapted from [48]).

2.3. Stereo Matching Pipelines

We focus on stereo matching algorithms that produce *dense* disparity maps, i.e., a disparity estimate at each pixel of the stereo images [1]. Dense depth maps are then generated from dense disparity maps using triangulation [28]. As opposed to using feature-based algorithms [31] stereo matching algorithms, producing sparse disparity maps, which is insufficient for reconstructing detailed 3D terrains [32]. Dense stereo matching algorithms can be divided into two pipelines based on, *traditional* methods [1] and *deep learning* methods [22]. The traditional pipeline in Figure 2.8 depict four steps [1]:

1. Matching cost computation: a cost is associated for a pixel in the left image with a pixel in the right image, for all pixels along the *epipolar* line, i.e., a horizontal line across the rectified stereo image pair, intersecting the pixel of interest;
2. Cost (support) aggregation: uncertainty of matching is minimized by aggregating the matching cost over a *support region*, i.e., a window centered on the pixel of interest. This step is performed by *local* methods;
3. Disparity computation/optimization: for *local* methods, the final disparities are computed by selecting, at each pixel, the disparity associated with the minimum cost value. For *global* methods, the final disparities are computed by finding a disparity function that minimizes a global energy;
4. Disparity refinement: disparity maps are improved by using regularization and interpolation.

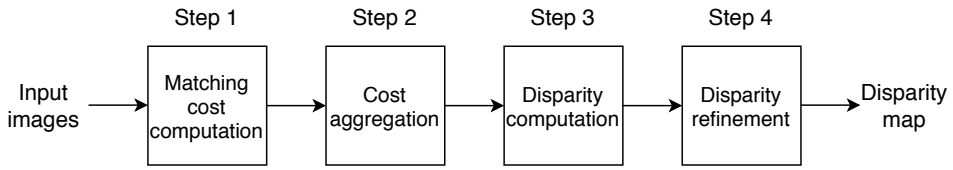


Figure 2.8: Traditional stereo matching pipeline (adapted from [50]).

Traditional methods can be classified into *local*, *global*, and *semi-global* matching methods [2, 32]. In local (window-based) methods [1], the disparity computation at a given pixel depends only on the intensity values within a predefined window. Local algorithms are characterized by their fast execution time, however the quality of the disparity map is low in textureless and repetitive regions [50]. Most of the local methods can be cleanly divided into steps 1, 2, 3 and 4 (Figure 2.8). A local method, sum of absolute differences (SAD) [28], has been used in the MER missions [4] and ExoMars mission [6]. Other local methods [1, 2, 50] include: rank transform (RT) [51], census transform (CT) [51] and bidirectional matching (BM) [27].

Global methods [1], compute the disparities using a disparity function. As opposed to local algorithms, global algorithms generate more accurate disparity maps in low texture and repetitive regions,

however they suffer from a slow execution time due to high computational complexity. The disparity function is obtained by minimizing a global energy function consisting of two terms, i.e., a data term expressed as a matching cost (step 1) and a smoothness term to retain smoothness in disparity among pixels in the same region. Typically, the global methods skip the cost (support) aggregation step (step 2), thus containing three steps. Examples of global methods are: dynamic programming (DP) [52], graph cuts (GC) [53] and belief propagation (BP) [54].

In semi-global matching (SGM) [26], steps from local and global methods are combined. Semi-global matching algorithms, are the middle ground between local and global algorithms. Their performance in textureless and repetitive regions, and the execution time is a combination of local and global algorithm performances. In the matching cost computation step (step 1), a matching cost, e.g., MI [55], SAD, CT, is used. In the cost aggregation step (step 2), the matching uncertainty is minimized by a global energy function [26]. The other steps are similar to the ones of the local methods. SGM has been used in the CE-3 mission [7].

Deep learning methods can be classified into *non-end-to-end learning*, *end-to-end learning*, and *unsupervised learning* methods [22]. Non-end-to-end learning methods use CNNs to replace one or more steps in the traditional stereo matching pipeline [22]. Examples of these methods are MC-CNN [56] and content-CNN [57], where the matching cost is computed (step 1) using neural networks. Other methods, e.g., SGM-Nets [58] and DRR [59], replace the cost aggregation step (step 3) and disparity refinement step (step 4), respectively.

End-to-end learning methods integrate all steps from the traditional stereo matching pipeline, producing dense disparity maps from stereo images directly [22]. Examples of these methods are DispNet [25], GC-Net [60] and GwcNet [61].

Unsupervised learning methods, similar to end-to-end learning methods, integrate all steps from the pipeline, producing dense disparity maps directly. However, require no ground truth depth data [22], as opposed to the other two deep learning methods. The work of [62] marked the maturity of the unsupervised approach and other approaches have been based on their proposed structure.

2.3.1. Traditional Stereo Matching Pipeline

From the building blocks arranged in four steps (see Figure 2.8), the traditional stereo matching algorithms are constructed. Generally, the following four steps are performed [1]:

1. Matching cost computation;
2. Cost (support) aggregation;
3. Disparity refinement.

Traditional methods can be divided into *local*, *global*, and *semi-global* matching methods [2, 32]. Local methods, compute the disparity at a given pixel based on the intensity values within a predefined window [1]. Most of these methods can be cleanly divided into the four steps, depicted in Figure 2.8. Global methods, compute the disparity function by minimizing a global energy function and typically skip the cost (support) aggregation step (step 2) [1]. Semi-global methods, combine steps from the local and global methods, where the cost (support aggregation) step is performed by minimizing a global energy function.

In Table 2.1 several methods are divided into the pipeline steps. The disparity refinement step (step 4) is applicable for all listed methods and is, therefore, not listed.

Method	Matching cost	Cost aggregation	Disparity computation
SSD (local)	SD	Square window	WTA
SAD [28] (local)	AD	Square window	WTA
DP [52] (global)	AD	-None-	DP
GC [53] (global)	AD	-None-	GC
SGM [26] (semi-global)	MI [55]	Paths	WTA

Table 2.1: Breakdown of several matching methods. Abbreviations: SSD – sum of squared differences, SAD – sum of absolute differences, DP – dynamic programming, GC – graph cuts, SGM – semi-global matching, SD – squared difference, AD – absolute difference, MI – mutual information, WTA – winner-take-all (adapted from [1, 2]).

Matching cost computation
The similarity of pixels at corresponding locations (x_0, y_0) and (x_1, y_1) can be expressed as a *cost* and are stored in a *disparity space image* (DSI) $C(x, y, d)$ for further processing [35]. The matching cost can be computed using, e.g., the sum of absolute differences (SAD) [28] algorithm, which considers the absolute difference (AD) [28] between the intensity of each pixel in the reference image I_0 and that of the corresponding pixel in the target image I_1 (Eq. 2.10) [50].

$$SAD(x, y, d) = \sum_{(x,y)\in w} |I_0(x, y) - I_1(x - d, y)| \tag{2.10}$$

where x and y are the possible pixel coordinates inside a support region w, and d the disparity. The matching cost values over all pixels and all disparities form the initial disparity space image $C_0(x, y, d)$ [1].

Cost (support) aggregation
For local methods, the matching cost is aggregated by averaging or summing over a *support region* in the DSI [1], with the support regions being either two-dimensional in the (x, y) at a fixed disparity, or three-dimensional in the (x, y, d) space [1]. The aggregated cost is computed using 2D or 3D convolution, depending on the chosen support region [1],

$$C(x, y, d) = w(x, y, d) \circledast C_0(x, y, d) \tag{2.11}$$

where w is the support region, x and y the pixel coordinates, d the disparity, and C_0 the initial DSI.

In semi-global methods, the matching costs are aggregated symmetrically from all directions in the DSI [26, 63], depicted in Figure 2.9. Eight paths from different directions $\vec{r}$ meet at the pixel $\vec{p}$, where for each pixel and each disparity d, the costs are summed over the eight paths [63].

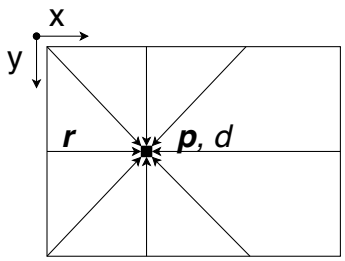


Figure 2.9: Aggregation of costs in disparity space (adapted from [63]).

Disparity computation
In local methods and semi-global methods [26], the final disparities are computed by selecting at each pixel the disparity associated with the minimum cost value [1], i.e., using the winner-take-all (WTA) strategy, defined by [50],

$$d_{x,y} = \arg\min_{d\in D} C(x, y, d) \tag{2.12}$$

where d is the disparity in the set of allowed disparities D, and $C(x, y, d)$ the aggregated cost.

In global methods, the objective is to find a disparity function $d = d(x, y)$ [50] that minimizes a global energy, described by [1],

$$E(d) = E_{data}(d) + \lambda E_{smooth}(d) \tag{2.13}$$

where the data term $E_{data}(d)$ represents the matching costs for all pixel coordinates (x, y) [50], i.e. [1],

$$E_{data}(d) = \sum_{(x,y)} C(x, y, d(x, y)) \tag{2.14}$$

where C is the aggregated cost. The weighting factor is λ and the smoothness term $E_{smooth}(d)$ encourages neighboring pixels to have similar disparities based on smoothness assumptions made by the algorithm, i.e. [1],

$$E_{smooth}(d) = \sum_{(x,y)} \rho(d(x, y) - d(x - 1, y)) + \rho(d(x, y) - d(x, y - 1)) \tag{2.15}$$

where ρ is a monotonically increasing, i.e, a non-decreasing [64], function of disparity difference.

Disparity refinement
The disparity refinement step is used to reduce noise and improve the disparity map by regularization and interpolation [50]. For regularization, Gaussian convolution is used to estimate disparities in combination with those of neighboring pixels in accordance with weights defined by a Gaussian distribution, from which noise in the disparity map reduced, however also reducing the amount of fine detail present [50]. For interpolation, a median filter is used to remove small, isolated mismatches in disparity map by selecting the median value within a window of pixels as the final result for the central pixel [50].

2.3.2. Deep Learning Stereo Matching Pipeline

Deep learning methods can be classified into three categories [22]:

1. Non-end-to-end learning methods;
2. End-to-end learning methods;
3. Unsupervised learning methods.

The main building block for deep learning methods is the convolutional neural network (CNN) [23]. The CNN is described in Section 2.2.

Non-end-to-end learning methods
Non-end-to-end learning methods replace one or more steps in the traditional stereo matching pipeline using CNNs [22]. The SGM-Nets [58] is a network proposed to replace the cost (support) aggregation step (step 2), where the penalties are tuned using a network with CNNs. For the disparity refinement step (step 4), DRR [59] is proposed to detect, replace and refine erroneous disparity predictions. The most commonly replaced step, however, is the matching cost computation step (step 1). In the works of [56, 57, 65], a Siamese network, i.e., two sub-networks joined at the head, is proposed to replace the matching cost computation step (step 1) of the traditional pipeline. The network is depicted in Figure 2.10. The left and right stereo image, $\vec{I}_0$ and $\vec{I}_1$, with the patches denoted as $\vec{I}_0(\vec{p})$ and $\vec{I}_1(\vec{p}-\vec{d})$, centered at $\vec{p}_0 = (x, y)$ and $\vec{p}_1 = (x - d, y)$, are the input of the network [65]. The sub-networks are composed of a number of convolutional layers with rectified linear units (ReLu) following all but the last layer [56], which extract the features of both image patches (feature maps). The extracted features are joined by performing the inner-product, producing a similarity score S, defined as [65],

$$S(\vec{p}, \vec{d}) = < f(\vec{I}_0(\vec{p})), f(\vec{I}_1(\vec{p} - \vec{d})) > \tag{2.16}$$

where $f(\vec{I})$ is the extracted feature of the image patches $\vec{I}_0(\vec{p})$ and $\vec{I}_1(\vec{p} - \vec{d})$.

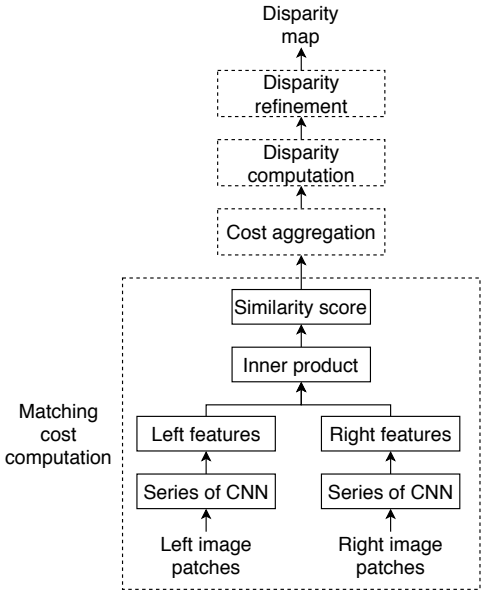


Figure 2.10: Siamese matching network (adapted from [22]).

End-to-end learning methods

End-to-end learning methods integrate all four steps in the traditional stereo matching pipeline for a joint optimization, producing dense disparity maps from stereo images directly [22]. The methods can be classified into two structures: 2D encoder-decoder and 3D regularization. Figure 2.11 depicts the two structures.

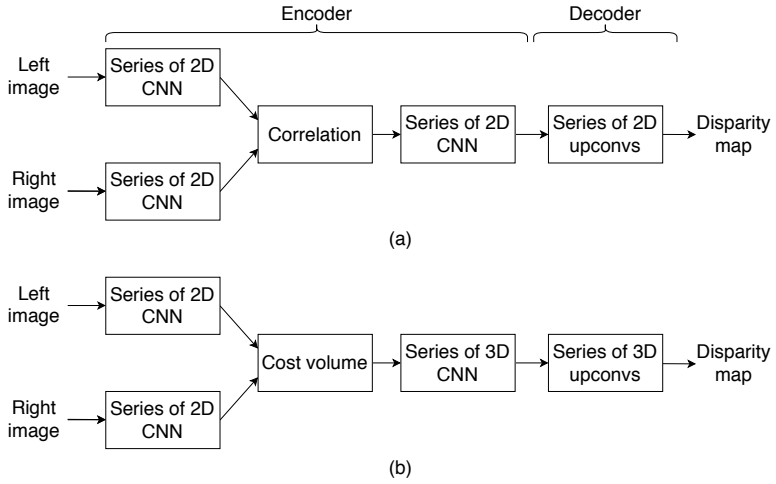


Figure 2.11: End-to-end architectures: (a) 2D encoder-decoder; (b) 3D regularization (adapted from [22, 66]).

In the work of [66], FlowNet, an encoder-decoder network is proposed, depicted in Figure 2.11(a). The *encoder*, i.e., the contractive part of the network, compresses information. The left and right images are passed into two separate, yet identical, processing streams consisting of a series of 2D CNNs and are combined in a later stage. With the two streams, the features are extracted from the left and right image, similar to the approach in the non-end-to-end methods. The matching process is performed in the *correlation layer*, where patches between two feature maps are compared, defined as [66],

$$c(\vec{x}_1, \vec{x}_2) = \sum_{\vec{o} \in [-k,k] \times [-k,k]} < \vec{f}_1(\vec{x}_1 + \vec{o}), \vec{f}_2(\vec{x}_2 + \vec{o}) > \tag{2.17}$$

where $\vec{f}_1$ and $\vec{f}_2$ are the extracted features (feature maps) from the left and right images, $\vec{x}_1$ and $\vec{x}_2$ the

centers of the patches from $\vec{f}_1$ and $\vec{f}_2$, $\vec{o}$ a single pixel of the patch, and k the size of the patch. The correlation layer, effectively, convolves the patches in the feature maps.

The *decoder*, i.e., the expanding part of the network, refines the information. In the refinement process, depicted in Figure 2.12, upconvolution (transpose convolution) (see Section 2.2) is applied to the feature maps and are concatenated with the corresponding feature maps from the encoder part of the network. The concatenation preserves both the high-level information passed from coarser feature maps, in the encoder part, and fine local information provided in lower layer feature maps, in the decoder part.

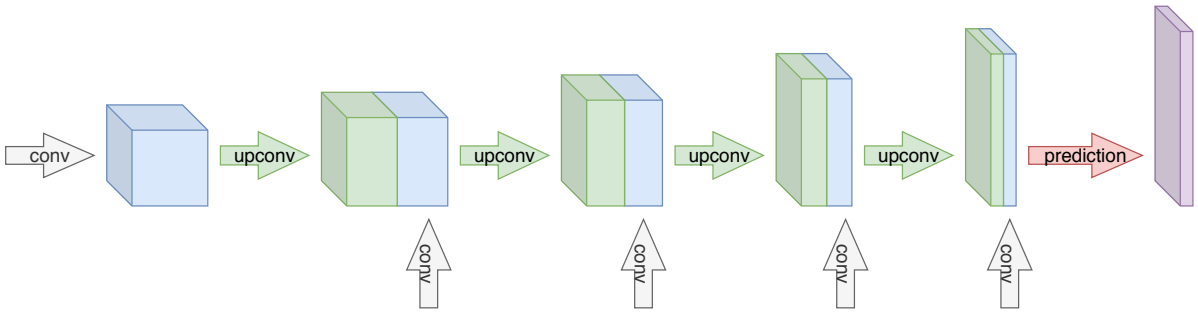


Figure 2.12: Decoder part: refinement of the coarse feature maps to the high resolution prediction of the disparity map. Abbreviations: conv – convolution, upconv – upconvolution (adapted from [66]).

Based on [66] several networks are proposed: DispNet [25], Cascade Residual Learning (CRL) [67] and iResNet [68]. Other networks include: EdgeStereo [69] and SegStereo [70].

In the work of [60], GC-Net, a 3D regularization network is proposed, depicted in Figure 2.11(b). Similar to the encoder-decoder network, the features of the left and right images are extracted through a series of 2D CNNs. The matching process of performed by forming a cost volume. The operation for the cost volume construction is depicted in Figure 2.13. The left feature map is set as the base feature map and the right feature map as the shift feature map [71]. The shift feature map slides, along the width, with an offset of 1 pixel in the feature map for each increasing disparity and is concatenated to the left feature map [72]. The resulting cost volume is of dimensionality $height \times width \times (max\ disparity + 1) \times feature\ size$, i.e., a 4D volume [60]. The cost volume is passed into a series of 3D CNNs to extract features from the height, width and disparity dimensions. Lastly, a series of 3D upconvolutions is applied to scale the feature map to the original input resolution.

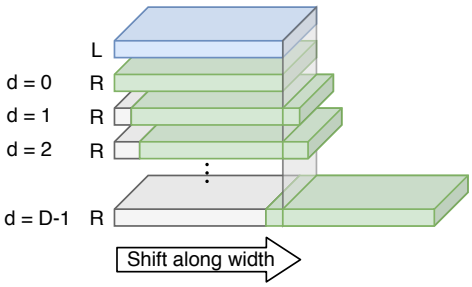


Figure 2.13: Cost volume construction operation. Abbreviations: L – left feature map, R – right feature map (adapted from [71, 72]).

Based on [60] several networks are proposed: PSMNet [73], SCV-Net [74] and PDS [75].

Unsupervised learning methods

Unsupervised learning methods, similar to end-to-end learning methods, integrate all steps from the pipeline, producing dense disparity maps directly. However, require no ground truth depth data [22], as opposed to the other two deep learning methods. These methods rely on minimizing *warping* error to drive the network in an unsupervised way [22]. In the work of [76], an unsupervised network is proposed, depicted in Figure 2.14. The network consists of three parts: encoder, decoder and loss. In

the *encoder* part, the left image $\vec{I}_0(x,y)$ is passed to a deep CNN network, based on the AlexNet [77] architecture, producing a predicted disparity map $\vec{D}(x,y)$ for the left image.

The *decoder* part, takes the predicted disparity map and right image $\vec{I}_1(x,y)$ and generates a warped image $\vec{I}_w(x,y)$, i.e., the reconstructed left image, defined as,

$$\vec{I}_w(x,y) = \vec{I}_2(x + \vec{D}(x,y), y) \tag{2.18}$$

where pixels from the right image are shifted along the epipolar lines.

In the *loss* part, the reconstructed image $\vec{I}_w(x,y)$ is matched with the left image $\vec{I}_0(x,y)$ via a reconstruction error, defined as,

$$Loss = \|\vec{I}_w(x,y) - \vec{I}_0(x,y)\| \tag{2.19}$$

Minimizing the reconstruction error results in an accurate predicted disparity map.

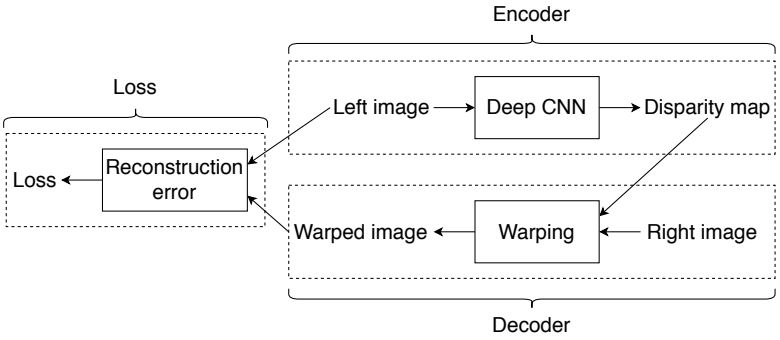


Figure 2.14: Unsupervised matching network (adapted from [76]).

Based on [76] several networks are proposed: [62] and [78].

3

ExoMars

In this chapter we describe the ExoMars rover. The rover serves as vehicle from which we derive our requirements and test bed (see Chapter 6). The ExoMars rover, developed by the European Space Agency (ESA), will land on the Martian surface with the aim of establishing if life ever existed on Mars. To maximise the number and variety of sites visited by the mission, the rover will be fitted with a mobility subsystem designed to allow it to traverse terrain without real-time assistance from Earth [79].

The remainder of this chapter is structured as follows. The hardware of the rover is described in Section 3.1. The mobility system architecture is described in Section 3.2. The perception system architecture is described in Section 3.3. In Section 3.4, we formulate the requirements based on the ExoMars rover.

3.1. Hardware

The ExoMars rover is depicted in Figure 3.1. It features a locomotion system with six wheels, each wheel is able to actuate independently. The mass of the rover is approximately 300 kg and its dimensions, when fully deployed, is comparable to a small car. It is solar powered and designed to carry out operations on Mars, traversing several kilometers during its mission. The rover features a drill capable of drilling down a depth of 2 meters to obtain a sample. The front mast holds the localization camera (LocCam) and navigation/panoramic camera (NavCam/PanCam) [79]. These two stereo cameras, i.e., the LocCam and PanCam, are of the same design. The baseline of the camera is 15 cm, the focal length is 4 mm, the resolution is 1024 x 1024 pixels, with a FOV is 65° [80].

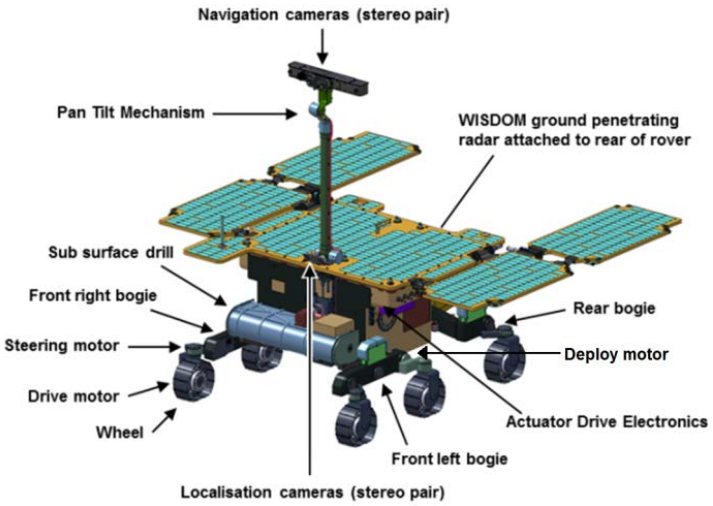


Figure 3.1: ExoMars rover [79]

3.2. Mobility System Architecture

To traverse terrain without real-time assistance from Earth, the ExoMars rover uses a mobility system [80] and its architecture is shown in Figure 3.2. The system has several inputs, the command module generates an x-y coordinate as the target for the rover. The localization module, produces the position and attitude, with the LocCam and IMU as its input. The navigation module, takes the PanCam images as its input and processes the images via a preparation, perception, and terrain evaluation pipeline. A navigation map is the output of the module. The path planning module generates a path from the navigation map, position, attitude, and target. The locomotion module, takes the position, attitude, and path sequence to drive the actuators of the wheels.

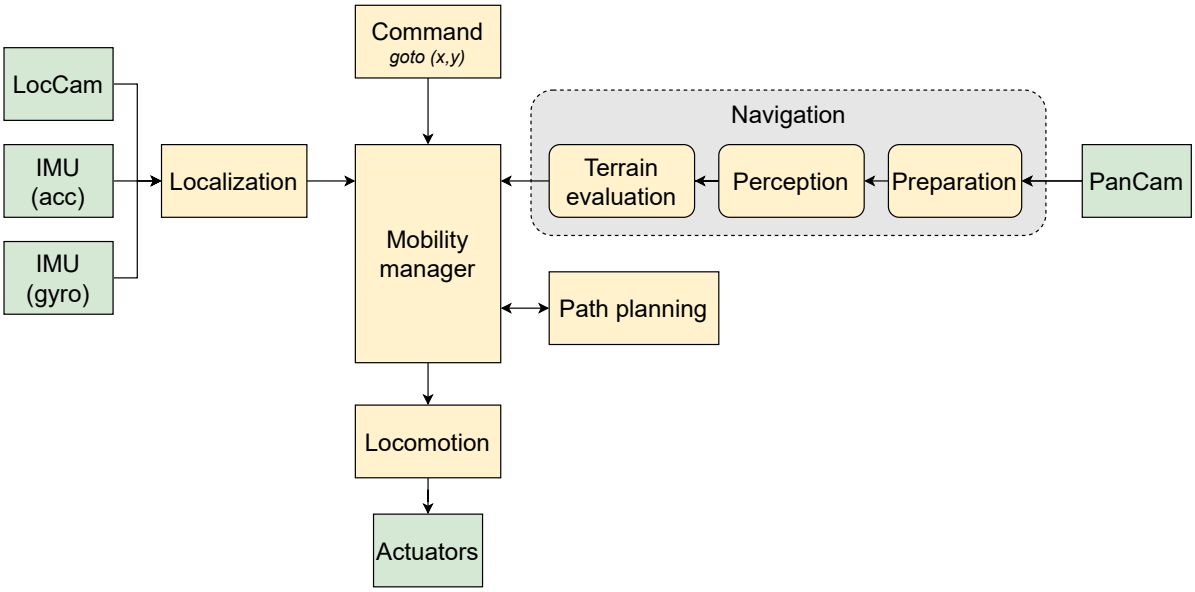


Figure 3.2: Mobility system architecture (simplified) (adapted from [80]).

3.3. Perception System Architecture

In this thesis we focus on the perception system. The system is part of the pipeline residing in the navigation module (see Figure 3.2). The role of the perception system is to analyse a pair of stereo images and produce a disparity map [6]. Figure 3.3 depicts the perception system architecture. The raw stereo image from the PanCam unit is the input for the navigation module. In the preparation module, the raw stereo image is transformed to a rectified stereo image. Using the rectified stereo image, the perception module generates a predicted disparity map using a stereo matching algorithm. In [6], the predicted disparity map are tested using simulated stereo image. However, these simulated stereo images are not publicly available, thus we use the Katwijk Beach Planetary Rover dataset [21] (see Chapter 6.4).

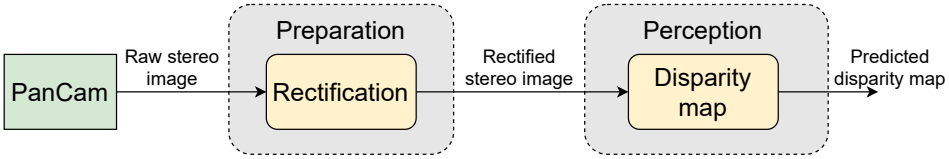


Figure 3.3: Perception system architecture (simplified) (adapted from [6]).

3.4. Requirements

In this work we need to develop requirements to generate the metrics (see Chapter 6.5) and validate our results (see Chapter 7). The ExoMars rover must satisfy the following requirements [6, 80, 81],

- **R1: The perception system must produce a disparity map accurate enough to adequately represent a specified region of terrain such that the navigation system can determine its**

safety. Approximately 6 meters of terrain in front of the rover must be sufficiently characterised by the output disparity map.
That is, the region of interest is a terrain of approximately 6 meters in front of the rover. Regions outside this terrain are not of interest and do not need to be evaluated.

- **R2: The perception system must execute fast enough on the flight hardware in order to meet system level requirements for the distance the rover must be able to drive during one sol. Each call of the perception system must take less than 20 seconds to run on the ExoMars Rover 96MHz LEON2 co-processor.**
  The time it takes to produce a predicted disparity map from a rectified image (see Figure 3.3) must take less than 20 seconds. The time is independent of the hardware, as it is derived from the mission duration, final target distance, and operating time per Martian day.
- **R3: The rover must maintain a position estimate accurate to within 10% of distance travelled.**
  E.g., if the rover traversed 1 meter forward, the target must be within a radius of 10 cm.
- **R4: The perception system must use less than 512 MiB of memory.**
  The 96MHz LEON2 co-processor has 512 MiB of RAM. To facilitate the memory usage requirement, we set the memory usage limit to maximum capacity.

4

Algorithm Selection

In this chapter we address our first research question: How to obtain a manageable selection of stereo matching algorithms for comparison? Our approach is first, the formulation of a set of selection criteria from literature and ExoMars requirements. Second, we search for algorithms from the two stereo vision datasets, Middlebury [30] and KITTI 2015 [29], which provide a ranking of stereo matching algorithms. Third, we reduce the list of algorithms to a manageable set by applying the formulated selection criteria.

The remainder of this chapter is structured as follows. The algorithm selection process is described in more detail in Section 4.1. In Section 4.2, the selection criteria are formulated. In Section 4.3, the benchmark performances are presented. The draft set of algorithms is generated in Section 4.4.

4.1. Selection Process

The algorithm selection process, depicted in Figure 4.1, consists of a number of components, including the data generation, benchmark performance, and selection module.

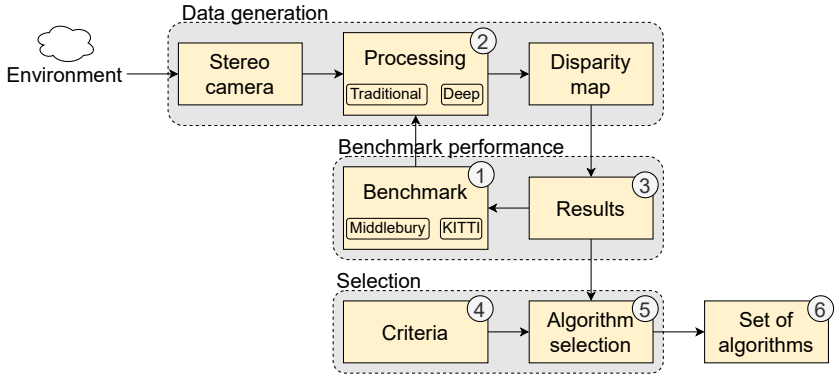


Figure 4.1: Algorithm selection process.

As input for the data generation module, the Middlebury and KITTI 2015 stereo benchmarks are used (1). These benchmarks provide test stereo images of indoor objects and dynamic outdoor driving scenes, respectively. The test images are passed to the processing component of the data generation module (2). This component includes the implementation of traditional (e.g., SGM) and deep learning (e.g., DispNet) stereo matching algorithms, each producing a predicted disparity map.

The predicted disparity map is verified for correctness, i.e., the disparity error is computed, according to the ground-truth (3). In addition, the results include the run time of the stereo matching algorithm in an environment, i.e., the processor used. Based on the disparity error, the algorithms are ranked in the benchmarks.

To aid the selection of algorithms, criteria are formulated based literature and the ExoMars requirements (4). The stereo matching algorithm results are evaluated against the set of criteria to narrow

down the large list of algorithms (5). This reduction of algorithms results in a set of algorithms for the implementation of the single-board computer (6).

4.2. Selection Criteria

The selection of a set of algorithms is the result of a list of algorithms, responding to a set of selection criteria. In this section we discuss the main criteria for the selection of algorithms.

(C1) Run time: the algorithm selection process must make use of the stereo matching run time as one of its selection criteria. From the ExoMars requirement, (R2) each call of the perception system must take less than 20 seconds to run on the ExoMars Rover 96MHz LEON2 co-processor. That is, the generation of a disparity map must take less than 20 s. In the perception pipeline, before generating the disparity map, a rectification step is performed. However, to facilitate the formulation of the run time, we neglect the time it takes to rectify the images, since the run time of the rectification step is in the order of milliseconds [4]. In addition, the run time of the algorithms in the Middlebury and KITTI 2015 stereo benchmarks are based on already rectified images. Thus, we set our maximum run time to 20 seconds.

(C2) Error rate: the algorithm selection process must make use of the error rate as one of its selection criteria. For the Middlebury and KITTI 2015 stereo benchmarks the bad matched pixels (BMP) percentage is used as the error measure. In these two datasets a lower BMP percentage increases the ranking of an algorithm. We set the error rate selection criteria equal to the BMP percentage. From the works of [1], a BMP < 10% is considered as performing *reasonably well*. To facilitate the formulation of the error rate, we use this 10% as our maximum allowable error rate.

(C3) Diverse, representative stereo algorithms: the list of selected algorithms must be broad, covering most, if not all, categories of each pipeline. In particular, the selection must not only be based on the lowest C1 and C2, which could be dominated by a single stereo matching category, e.g., a local stereo matching algorithm. Without representativeness, the list of selected algorithms could bias a single category and neglect the advantages of other categories. In the draft set of selected algorithms we include all three categories of the traditional pipeline, local, semi-global, and global. For deep learning algorithms we include one of the three categories, non-end-to-end, end-to-end, unsupervised.

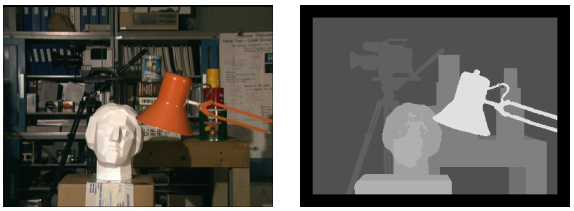
(C4) Deep learning library: deep learning algorithms are implemented using a deep learning library. Various libraries exist, such as Tensorflow [82] (Google), PyTorch (Facebook), Caffe (Berkeley AI Research). TensorFlow achieves a minimum average speedup of 2.87×, in terms of inference, over PyTorch and Caffe on various deep neural network models executed on a Raspberry Pi 3 Model B [83]. We opt to use the TensorFlow library, because of the lowest inference time.

4.3. Benchmark Performances

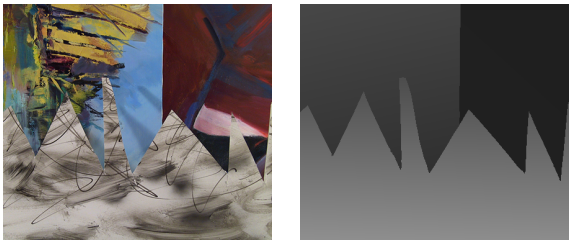
We make use of two stereo vision benchmarks, the Middlebury and KITTI 2015 stereo benchmarks, described in Section 4.3.1. These two stereo vision datasets are not representative of the Martian environment, however as far as we know, there is no ranking of algorithms based on a (simulated) Martian dataset, such as the Katwijk Beach Planetary Rover dataset [21] (described in Chapter 6.4.1). For this reason, we obtain algorithms from these two datasets, even though there is a lack of representativeness. The lists of algorithms are presented in Section 4.3.2.

4.3.1. Datasets

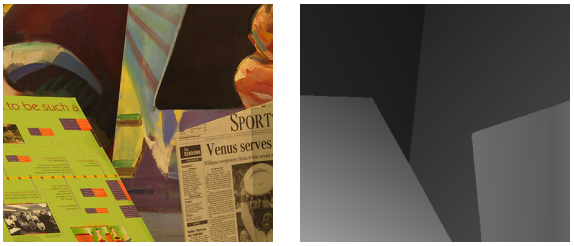
The images in the Middlebury dataset are static indoor scenes with varying difficulties. They include repetitive structures, occlusions, and untextured areas. The Middlebury stereo dataset contains different datasets depending on the version. In Section 4.3.2, the 2014 datasets are used for the rankings on the benchmark (Table 4.1), while in both surveys (Table 4.2) the 2001 datasets are used. The 2001 datasets are depicted in Figure 4.2. On the left, the left stereo pair image, and on the right, the ground-truth disparity map.



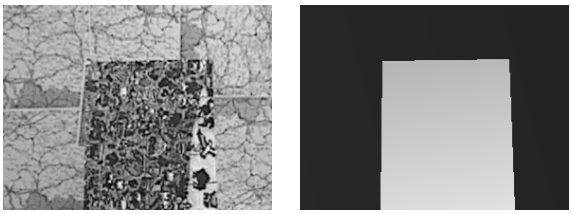
(a) Tsukuba: (left) reference; (right) ground-truth.



(b) Sawtooth: (left) reference; (right) ground-truth.



(c) Venus: (left) reference; (right) ground-truth.



(d) Map: (left) reference; (right) ground-truth.

Figure 4.2: Middlebury 2001 stereo datasets.

The KITTI 2015 stereo benchmark is a dataset for 3D depth estimation with an application to autonomous driving. The dataset contains 400 annotated, i.e., with ground-truth, outdoor scenes in a city setting. An example of outdoor scenes with the ground-truth is depicted in Figure 4.3. On the left, the left stereo camera image is shown, and on the right, the ground-truth LIDAR scan.

(a) Outdoor scene: (left) reference; (right) ground-truth.

(b) Outdoor scene: (left) reference; (right) ground-truth.

Figure 4.3: KITTI 2015 stereo dataset.

4.3.2. Algorithms

The stereo matching algorithms from the Middlebury and KITTI 2015 stereo benchmarks are listed in Table 4.1 and Table 4.3, respectively. The tables are sorted by BMP in ascending order. The listed algorithms are published methods with available code. We choose to select algorithms with available code to reduce the time allocated for the implementation of the algorithms.

In Tables 4.1 and 4.3 we observe for both benchmarks the dominating effect of deep learning methods, especially end-to-end (deep learning) methods. Traditional stereo matching algorithms are overtaken by deep learning algorithms, such that the Middlebury and KITTI 2015 benchmarks lack traditional algorithms, except for a single traditional global method in Table 4.1.

To obtain traditional local, global, and semi-global methods we refer to surveys using the older Middlebury 2001 benchmark, listed in Table 4.2. The first survey [1], has developed a taxonomy of traditional stereo matching algorithms, i.e., a local method and several global methods. The benchmark used is the 2001 version of the Middlebury stereo benchmark, which uses different image sets compared to the recent version. The second survey [2], provides a review of the first survey. The results

of the second survey are listed in Table 4.2. However, the run time is not provided in the survey and is therefore left blank.

The tables contain the following information:

- **Method**: stereo matching algorithm name taken from its corresponding literature or generated by the benchmark if the name is not given.
- **Category**: stereo matching pipeline category, described in Section 2.3.
- **BMP**: error measure used is the bad matched pixels percentage.
- **Run time**: time required to produce a predicted disparity map from *rectified* stereo images, measured in seconds.
- **Environment**: hardware used to execute the stereo matching algorithm. For deep learning algorithm, this also is the hardware used to train the neural network.
- **Library**: deep learning library or stereo vision library used for the implementation of the method.

Method	Category	BMP (%)	Run time (s)	Environment	Library
LocalExp [84]	TGLO	5.43	881	4 core @ 3.5 Ghz + NVIDIA Titan X	OpenCV
CBMV [85]	NETE	7.65	1001	6 core @ 3.0 Ghz	OpenCV
SDR [86]	NETE	7.69	15.3	1 core @ 2.6 GHz	OpenCV
MC-CNN-acrt [56]	NETE	8.08	150	Nvidia GTX Titan X	Torch
OVOD [87]	ETE	8.87	69.1	-	C++
MC-CNN-fst [56]	NETE	9.47	1.69	Nvidia GTX Titan X	Torch
LBPS [88]	NETE	9.68	1.05	GPU @ 2.5 Ghz	PyTorch
HSM [89]	ETE	10.2	0.51	Titan X Pascal	PyTorch
JMR [90]	ETE	12.5	4.46	-	OpenCV
FBS [91]	NETE	18.1	188	6 core @ 3.2 Ghz	Python
PSMNet [73]	ETE	28.9	0.64	Nvidia GTX Titan Xp	PyTorch
DeepPruner [92]	ETE	30.1	0.13	1 core @ 2.5 Ghz	PyTorch
DDL [93]	ETE	30.1	112	1 core @ 4.0 Ghz	Matlab/C
ELAS [94]	ETE	32.3	0.48	1 core @ 2.5 Ghz	C++

Table 4.1: Evaluation of stereo matching algorithms using the Middlebury Stereo Version 3 benchmark; retrieved September 2020. Abbreviations: ETE – end-to-end learning method, NETE – non-end-to-end learning method, TGLO – traditional global method.

First survey [1]									
Method	**Category**	**BMP (%)**				**Run time (s)**			
		Tsukuba	**Sawtooth**	**Venus**	**Map**	**Tsukuba**	**Sawtooth**	**Venus**	**Map**
SSD	Local	$5.23^{(4)}$	$2.21^{(3)}$	$3.74^{(2)}$	$0.66^{(3)}$	1.1	1.5	1.7	0.8
DP	Global	$4.12^{(2)}$	$4.84^{(5)}$	$10.10^{(5)}$	$3.33^{(5)}$	1.0	1.8	1.9	0.8
SO	Global	$5.08^{(3)}$	$4.06^{(4)}$	$9.44^{(4)}$	$1.84^{(4)}$	1.1	2.2	2.3	1.3
GC	Global	$1.94^{(1)}$	$1.30^{(1)}$	$1.79^{(1)}$	$0.31^{(2)}$	662.0	735.0	829.0	480.0
Bay	Global	$6.49^{(5)}$	$1.45^{(2)}$	$4.00^{(3)}$	$0.20^{(1)}$	1055.0	2049.0	2047.0	1236.0
Second survey [2]									
SSD	Local	$5.3337^{(3)}$	$2.2051^{(3)}$	$3.7441^{(2)}$	$0.6638^{(3)}$	-	-	-	-
DP	Global	$4.5472^{(2)}$	$4.3094^{(4)}$	$8.8257^{(5)}$	$0.7748^{(4)}$	-	-	-	-
SO	Global	$6.0406^{(4)}$	$4.4275^{(5)}$	$7.7776^{(4)}$	$1.1016^{(5)}$	-	-	-	-
GC	Global	$1.9184^{(1)}$	$1.4049^{(1)}$	$1.8324^{(1)}$	$0.3103^{(2)}$	-	-	-	-
Bay	Global	$6.4924^{(5)}$	$1.4533^{(2)}$	$4.0044^{(3)}$	$0.1973^{(1)}$	-	-	-	-

Table 4.2: Evaluation of traditional stereo matching algorithms from surveys. Ranking of each algorithm is denoted by the superscript.

Method	Category	BMP (%)	Run time (s)	Environment	Library
GA-Net [95]	ETE	1.81	1.8	NVIDIA Tesla P40	PyTorch
Stereo expansion [96]	ETE	1.81	2.0	NVIDIA GeForce GTX TITAN X	PyTorch
AcfNet [97]	ETE	1.89	0.48	NVIDIA GTX 1080Ti	PyTorch
CSN [98]	ETE	2.00	0.6	NVIDIA GTX 1080Ti	PyTorch
HD^3-Stereo [99]	ETE	2.02	0.14	NVIDIA TITAN Xp	PyTorch
AANet [100]	ETE	2.03	0.06	NVIDIA Tesla V100	PyTorch
GwcNet [61]	ETE	2.11	0.32	NVIDIA TITAN Xp	PyTorch
WSMCnet [101]	ETE	2.13	0.39	NVIDIA GTX 1070	PyTorch
HSM [89]	ETE	2.14	0.14	NVIDIA TITAN X Pascal	PyTorch
DeepPruner [92]	ETE	2.15	0.18	NVIDIA TITAN Xp	PyTorch
AutoDispNet [102]	ETE	2.18	0.9	NVIDIA GTX 1080Ti	TensorFlow
Bi3D [103]	ETE	2.21	0.48	NVIDIA Tesla V100	PyTorch
SENSE [104]	ETE	2.22	0.32	NVIDIA GeForce RTX 2080 Ti	PyTorch
SegStereo [70]	ETE	2.25	0.6	NVIDIA GTX TITAN Xp	Caffe
CFP-Net [105]	ETE	2.31	0.9	NVIDIA GeForce GTX TITAN X	PyTorch
PSMNet [73]	ETE	2.32	0.41	NVIDIA TITAN Xp	PyTorch
SCV-Net [74]	ETE	2.61	0.36	NVIDIA GTX 1080Ti	PyTorch
CRL [67]	ETE	2.67	0.47	NVIDIA GTX 1080	Caffe
FADNet [106]	ETE	2.82	0.05	NVIDIA Tesla V100	PyTorch
MBFNet [107]	ETE	2.96	0.05	NVIDIA GTX 2070	PyTorch
Fast DS-CS [108]	ETE	3.08	0.02	NVIDIA GTX 1080Ti	TensorFlow
RecResNet [109]	ETE	3.10	0.3	NVIDIA GeForce GTX TITAN X	TensorFlow
NVStereoNet [110]	ETE	3.13	0.6	NVIDIA TITAN Xp	TensorFlow
L-ResMatch [111]	ETE	3.42	48	1 core @ 2.5 Ghz	Torch
Displets [112]	NETE	3.43	265	>8 cores @ 3.0 Ghz	Matlab
LBPS [88]	NETE	3.44	0.39	GPU @ 2.5 Ghz	PyTorch
MC-CNN-acrt [56]	NETE	3.89	67	Nvidia GTX Titan X	Torch
Reversing-PSMNet [113]	UNS	4.06	0.41	1 core @ 1.5 Ghz	PyTorch
PRSM [114]	ETE	4.27	300	1 core @ 2.5 Ghz	Matlab
DispNetC [25]	ETE	4.34	0.06	Nvidia GTX Titan X	Caffe
MADnet [115]	ETE	4.66	0.02	NVIDIA GTX 1080Ti	TensorFlow
MC-CNN-WS [116]	NETE	4.97	1.35	1 core 2.5 Ghz + K40 NVIDIA	Torch
CBMV [85]	NETE	4.97	250	6 core @ 3.0 Ghz	OpenCV
PWOC-3D [117]	ETE	5.13	0.13	NVIDIA GTX 1080Ti	TensorFlow
OSF [118]	ETE	5.28	390	1 core @ 2.5 Ghz	Matlab
SPS-St [119]	ETE	5.31	2	1 core @ 3.5 Ghz	C++
PR-Sceneflow [120]	ETE	6.24	150	4 core @ 3.0 Ghz	Matlab
DeepCostAggr [121]	ETE	6.34	0.03	GPU @ 2.5 Ghz	Lasagne
MeshStereo [122]	ETE	8.38	87	1 core @ 2.5 Ghz	C++
ELAS [94]	ETE	22.16	0.09	NVIDIA GTX 1080 Ti	PyTorch
VSF [123]	ETE	26.38	7500	1 core @ 2.5 Ghz	C++

Table 4.3: Evaluation of stereo matching algorithms using the KITTI 2015 benchmark; retrieved September 2020. Abbreviations: ETE – end-to-end learning method, NETE – non-end-to-end learning method, UNS – unsupervised learning method.

4.4. Draft Set of Algorithms

In the previous section, we have generated lists of stereo matching algorithm from benchmarks and surveys. To facilitate the selection of a draft set of algorithms, we narrow down the list to the best performing algorithms using selection criteria C1, C2, C3, and C4. We call this the draft set of algorithms instead of the final set, because the set is subject to change.

Traditional Algorithm Selection

For the selection of algorithms from the traditional pipeline, we apply selected criteria C1, C2, and C4. We select the SSD algorithm as the local algorithm and the DP algorithm as the global algorithm. For

the semi-global algorithm, we select the semi-global matching [26] (SGM) algorithm, because this is the only one in its category.

Deep Learning Algorithm Selection
For the selection of algorithms from the deep learning pipeline, we first must normalize the deep learning algorithms, i.e., evaluate the run time on the same environment using the same library. In Table 4.3, we observe different environments (e.g., NVIDIA Tesla P40, NVIDIA TITAN Xp, NVIDIA GTX 1080Ti, etc.) and libraries (e.g., PyTorch, TensorFlow, Caffe, etc.).

From C4, that is using the Tensorflow library, the number of algorithms is reduced to six, focusing solely on Table 4.3. Two of the six algorithms, Fast DS-CS and MADnet, have the lowest run time in the list of deep learning algorithms.

To investigate how well each of the six methods performs, the methods should execute on the same environment, such that there is a fair comparison. Several options are proposed and we discuss the advantages and disadvantages of each option. (1) Execute the methods on the same environment, e.g., an NVIDIA TITAN Xp. Advantage: accurate comparison of run times; disadvantage: high cost to obtain a GPU. (2) Execute the methods on a laptop without GPU support, i.e., only using CPU. Advantage: accurate comparison of run times, availability; disadvantage: changing the code which originally supports using a GPU, to executing on a CPU only, is time consuming. (3) Estimate run times by comparing the inference performance of environments. Advantage: no hardware required; disadvantage: rough estimate of the run times.

We use option (3), as it requires no additional hardware purchases. The approach is to compare the inference run times of neural network models on 3 environments, i.e., NVIDIA GeForce GTX TITAN X, NVIDIA GTX 1080Ti, and NVIDIA TITAN XP. On the AI-Benchmark [124], 19 neural network models are run on various environments, including the ones of interest. The inference run times of the models on the three environments are listed in Table 4.5. For each model, we calculate the inference time of each environment compared to a reference environment, i.e., an inference scale factor. The reference environment is set to NVIDIA GTX 1080Ti. In Table 4.6, the scale factors are shown and in the last column the mean inference scale factors are listed. E.g., the inference scale factor of MobileNet-V2 on a NVIDIA GeForce GTX TITAN X is $\frac{48}{64} = 0.75$. We multiply the run times of the six methods with the inference mean scale factors to produce the scaled (estimated) run times. In Table 4.7, the resulting scaled run times are presented. The Fast DS-CS method is selected, since it has a lower matching error (BMP), 3.08 versus 4.66, compared to the MADnet method with a similar run time.

Draft set of selected algorithms
Applying the selection criteria for diverse, representative stereo algorithms (C3), we select algorithms from each category to form the draft set of algorithms, i.e., for the traditional pipeline - local, global and semi-global methods, and for the deep learning pipeline - end-to-end method. Table 4.4 summarizes the four selected stereo matching algorithms.

Method	Pipeline	Category
SSD	Traditional	Local
DP	Traditional	Global
SGM	Traditional	Semi-global
Fast DS-CS	Deep	End-to-end

Table 4.4: Draft set of stereo matching algorithms.

Environment	Network inference time (ms)																		
	MobileNet-V2	Inception-V3	Inception-V4	Inc-ResNet-V2	ResNet-V2-50	ResNet-V2-152	VGG-16	SRCNN 9-5-5	VGG-19 Super-Res	ResNet-SRGAN	ResNet-DPED	U-Net	Nvidia-SPADE	ICNet	PSPNet	DeepLab	Pixel-RNN	LSTM	GNMT
NVIDIA GeForce GTX TITAN X	64	94	96	129	63	85	168	131	273	145	319	363	183	211	752	208	579	679	225
NVIDIA GTX 1080Ti	48	62	61	81	39	51	104	79	158	95	198	207	107	141	441	116	372	430	141
NVIDIA TITAN Xp	40	62	58	76	37	51	97	78	149	89	182	192	102	168	423	111	527	533	218

Table 4.5: Inference time (run-time) of various networks on GPUs.

Environment	Network inference scale factors																			Inference mean scale factor
	MobileNet-V2	Inception-V3	Inception-V4	Inc-ResNet-V2	ResNet-V2-50	ResNet-V2-152	VGG-16	SRCNN 9-5-5	VGG-19 Super-Res	ResNet-SRGAN	ResNet-DPED	U-Net	Nvidia-SPADE	ICNet	PSPNet	DeepLab	Pixel-RNN	LSTM	GNMT	
NVIDIA GeForce GTX TITAN X	0.75	0.66	0.64	0.63	0.62	0.6	0.62	0.6	0.58	0.66	0.62	0.57	0.58	0.67	0.59	0.56	0.64	0.63	0.63	0.62
NVIDIA GTX 1080Ti	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
NVIDIA TITAN Xp	1.2	1	1.05	1.07	1.05	1	1.07	1.01	1.06	1.07	1.09	1.08	1.05	0.84	1.04	1.05	0.71	0.81	0.65	0.99

Table 4.6: Mean inference scale factors for GPUs with the *NVIDIA GTX 1080Ti* GPU as reference environment.

Method	Run time (s)	Environment	Inference mean scale factor	Scaled run time (s)
AutoDispNet	0.9	NVIDIA GTX 1080Ti	1	0.9
Fast DS-CS	0.02	NVIDIA GTX 1080Ti	1	0.02
RecResNet	0.3	NVIDIA GeForce GTX TITAN X	0.62	0.19
NVStereoNet	0.6	NVIDIA TITAN Xp	0.99	0.59
MADnet	0.02	NVIDIA GTX 1080Ti	1	0.02
PWOC-3D	0.13	NVIDIA GTX 1080Ti	1	0.13

Table 4.7: Scaled run time of stereo matching algorithms using the TensorFlow library.

5

Algorithm Verification

We address in this chapter the verification of the selected algorithms, i.e., three traditional and one deep learning algorithm from Chapter 4. Our approach is executing the algorithms and comparing the results with literature. This step is necessary, as it verifies the working of the implementation of the algorithms. We compare the results of each implementation and construct the final set of selected algorithms for the validation phase in Chapter 7. In addition, it is an important note that the pipeline of predicting disparity maps is memory-less, i.e., single still stereo images are processed without the dependence of previous stereo images. We use the memory-less pipeline in Chapter 7 for processing the video dataset.

The remainder of this chapter is structured as follows. The verification is performed in Section 5.1. The final set of selected algorithms is presented in Section 5.2.

5.1. Verification

To verify the working of the algorithms, we search for implementations of each algorithm and compare the error rate and run time with the one provided in literature. This ensures the implementation is correct (or as close as possible). For the traditional algorithms we have to look for 3rd party implementations, since the authors have not released their implementation to the public. As opposed to the deep learning algorithm, where the authors have released their implementation.

Additionally, we take note that our environment (hardware) differs from the ones used in literature. That is, we assume the traditional algorithms are executed on environments less powerful than our environment, since the publication date of [1] is in 2002. This means that the run time of the implementations must be faster than the timings provided in literature. For the deep learning algorithm, the environment is a NVIDIA GTX 1080Ti, i.e., a more powerful environment. Resulting in a slower run time on our environment. The environment we use in our verification process can be seen in Table 5.1.

Component	Specification
CPU	Intel Core i5-6200U @ 2.30GHz
Cores	4
Memory	8 GiB

Table 5.1: Hardware specifications of our environment.

We have two separate approaches for the verification of traditional and deep learning algorithms. Traditional algorithms have numerous implementations for the same algorithm, because there is no single official implementation, various implementations are available. The approach is to generate predicted disparity maps using the Tsukuba, Sawtooth, Venus, Map, Teddy, and Cones Middlebury dataset images, depending on the algorithm. Next, we compare the results of the implementations, in terms of error rate and run time, and select the one with the lowest values. As a pre-validation check, we pass a sample Katwijk stereo image to each algorithm, in order to determine if the selected algorithm is usable for validation. Since we do not have the disparity map ground-truth for the sample stereo image, we compute a noise percentage estimate.

Deep learning algorithms, on the other hand, have official implementations. They are pre-trained on the FlyingThings3D [25] dataset and from there the pre-trained model is fine-tuned to the KITTI dataset. The pre-trained model contains the weights to reproduce the predicted disparity maps from the dataset. Normally, the deep learning algorithms are trained and run on a GPU. To check if the algorithm is able to run on a CPU we first generate a predicted disparity map using a sample image of the KITTI 2015 dataset. Next, the error rate and run time are computed. Again, as a pre-validation check, we pass a sample Katwijk stereo image to each algorithm and compute a noise percentage estimate, in order to determine if the selected algorithm is usable for validation.

Our verification approach for the *traditional* algorithms will be as follows:

1. Find implementations of the algorithm, e.g., SSD. To facilitate the search of implementations, we use Github and set the programming language to Python,
2. For each implementation, compare the error rate and run time with literature based on the Middlebury dataset,
3. Select the implementation with the lowest error rate and run time,
4. Generate a predicted disparity map using a Katwijk sample stereo image and compute a noise percentage estimatelofi coding.

The approach for the *deep learning* algorithm will be as follows:

1. Check if the algorithm is able to run on a CPU, using a sample KITTI 2015 stereo image,
2. Compute the error rate and run time, and compare the results with literature based on the KITTI 2015 dataset,
3. Generate a predicted disparity map using a Katwijk sample stereo image and compute a noise percentage estimate.

5.1.1. SSD

The SSD algorithm is described in Chapter 2.3.1, it differs only from the SAD by squaring the term, instead of taking the absolute value of the term.

The first step in our approach is finding implementations of the SSD algorithm. The list of implementations of the SSD algorithm is shown in Table 5.2, with their results on the four test images and the last row showing the results from literature. The error rate is computed with a disparity threshold $\delta_d = 1$, as used in the works of [1]. For the run time, the algorithms are iterated a number of times to obtain a mean run time with a standard deviation, where the number of iterations $N = 10$.

In the SSD algorithm, a variable parameter is the window size. To determine the lowest error rate, we vary the window sizes with the values 3, 5, 7,...29, as in the works of [1]. The results from varying the window sizes is depicted in Figure 5.1. Selecting the lowest error rate from the figures results in the error rates listed in Table 5.2. It can be observed that the StereoBM algorithm (Figure 5.1c) starts with a window size of 5 instead of 3, since this implementation takes a minimum window size of 5.

The *Flatscher* implementation, takes the SSD equation and implements it one-to-one. Normally, the five nested for-loops, i.e., loop over the image (takes 2 for-loops), loop over the window (takes 2 for-loops), and loop over all possible disparities (1 for-loop), take a long time to run. In the order of minutes, instead of seconds, based on our own experiments. This implementation, however, uses the Numba library [125], which translates the Python code to fast machine code, reducing the run time from minutes to seconds.

The *SSD+MF* implementation, is based on the works of [1]. It applies a minimum filter to the resulting disparity map to smooth out the outliers in the image.

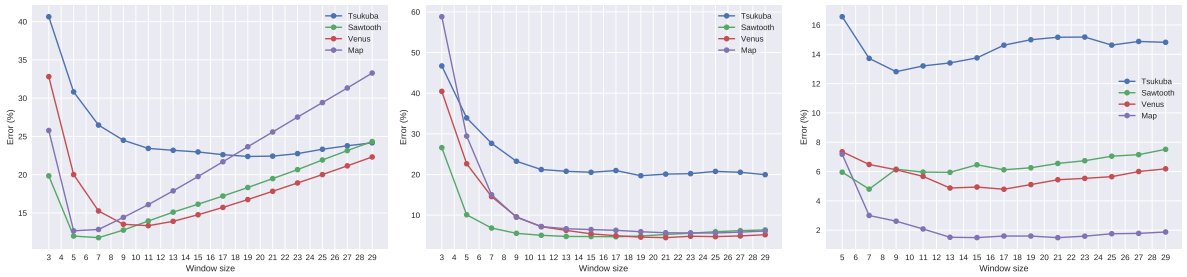
The *StereoBM* implementation, is contained in the OpenCV library. The code is optimized and uses the SAD algorithm instead of the SSD algorithm. SAD takes the absolute value of the difference between two pixels, and SSD takes the squared value of the difference between two pixels. Since the results of SAD and SSD are similar [1], we opt not to change the StereoBM source code.

In Figure 5.2, the predicted disparity map on the Tsukuba, Sawtooth, Venus, and Map image using the three implementation are depicted. We observe that the *StereoBM* algorithm corresponds the best

of the three implementations to the ground-truth. This is reflected in the error rate in Table 5.2, and this implementation has the lowest run time.

	Error rate (%)				Run time (s)			
Implementation	Tsukuba	Sawtooth	Venus	Map	Tsukuba	Sawtooth	Venus	Map
Flatscher [126]	22.4	11.8	13.3	12.7	1.66 ± 0.0440	2.47 ± 0.0211	2.88 ± 0.415	1.08 ± 0.0891
SSD+MF [1]	19.7	4.67	4.45	5.60	0.168 ± 0.00264	0.241 ± 0.00629	0.248 ± 0.00351	0.0661 ± 0.0115
StereoBM [127] (OpenCV)	12.8	4.30	4.29	1.49	0.0208 ± 0.00287	0.0334 ± 0.00278	0.0319 ± 0.00435	0.0160 ± 0.00127
Scharstein & Szeliski [1]	5.23	2.21	3.74	0.66	1.1	1.5	1.7	0.8

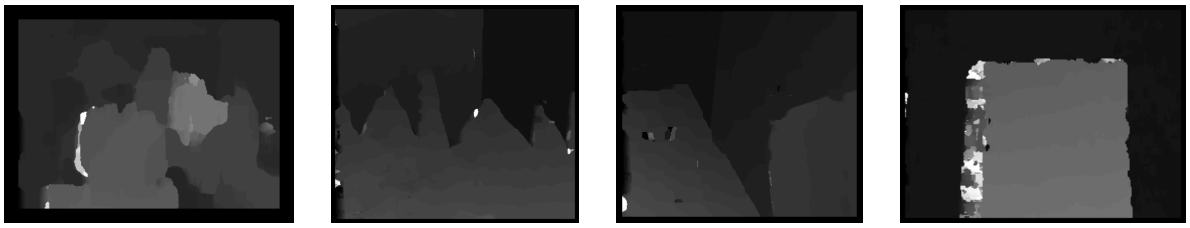
Table 5.2: SSD implementations and results on the Middlebury 2001 dataset and results from literature.



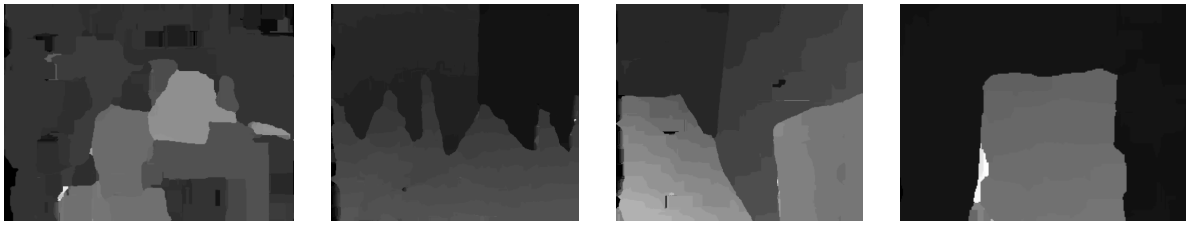
(a) Flatscher (b) SSD+MF (c) StereoBM

Figure 5.1: Results of the SSD implementations with varying window sizes.

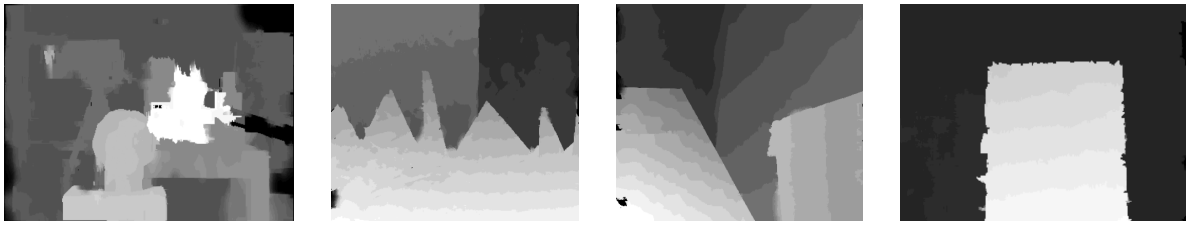
(a) Ground-truth: Tsukuba, Sawtooth, Venus, Map.



(b) Flatscher: Tsukuba, Sawtooth, Venus, Map.



(c) SSD+MF: Tsukuba, Sawtooth, Venus, Map.



(d) StereoBM: Tsukuba, Sawtooth, Venus, Map.

Figure 5.2: Predicted disparity maps using SSD implementations with optimal window sizes.

The last step in our approach is to generate a predicted disparity map using a Katwijk sample stereo image. In Appendix A.1, the resulting predicted disparity maps with varying window sizes are shown. We select the predicted disparity map with the least amount of noise. The noise consists of two parts: zero value disparities in the ground terrain ($y \gtrapprox 150$ pixels), and non-zero value disparities in the sky region ($y \lessapprox 150$ pixels). Normally, the ground terrain contains disparities that are greater than zero, as it is a finite distance away. For the sky region, the distance is an 'infinite' distance away, such that the disparity is zero. We compute the noise in the ground terrain by counting the number of zero value disparities, and the noise in the sky region by counting the number of non-zero value disparities. The noise percentage is equal to the number of erroneous disparities and dividing them by their respective areas. Next, we select the predicted disparity map with the lowest noise percentage. Figure A.2 shows the noise percentages with varying window sizes. In Figure 5.3, the left stereo image and the selected predicted disparity map with a window size of 23 is shown.

5.1.2. DP

The DP algorithm [128] tries to minimize the global energy function, described in Chapter 2.3.1. Instead of matching each individual pixel, the DP algorithm takes a corresponding row (scanline) in the left and right stereo image and finds the optimal alignment. By performing the alignment for each row, a disparity map is obtained. In Figure 5.4, the scanline alignment process is shown. The left and right scanline are lined up to compare each pixel. If the two scanlines are identical, the path (orange) is a diagonal. However, in a practical case the scanlines differ from each other. The path is not a diagonal line, which

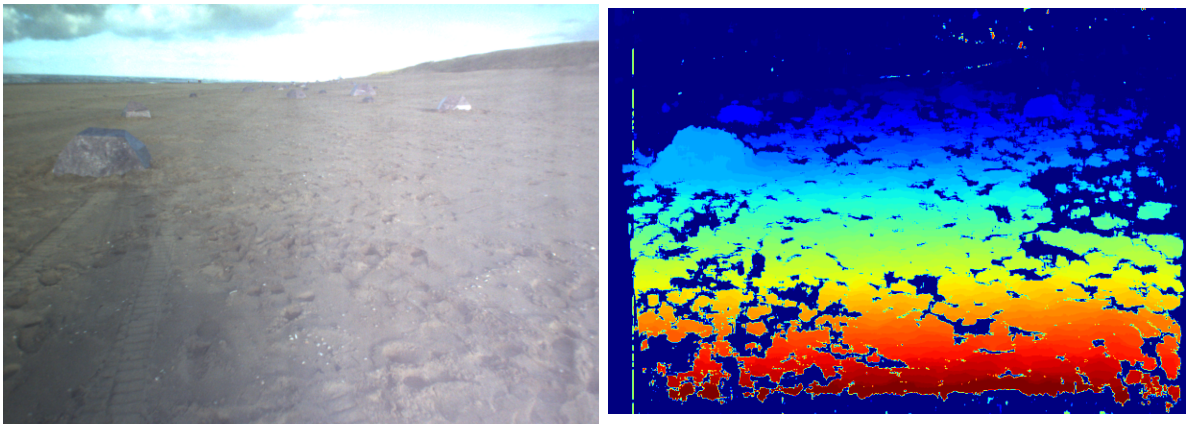


Figure 5.3: StereoBM disparity map on a sample Katwijk image using a window size of 23.

means that a pixel in the left scanline has no corresponding adjacent pixel in the right scanline. That is, the matching pixel in the right image is occluded by other pixels. The DP algorithm associates costs for pixels that occlude and pixels that match, the *occlusion cost* and matching cost. The occlusion cost influences the shape of the path and therefore also the disparity map.

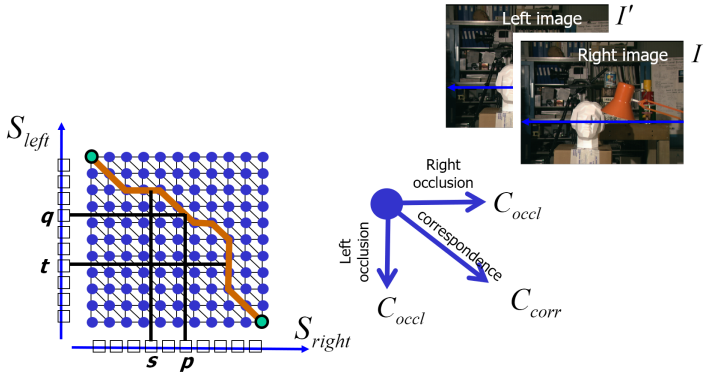


Figure 5.4: DP scanline alignment.

The list of implementations of the DP algorithm is shown in Table 5.3, with their results on the four test images and the last row showing the results from literature. The error rate is computed with a disparity threshold $\delta_d = 1$, as used in the works of [1]. For the run time, the algorithms are iterated a number of times to obtain a mean run time with a standard deviation, where the number of iterations $N = 10$.

In the DP algorithm, a variable parameter is the occlusion cost. To determine the lowest error rate, we vary the occlusion costs with the values 1, 10, 20, 30,...100, expanding the values compared to the works of [1]. The results from varying the occlusion costs is depicted in Figure 5.5. Selecting the lowest error rate from the figures results in the error rates listed in Table 5.3.

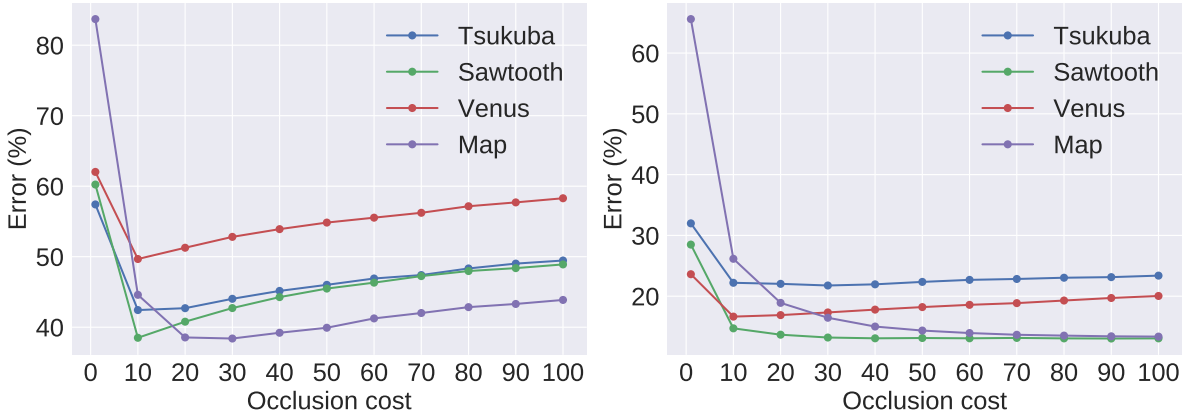
The *Siddhad* implementation, takes the DP pseudo-code from the works of [128], and implements it one-to-one. Normally, the for-loops slow down the execution of the code, however the Numba library translates the Python code to fast machine code.

The *Hartley* implementation, takes the optimized DP pseudo-code from the works of [128]. This optimized code improves the matching cost, such that it improves the provides a better predicted disparity value. Again, this implementation uses the Numba library to speed up the run time.

In Figure 5.6, the predicted disparity map on the Tsukuba, Sawtooth, Venus, and Map images using the two implementations are depicted. We observe that the *Hartley* algorithm corresponds the best. This is reflected in the error rate in Table 5.3 and this implementation has the lowest run time.

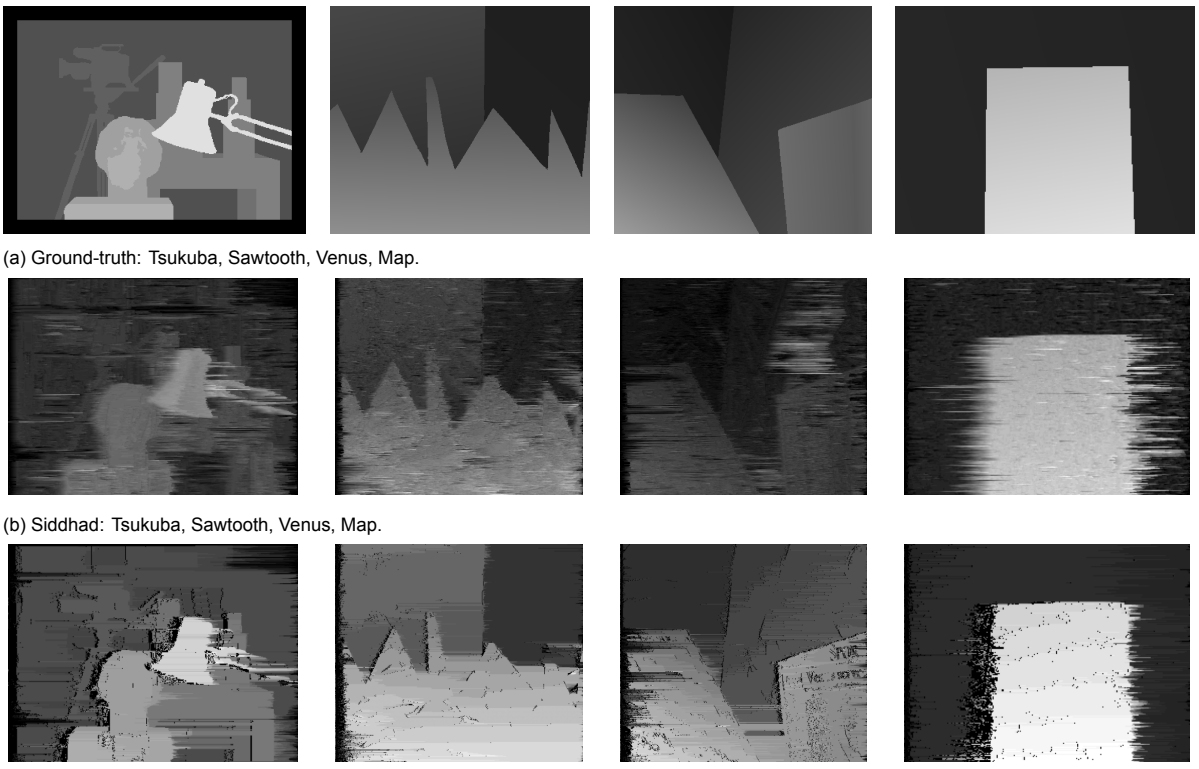
	Error rate (%)				Run time (s)			
Implementation	Tsukuba	Sawtooth	Venus	Map	Tsukuba	Sawtooth	Venus	Map
Siddhad [129]	42.4	38.5	49.7	38.4	5.70 ± 0.219	8.54 ± 0.335	8.45 ± 0.202	2.09 ± 0.256
Hartley [130]	21.8	13.1	16.6	13.4	0.455 ± 0.0479	0.726 ± 0.0354	0.706 ± 0.0254	0.221 ± 0.0621
Scharstein & Szeliski [1]	4.12	4.84	10.10	3.33	1.0	1.8	1.9	0.8

Table 5.3: DP implementations and results on the Middlebury 2001 dataset and results from literature.



(a) Siddhad (b) Hartley

Figure 5.5: Results of the DP implementations with varying occlusion costs.



(a) Ground-truth: Tsukuba, Sawtooth, Venus, Map.

(b) Siddhad: Tsukuba, Sawtooth, Venus, Map.

(c) Hartley: Tsukuba, Sawtooth, Venus, Map.

Figure 5.6: Predicted disparity maps using DP implementations with optimal occlusion costs.

The last step in our approach is to generate a predicted disparity map using a Katwijk sample stereo image. In Appendix A.2, the resulting predicted disparity maps with varying window sizes are shown. The noise consists of two parts: zero value disparities in the ground terrain (y $\gtrapprox$ 150 pixels), and non-

zero value disparities in the sky region ($y \lessapprox 150$ pixels). We compute the noise in the ground terrain by counting the number of zero value disparities, and the noise in the sky region by counting the number of non-zero value disparities. The noise percentage is equal to the number of erroneous disparities and dividing them by their respective areas. The resulting noise percentages with varying occlusion costs is shown in Figure A.4. In Figure 5.7, the left stereo image and the selected predicted disparity map with an occlusion cost of 40 is depicted.

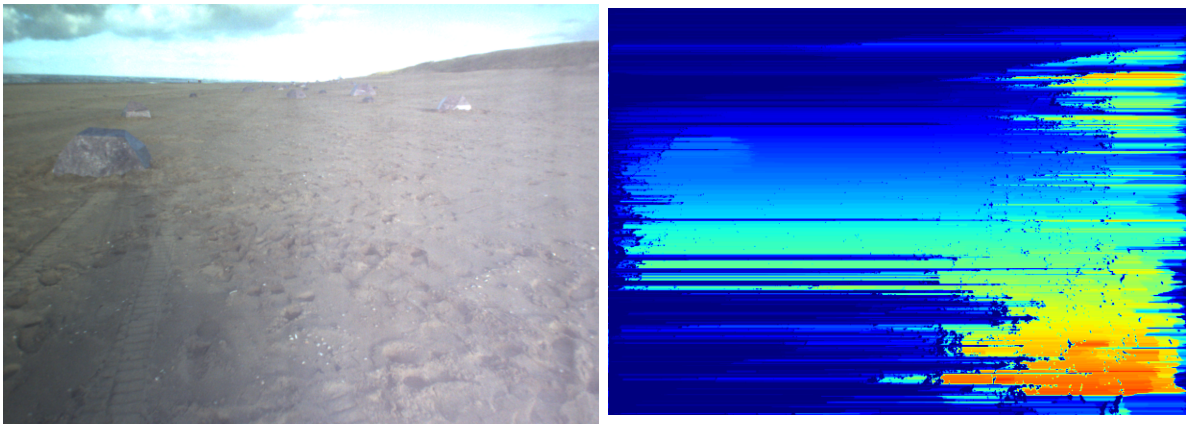


Figure 5.7: Hartley disparity map on a sample Katwijk image using an occlusion cost of 40.

5.1.3. SGM

The SGM algorithm is described in Chapter 2.3.1. The list of implementations of the SGM algorithm is shown in Table 5.4, with their results on the Middlebury 2001 and 2003 datasets, and results from literature. The error rate is computed with a disparity threshold $\delta_d = 1$, as used in the works of [1]. For the run time, the algorithms are iterated a number of times to obtain a mean run time with a standard deviation, where the number of iterations $N = 10$.

Similar to the SSD algorithm, the SGM makes use of a variable parameter in the matching cost computation step (we use the SSD algorithm for the cost computation), i.e., the window size. The results from varying the window sizes is depicted in Figure 5.8. Only the plot of the StereoSGBM with varying windows sizes is depicted, as the *Flatscher* implementation takes around 20 minutes per iteration of the window size. Selecting the lowest error rate from the figures results in the error rates listed in Table 5.4.

The *Flatscher* implementation, takes the SGM equations and implements it one-to-one. This implementation uses the Numba library in some parts of the code, where it is possible.

The *StereoSGBM* implementation, is contained in the OpenCV library. The code is optimized and focuses on speed by computing the matching cost in fewer directions, instead of all eight directions, such that the run time is reduced.

In Figure 5.9, the predicted disparity map on the Tsukuba, Sawtooth, Venus, and Map image using the three implementation are depicted. We observe that the *StereoSGBM* algorithm corresponds the best of the three implementations to the ground-truth. This is reflected in the error rate in Table 5.4, and this implementation has the lowest run time.

	Error rate (%)				Run time (s)			
Implementation	Tsukuba	Venus	Teddy	Cones	Tsukuba	Venus	Teddy	Cones
Flatscher [126]	22.4	10.3	13.3	12.7	271 ± 58.6	310 ± 7.49	315 ± 9.19	313 ± 5.49
StereoSGBM [131] (OpenCV)	8.86	2.83	10.5	6.15	0.123 ± 0.0288	0.189 ± 0.0227	0.188 ± 0.0261	0.200 ± 0.0265
Hirschmuller [26]	3.26	1.00	6.02	3.06	1-2	1-2	1-2	1-2

Table 5.4: SGM implementations and results on the Middlebury 2001 & 2003 datasets and results from literature.

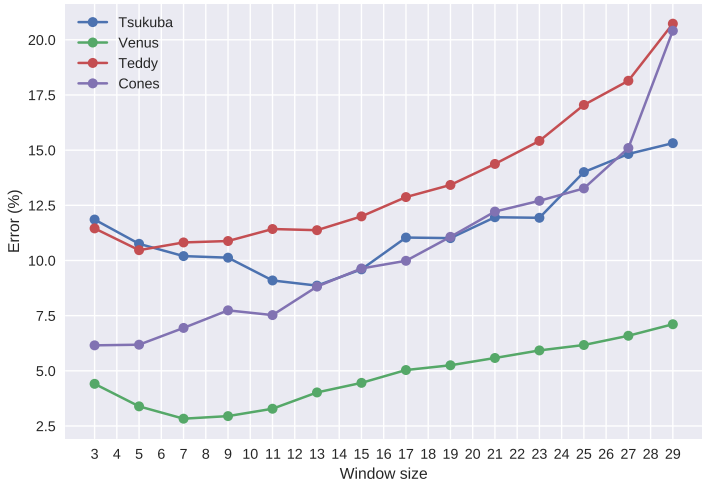
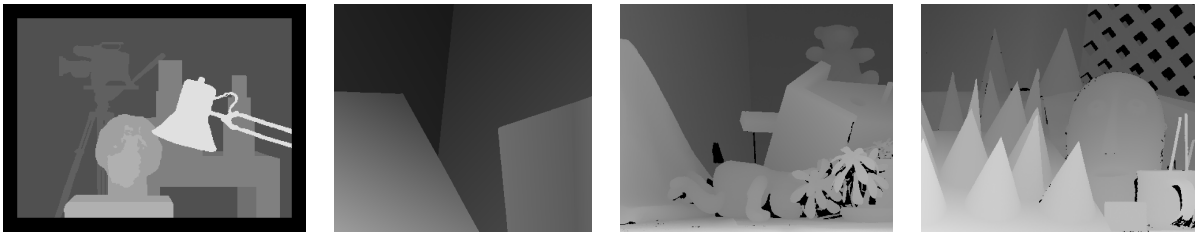
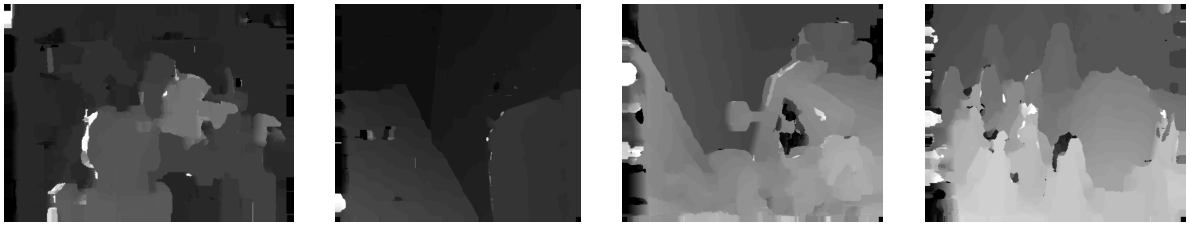


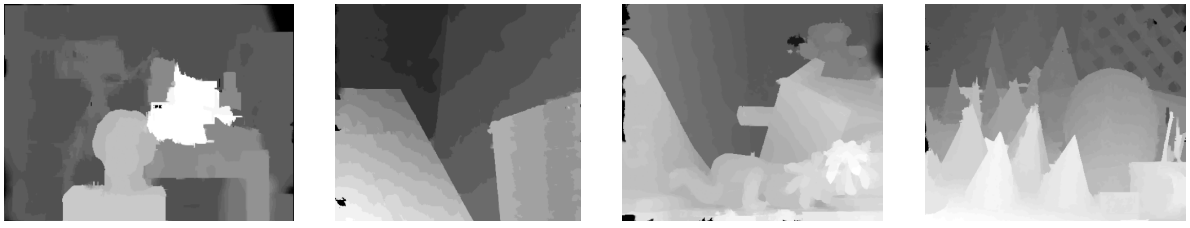
Figure 5.8: Results of the StereoSGBM implementation with varying window sizes.



(a) Ground-truth: Tsukuba, Venus, Teddy, Cones.



(b) Flatscher: Tsukuba, Venus, Teddy, Cones.



(c) StereoSGBM: Tsukuba, Venus, Teddy, Cones.

Figure 5.9: Predicted disparity maps using SGM implementations with window sizes.

The last step in our approach is to generate a predicted disparity map using a Katwijk sample stereo image. In Appendix A.3, the resulting predicted disparity maps with varying window sizes are shown. We observe high valued disparity areas in the upper right corner of the predicted disparity maps. These areas, in addition to the zero valued disparities in the ground terrain ($y \gtrapprox 150$ pixels), contribute to the noise. The noise is computed by summing the occurrences of erroneous high valued disparities and zero valued disparities in the ground terrain, and dividing by the total number of pixels. The noise percentages with varying window sizes are shown in Figure A.6. In Figure 5.10, the left stereo image and the selected predicted disparity map with a window size of 27 is depicted.

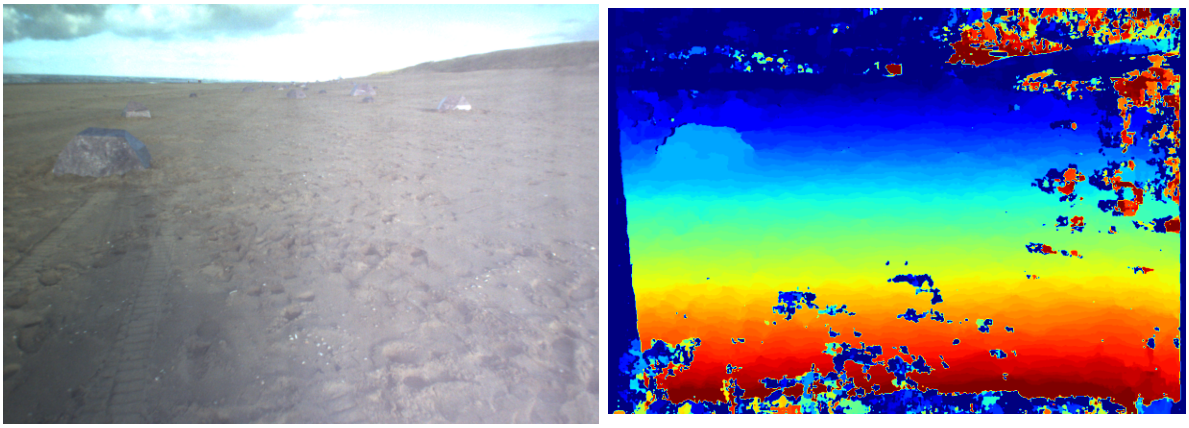


Figure 5.10: StereoSGBM disparity map on a sample Katwijk image using a window size of 27.

5.1.4. Deep Learning

In Chapter 4, the Fast DS-CS algorithm was selected as the deep learning algorithm. In this section we take a look at the results of Fast DS-CS and explore another algorithm.

Fast DS-CS

We described the fundamentals of neural networks, in particular, the end-to-end deep learning network in Chapter 2.3.2. Figure 5.11 depicts the Fast DS-CS network. The left and right image are first processed using traditional cost computation. Next, the cost volume is constructed and enters the encoder-decoder network, which down-samples and up-samples the layers. Performing a last up-sample operation generates the predicted disparity map.

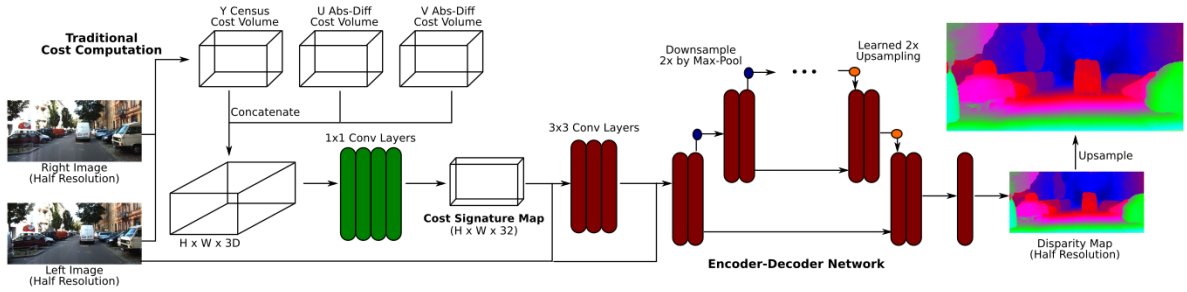
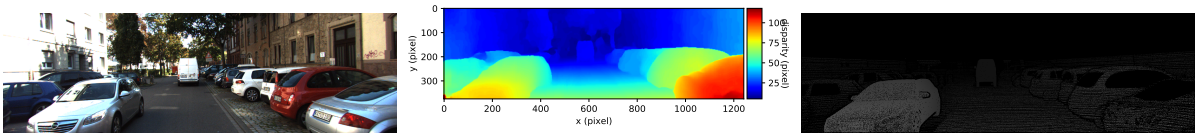


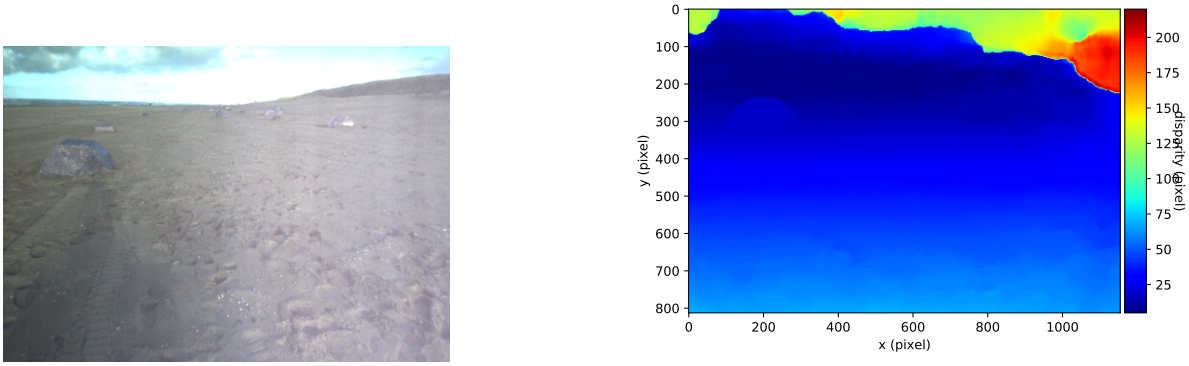
Figure 5.11: Fast DS-CS network (adopted from [108]).

The Tensorflow implementation of the Fast DS-CS algorithm is available on Github [108]. The first step in our approach is to generate a predicted disparity map using the KITTI pre-trained model using a sample KITTI stereo image. To execute the code, a custom Tensorflow ops, i.e., a custom function that does not exist in the Tensorflow library needs to be compiled. The custom functions in the `slib/slib.cc`, computes the census transform and Hamming distance on a GPU. We rewrite the custom function, such that it compiles on our CPU environment. The predicted disparity map using a sample KITTI stereo image is depicted in Figure 5.12a. We observe a gradient of disparity values, such that the cars closer to the camera have a larger disparity (smaller distance), than the ones farther down. In addition, the predicted disparity map outlines the shape of the cars and other objects in the scene.

To compute the error rate, we use the metrics function provided in the code repository. This function transforms the LIDAR scan into a (sparse) ground-truth disparity map and compares it with the predicted disparity map. The disparity threshold is 2 pixels, i.e., if a value in the predicted disparity map is 2 pixels larger or smaller than the ground-truth value, the incorrect pixels count is incremented. We use 10 training images, instead of the testing images to compute the mean error rate and run time, because the ground-truth of the testing images are not provided in the KITTI 2015 dataset. The results of the error rate and run time are listed in Table 5.5. We observe an error rate increase of 2.04% and a run



(a) KITTI sample image: (left) left stereo image; (middle) predicted disparity map; (right) ground-truth LIDAR scan.



(b) Katwijk sample image: (left) left stereo image; (right) predicted disparity map.

Figure 5.12: Fast DS-CS predicted disparity maps using sample images of KITTI and Katwijk.

time increase of a factor 130. The error rate reported in literature are from the testing images and are computed by the KITTI evaluation server. This may explain the increased error rate from our 10 training images.

Environment	Error rate (%)	Run time (s)
Intel Core i5-6200U	5.12 ± 0.211	2.59 ± 0.324
NVIDIA GTX 1080Ti	3.08	0.02

Table 5.5: Fast DS-CS error rate and run time on our and literature's environment.

The last step of our approach is generating a predicted disparity map using a Katwijk sample stereo image. In Figure 5.12b, the resulting predicted disparity map can be seen. The noise consists of two parts: zero value disparities in the ground terrain ($y \gtrapprox 150$ pixels), and non-zero value disparities in the sky region ($y \lessapprox 150$ pixels). We compute the noise in the ground terrain by counting the number of zero value disparities, and the noise in the sky region by counting the number of non-zero value disparities. The noise percentage is equal to the number of erroneous disparities and dividing them by their respective areas. The resulting noise percentage is 15.8%.

MADnet

The Tensorflow implementation of the MADnet algorithm is available on Github [115]. This algorithm has a similar run time as the previously examined Fast DS-CS algorithm. However, when trying to run the algorithm we get the following execution error, `ZeroDivisionError: division by zero`. To examine this error, the issues page of the code repository, which lists issues flagged by users, is consulted. A possible reason for this execution error is provided in [132]. The user running a 4GB GPU (GTX 1050Ti), states that his environment is not powerful enough run the code. For this reason, our CPU environment is also not able to run the code.

PWOC-3D

In Figure 5.13 the PWOC-3D network is depicted. The network contains an encoder-decoder network, the pyramid on the left forms the encoder, and the pyramid on the right forms the decoder. During the decoding stage, several processes (orange boxes) occur. In order of occurrence the processes are: warping, occlusion estimation, cost volume computation, and scene flow prediction. Following the processes generates a predicted disparity map, and is improved using the context network.

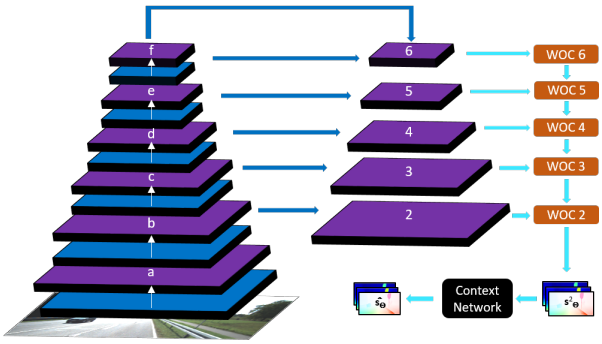
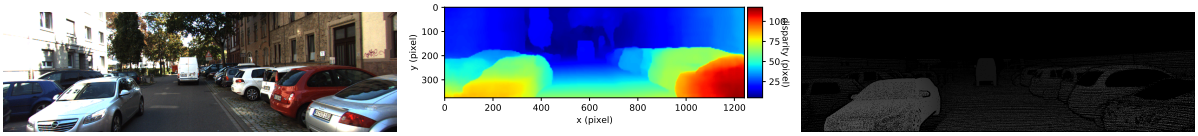
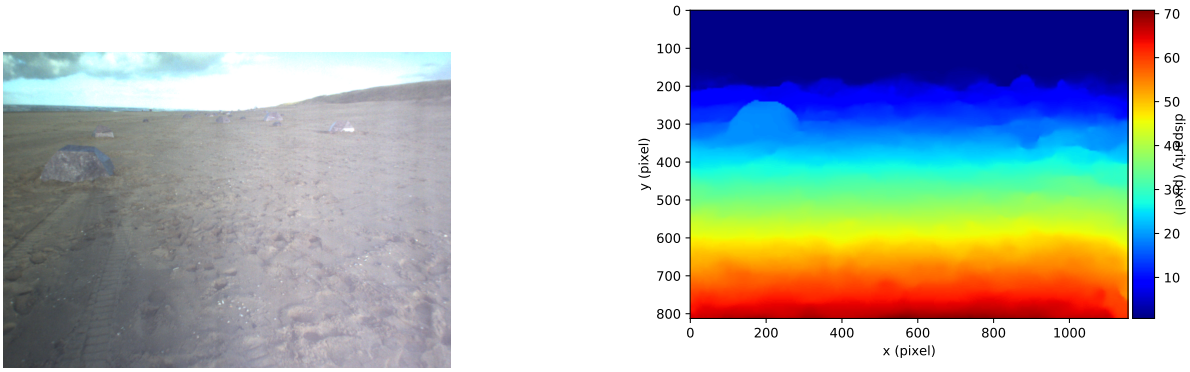


Figure 5.13: PWOC-3D network (adopted from [117]).

The Tensorflow implementation of the PWOC-3D algorithm is available on Github [117]. The first step in our approach is to generate a predicted disparity map using the KITTI pre-trained model using a sample KITTI stereo image. The predicted disparity map using a sample KITTI stereo image is depicted in Figure 5.14a. Similar to the Fast DS-CS algorithm, we observe a gradient of disparity values, such that the cars closer to the camera have a larger disparity (smaller distance), than the ones farther down. In addition, the predicted disparity map outlines the shape of the cars and other objects in the scene.



(a) KITTI sample image: (left) left stereo image; (middle) predicted disparity map; (right) ground-truth LIDAR scan.



(b) Katwijk sample image: (left) left stereo image; (right) predicted disparity map.

Figure 5.14: PWOC-3D predicted disparity maps using sample images of KITTI and Katwijk.

To compute the error rate, we use the metrics function provided in the code repository. This function transforms the LIDAR scan into a (sparse) ground-truth disparity map and compares it with the predicted disparity map. The disparity threshold is 2 pixels, i.e., if a value in the predicted disparity map is 2 pixels larger or smaller than the ground-truth value, the incorrect pixels count is incremented. We use 10 training images, instead of the testing images to compute the mean error rate and run time, because the ground-truth of the testing images are not provided in the KITTI 2015 dataset. The results of the error rate and run time are listed in Table 5.6. We observe an error rate increase of 2.90% and a run time increase of a factor 35. The error rate reported in literature are from the testing images and are computed by the KITTI evaluation server. This may explain the increased error rate from our 10 training images.

Environment	Error rate (%)	Run time (s)
Intel Core i5-6200U	8.03 ± 0.112	4.48 ± 0.603
NVIDIA GTX 1080Ti	5.13	0.13

Table 5.6: PWOC-3D error rate and run time on our and literature's environment.

The last step of our approach is generating a predicted disparity map using a Katwijk sample stereo image. In Figure 5.14b, the resulting predicted disparity map can be seen. We compute the noise in the ground terrain (y $\gtrapprox$ 150 pixels) by counting the number of zero value disparities, and the noise in the sky region (y $\lessapprox$ 150 pixels) by counting the number of non-zero value disparities. The noise percentage is equal to the number of erroneous disparities and dividing them by their respective areas. The resulting noise percentage is 3.2%.

5.2. Final Set of Algorithms

In Section 5.1.1- 5.1.3, we have examined implementations of the SSD, DP, and SGM algorithms. One of the SSD implementations, the StereoBM algorithm, implements the SAD algorithm instead of the SSD algorithm. From the works of [1], the SAD and SSD algorithms have similar performances in terms of error rate and run time. We select the StereoBM algorithm, as it yields the lowest error rate and run time compared to the other SSD implementations. The set of traditional algorithms is listed in Table 5.7.

In Section 5.1.4 we have examined three deep learning algorithms, Fast DS-CS, MADnet, and PWOC-3D. Fast DS-CS, generates a disparity map with corresponding disparity values and shapes. However, using the Katwijk sample stereo image, the predicted disparity map contains noise, with a noise percentage estimate of 15.8%. Because of the noise, we attempt to find an algorithm with a smooth predicted disparity map. MADnet, has a similar run time, however the execution of the code returns an error related to our environment. The PWOC-3D algorithm, yields a smooth predicted disparity map using the Katwijk sample stereo image and a lower noise percentage estimate of 3.2%. Thus, we select the PWOC-3D algorithm as our deep learning algorithm. The final set of algorithms is listed in Table 5.7.

Implementation	Method	Pipeline	Category
StereoBM [127]	SAD	Traditional	Local
Hartley [130]	DP	Traditional	Global
StereoSGBM [131]	SGM	Traditional	Semi-global
PWOC-3D [117]	-	Deep	End-to-end

Table 5.7: Final set of stereo matching algorithms.

6

Experimental Setup

We address in this chapter our second research question: How to efficiently compare the performance of stereo matching algorithms? Our approach is to setup experiments to compare the performance of the algorithms. This thesis contains two experiments. The first experiment evaluates the generated predicted disparity maps where the sun is positioned behind the rover. The second experiment uses the dataset where the sun is positioned in front of the rover. The two different sun positions provide a realistic setting for the rover. As the rover traverses through the scene, the sun can be positioned in front or behind it.

The remainder of this chapter is structured as follows. An overview of the experiments is presented in Section 6.1. The design of the experiments is described in Section 6.2. The experimental environment is presented in Section 6.3. In Section 6.4, the Katwijk dataset is described and its setup. Finally, the metrics are formulated in Section 6.5.

6.1. Experiments Overview

In our work we present two experiments, shown in Table 6.1. The *section* column refers to in which chapter the experiment is described. The *dataset* column indicates the dataset used for the experiments. The dataset is described in Section 6.4. The *traverse* column indicates which traverse of the dataset is used. The *position* column refers to sunlight position, influencing the captured stereo images. The *algorithms* column indicates the algorithms used to generate predicted disparity maps, these algorithms are established in Chapter 5. The *metrics* column lists the metrics used to evaluate the predicted disparity maps. The metrics for the experiments are described in Section 6.5.

Section	Traverse	Dataset	Position	Algorithms	Metrics
7.1	1	Katwijk	sunlight behind the rover	SAD, DP, SGM, PWOC-3D	error rate, run time, memory usage
7.2	2	Katwijk	sunlight in front of the rover	SAD, DP, SGM, PWOC-3D	error rate, run time, memory usage

Table 6.1: Summary of the two experiments presented in our work.

6.2. Experiment Design

Our goal is to evaluate the predicted distance measurements from the four stereo matching algorithms. In Figure 6.1, the experiment design is shown. The design consists of four modules. The *Katwijk dataset* module (Section 6.4), processes the raw stereo image (Section 6.4.1) of the LocCam in the rectification block to generate the rectified stereo image (Section 6.4.2). The *algorithms* module (Chapter 5), takes the rectified stereo image and passes it to the four algorithms, i.e., SAD, DP, SGM, and PWOC-3D. The algorithm output is a predicted disparity map. In addition, from the execution of each algorithm the run time and memory usage is computed. The *error rate metric* module (Section 6.5.1), computes two distances, the ground-truth distance to a rock (Section 6.5.2) and the predicted distance

to a rock (Section 6.5.3), to obtain the distance error. The ground-truth distance is obtained by transforming the rover GPS coordinates and rock(s) coordinates to Cartesian and computing the distance between them. The predicted distance is computed by transforming the predicted disparity map to a U-disparity map, followed by an edge detector, Hough line transform, and a Hough to Cartesian transform. The *metrics* module (Section 6.5), contains the metrics for our experiments, i.e., run time, memory usage, and error rate.

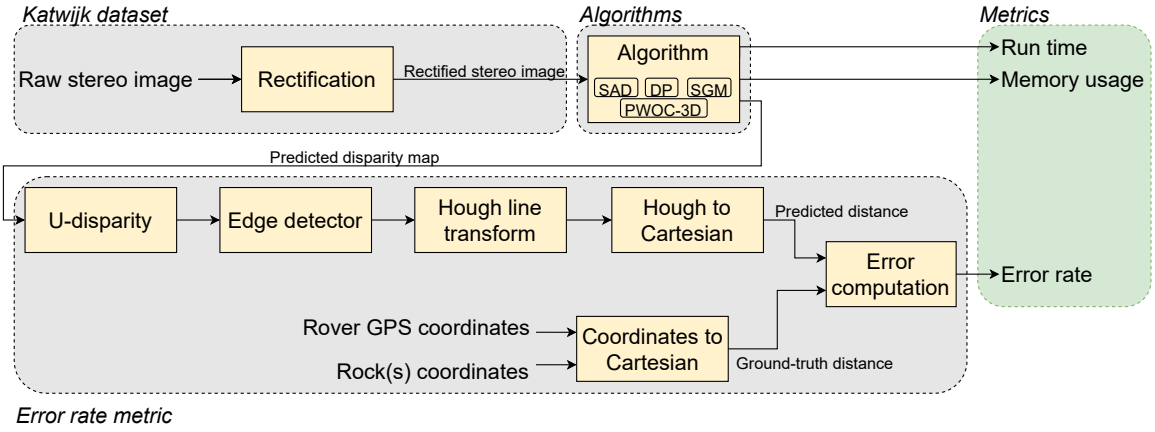


Figure 6.1: Experiment design.

6.3. Environment

We perform our experiments on the Raspberry Pi 4 Model B [14], a commercial off-the-shelf single-board computer. The hardware specifications of the environment are listed in Table 6.2.

The environment we use in our experiments is not representative of the ExoMars rover environment, a 96MHz LEON2 processor with 512 MiB of RAM [81, 133]. However, the Raspberry Pi 4 Model B is used, as it is the single-board computer on the Lunar Zebro [12] during the testing phase. The Lunar Zebro is developed at the Delft University of Technology, the Delft Robotics Institute, to study swarm behaviour on the Lunar surface [13]. This indicates that this environment is a suitable option for our purpose.

Component	Specification
CPU	Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memory	8 GiB

Table 6.2: Hardware specifications of the Raspberry Pi 4 Model B.

6.4. Katwijk Beach Planetary Rover Dataset

Validation of the algorithms requires a dataset that is representative of Martian surface. For the experiments we use the Katwijk Beach Planetary Rover Dataset [21]. Other datasets have presented Mars-like terrains, such as the Devon Island rover navigation dataset [134] and the Canadian Planetary Emulation Terrain Energy-Aware Rover Navigation Dataset [135]. However, they are not based on a real planetary rover, from which no requirements can be derived.

The remainder of this section is structured as follows. In Section 6.4.1 an overview of the Katwijk dataset is presented. The rectification process of raw stereo images is described in Section 6.4.2. The ground-truth data is developed in Section 6.4.3.

6.4.1. Overview

The Katwijk beach planetary rover dataset [21] is a large dataset collected on a section of beach near Katwijk, The Netherlands. The beach is selected as a Mars analog since it is a natural, unstructured,and sandy terrain. The dataset contains two separate test runs, depicted in Figure 6.2. Traverse 1, comprises of two long traverses of >1km each, where the rover traversed through a boulder field. It is made up of 212 artificial boulders that were custom made and carefully distributed to resemble boulders of a

typical boulder field. The sunlight position is in the back of the rover. Traverse 2, is a shorter traverse of 200 m, where the artificial boulders were placed such that the density is approximately twice that of the previous traverse. The sunlight position is facing the rover.

The dataset serves to build up and evaluate capabilities for future field experiments at analog sites. The sensors on the rover are configured to emulate the ExoMars rover, including stereo cameras. An overview of on-board sensors are listed in Table 6.3.

The sensors can be divided into two groups, input and output. Input sensors are used for navigation, i.e., the LocCam and PanCam. The LocCam is mounted at the same height and angle as the ExoMars rover, at 1 meter height. Its primary use is for visual odometry. The PanCam is mounted on a pan-tilt unit at the same height and angle as the ExoMars rover, at 2 meters height. It is intended for scene reconstruction and images are taken at varying pan angles throughout the traverses. Even though PanCam images are used for scene reconstruction, the images acquired from the stereo *LocCam* are used as the raw stereo images, as they are non-blurry and are acquired at a consistent interval. Whereas the PanCam stereo images are blurry due to a different kind of stereo camera, and the interval is inconsistent, where in some parts a 4 second stereo image sequence is missing. Additionally, the specifications of the LocCam of the Katwijk dataset (baseline 12 cm, focal length 4 mm) are comparable with the PanCam and LocCam of the ExoMars rover (see Chapter 3.1), whereas the Katwijk PanCam's baseline is significantly larger, i.e., 50 cm.

Output sensors are used for depth extraction of the scene, i.e., the ToF camera, 3D LiDAR, and RTK GPS. The time-of-flight (ToF) camera and the three-dimensional (3D) scanning light detection and ranging (LiDAR) sensor are active sensors. These two sensors generate a sparse ground-truth of the objects in the scene, i.e., a limited number of data points are extracted. We select the *RTK GPS* as our output sensor, since it gives a rover position estimate with an accuracy of 2 mm. Combining this with the geo-referenced image, gives us our ground-truth, described in Section 6.4.3.

Sensor	Description	Data logged	Measurements	Capture rate
LocCam	PointGrey Bumblebee2 (BB2-08S2C-38) 12 cm baseline stereo camera	Color 1024×768 images	20,000 pairs	3.75 Hz
PanCam	PointGrey GrassHopper2 (GS2-FW-14S5C-C) ×2 custom 50 cm baseline stereo camera	Color 1280×960 images	12,000 pairs	3.75 Hz
ToF camera	MESA SwissRanger 4500	Depth and intensity 176×144 images	47,000 pairs	10 Hz
3D LiDAR	Velodyne VLP-16	Range, azimuth, intensity 1808×16 images	33,000 triples	10 Hz
RTK GPS	Trimble BD 970 receiver with Zephyr Model 2 antenna (rover) Trimble BX 982 Receiver with Zephyr Geodetic antenna (base station)	Latitude, longitude and altitude expressed on WGS 84 ellipsoid	1790 sets	0.3 Hz

Table 6.3: Overview of sensors [21].

File name	Description	Format
LocCam_YYYY_MM_DD_hh_mm_ss_fff_{0,1}	Color, 1024×768 raw stereo image pairs	
PanCam_YYYY_MM_DD_hh_mm_ss_fff_{0,1}	Color, 1280×960 raw stereo image pairs	
ToF_YYYY_MM_DD_hh_mm_ss_fff_{range, intensity}	16-bit grayscale, 176×144 raw images	
Velodyne_YYYY_MM_DD_hh_mm_ss_fff_{range, intensity}	16-bit grayscale, 1808×16raw images (note: intensity images are 8-bit)	
gps-latlong.txt	GPS measurements of latitude, longitude and altitude expressed on WGS 84 ellipsoid	Timestamp, signal quality, latitude (◦), longitude (◦), altitude (m), latitude standard deviation (m), longitude standard deviation (m), altitude standard deviation (m)
imu.txt	Accelerometer, gyroscope and inclinometer measurements	Timestamp, x, y, z, acceleration (m/s^2), x, y, z, angular velocity (rad/s), x, y, z, acceleration (m/s^2)
odometry.txt	Wheel and steering angular displacement and rocker-bogie potentiometer measurements	Timestamp, front left, front right, center left, center right, back left, back right wheel angular displacement (rad), front left, front right, back left, back right steering angular displacement (rad), rocker, left bogie, right bogie orientation (rad)
ptu.txt	Pan–tilt unit orientation	Timestamp, pan (rad), tilt (rad)
{small, medium, large}-rocks-traverse{12,3}.txt	Artificial rock coordinates	Northing (m), easting (m)
{small, medium, large}-rock.{sldprt,step}	Artificial rock 3D models	.sldprt, .step
dsm.tif	Digital elevation model	.tif
mosaic.tif	Georeferenced image	.tif

Table 6.4: Overview of the file formats for all data collected. In stereo images, 0,1 corresponds to the left and right images respectively [21].

In Figure 6.3a, a scene from traverse 1 with several boulders is depicted. The sunlight positioned in the back of the rover is depicted in Figure 6.4a, where the shadow of the rover and a large boulder can be seen.

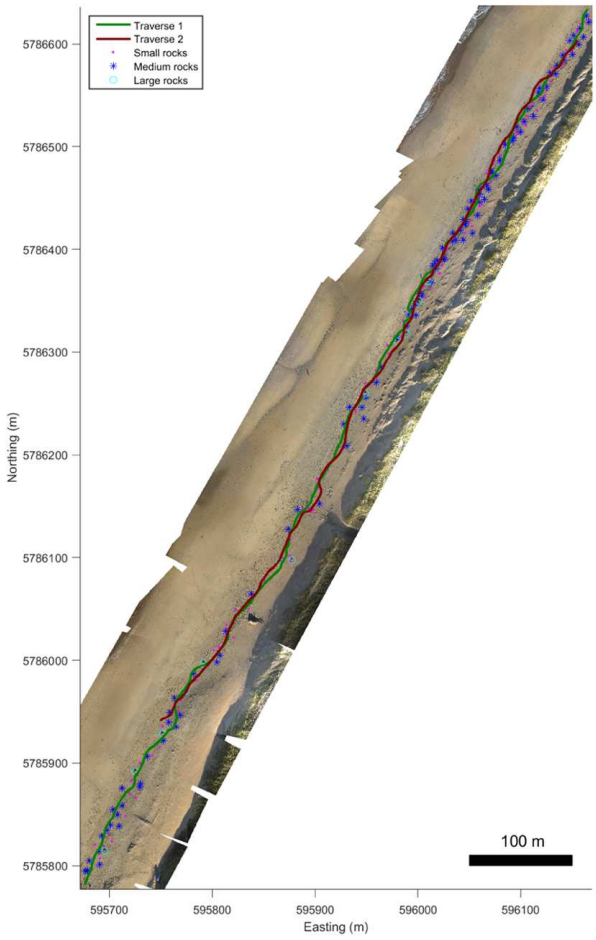
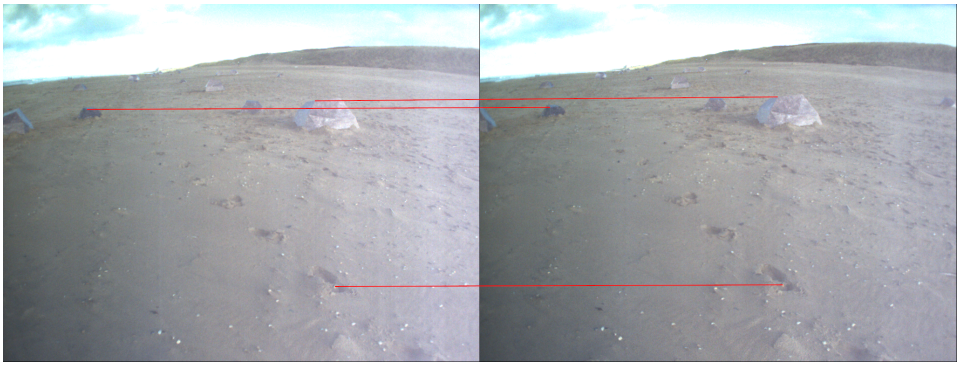


Figure 6.2: Overview of the traverse 1 and 2.

6.4.2. Rectification

The dataset contains raw stereo images, while the stereo matching process requires rectified stereo images as the input. The dataset provides a Matlab script, `image2points.m`, that calls the `rectifyStereoImages` function to rectify the raw stereo images. This function takes the left and right images, and the stereo calibration data from the `LocCam_calibration.mat` data file.

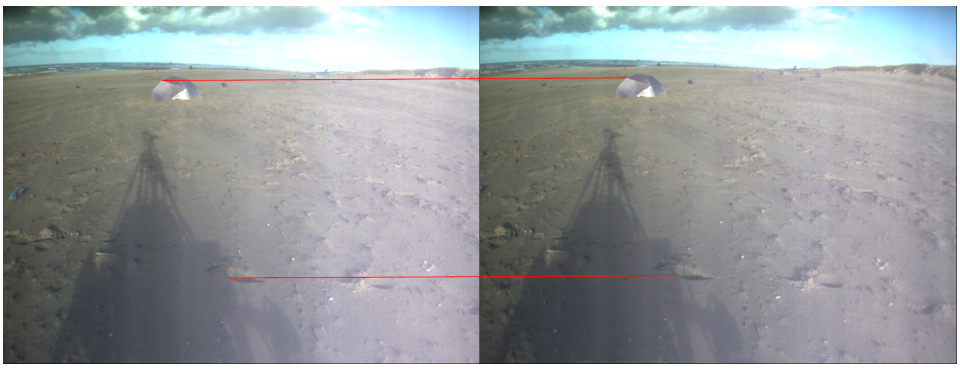
In Figure 6.3a, the left and right stereo image pair of terrain 1 (traverse 1) is depicted with matching lines to show the matching points are not along the same horizontal scanline. The raw image shows a slight upward slope, from the left to right matching point (zoom in to observe the slope). In Figure 6.3b, the rectified image pair is depicted. The matching points lie on the same horizontal scanline, i.e., the epipolar line. The same procedure is depicted in Figure 6.4a, raw stereo image pair with matching lines, and Figure 6.4b, rectified stereo image pair with matching lines, for terrain 2 (traverse 1).



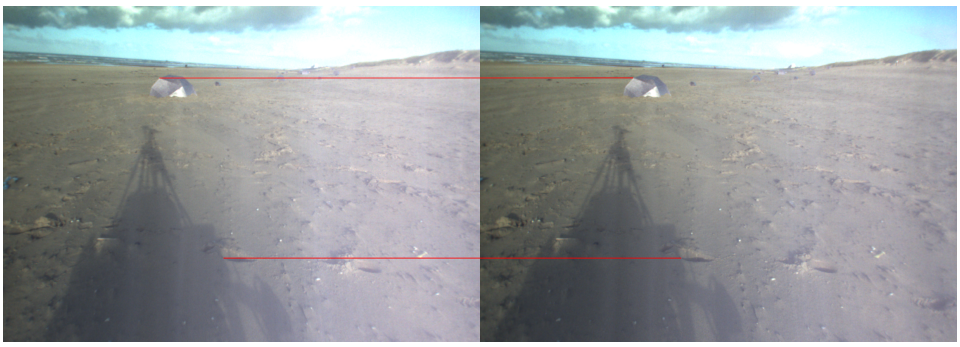
(a) Raw image pair

(b) Rectified image pair

Figure 6.3: Terrain 1 (traverse 1) image pair with matching lines.



(a) Raw image pair

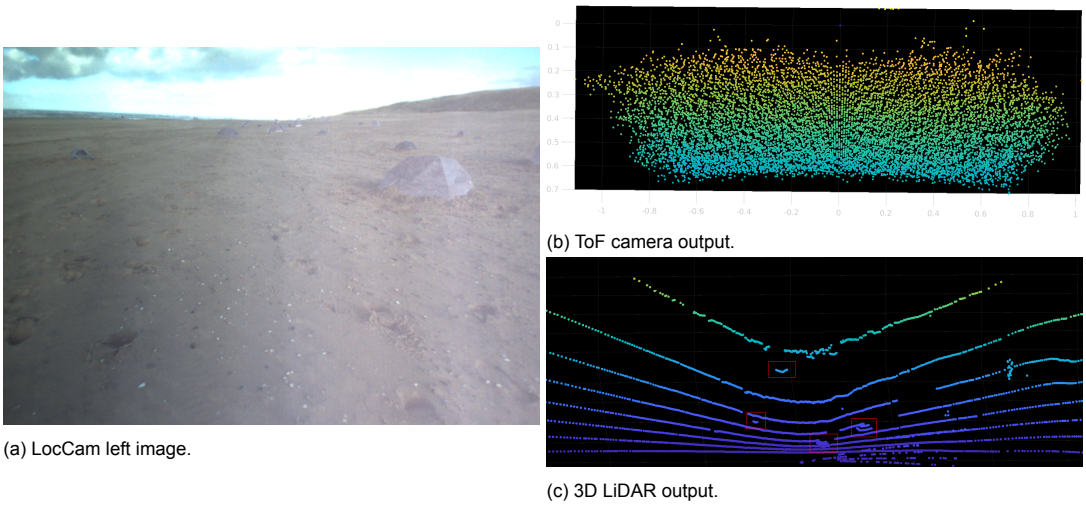


(b) Rectified image pair

Figure 6.4: Terrain 2 (traverse 1) image pair with matching lines.

6.4.3. Ground-truth

The ground-truth is used to evaluate the predicted disparity map. Unlike the Middlebury and KITTI 2015 dataset, the Katwijk dataset does not contain ground-truth disparity maps. The dataset contains other sensors, that actively capture the scene, those are the ToF camera and 3D LiDAR. In Figure 6.5a, a left LocCam image of traverse 1 is shown. The corresponding ToF camera and 3D LiDAR outputs are depicted in Figure 6.5b and 6.5c. We attempt to identify the rocks in the sensor outputs. For the ToF camera, blue points represent closer points to the camera, with gradually increasing distances represented by a gradient to yellow points. We observe no resemblance of rocks or other elements in the scene for the ToF camera output. For the 3D LiDAR, again, the coloured points represent the distances. We observe some resemblance of rocks in the scene, bounded by a red box. However, identifying rocks in the 3D LiDAR output is an inaccurate process as the scanlines of the sensor are sparse. To generate an accurate ground-truth we explore another sensor listed in Table 6.3, the RTK GPS, in Section 6.5.2.



(a) LocCam left image.

(b) ToF camera output.

(c) 3D LiDAR output.

Figure 6.5: Active sensor outputs.

6.5. Metrics

In this section we develop metrics to evaluate the predicted disparity maps. In addition to the *run time* of generating disparity maps and the *memory usage*, we develop an error rate metric to evaluate the quality of the disparity maps. In Section 6.5.1, we discuss the pitfall of using the BMP metric. The ground-truth distance computation is described in Section 6.5.2. The predicted distance computation is described in Section 6.5.3. In Section 6.5.4, the ground-truth and predicted distance are combined to form a metric.

6.5.1. Pitfall of the BMP metric

As far as we know, the standard metric to evaluate the predicted disparity maps is using the bad matched pixel (BMP) metric. This metric essentially counts how many pixels in the predicted disparity map are not the same as the ground-truth disparity map, with a threshold of how much it may vary. We argue that the BMP metric is not a metric which is representative of how well the rover performs the traverse. To show this, let us have two algorithms with a BMP of 5% and 50%, respectively. A lower BMP would indicate a better predicted disparity map, as used in literature. However, in our case, the BMP does not tell anything about how well it measures the distance of rocks in the scene. It could even be that the algorithm with a 50% BMP performs better, if the errors are located at uninteresting places, such as on the edges of the disparity map. The 5% BMP algorithm could have errors at important places, such as on rocks, which affects the rover navigation. Thus, a lower BMP of an algorithm, i.e., a more accurate disparity map, does not correspond directly to a better algorithm.

This applies to the algorithm selection process (Chapter 4), where we use the Middlebury and KITTI 2015 datasets and base our algorithm selection on the BMP (and run time). One could argue that because of this pitfall the selection based on the BMP is not valid, i.e., an algorithm with a higher

BMP than 10% could perform better than one with a lower BMP. However, to facilitate the algorithm selection process we opt to use the BMP as a selection criteria.

We need to develop a metric which is based on a physical parameter, instead of the percentage of bad matched pixels in the disparity map. When taking a closer look at path planning, it is crucial to determine the distance of obstacles in the scene. From this we can base our metric on the quality of the distance computation of obstacles. To determine the distance of obstacles in the scene some options are possible. The ToF camera and LIDAR scans are used to determine the distance of obstacles, however the ToF camera output is unclear and the LIDAR scans in the dataset are sparse and would not be sufficient (see Section 6.4.3).

We develop a metric which computes the difference between the ground-truth distance and the predicted distance to rocks in the scene. In Section 6.5.2, the ground-truth distance computation is described. In Section 6.5.3, the predicted distance computation is described. These two distances are used to form a metric, described in Section 6.5.4.

6.5.2. Ground-truth distance

To determine the ground-truth distances to rocks in the scene, we use the geo-referenced image, `mosaic.tif` (see Table 6.4), in the dataset. A Sensefly eBee unmanned aerial vehicle (UAV), a fixed-wing aircraft with a camera mounted to its belly, was flown before each traverse to capture geo-referenced images with a resolution of approximately 2 cm. By imaging multiple overlapping areas, the UAV bundles the images to create a geotiff, a single image that covers the entire area of the traverse [21], resulting in a large geotiff file of 2.4 GiB. Each pixel has a corresponding 2D position in the global UTM 31 N reference system. In Figure 6.6a, a portion of the mosaic, of traverse 1, is depicted. The mosaic is processed using the Rasterio [136] library. The artificial rocks in three sizes, small, medium, and large can be seen. We are unable to render the entire mosaic, as processing the large geotiff file uses the maximum memory capacity of our environment, i.e., 8 GiB.

To obtain the positions of the small, medium, and large rocks, we extract the northing and easting, i.e., UTM 31 N coordinates, from the `{small,medium,large}-rocks-traverse{12}.txt` text file (see Table 6.4), where each row in the file corresponds to a rock position. Using the Rasterio `transform` function we obtain an affine transformation matrix that maps pixel locations to UTM 31 N spatial positions. The inverse of the affine transformation matrix maps the reverse, i.e., the UTM 31 N spatial positions to pixel locations. This transformation is described in Eq. 6.1.

$$(x_p, y_p) = \begin{bmatrix} 0.02 & 0.00 & 595591.45 \\ 0.00 & -0.02 & 5786848.79 \\ 0.00 & 0.00 & 1.00 \end{bmatrix} \begin{bmatrix} x_{UTM} \\ y_{UTM} \\ 1 \end{bmatrix} \tag{6.1}$$

Where (x_p, y_p) are the pixel coordinates, the left matrix the affine transformation matrix, x_{UTM} and y_{UTM} the UTM 31 N spatial positions. The three sizes of rocks are plotted in the mosaic using the inverse affine transformation matrix, depicted in Figure 6.6b.

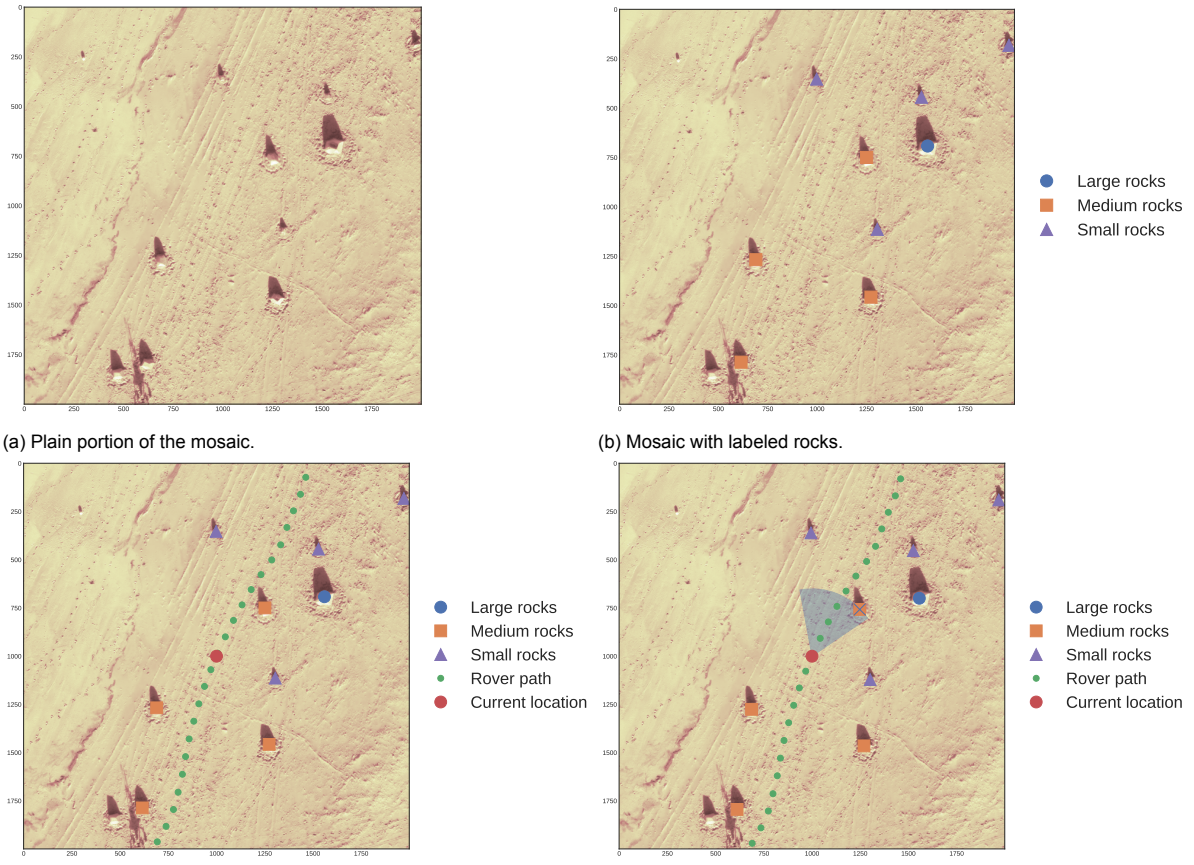
The GPS measurements of the rover are found in the `gps-latlong.txt` text file (see Table 6.4), where each row contains a timestamp, with an interval of 3 seconds, the latitude, and longitude. To plot the rover positions onto the mosaic we have to transform the latitude and longitude to pixel coordinates. The transform consists of two steps: (1) latitude and longitude to UTM 31 N coordinates; (2) UTM 31 N coordinates to pixel coordinates. The first step is performed using the `Proj` function from the pyproj [137] library, which takes the latitude and longitude coordinates as the input and outputs the UTM 31 N coordinates. The second step is performed using Eq. 6.1, to transform the UTM 31 N coordinates to pixel coordinates. In Figure 6.6c, the GPS measurements of the rover are depicted, where the red dot represents the current location centered in the mosaic.line We observe a sparse path, as the time between each GPS measurement is 3 seconds, while LocCam images are taken every 250 milliseconds. To obtain a rover position for every LocCam image, we linearly interpolate between each two GPS measurements to generate a GPS measurement for every LocCam image.

From RQ1 (see Section 3.4), the rover characterises approximately 6 meters of terrain in front of it, such that we can create a region of interest. In combination with the horizontal FOV of the LocCam stereo camera, a wedge shaped region of interest can be created. The horizontal FOV of the LocCam stereo camera, i.e., PointGrey Bumblebee2 (BB2-08S2C-38), is 66° [138]. In Figure 6.6d, the wedge shaped region of interest is depicted. The observable range is set to 6 meters and rocks inside of this region are marked with a cross.

Now we have all elements to determine the ground-truth distance to rocks in the scene. That are, the rock positions, rover position, and region of interest. When a rock or multiple rocks enter the wedge, the distance to the rock from the rover is computed using,

$$dist_{true} = \sqrt{(x_c - x_r)^2 + (y_c - y_r)^2} - d_{gl} - r_r \tag{6.2}$$

Where $dist_{true}$ is the ground-truth distance from the rover to the rock in meters, x_c and y_c the UTM 31 N coordinates of the rover, x_r and y_r the UTM 31 N coordinates of the rock, d_{gl} the distance between the GPS antenna and LocCam stereo camera on the rover (0.40 m), and r_r the radius of the detected rock; small rock: 0.37 m; medium rock: 0.66 m; large rock: 0.95 m. Because the center point coordinates of the rocks are given, the radius is subtracted to determine the distance to the outer surface of the rocks.



(a) Plain portion of the mosaic.

(b) Mosaic with labeled rocks.

(c) Mosaic with labeled rocks, path, and current location.

(d) Mosaic with labeled rocks, path, current location, and FOV wedge.

Figure 6.6: Mosaic

6.5.3. Predicted distance

To determine the predicted distances to rocks in the scene, we process the predicted disparity maps and extract the distances. We make use of the U-disparity concept, transforming the predicted disparity map to a top-view map. Followed by extracting the rocks using edge detection and Hough lines. Finally, the Hough lines are transformed to Cartesian coordinates. These steps are described in more detail in the following sections.

U-Disparity Map

To transform the predicted disparity map to a top-view map, we make use of the U-disparity concept [139]. The U-disparity map counts the appearances of disparity values in a column wise manner. Meaning that a column in the U-disparity map is the histogram of disparity values, on the corresponding disparity image column. The number of rows correspond to the number of disparity bins, while the number of columns equals the disparity image width. Constructing the U-disparity map consists of iterating through the disparity image and for each pixel disparity image, increment the pixel in the U-disparity

map, where the intensity of the disparity map image corresponds to the U-disparity map row position, and the x-location of the disparity image pixel corresponds to the U-disparity map column position. In Figure 6.7c, the U-disparity map of a sample predicted disparity map (see Figure 6.7b), using the SAD algorithm, is depicted. We observe a horizontal line (u = 850, d = 13) corresponding to a rock in the scene, with additional vague horizontal lines due to imperfect stereo matching and a bumpy terrain.

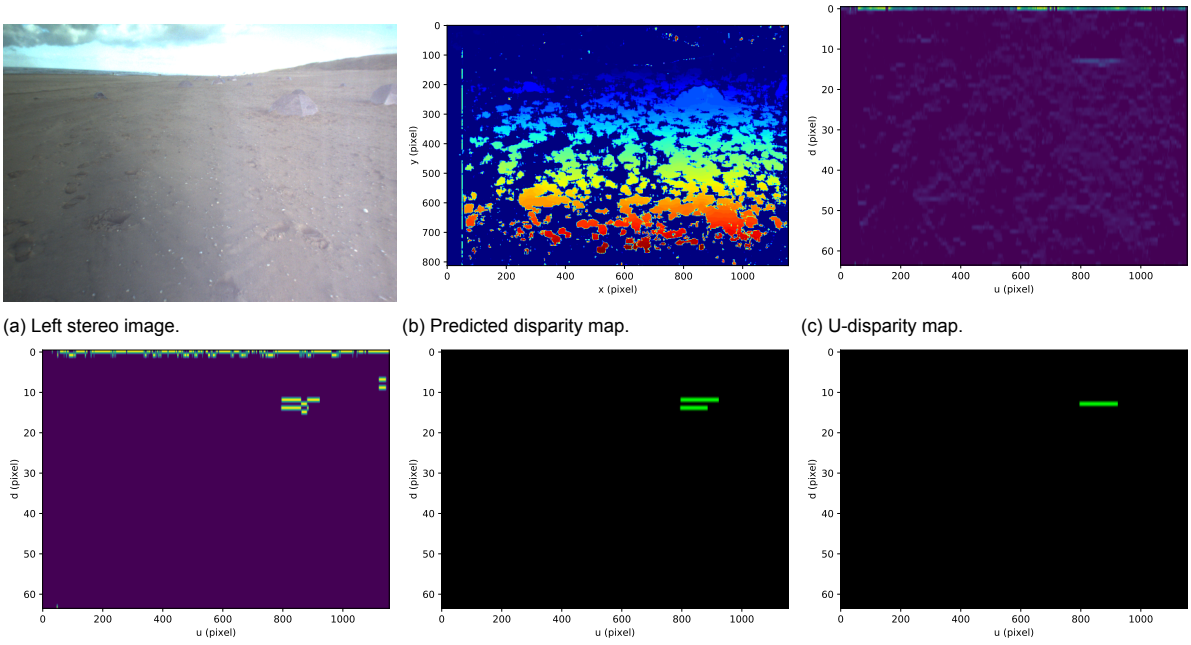
Edge detection

To filter out the vague horizontal lines in the U-disparity map, we apply edge detection on the U-disparity map. We use the Canny edge detector [140] in OpenCV. The edge detector transforms the U-disparity map to a binary map, where detected edges of rocks are labeled with a value of 1, and the rest with a 0 value. In Figure 6.7d the result of the edge detector applied to the U-disparity map is shown. We observe the detected edges of rocks in the scene, along with edges near disparity values of 0. Since the predicted disparity map contains numerous disparity values of zero, these show up as brighter regions in the U-disparity map, thus are detected as edges.

Hough line transform

The Hough line transform [141] is applied to the edge detected image, such that straight lines are extracted. These each extracted line is represented by a start coordinate (u_1, d_1) and an end coordinate (u_2, d_2). The Hough line transform can detect detect disconnected and distorted lines in the edge detected image. In Figure 6.7e, the Hough lines are depicted. We observe two straight lines corresponding to the top and bottom edge of the horizontal line in the U-disparity map.

The correct line corresponding to the horizontal line in the U-disparity map is between the top and bottom Hough lines. In Figure 6.7f, the correct Hough line is shown.



(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure 6.7: Transformation of a sample predicted disparity map to filtered Hough lines.

U-D space to Cartesian

The Hough lines in the U-D space are transformed, such that the predicted distances are in Cartesian coordinates, i.e., X-Z space. In Figure 6.8, the left stereo camera geometry for the transformation from U-D space to X-Z space is depicted. Where I_0 is the left image plane, c_0 the left camera center, p the object in the scene, f the focal length in pixels , u the x-coordinate in the image plane in pixels, $dist'$ the distance from the camera center to the image plane, z the perpendicular distance to the point in meters, x the x-coordinate in the Cartesian plane in meters, and $dist$ the distance to the point in meters.

Two similar triangles, with sides (z, x, dist) and sides (f, u, dist'), can be seen. Such that the following holds,

$$\frac{z}{f} = \frac{x}{u} \quad \rightarrow \quad x = u\frac{z}{f} \quad \rightarrow \quad x = \frac{Bu}{d} \quad ; \text{ where } z = \frac{fB}{d} \tag{6.3}$$

Where the focal length f is 834 pixels (from the `LocCam_calibration.mat` data file) and the baseline of the stereo camera is 0.12 meters (see Table 6.3).

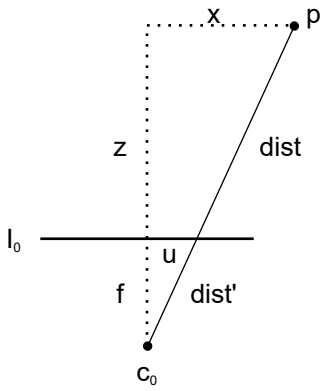


Figure 6.8: Geometry for the transformation from U-D space to X-Y space. The left stereo camera is depicted.

The predicted distance is described as,

$$dist_{pred} = \sqrt{x^2 + z^2} \tag{6.4}$$

6.5.4. Error rate metric

Combining the true distance and predicted distance, we develop the following error rate metric,

$$e = \left| dist_{pred} - dist_{true} \right| \tag{6.5}$$

Where $dist_{pred}$ is formulated in Eq. 6.4 and $dist_{true}$ in Eq. 6.2. From R3 (see Chapter 3.4), the rover must maintain a position estimate accurate to within 10% of distance travelled, we can formulate the following,

$$e \leq 0.10 \cdot dist_{true} \tag{6.6}$$

For example, if the rover transverses 1 meter to a target, the target must lie within a radius of 0.1 meter of the rover.

7

Experimental Results

We address in this chapter the results of our experiments to address our second research question. The setup of our two experiments is described in detail in Chapter 6. We use the metrics developed in Chapter 6 to evaluate the results. Our approach to analyze the distance estimation error rate is inspecting the maps generated from the stereo image, i.e., the predicted disparity map, U-disparity map, edge detection map, and (filtered) Hough line transform maps.

The remainder of this chapter is structured as follows. In Section 7.1, we perform the first experiment and analyze the results of the error rate. In Section 7.2, we perform the second experiment and analyze the results of the error rate. The run time of the four algorithms during the two traverses are presented in Section 7.3. The memory usage of the four algorithms to produce a single predicted disparity map is presented in Section 7.4 The results of the first and second experiment are combined and discussed in Section 7.5.

7.1. Experiment 1

The first experiment evaluates the predicted disparity maps in a scene where the sun is positioned behind the rover. During the traverse of the rover, it encounters small, medium, and large rocks within its detection region, i.e., a wedge shape with a range of 6 meters and a FOV of 66° as described in Chapter 6.5.2. To accommodate requirements R1 and R3, we developed an error rate metric (see Chapter 6.5.1). In the following figures, we depict the time-distance plots of each algorithm during traverse 1, with separate small, medium, and large plots. The horizontal axes depicts the timestamp in hh:mm:ss:fff format, that is hours, minutes, seconds, and milliseconds. The vertical axes depicts the distance to the rock in meters. The true distance curve is marked in green, the predicted distance from the algorithm is marked in blue, and points outside the error band (light red) are marked in red. Different encounters of rocks are separated by a vertical dashed black line, we should ignore the line connecting the two different rock encounters.

It is important to mention again, that the pipeline of predicting disparity maps is memory-less, i.e., single still stereo images are processed without the dependence of previous stereo images. This means that the predicted distance of each single stereo image is not dependent on the previous predicted distance(s).

For the StereoBM (SAD) and StereoSGBM (SGM) algorithms, the window size is the variable parameter. We use a window size of 23 and 27, respectively, determined in Chapter 5.1.1 and Chapter 5.1.3 by selecting the lowest noise percentage estimate of the disparity maps. For the Hartley (DP) algorithm, the occlusion cost is the variable parameter. We use an occlusion cost of 40, determined in Chapter 5.1.2 by selecting the lowest noise percentage estimate of the disparity maps.

In Section 7.1.1, the results of the StereoBM (SAD) algorithm are analyzed, where we present the analysis of erroneous points in detail, i.e., showing the U-disparity map, edge detection map, and (filtered) Hough line transform map. In the following sections the maps are omitted to improve readability and are referred to in Appendix B.1. The results of the Hartley (DP) algorithm are analyzed in Section 7.1.2. The results of the StereoSGBM (SGM) algorithm are analyzed in Section 7.1.3. In Section 7.1.3, results of the PWOC-3D algorithm are analyzed.

7.1.1. StereoBM (SAD)

Figure 7.1 shows the true and predicted distance points of small rock encounters. Overall, we observe 9 erroneous points over a total of 55 points. We inspect the erroneous points by looking at the U-disparity map, edge detection map, and (filtered) Hough line transform map. In Figure B.1 the maps are shown. Figure 7.2d shows the edge detection map with two detected rocks, at (u = 226, d = 8) and (u = 1032, d = 19). However, in the Hough line transform map in Figure 7.2e the rock the right side is not detected by the Hough transform. Such that the left rock is taken as the predicted distance. We compute the distance of the rock on the right side, which is 6.00 meters. This means that if the right rock would be detected, the predicted distance would be within the error band. For the same reason, the other erroneous points can be explained. The point at time 12:56:19:924 seems to be detected correctly and the error could be attributed to a stereo matching error, however upon closer inspection we find that it also is a point corresponding to a wrong rock (see Figure B.1). The correct rock at (u = 532, d = 17) in the edge detection map, results in a predicted distance of 5.90 meters, which is within the error band.

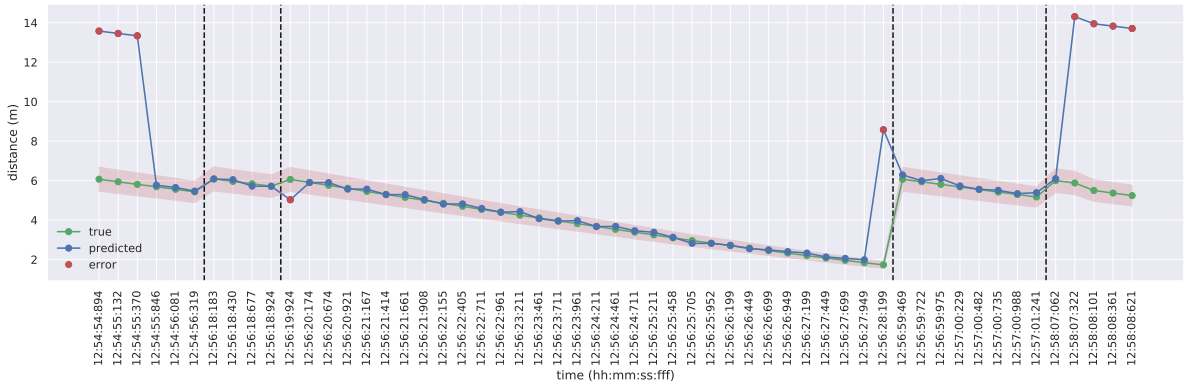
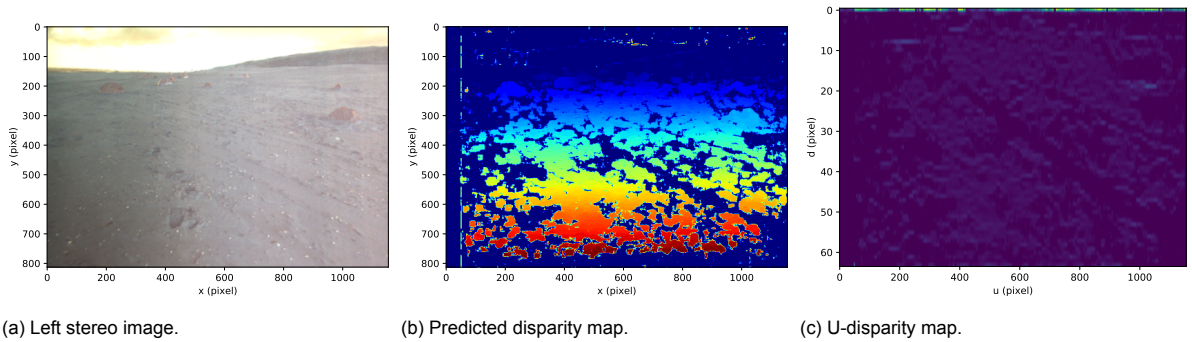


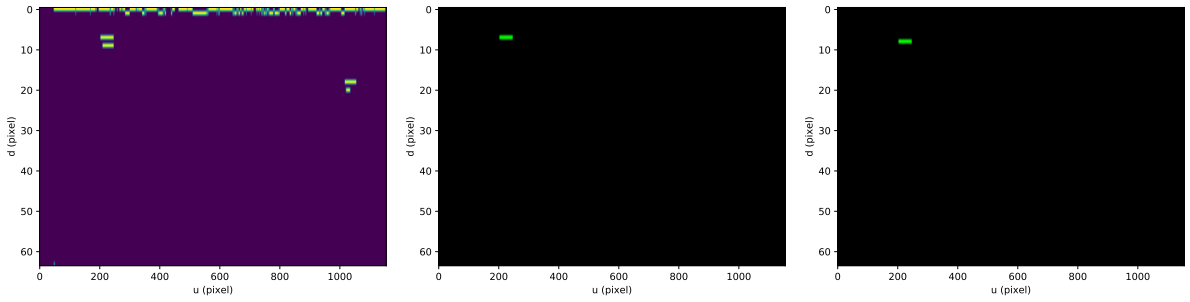
Figure 7.1: Small rocks.



(a) Left stereo image.

(b) Predicted disparity map.

(c) U-disparity map.



(d) Edge detection.

(e) Hough lines.

(f) Filtered Hough lines.

Figure 7.2: First erroneous point analysis of the small rocks.

Figure 7.3 shows the true and predicted distance points of medium rock encounters. Overall, we

observe 8 erroneous points over a total of 107 points. Using the same procedure as before we analyze the erroneous points. The four erroneous points in the third medium rock encounter can be attributed to the stereo matching performance of the StereoBM algorithm. The rock is detected (see Figure B.2), however the disparity value is larger than expected, resulting in a smaller predicted distance. In the fourth and seventh medium rock encounter the errors are due to undetected rocks in the scene, this causes the predicted distance to be set to the rock farther away.

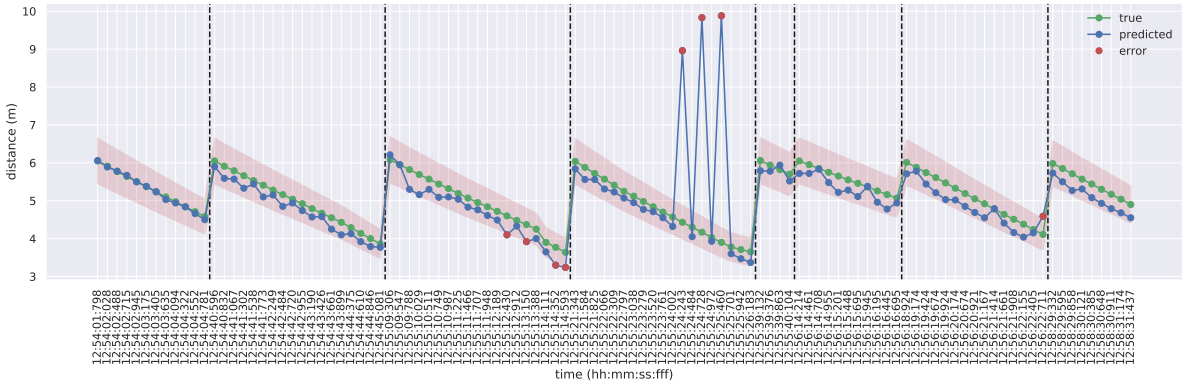


Figure 7.3: Medium rocks.

Figure 7.4 shows the true and predicted distance points of large rock encounters. Overall, we observe 1 erroneous point over a total of 14 points. Using the same procedure as before we analyze the erroneous point. In Figure B.3, we observe the detected rock, however we attribute the error due to the stereo matching performance.

Figure 7.4: Large rocks.

Combining Figures 7.1, Figure 7.3 and Figure 7.4 shows that the error rate is $(18/176) \cdot 100\% = 10.2\%$.

7.1.2. Hartley (DP)

Figure 7.5 shows the true and predicted distance points of small rock encounters. Overall, we observe 50 erroneous points over a total of 55 points. The last five points are within the error band, however the rest is outside the error band. We compare the maps of the points where the predicted distance is within the error band and points where they are outside the error band. Figure B.4 shows the maps of a predicted distance within the error band, i.e., one of the last five points. We observe the detected small rock in the disparity map, U-disparity map, edge detection map, and (filtered) Hough line transform

map. Figure B.5 shows the maps of an erroneous point. The small rock in the middle of the scene is not visible in the predicted disparity map, because of the matching performance. It leads to the rock is not being present in the U-disparity map and following maps. We attribute the erroneous points due to the lighting conditions in the scene. Comparing the left stereo image of both figures, we observe a high contrast between the rock and the ground in Figure B.4a, while in Figure B.5a a shadow is cast over the rock because of the angle of the light hitting the camera, such that the contrast is reduced.

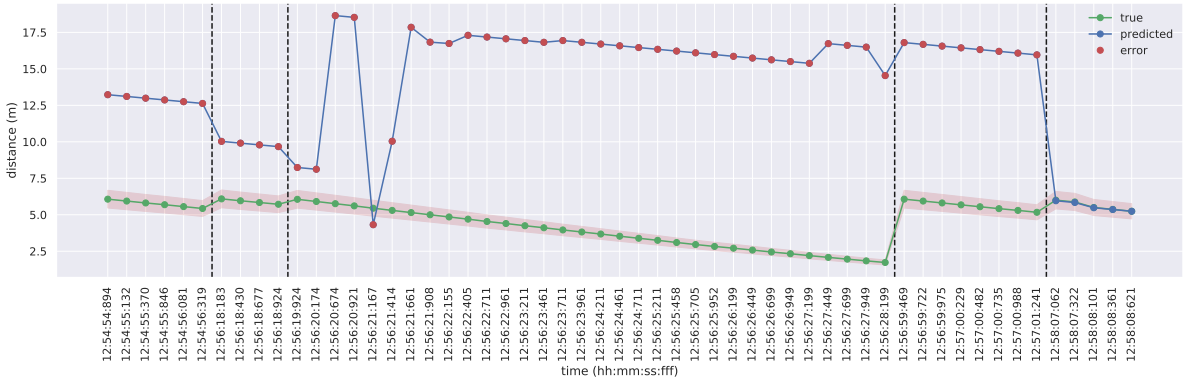


Figure 7.5: Small rocks.

Figure 7.6 shows the true and predicted distance points of medium rock encounters. Overall, we observe 57 erroneous points over a total of 107 points. Again, we compare the maps of the points where the predicted distance is within the error band and points where they are outside the error band. Figure B.6 shows the maps of a predicted distance within the error band. We observe the detected medium rock in the disparity map, U-disparity map, edge detection map, and (filtered) Hough line transform map. Figure B.7 shows the maps of an erroneous point. The medium rock on the left of the scene is not visible in the predicted disparity map and instead the middle rock is detected. Comparing the left stereo image of both figures, we observe a high contrast between the rock (a bright area on the rock) and the ground in Figure B.6a, while in Figure B.7a a shadow is cast over the rock because of the angle of the light hitting the camera, such that the contrast is reduced.

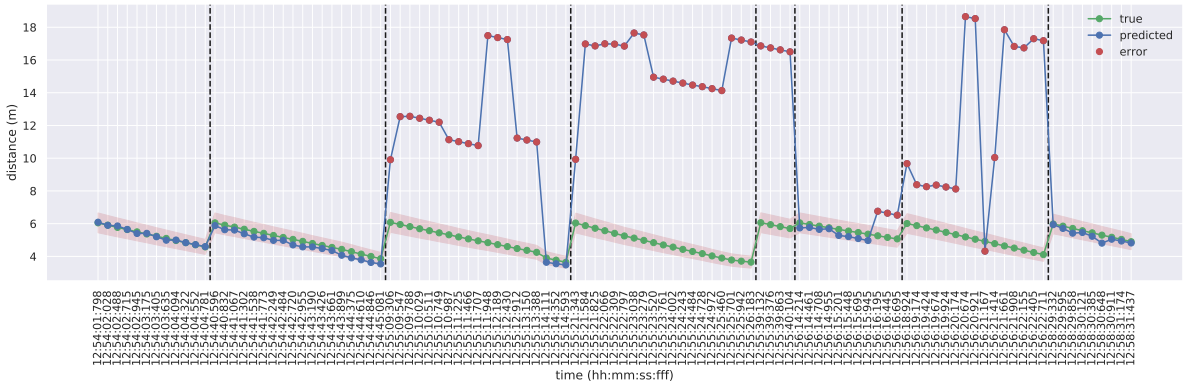


Figure 7.6: Medium rocks.

Figure 7.7 shows the true and predicted distance points of large rock encounters. Overall, we observe 2 erroneous points over a total of 14 points. Figure B.8 shows the detected rock in all maps of an erroneous point, however the stereo matching process did not infer the correct disparities.

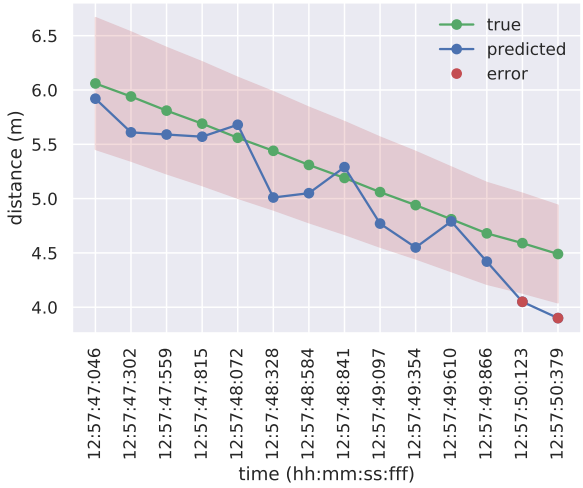


Figure 7.7: Large rocks.

Combining Figures 7.5, Figure 7.6 and Figure 7.7 shows that the error rate is $(109/176) \cdot 100\% = 61.9\%$.

7.1.3. StereoSGBM (SGM)

Figure 7.8 shows the true and predicted distance points of small rock encounters. Overall, we observe 13 erroneous points over a total of 55 points. Figure B.9 shows the maps of an erroneous predicted distance during the first rock encounter. The U-disparity map contains indistinct lines corresponding to the small rock, such that the following maps do not detect the small rock. Another prominent erroneous point is during the fourth rock encounter with the maps shown in Figure B.10. The small rock is detected, however during filtering of the Hough line transform map, the lines are removed. This is because the line corresponding to the rock in the U-disparity map is not completely detected.

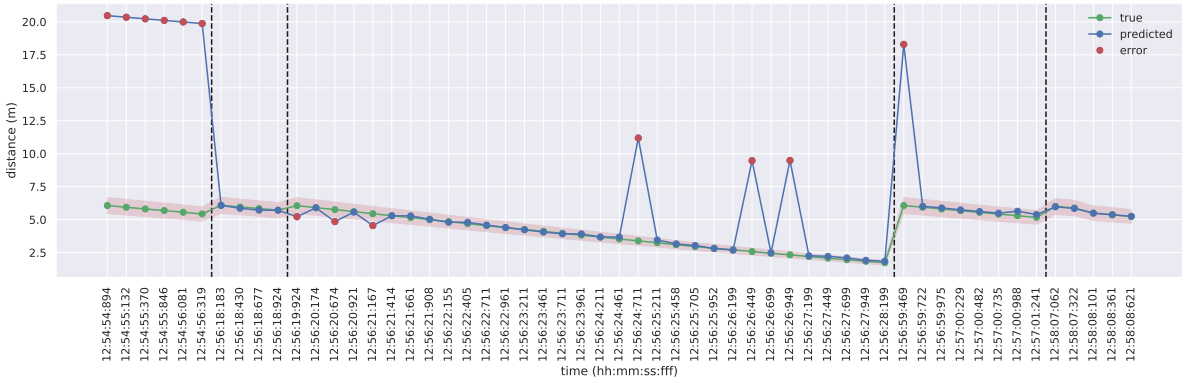


Figure 7.8: Small rocks.

Figure 7.9 shows the true and predicted distance points of medium rock encounters. Overall, we observe 5 erroneous points over a total of 107 points. Figure B.11 shows the maps of an erroneous point during the fifth rock encounter. The edge detection map contains a large amount of edges, however in the Hough line transform map the line corresponding to the rock is not detected.

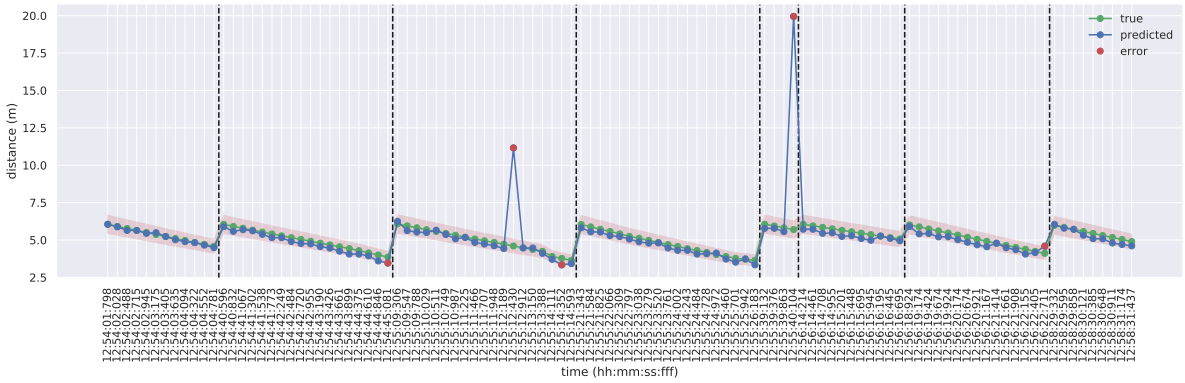


Figure 7.9: Medium rocks.

Figure 7.10 shows the true and predicted distance points of large rock encounters. Overall, we observe 1 erroneous points over a total of 14 points. Figure B.12 shows the maps of the erroneous point. It can be observed that the large rock is detected in the maps, however due to the stereo matching performance the predicted distance is closer than the true distance.

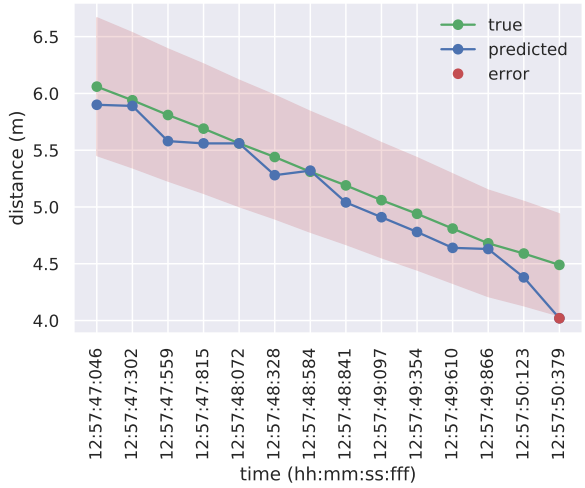


Figure 7.10: Large rocks.

Combining Figures 7.8, Figure 7.9 and Figure 7.10 shows that the error rate is $(19/176) \cdot 100\% = 10.8\%$.

7.1.4. PWOC-3D

Figure 7.11 shows the true and predicted distance points of small rock encounters. Overall, we observe 19 erroneous points over a total of 55 points. Figure B.13 shows the maps of an erroneous point during the third encounter. Inspecting the U-disparity map, edge detection map, and (filtered) Hough line transform, we observe the detection of additional rocks not present in the scene. This is caused by the stereo matching performance to generate a smooth disparity map, i.e., without unmatched areas. The filled in areas can lead to the addition of rocks. The small rock in the scene is detected, however disparities from stereo matching are too high.

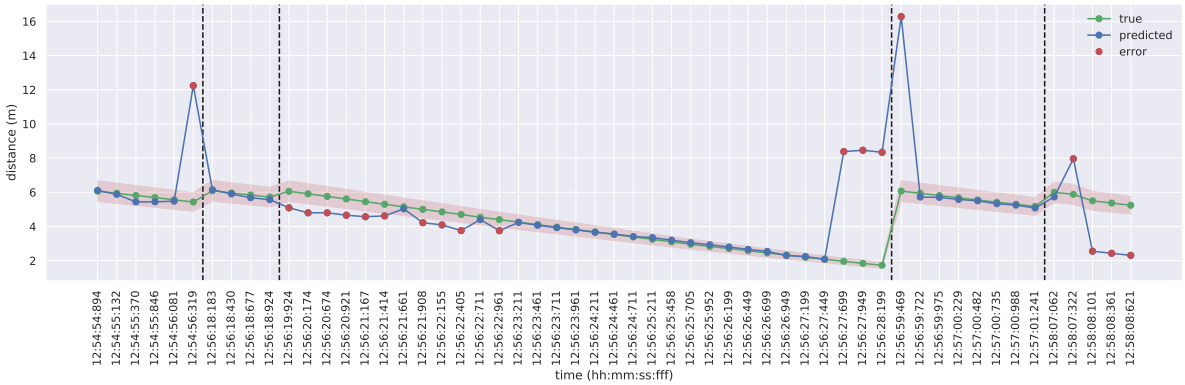


Figure 7.11: Small rocks.

Figure 7.12 shows the true and predicted distance points of medium rock encounters. Overall, we observe 9 erroneous points over a total of 107 points. Figure B.14 shows the maps of an erroneous point during the third encounter. Inspecting the U-disparity map, edge detection map, and (filtered) Hough line transform, we observe the detection of the medium rock, however due to the stereo matching performance the predicted distance of the rock is closer than the true distance.

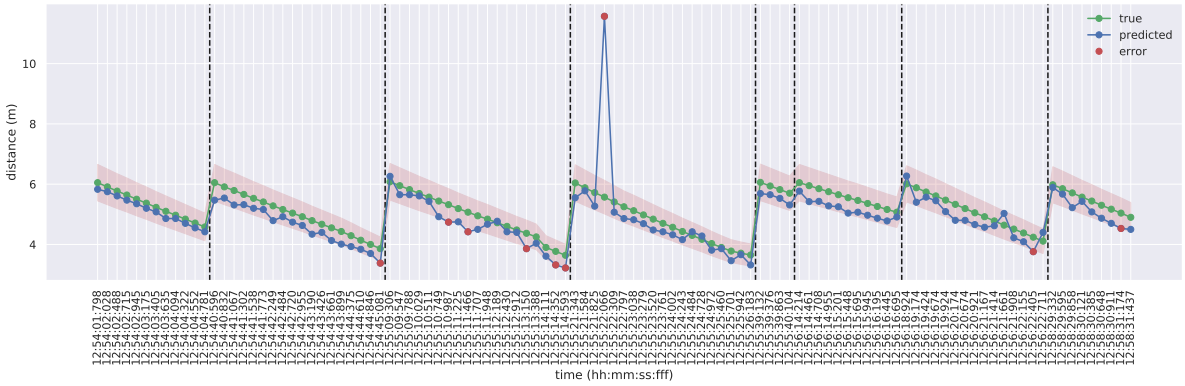


Figure 7.12: Medium rocks.

Figure 7.13 shows the true and predicted distance points of large rock encounters. Overall, we observe 0 erroneous points over a total of 14 points.

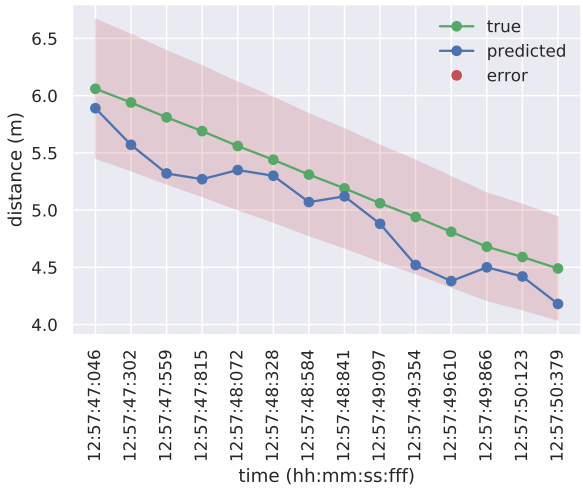


Figure 7.13: Large rocks.

Combining Figures 7.11, Figure 7.12 and Figure 7.13 shows that the error rate is $(28/176) \cdot 100\% = 15.9\%$.

7.2. Experiment 2

The second experiment evaluates the predicted disparity maps in a scene where the sun is positioned in front of the rover. Similar to the first experiment, the following figures depict the encounter of small, medium, and large rocks. The general description of the figures are described in Section 7.1. We use the same parameters for the window sizes, for the StereoBM and StereoSGBM algorithms, and occlusion cost, for the Hartley algorithm. This is because we assume the parameters are hard-coded.

In Section 7.2.1, the results of the StereoBM (SAD) algorithm are analyzed. The results of the Hartley (DP) algorithm are analyzed in Section 7.2.2. The results of the StereoSGBM (SGM) algorithm are analyzed in Section 7.2.3. In Section 7.2.3, results of the PWOC-3D algorithm are analyzed. In the following sections the maps are omitted to improve readability. They are referred to in Appendix B.2.

7.2.1. StereoBM (SAD)

Figure 7.14 shows the true and predicted distance points of small rock encounters. Overall, we observe 109 erroneous points over a total of 176 points. Figure B.15 shows the maps of a correctly predicted distance during the fourth rock encounter. The disparity map are mostly unmatched due to the shadows cast on the rocks, i.e., it contains zero valued disparities. In general the edges of the rocks are matched, because of the contrast between the rock and ground, however the regions within the edges are unmatched. In points where the predicted distance is close to the true distance, the line in the Hough line transform corresponding to a rock is not fully present, such that the predicted distance is off.

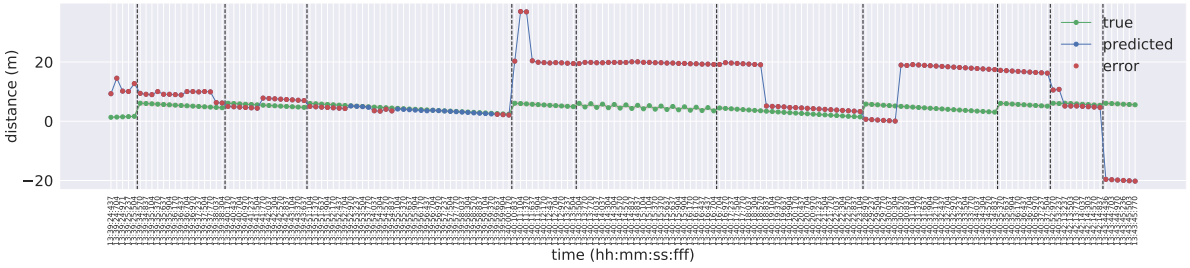


Figure 7.14: Small rocks.

Figure 7.15 shows the true and predicted distance points of medium rock encounters. Overall, we observe 109 erroneous points over a total of 109 points. We observe intervals during the encounters where the predicted distance is close to the true distance. Because the rock in the predicted disparity map is not fully matched, the predicted distance is off.

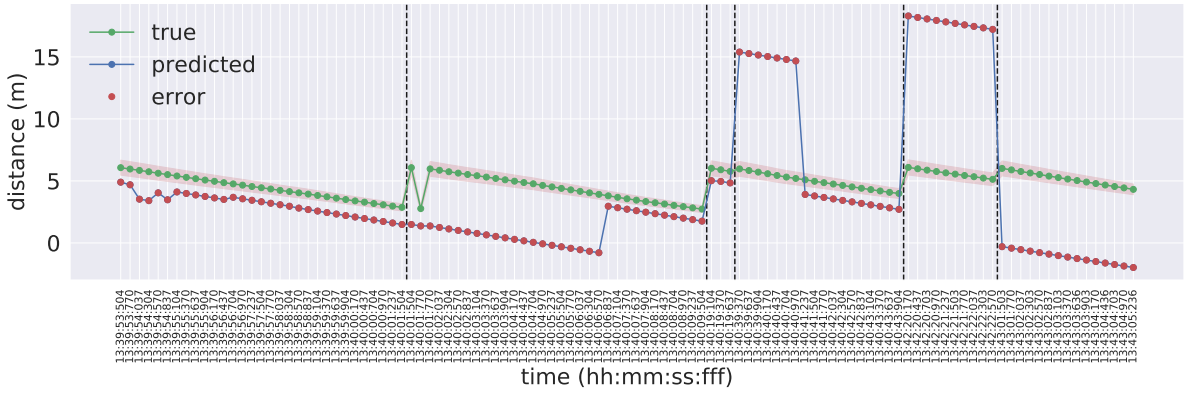


Figure 7.15: Medium rocks.

Combining Figures 7.14 and Figure 7.15 shows that the error rate is $(218/285) \cdot 100\% = 76.5\%$.

7.2.2. Hartley (DP)

Figure 7.16 shows the true and predicted distance points of small rock encounters. Overall, we observe 162 erroneous points over a total of 176 points. Figure B.16 shows a correct predicted distance, while Figure B.17 shows an incorrect predicted distance. Comparing the disparity maps of the two points, we observe that the rock in correct predicted case is fully matched, whereas in the erroneous case the sunlight obstructs the rock. This results in the rock not being visible in the U-disparity map.

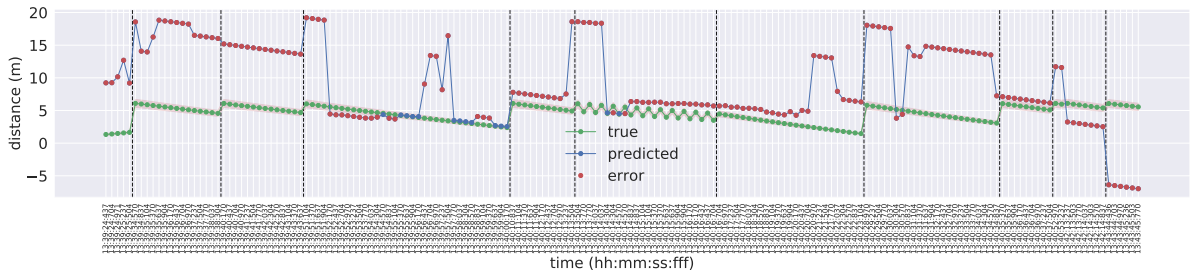


Figure 7.16: Small rocks.

Figure 7.17 shows the true and predicted distance points of medium rock encounters. Overall, we observe 101 erroneous points over a total of 109 points.

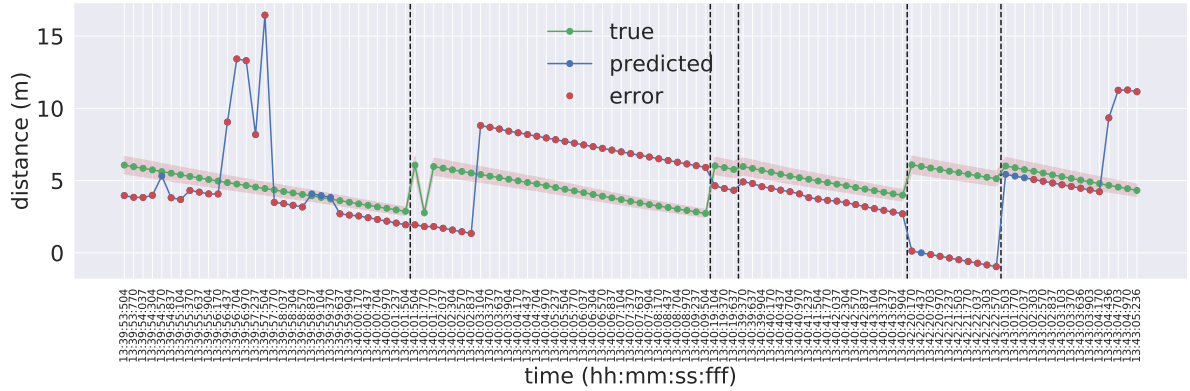


Figure 7.17: Medium rocks.

Combining Figures 7.16 and Figure 7.17 shows that the error rate is $(263/285) \cdot 100\% = 92.3\%$.

7.2.3. StereoSGBM (SGM)

Figure 7.18 shows the true and predicted distance points of small rock encounters. Overall, we observe 139 erroneous points over a total of 176 points. Figure B.18 shows the maps of an erroneous point during the second encounter. We observe the small rock is not clearly visible in the U-disparity map, due to inaccurate stereo matching.

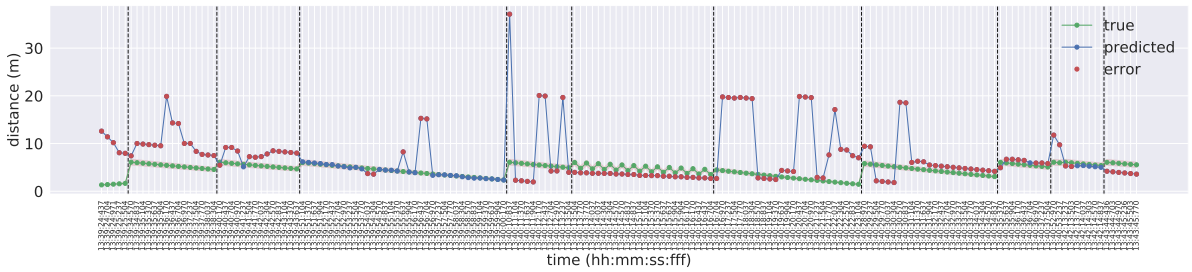


Figure 7.18: Small rocks.

Figure 7.19 shows the true and predicted distance points of medium rock encounters. Overall, we observe 98 erroneous points over a total of 109 points.

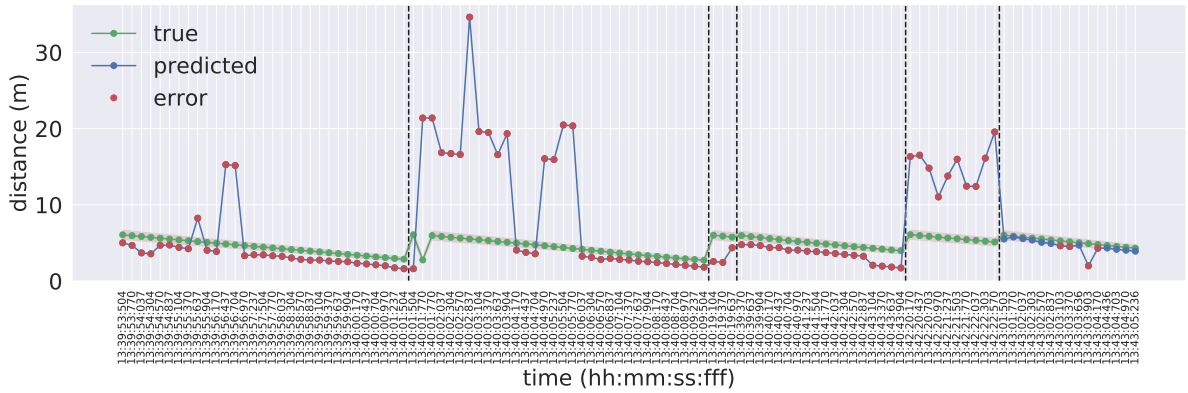


Figure 7.19: Medium rocks.

Combining Figures 7.18 and Figure 7.19 shows that the error rate is $(218/285) \cdot 100\% = 83.2\%$.

7.2.4. PWOC-3D

Figure 7.20 shows the true and predicted distance points of small rock encounters. Overall, we observe 111 erroneous points over a total of 176 points. We observe no errors during the fourth encounter.

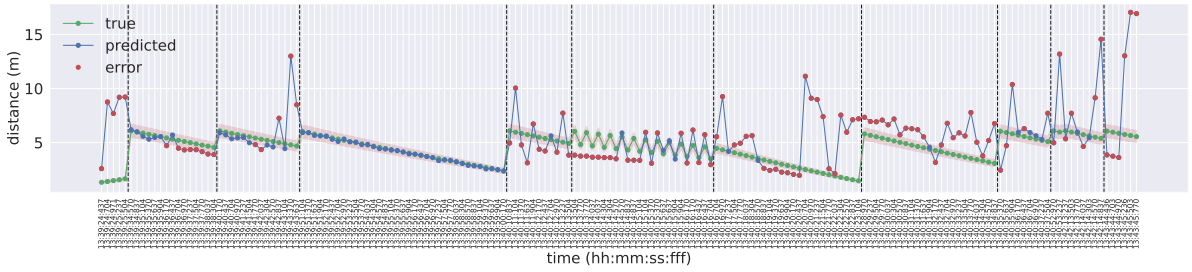


Figure 7.20: Small rocks.

Figure 7.21 shows the true and predicted distance points of medium rock encounters. Overall, we observe 103 erroneous points over a total of 109 points. The predicted distances are mostly following the true distances, however they are on average 0.90 meters less than the true distances.

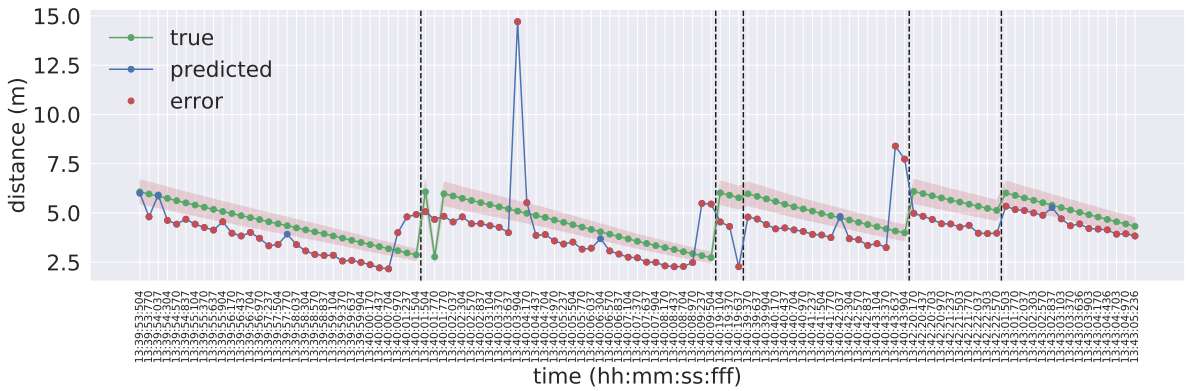


Figure 7.21: Medium rocks.

Combining Figures 7.14 and Figure 7.15 shows that the error rate is $(214/285) \cdot 100\% = 75.1\%$.

7.2.5. Discussion

The error rate is determined during two experiments for each algorithm, listed in Table 7.1. In the first experiment, where the sun is behind the rover, the algorithm with the lowest error rate is StereoBM (SAD) with an error rate of 10.2%. The StereoBM (SAD) algorithm, leaves parts of the terrain unmatched, however the rocks are matched and are visible in the predicted disparity map. The Hartley (DP) algorithm, performs different depending on the lighting condition, which affects the contrast. A high contrast between the rock and the ground, allows for better stereo matching. The StereoSGBM (SGM) algorithm performs similar to the StereoBM algorithm. The predicted disparity map is more smooth, however there are incorrectly matched areas where the disparity value is near maximum. The PWOC-3D algorithm, provides a full predicted disparity map, however this leads to additional rocks in the scene that are not present.

In the second experiment, where the sun is in front of the rover, the algorithm with the lowest error rate is PWOC-3D, with an error rate of 75.1%, followed by StereoBM (SAD) with an error rate of 76.5%. The StereoBM (SAD) algorithm, generates the edges of the rock, however leave the area inside unmatched causing the predicted distance to be off. The Hartley (DP) algorithm, performs different depending on the lighting condition, which affects the contrast. The StereoSGBM (SGM) algorithm performs similar to the StereoBM algorithm. The predicted disparity map is more smooth, however there are incorrectly matched areas where the disparity value is near maximum. The PWOC-3D algorithm, computes predicted distances which follow the shape of the true distances, with an offset.

In general, we observe a high error rate for all algorithms during the second traverse, compared to the first traverse. This is because of the harsh lighting conditions, where the sun is facing the front of the rover. Looking at each erroneous predicted distance, the errors originate from one of two sources. (1) The stereo matching performance of the algorithm, which contributes to smaller errors. (2) Line detection in the U-disparity map, edge detection map, and (filtered) Hough line transform map. This is because of the variable parameters in the generation of these maps, and contribute to large errors. Tweaking these variable parameters would improve the line detection in the maps.

Implementation	Method	Traverse	Error (%)
StereoBM	SAD	1	10.2
StereoBM	SAD	2	76.5
Hartley	DP	1	61.9
Hartley	DP	2	92.3
StereoSGBM	SGM	1	10.8
StereoSGBM	SGM	2	83.2
PWOC-3D	-	1	15.9
PWOC-3D	-	2	75.1

Table 7.1: Error rates of the algorithms on traverse 1 and 2.

7.3. Run time

The run time of the algorithms on our experimental environment, i.e., the Raspberry Pi 4 Model B, is listed in Table 7.2. The table lists the implementation, method, traverse, and the run time in seconds. To compute the mean run time and standard deviation, the algorithms are iterated for 10 times on different input images. We observe that for traverse 1 and 2, the run time is similar. This is expected behaviour, as it depends on the resolution of the input image, which is 1155 x 813 for both traverses.

To accommodate with requirement R2, i.e., each call of the perception system must take less than 20 seconds on our environment, we select the StereoBM and StereoSGBM algorithms. The algorithms are part of the OpenCV library and are optimized for speed. The Hartley and PWOC-3D algorithm have a run time larger than 20 seconds. The Hartley code is not optimized and thus is slow compared to the OpenCV algorithms. The PWOC-3D code uses a neural network typically run on a GPU, the larger run time is explained by the use of the less powerful processor of our experimental environment.

Implementation	Method	Traverse	Run time (s)
StereoBM	SAD	1	0.142 ± 0.00591
StereoBM	SAD	2	0.136 ± 0.00509
Hartley	DP	1	21.2 ± 1.57
Hartley	DP	2	21.5 ± 1.59
StereoSGBM	SGM	1	2.04 ± 0.0341
StereoSGBM	SGM	2	2.00 ± 0.0146
PWOC-3D	-	1	30.1 ± 1.72
PWOC-3D	-	2	30.3 ± 1.65

Table 7.2: Run time of the algorithms on traverse 1 and 2.

7.4. Memory Usage

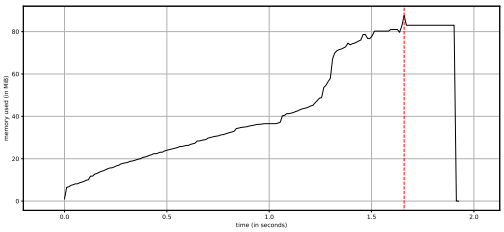
To compute the memory usage of the algorithms we use the memory_profiler [142] Python module. This module reports the memory usage as a function of time. In Figure 7.22, the memory usage plots are depicted of the algorithms during both traverses. The horizontal axes shows the time in seconds and the vertical axes shows the memory used in MiB. The vertical dashed red line shows the time at which the maximum memory is used. In Table 7.3, the maximum memory usage of each algorithm is listed.

Implementation	Method	Memory usage (MiB)
StereoBM	SAD	90
Hartley	DP	166
StereoSGBM	SGM	85
PWOC-3D	-	1428

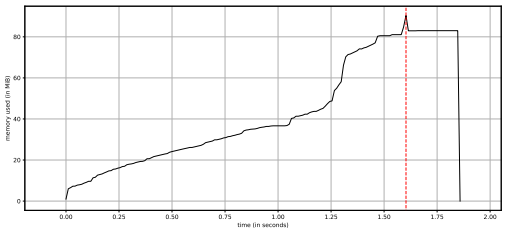
Table 7.3: Maximum memory usage of the algorithms.

Figure 7.22a and Figure 7.22b show the memory usage of StereoBM (SAD). The maximum memory used during traverse 1 and 2 is 88 MiB and 90 MiB, respectively, at time 1.6 seconds. Figure 7.22c and Figure 7.22d show the memory usage of Hartley (DP). The maximum memory used during traverse 1 and 2 is 151 MiB and 166 MiB, respectively, at time 2.8 seconds and 8.0 seconds. The difference in time is explained by the loading time of the libraries. Figure 7.22e and Figure 7.22f show the memory usage of StereoSGBM (SGM). The maximum memory used during traverse 1 and 2 is 85 MiB and 85 MiB, respectively, at time 3.5 seconds. Figure 7.22g and Figure 7.22h show the memory usage of PWOC-3D. The maximum memory used during traverse 1 and 2 is 1428 MiB and 1316 MiB, respectively, at time 50 seconds and 49 seconds. The difference in time is explained by the loading time of the libraries.

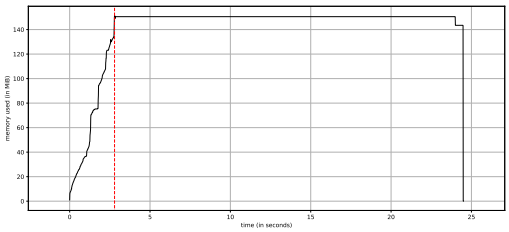
To accommodate with requirement R4, the perception system must use less than 512 MiB of memory, the StereoBM, Hartley, and StereoSGM implementations are selected. Both OpenCV implementations, i.e., StereoBM and StereoSGM, are optimized and run efficiently. The Hartley implementation uses the third most memory, while the PWOC-3D implementation uses the most memory. Since the PWOC-3D code uses convolutional neural networks, it needs to store a number of large matrices.



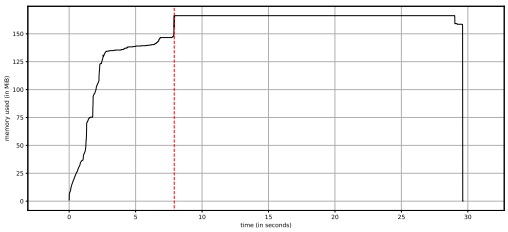
(a) StereoBM memory usage during traverse 1.



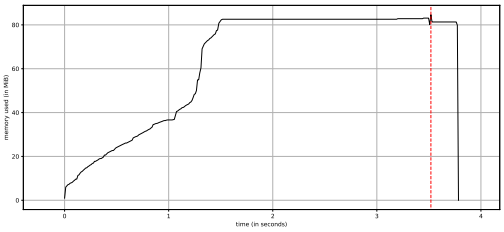
(b) StereoBM memory usage during traverse 2.



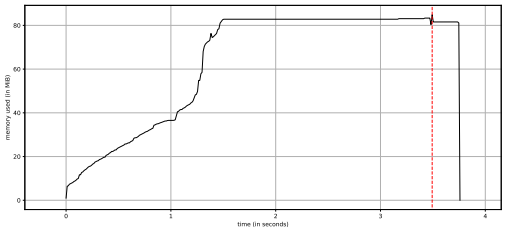
(c) Hartley (DP) memory usage during traverse 1.



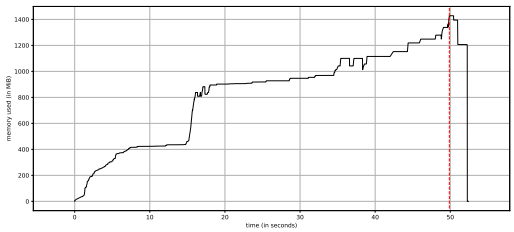
(d) Hartley (DP) memory usage during traverse 2.



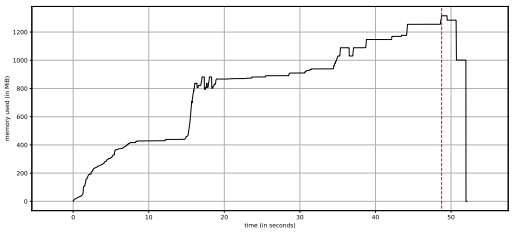
(e) StereoSGBM (SGM) memory usage during traverse 1.



(f) StereoSGBM (SGM) memory usage during traverse 2.



(g) PWOC-3D memory usage during traverse 1.



(h) PWOC-3D memory usage during traverse 2.

Figure 7.22: Memory usage of a single disparity map computation.

7.5. Discussion

In this section we address our third research question: What algorithm best applies to the requirements? In this chapter we analyzed the error rate, run time, and memory usage of four algorithms during two traverses. In the error rate section, we determined that the StereoBM performs best (lowest error rate) during the first traverse, and PWOC-3D during the second traverse. We observe a degrading quality in the predicted distances during the second traverse, due to the harsh lighting conditions, compared to the first traverse. The errors are traced back to their sources, which are either from the stereo matching performance or the variable parameters in the generation of the maps. In the run time section, we found that the StereoBM and StereoSGBM algorithms accommodate with requirement R2. In the memory usage section, we determined that StereoBM, Hartley, and StereoSGBM accommodate with requirement R4. Overall, the *StereoBM* algorithm best applies to the requirements in terms of error rate, run time, and memory usage.

8

Conclusion and Future Work

Stereo matching is part of the first module of navigation systems of planetary rovers, e.g., from NASA's Mars Exploration Rover missions [3, 4], NASA's Mars Science Laboratory (MSL) mission [5], ESA's ExoMars mission [6], and Chang'e-3 (CE-3) and Chang'e-4 (CE-4) missions [7, 8]. As (new) algorithms are being designed, their performance are typically evaluated on the Middlebury Stereo Vision benchmark [1], containing indoor scenes with objects, or the KITTI Vision Benchmark Suite [29], containing outdoor city scenes with roads and buildings. However, the results of their performance are not comparable for the Martian surface, as texture, colour, and lighting of indoor objects and outdoor city scenes differ from the Martian surface.

In this thesis, our goal is to improve insight into the performance of dense stereo matching algorithms, for the Martian surface on a single-board computer. To realise this goal, we searched for and compared stereo matching algorithms ranked on stereo vision datasets to obtain a manageable set of algorithms and verified their implementations. Next, we derived performance metrics from the ExoMars requirements and validated our set of verified algorithms on the Katwijk Beach Planetary Rover dataset [21]. A final stereo matching algorithm is selected that best applies to the requirements.

In this chapter we conclude the thesis (see Section 8.1) by presenting our contributions to address the research questions posed in Chapter 1. We discuss directions for future work in extending this work (see Section 8.2).

8.1. Conclusion

In this section we provide our answers to the three main research questions:

RQ1: How to obtain a manageable selection of stereo matching algorithms for comparison?
Obtaining a manageable selection of stereo matching algorithms is crucial for the comparison of algorithms. We have obtained a list of algorithms from the Middlebury and KITTI 2015 datasets, as there are algorithms ranked on the Katwijk dataset. We have developed selection criteria to reduce the list to a manageable size, i.e., to four algorithms (three traditional algorithms, one deep learning algorithm); SSD (local traditional), DP (global traditional), SGM (semi-global traditional), and Fast DS-CS (end-to-end deep learning). The set of algorithms were then verified by executing several implementations of the same algorithm and comparing them with values listed in literature. We made changes to the selected set, as the StereoBM (SAD) outperforms other SSD implementations and PWOC-3D generates a better disparity map for a sample Katwijk stereo image compared to the Fast-DS-CS algorithm. We have obtained a final set of algorithm implementations, i.e., StereoBM (SAD), Hartley (DP), StereoSGBM (SGM), and PWOC-3D (end-to-end deep learning).

RQ2: How to systematically compare the performance of stereo matching algorithms?
Comparing the selected algorithms using metrics and requirements gives us insight into the performance of each algorithm. We have presented the requirements of the ExoMars rover and developed a new practical error rate metric that is not based on the bad matched pixels percentage, as used in other datasets. We used the Katwijk dataset for our two experiments, which differ in sunlight position,

as the dataset used by ESA for the ExoMars rover is not publicly available. Additionally, the run time and memory usage are metrics for each algorithm during the two experiments.

RQ3: What algorithm best applies to the requirements?
We analyzed the results of each algorithm during the experiments and found that the StereoBM (SAD) algorithm best applies to the requirements in terms of error rate, run time, and memory usage. In general, the four stereo matching algorithms have degrading results in the second traverse, where the sun is in front of the rover, compared to the first traverse, where the sun is behind the rover. The harsh lighting conditions affect the stereo matching performance of all algorithms. Looking at each erroneous predicted distance from both traverses, the errors originate from either the stereo matching performance, contributing to small errors, or the variable parameters in the generation of the U-disparity map, edge detection map, and (filtered) Hough line transform map, contributing to large errors.

8.2. Future Work

This thesis presents promising results of the performance of stereo matching for the Martian surface on a single-board computer. Building on these results, we suggest three research directions that can be investigated.

Determine optimal parameters
The transform from a predicted disparity map to a predicted distance contains a number of variable parameters. The variable parameters are in the edge detection map, and (filtered) Hough line transform map process. However, several erroneous predicted distances are caused by not detecting the lines corresponding to the rocks in the U-disparity map. Tweaking the parameters would allow for the detection of the rocks and correct predicted distances.

Fine-tune the weights of the deep learning algorithm
We have used the PWOC-3D [117] deep learning stereo matching algorithm without fine-tuning the weights. The weights used are obtained from pre-training the deep learning network on the FlyingThings3D dataset [25]. However, the stereo matching performance would increase by fine-tuning the weights on a Martian-like dataset containing the ground-truth disparity maps.

Use TensorFlow Lite for the deep learning algorithm
We have used the TensorFlow library for the implementation of the PWOC-3D algorithm in the Raspberry Pi 4 Model B [14]. However, the TensorFlow [82] library is not optimized for the Raspberry Pi and results in more memory usage and slower execution times. The TensorFlow Lite [143] library is optimized for devices such as the Raspberry Pi and reduces the memory usage and results in faster execution times of the deep learning algorithm. In this thesis, the combined effect of the previous research direction and this direction could be a deep learning algorithm that complies with the requirements.

Bibliography

[1] D. Scharstein and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms”, *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[2] E. Bebeselea-Sterp, R. Brad, and R. Brad, “A comparative study of stereovision algorithms”, *International Journal of Advanced Computer Science and Applications*, vol. 8, Jan. 2017. DOI: 10.14569/IJACSA.2017.081144.

[3] S. Singh, R. Simmons, T. Smith, A. Stentz, V. Verma, A. Yahja, and K. Schwehr, “Recent progress in local and global traversability for planetary rovers”, in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, 1194–1200 vol.2.

[4] S. B. Goldberg, M. W. Maimone, and L. Matthies, “Stereo vision and rover navigation software for planetary exploration”, in *Proceedings, IEEE Aerospace Conference*, vol. 5, 2002, pp. 5–5.

[5] J. F. Bell III, A. Godber, S. McNair, M. A. Caplinger, J. N. Maki, M. T. Lemmon, J. Van Beek, M. C. Malin, D. Wellington, K. M. Kinch, M. B. Madsen, C. Hardgrove, M. A. Ravine, E. Jensen, D. Harker, R. B. Anderson, K. E. Herkenhoff, R. V. Morris, E. Cisneros, and R. G. Deen, “The mars science laboratory curiosity rover mastcam instruments: Preflight and in-flight calibration, validation, and data archiving”, *Earth and Space Science*, vol. 4, no. 7, pp. 396–452, 2017. DOI: 10.1002/2016EA000219. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1002/2016EA000219. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2016EA000219.

[6] K. McManamon, R. Lancaster, and N. Silva, “Exomars rover vehicle perception system architecture and test results”, in *Proceedings of the 12th Symposium on Advanced Space Technologies in Robotics and Automation, Noordwijk, The Netherlands*, 2013, pp. 15–17.

[7] M. Li, Z. Sun, S. Liu, Y. Ma, H. Ma, C. Sun, and Y. Jia, “Stereo vision technologies for china’s lunar rover exploration mission”, *International Journal of Robotics and Automation*, vol. 31, pp. 128–136, Mar. 2016. DOI: 10.2316/Journal.206.2016.2.206-4440.

[8] Y. Wang, W. Wan, S. Gou, M. Peng, Z. Liu, K. Di, L. Li, T. Yu, J. Wang, and X. Cheng, “Vision-based decision support for rover path planning in the chang’e-4 mission”, *Remote Sensing*, vol. 12, p. 624, Feb. 2020. DOI: 10.3390/rs12040624.

[9] L. Zhao and C. E. Thorpe, “Stereo-and neural network-based pedestrian detection”, *IEEE Transactions on intelligent transportation systems*, vol. 1, no. 3, pp. 148–154, 2000.

[10] D. E. Shean, O. Alexandrov, Z. M. Moratto, B. E. Smith, I. R. Joughin, C. Porter, and P. Morin, “An automated, open-source pipeline for mass production of digital elevation models (dems) from very-high-resolution commercial stereo satellite imagery”, *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 116, pp. 101–117, 2016.

[11] M. Menze and A. Geiger, “Object scene flow for autonomous vehicles”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3061–3070.

[12] Lunar Zebro. (2020). Lunar Zebro | Nano Rover TU Delft, [Online]. Available: http://zebro.space/ (visited on 07/06/2020).

[13] M. Otten, “Decizebro: The design of a modular bio-inspired robotic swarming platform”, PhD thesis, Delft University of Technology, 2017.

[14] (). Raspberry pi 4 tech specs, [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/ (visited on 07/25/2021).

[15] F. Wild, *What is mars?*, https://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-is-mars-58.html.

[16] W. Fink, V. R. Baker, D. Schulze-Makuch, C. W. Hamilton, and M. A. Tarbell, “Autonomous exploration of planetary lava tubes using a multi-rover framework”, in *2015 IEEE Aerospace Conference*, IEEE, 2015, pp. 1–9.

[17] V. Stamenković, L. Beegle, K. Zacny, D. Arumugam, P. Baglioni, N. Barba, J. Baross, M. Bell, R. Bhartia, J. Blank, *et al.*, “The next frontier for planetary and human exploration”, *Nature Astronomy*, vol. 3, no. 2, pp. 116–120, 2019.

[18] T. Vaquero, M. Troesch, M. S. Net, J. Gao, and S. Chien, “Energy-aware data routing for disruption tolerant networks in planetary cave exploration”, 2019.

[19] F. Tavares. (2019). Using a ’cave rover,’ nasa learns to search for life underground, [Online]. Available: `https://www.nasa.gov/feature/ames/braille` (visited on 07/12/2020).

[20] (). A new journey into earth for space exploration, [Online]. Available: `https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/CAVES_and_Pangaea/A_new_journey_into_Earth_for_space_exploration` (visited on 07/12/2020).

[21] R. A. Hewitt, E. Boukas, M. Azkarate, M. Pagnamenta, J. A. Marshall, A. Gasteratos, and G. Visentin, “The katwijk beach planetary rover dataset”, *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 3–12, 2018.

[22] K. Zhou, X. Meng, and B. Cheng, “Review of stereo matching algorithms based on deep learning”, *Computational Intelligence and Neuroscience*, vol. 2020, 2020.

[23] Y. LeCun, K. Kavukcuoglu, and C. Farabet, “Convolutional networks and applications in vision”, in *Proceedings of 2010 IEEE international symposium on circuits and systems*, IEEE, 2010, pp. 253–256.

[24] Q. Jia, X. Wan, B. Hei, and S. Li, “Dispnet based stereo matching for planetary scene depth estimation using remote sensing images”, in *2018 10th IAPR Workshop on Pattern Recognition in Remote Sensing (PRRS)*, 2018, pp. 1–5.

[25] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, “A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4040–4048.

[26] H. Hirschmuller, “Accurate and efficient stereo processing by semi-global matching and mutual information”, in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 2, 2005, 807–814 vol. 2.

[27] J. Vanne, E. Aho, T. D. Hamalainen, and K. Kuusilinna, “A high-performance sum of absolute difference implementation for motion estimation”, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 7, pp. 876–883, 2006.

[28] T. Kanade, H. Kano, S. Kimura, A. Yoshida, and K. Oda, “Development of a video-rate stereo machine”, in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, IEEE, vol. 3, 1995, pp. 95–100.

[29] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite”, in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2012, pp. 3354–3361.

[30] D. Scharstein, R. Szeliski, and H. Hirschmüller, *Middlebury stereo datasets*, `https://vision.middlebury.edu/stereo/`.

[31] L. Wang, Z. Liu, and Z. Zhang, “Feature based stereo matching using two-step expansion”, *Mathematical Problems in Engineering*, vol. 2014, Dec. 2014. DOI: `10.1155/2014/452803`.

[32] H. Li, L. Chen, and F. Li, “An efficient dense stereo matching method for planetary rover”, *IEEE Access*, vol. 7, pp. 48 551–48 564, 2019.

[33] S. Hadjis, C. Zhang, I. Mitliagkas, D. Iter, and C. Ré, “Omnivore: An optimizer for multi-device deep learning on cpus and gpus”, *arXiv preprint arXiv:1606.04487*, 2016.

[34] D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán, and Á. Rodríguez-Vázquez, “Performance analysis of real-time DNN inference on Raspberry Pi”, in *Real-Time Image and Video Processing 2018*, N. Kehtarnavaz and M. F. Carlsohn, Eds., International Society for Optics and Photonics, vol. 10670, SPIE, 2018, pp. 115–123. DOI: 10.1117/12.2309763. [Online]. Available: https://doi.org/10.1117/12.2309763.

[35] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st. Berlin, Heidelberg: Springer-Verlag, 2010, ISBN: 1848829345.

[36] R. C. Bolles, H. H. Baker, and D. H. Marimont, “Epipolar-plane image analysis: An approach to determining structure from motion”, *International journal of computer vision*, vol. 1, no. 1, pp. 7–55, 1987.

[37] Y. Kang and Y. Ho, “An efficient image rectification method for parallel multi-camera arrangement”, *IEEE Transactions on Consumer Electronics*, vol. 57, no. 3, pp. 1041–1048, 2011.

[38] A. Fusiello, E. Trucco, and A. Verri, “A compact algorithm for rectification of stereo pairs”, vol. 12, Oct. 2000. DOI: 10.1007/s001380050120.

[39] Y. Lecun and Y. Bengio, “Convolutional networks for images, speech, and time-series”, in *The handbook of brain theory and neural networks*, MIT Press, 1995.

[40] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition”, *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[42] D. Svozil, V. Kvasnicka, and J. Pospíchal, “Introduction to multi-layer feed-forward neural networks”, *Chemometrics and Intelligent Laboratory Systems*, vol. 39, pp. 43–62, Nov. 1997. DOI: 10.1016/S0169-7439(97)00061-0.

[43] B. MacLennan, “Chapter 3 - field computation: A framework for quantum-inspired computing”, in *Quantum Inspired Computational Intelligence*, S. Bhattacharyya, U. Maulik, and P. Dutta, Eds., Boston: Morgan Kaufmann, 2017, pp. 85–110, ISBN: 978-0-12-804409-4. DOI: https://doi.org/10.1016/B978-0-12-804409-4.00003-6. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128044094000036.

[44] S. Sharma, “Activation functions in neural networks”, *Towards Data Science*, vol. 6, 2017.

[45] M. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: The rprop algorithm”, in *IEEE International Conference on Neural Networks*, 1993, 586–591 vol.1.

[46] M. Mazur, *A Step by Step Backpropagation Example*. [Online]. Available: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ (visited on 09/01/2020).

[47] P. Radhakrishnan, “What are hyperparameters ? and how to tune the hyperparameters in a deep neural network?”, *Towards Data Science*, 2017. [Online]. Available: https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a (visited on 09/02/2020).

[48] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, 2016. arXiv: 1603.07285 [stat.ML].

[49] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, “Deconvolutional networks”, in *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, IEEE, 2010, pp. 2528–2535.

[50] R. A. Hamzah and H. Ibrahim, “Literature survey on stereo vision disparity map algorithms”, *Journal of Sensors*, vol. 2016,

[51] R. Zabih and J. Woodfill, “Non-parametric local transforms for computing visual correspondence”, in *Computer Vision — ECCV ’94*, J.-O. Eklundh, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 151–158, ISBN: 978-3-540-48400-4.

[52] O. Veksler, “Stereo correspondence by dynamic programming on a tree”, in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 2, 2005, 384–390 vol. 2.

[53] Y. Boykov, O. Veksler, and R. Zabih, “Fast approximate energy minimization via graph cuts”, *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 11, pp. 1222–1239, 2001.

[54] J. Sun, N.-N. Zheng, and H.-Y. Shum, “Stereo matching using belief propagation”, *IEEE Transactions on pattern analysis and machine intelligence*, vol. 25, no. 7, pp. 787–800, 2003.

[55] P. Viola and W. M. Wells III, “Alignment by maximization of mutual information”, *International journal of computer vision*, vol. 24, no. 2, pp. 137–154, 1997.

[56] J. Žbontar and Y. LeCun, “Stereo matching by training a convolutional neural network to compare image patches”, *The journal of machine learning research*, vol. 17, no. 1, pp. 2287–2318, 2016.

[57] W. Luo, A. G. Schwing, and R. Urtasun, “Efficient deep learning for stereo matching”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5695–5703.

[58] A. Seki and M. Pollefeys, “Sgm-nets: Semi-global matching with neural networks”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 231–240.

[59] S. Gidaris and N. Komodakis, *Detect, replace, refine: Deep structured prediction for pixel wise labeling*, 2016. arXiv: 1612.04770 [cs.CV].

[60] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry, *End-to-end learning of geometry and context for deep stereo regression*, 2017. arXiv: 1703.04309 [cs.CV].

[61] X. Guo, K. Yang, W. Yang, X. Wang, and H. Li, *Group-wise correlation stereo network*, 2019. arXiv: 1903.04025 [cs.CV].

[62] C. Godard, O. Mac Aodha, and G. J. Brostow, “Unsupervised monocular depth estimation with left-right consistency”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 270–279.

[63] H. Hirschmüller, “Semi-global matching-motivation, developments and applications”, *Photogrammetric Week 11*, pp. 173–184, 2011.

[64] L. Kudryavtsev, *Monotone function*. Encyclopedia of Mathematics. [Online]. Available: http://encyclopediaofmath.org/index.php?title=Monotone_function&oldid=18679 (visited on 08/09/2020).

[65] Z. Chen, X. Sun, L. Wang, Y. Yu, and C. Huang, “A deep visual correspondence embedding model for stereo matching costs”, vol. 2015, Dec. 2015, pp. 972–980. DOI: 10.1109/ICCV.2015.117.

[66] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, *Flownet: Learning optical flow with convolutional networks*, 2015. arXiv: 1504.06852 [cs.CV].

[67] J. Pang, W. Sun, J. S. Ren, C. Yang, and Q. Yan, *Cascade residual learning: A two-stage convolutional neural network for stereo matching*, 2017. arXiv: 1708.09204 [cs.CV].

[68] Z. Liang, Y. Feng, Y. Guo, H. Liu, W. Chen, L. Qiao, L. Zhou, and J. Zhang, *Learning for disparity estimation through feature constancy*, 2017. arXiv: 1712.01039 [cs.CV].

[69] X. Song, X. Zhao, H. Hu, and L. Fang, *Edgestereo: A context integrated residual pyramid network for stereo matching*, 2018. arXiv: 1803.05196 [cs.CV].

[70] G. Yang, H. Zhao, J. Shi, Z. Deng, and J. Jia, *Segstereo: Exploiting semantic information for disparity estimation*, 2018. arXiv: 1807.11699 [cs.CV].

[71] L. Yu, Y. Wang, Y. Wu, and Y. Jia, *Deep stereo matching with explicit cost aggregation sub-architecture*, 2018. arXiv: 1801.04065 [cs.CV].

[72] J. Liu, H. Li, R. Wu, Q. Zhao, Y. Guo, and L. Chen, “A survey on deep learning methods for scene flow estimation”, *Pattern Recognition*, vol. 106, p. 107 378, Oct. 2020. DOI: 10.1016/j.patcog.2020.107378.

[73] J.-R. Chang and Y.-S. Chen, *Pyramid stereo matching network*, 2018. arXiv: 1803.08669 [cs.CV].

[74] C. Lu, H. Uchiyama, D. Thomas, A. Shimada, and R.-i. Taniguchi, “Sparse cost volume for efficient stereo matching”, *Remote Sensing*, vol. 10, p. 1844, Nov. 2018. DOI: 10.3390/rs10111844.

[75] S. Tulyakov, A. Ivanov, and F. Fleuret, *Practical deep stereo (pds): Toward applications-friendly deep stereo matching*, 2018. arXiv: 1806.01677 [cs.CV].

[76] R. Garg, V. K. BG, G. Carneiro, and I. Reid, *Unsupervised cnn for single view depth estimation: Geometry to the rescue*, 2016. arXiv: 1603.04992 [cs.CV].

[77] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[78] Y. Zhong, Y. Dai, and H. Li, *Self-supervised learning for stereo matching with self-improving ability*, 2017. arXiv: 1709.00930 [cs.CV].

[79] L. Bora, B. Nye, R. Lancaster, C. Barclay, and M. Winter, “Exomars rover control, localisation and path planning in an hazardous and high disturbance environment”, in *14th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2017, pp. 20–22.

[80] M. Winter, C. Barcaly, V. Pereira, R. Lancaster, M. Caceres, N. Mc-Manamon, N. Silva, D. Lachat, and M. Campana, “Exomars rover vehicle: Detailed description of the gnc system”, *Astra*, 2015.

[81] Y. Gao, *Contemporary Planetary Robotics: An approach toward autonomous systems*. John Wiley & Sons, 2016.

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

[83] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, “Characterizing the deployment of deep neural networks on commercial edge devices”, in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, 2019, pp. 35–48.

[84] T. Taniai, Y. Matsushita, Y. Sato, and T. Naemura, “Continuous 3d label stereo matching using local expansion moves”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 11, pp. 2725–2739, Nov. 2018, ISSN: 1939-3539. DOI: 10.1109/tpami.2017.2766072. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2017.2766072.

[85] K. Batsos, C. Cai, and P. Mordohai, “Cbmv: A coalesced bidirectional matching volume for disparity estimation”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2060–2069.

[86] T. Yan, Y. Gan, Z. Xia, and Q. Zhao, “Segment-based disparity refinement with occlusion handling for stereo matching”, *IEEE Transactions on Image Processing*, vol. 28, no. 8, pp. 3885–3897, 2019.

[87] M. G. Mozerov and J. van de Weijer, “One-view occlusion detection for stereo matching with a fully connected crf model”, *IEEE Transactions on Image Processing*, vol. 28, no. 6, pp. 2936–2947, 2019.

[88] P. Knobelreiter, C. Sormann, A. Shekhovtsov, F. Fraundorfer, and T. Pock, “Belief propagation reloaded: Learning bp-layers for labeling problems”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7900–7909.

[89] G. Yang, J. Manela, M. Happold, and D. Ramanan, “Hierarchical deep stereo matching on high-resolution images”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5515–5524.

[90] P. Knöbelreiter, C. Reinbacher, A. Shekhovtsov, and T. Pock, *End-to-end training of hybrid cnn-crf models for stereo*, 2016. arXiv: 1611.10229 [cs.CV].

[91] J. T. Barron and B. Poole, “The fast bilateral solver”, *ECCV*, 2016.

[92] S. Duggal, S. Wang, W.-C. Ma, R. Hu, and R. Urtasun, “Deeppruner: Learning efficient stereo matching via differentiable patchmatch”, in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 4384–4393.

[93] J. Yin, H. Zhu, D. Yuan, and T. Xue, “Sparse representation over discriminative dictionary for stereo matching”, *Pattern Recognition*, vol. 71, pp. 278–289, 2017, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.06.015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320317302364.

[94] A. Geiger, M. Roser, and R. Urtasun, “Efficient large-scale stereo matching”, in *Asian Conference on Computer Vision (ACCV)*, 2010.

[95] F. Zhang, V. Prisacariu, R. Yang, and P. H. Torr, “Ga-net: Guided aggregation net for end-to-end stereo matching”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[96] G. Yang and D. Ramanan, “Upgrading optical flow to 3d scene flow through optical expansion”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[97] Y. Zhang, Y. Chen, X. Bai, S. Yu, K. Yu, Z. Li, and K. Yang, “Adaptive unimodal cost volume filtering for deep stereo matching.”, in *AAAI*, 2020, pp. 12 926–12 934.

[98] X. Gu, Z. Fan, S. Zhu, Z. Dai, F. Tan, and P. Tan, “Cascade cost volume for high-resolution multi-view stereo and stereo matching”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2495–2504.

[99] Z. Yin, T. Darrell, and F. Yu, “Hierarchical discrete distribution decomposition for match density estimation”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6044–6053.

[100] H. Xu and J. Zhang, “Aanet: Adaptive aggregation network for efficient stereo matching”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1959–1968.

[101] Y. Wang, H. Wang, G. Yu, M. Yang, Y. Yuan, and J. Quan, “Stereo matching algorithm based on three-dimensional convolutional neural network”, *Acta Optica Sinica*, vol. 39, no. 11, p. 1 115 001, Nov. 2019. [Online]. Available: http://www.clp.ac.cn/EN/Article/OJe48a4e2a94dac30.

[102] T. Saikia, Y. Marrakchi, A. Zela, F. Hutter, and T. Brox, “Autodispnet: Improving disparity estimation with automl”, in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1812–1823.

[103] A. Badki, A. Troccoli, K. Kim, J. Kautz, P. Sen, and O. Gallo, “Bi3d: Stereo depth estimation via binary classifications”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1600–1608.

[104] H. Jiang, D. Sun, V. Jampani, Z. Lv, E. Learned-Miller, and J. Kautz, “Sense: A shared encoder network for scene-flow estimation”, in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 3195–3204.

[105] Z. Zhu, M. He, Y. Dai, Z. Rao, and B. Li, “Multi-scale cross-form pyramid network for stereo matching”, in *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, IEEE, 2019, pp. 1789–1794.

[106] Q. Wang, S. Shi, S. Zheng, K. Zhao, and X. Chu, “Fadnet: A fast and accurate network for disparity estimation”, *arXiv preprint arXiv:2003.10758*, 2020.

[107] Y. Wang, H. Wang, Y. Liu, M. Yang, and J. Quan, “Real-time stereo matching algorithm with hierarchical refinement”, *Acta Optica Sinica*, vol. 40, no. 11, p. 0 915 002, Sep. 2020. [Online]. Available: http://www.clp.ac.cn/EN/Article/OJ904f9e957b1721df.

[108] K. Yee and A. Chakrabarti, “Fast deep stereo with 2d convolutional processing of cost signatures”, in *The IEEE Winter Conference on Applications of Computer Vision*, 2020, pp. 183–191.

[109] K. Batsos and P. Mordohai, “Recresnet: A recurrent residual cnn architecture for disparity map enhancement”, in *2018 International Conference on 3D Vision (3DV)*, IEEE, 2018, pp. 238–247.

[110] N. Smolyanskiy, A. Kamenev, and S. Birchfield, “On the importance of stereo for accurate depth estimation: An efficient semi-supervised deep neural network approach”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1007–1015.

[111] A. Shaked and L. Wolf, “Improved stereo matching with constant highway networks and reflective confidence learning”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4641–4650.

[112] F. Guney and A. Geiger, “Displets: Resolving stereo ambiguities using object knowledge”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4165–4175.

[113] F. Aleotti, F. Tosi, L. Zhang, M. Poggi, and S. Mattoccia, *Reversing the cycle: Self-supervised deep stereo through enhanced monocular distillation*, 2020. arXiv: 2008.07130 [cs.CV].

[114] C. Vogel, K. Schindler, and S. Roth, “3d scene flow estimation with a piecewise rigid scene model”, *International Journal of Computer Vision*, vol. 115, no. 1, pp. 1–28, 2015.

[115] A. Tonioni, F. Tosi, M. Poggi, S. Mattoccia, and L. D. Stefano, “Real-time self-adaptive deep stereo”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 195–204.

[116] S. Tulyakov, A. Ivanov, and F. Fleuret, “Weakly supervised learning of deep metrics for stereo reconstruction”, in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1339–1348.

[117] R. Saxena, R. Schuster, O. Wasenmuller, and D. Stricker, “Pwoc-3d: Deep occlusion-aware end-to-end scene flow estimation”, in *2019 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2019, pp. 324–331.

[118] M. Menze and A. Geiger, “Object scene flow for autonomous vehicles”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3061–3070.

[119] K. Yamaguchi, D. McAllester, and R. Urtasun, “Efficient joint segmentation, occlusion labeling, stereo and flow estimation”, in *European Conference on Computer Vision*, Springer, 2014, pp. 756–771.

[120] C. Vogel, K. Schindler, and S. Roth, “Piecewise rigid scene flow”, in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 1377–1384.

[121] A. Kuzmin, D. Mikushin, and V. Lempitsky, “End-to-end learning of cost-volume aggregation for real-time dense stereo”, in *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*, IEEE, 2017, pp. 1–6.

[122] C. Zhang, Z. Li, Y. Cheng, R. Cai, H. Chao, and Y. Rui, “Meshstereo: A global stereo model with mesh alignment regularization for view interpolation”, in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2057–2065.

[123] F. Huguet and F. Devernay, “A variational method for scene flow estimation from stereo sequences”, in *2007 IEEE 11th International Conference on Computer Vision*, IEEE, 2007, pp. 1–7.

[124] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. Van Gool, “Ai benchmark: All about deep learning on smartphones in 2019”, in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, IEEE, 2019, pp. 3617–3635.

[125] S. K. Lam, A. Pitrou, and S. Seibert, “Numba: A llvm-based python jit compiler”, in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, ser. LLVM ’15, Austin, Texas: Association for Computing Machinery, 2015, ISBN: 9781450340052. DOI: 10.1145/2833157.2833162. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/2833157.2833162.

[126] T. Flatscher, *Stereo matching*, https://github.com/2b-t/SciPy-stereo-sgm, 2020.

[127] K. Konolige, *Cv::stereobm class reference*, OpenCV. [Online]. Available: https://docs.opencv.org/3.4/d9/dba/classcv_1_1StereoBM.html.

[128] I. J. Cox, S. L. Hingorani, S. B. Rao, and B. M. Maggs, “A maximum likelihood stereo algorithm”, *Computer Vision and Image Understanding*, vol. 63, no. 3, pp. 542–567, 1996, ISSN: 1077-3142. DOI: https://doi.org/10.1006/cviu.1996.0040. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314296900405.

[129] G. Siddhad, *Stereo vision dp*, https://github.com/Gourav1607/Stereo-Vision-DP, 2019.

[130] J. Hartley, *Stereo matching algorithm*, https://github.com/jadehartley/StereoMatchingAlgorit 2020.

[131] K. Konolige, *Cv::stereosgbm class reference*, OpenCV. [Online]. Available: https://docs.opencv.org/3.4/d2/d85/classcv_1_1StereoSGBM.html.

[132] S. Mehdi, *Division by zero error #53*, https://github.com/CVLAB-Unibo/Real-time-self-adaptive-deep-stereo/issues/53, 2020.

[133] K. McManamon, R. Lancaster, and N. Silva, “Exomars rover vehicle perception system architecture and test results”, in *Proceedings of the 12th Symposium on Advanced Space Technologies in Robotics and Automation, Noordwijk, The Netherlands*, 2013, pp. 15–17.

[134] P. Furgale, P. Carle, J. Enright, and T. D. Barfoot, “The devon island rover navigation dataset”, *The International Journal of Robotics Research*, vol. 31, no. 6, pp. 707–713, 2012. DOI: 10.1177/0278364911433135. eprint: https://doi.org/10.1177/0278364911433135. [Online]. Available: https://doi.org/10.1177/0278364911433135.

[135] O. Lamarre, O. Limoyo, F. Marić, and J. Kelly, “The canadian planetary emulation terrain energy-aware rover navigation dataset”, *The International Journal of Robotics Research*, vol. 39, no. 6, pp. 641–650, 2020. DOI: 10.1177/0278364920908922. eprint: https://doi.org/10.1177/0278364920908922. [Online]. Available: https://doi.org/10.1177/0278364920908922.

[136] S. Gillies *et al.*, *Rasterio: Geospatial raster i/o for Python programmers*, Mapbox, 2013–. [Online]. Available: https://github.com/mapbox/rasterio.

[137] J. Whitaker, *Proj*, pyproj4, 2006–. [Online]. Available: https://github.com/pyproj4/pyproj.

[138] PointGrey, *Camera catalog and sensor review*, www.ptgrey.com, 2016.

[139] Z. Hu and K. Uchimura, “U-v-disparity: An efficient algorithm for stereovision based scene analysis”, in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, 2005, pp. 48–54. DOI: 10.1109/IVS.2005.1505076.

[140] OpenCV, *Canny edge detection*, https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html.

[141] ——, *Hough line transform*, https://docs.opencv.org/4.5.2/d6/d10/tutorial_py_houghlines.html.

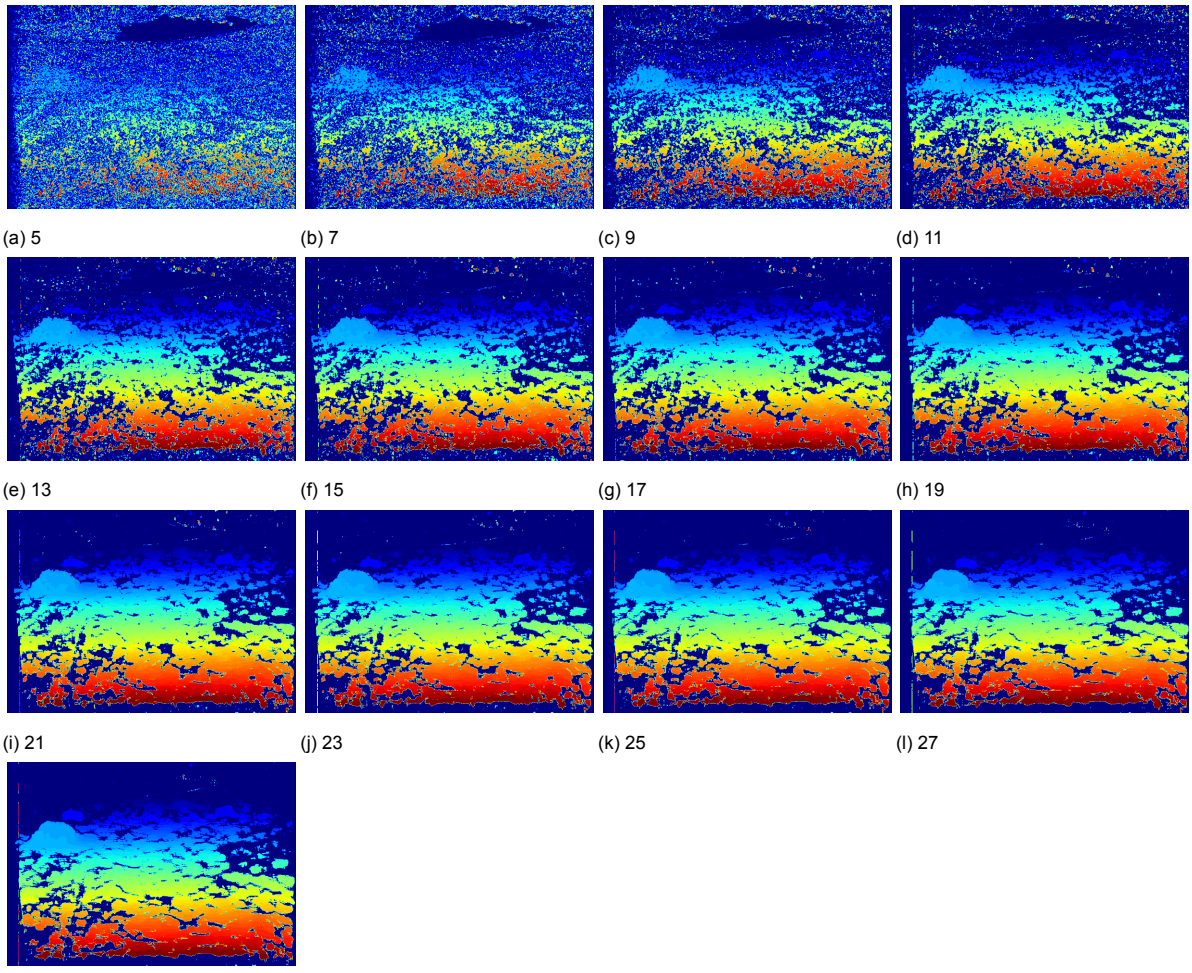
[142] F. Pedregosa, *Memory-profiler: A module for monitoring memory usage of a python program*, https://pypi.org/project/memory-profiler/.

[143] L. Shuangfeng, “Tensorflow lite: On-device machine learning framework”, *Journal of Computer Research and Development*, vol. 57, no. 9, 1839, p. 1839, 2020. DOI: 10.7544/issn1000-1239.2020.20200291. [Online]. Available: https://crad.ict.ac.cn/EN/abstract/article_4251.shtml.

A

Predicted disparity maps parameters

A.1. StereoBM (SAD)



(a) 5 (b) 7 (c) 9 (d) 11

(e) 13 (f) 15 (g) 17 (h) 19

(i) 21 (j) 23 (k) 25 (l) 27

(m) 29

Figure A.1: Predicted disparity map of the sample Katwijk stereo using the StereoBM algorithm with varying window sizes.

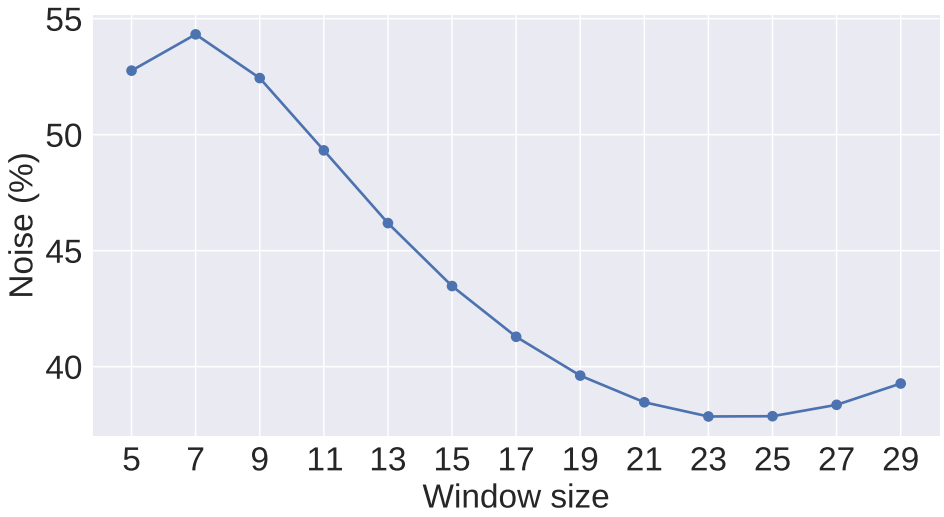
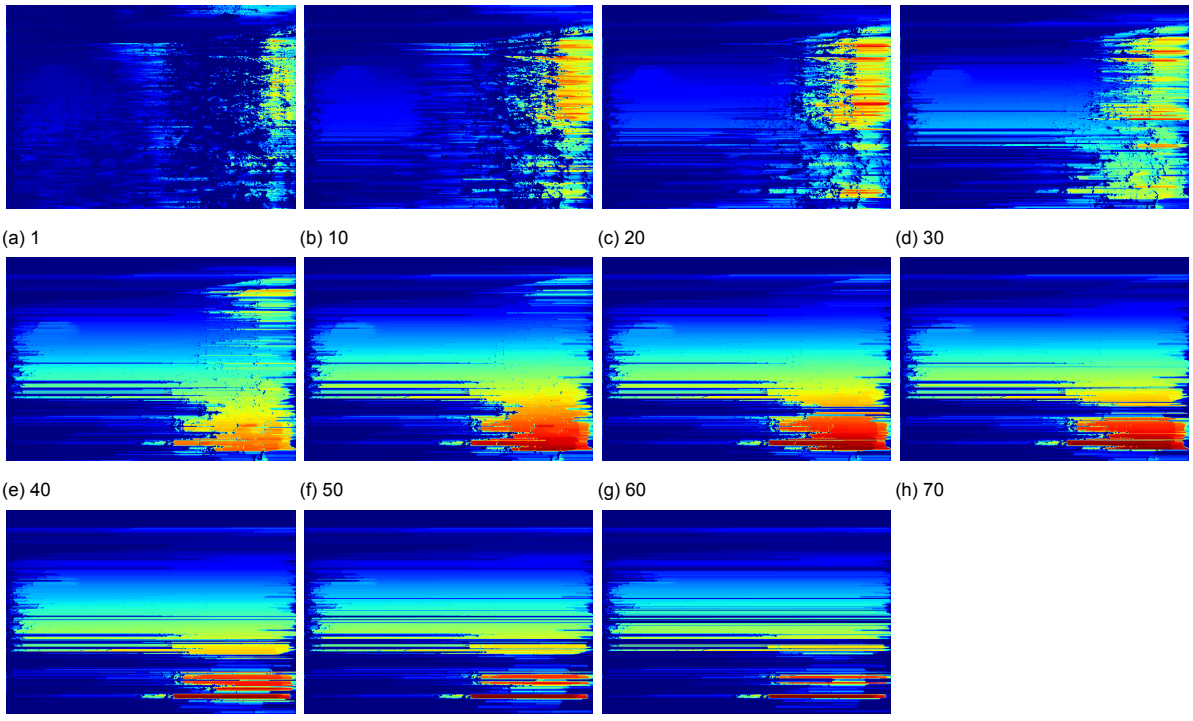


Figure A.2: StereoBM noise with varying window sizes.

A.2. Hartley (DP)



(a) 1 (b) 10 (c) 20 (d) 30

(e) 40 (f) 50 (g) 60 (h) 70

(i) 80 (j) 90 (k) 100

Figure A.3: Predicted disparity map of the sample Katwijk stereo using the Hartley algorithm with varying occlusion costs.

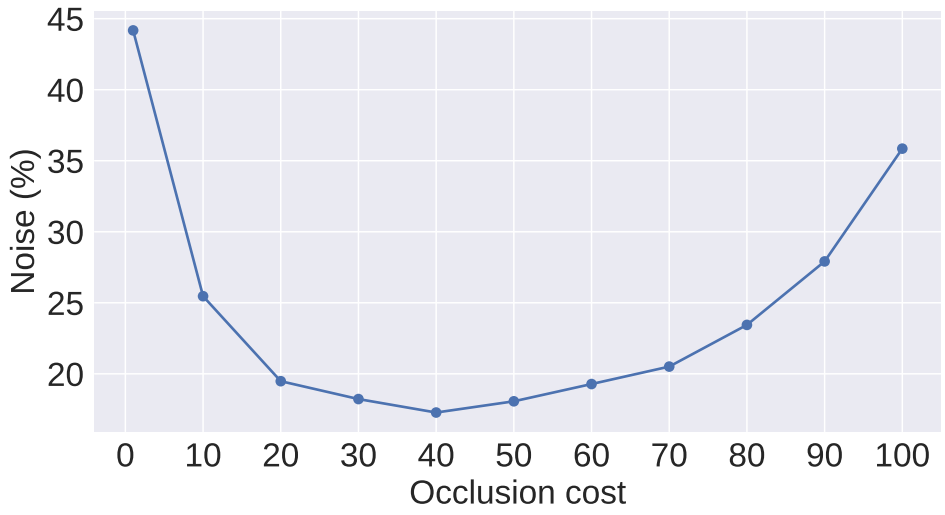


Figure A.4: Hartley noise with varying occlusion costs.

A.3. StereoSGBM (SGM)

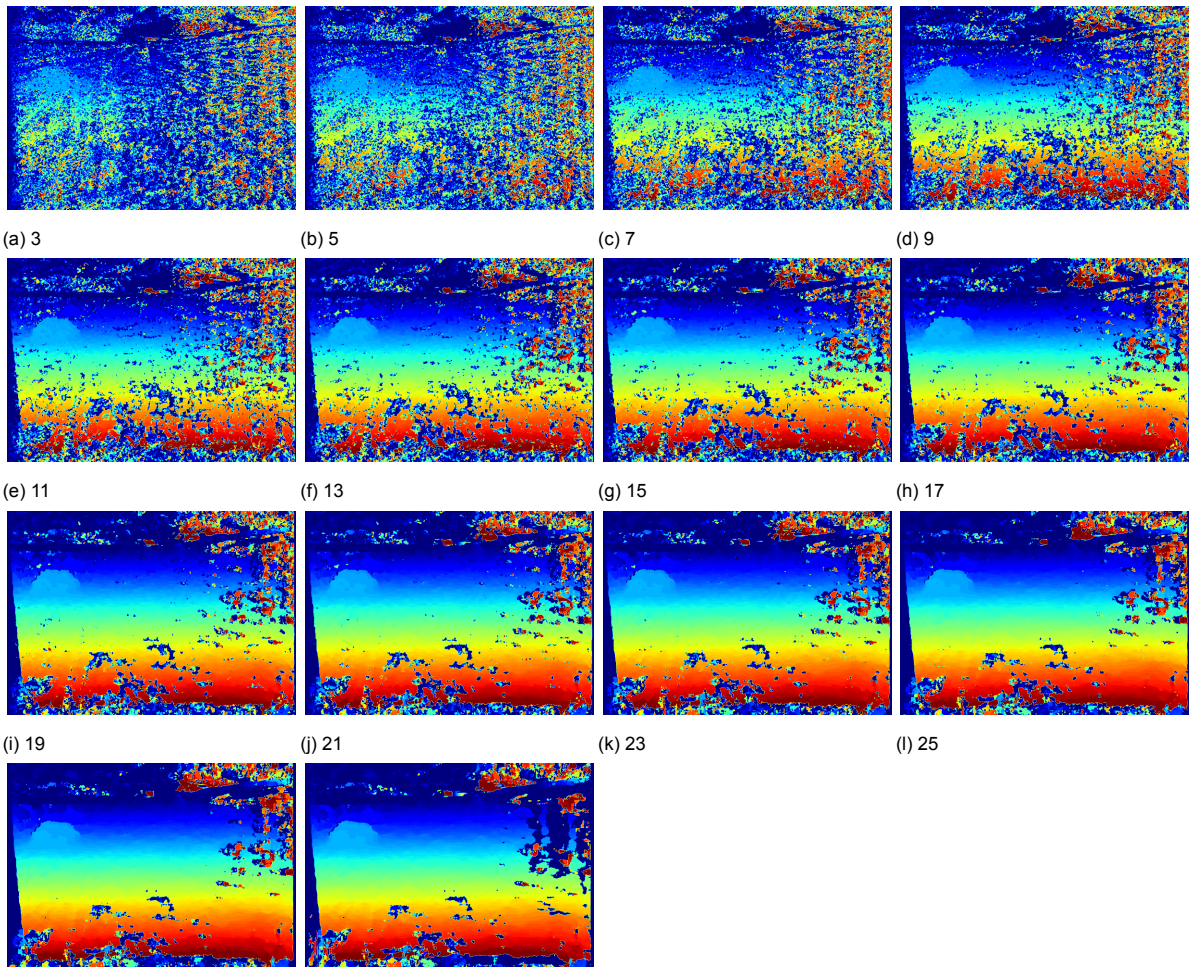


Figure A.5: Predicted disparity map of the sample Katwijk stereo using the StereoSGBM algorithm with varying window sizes.

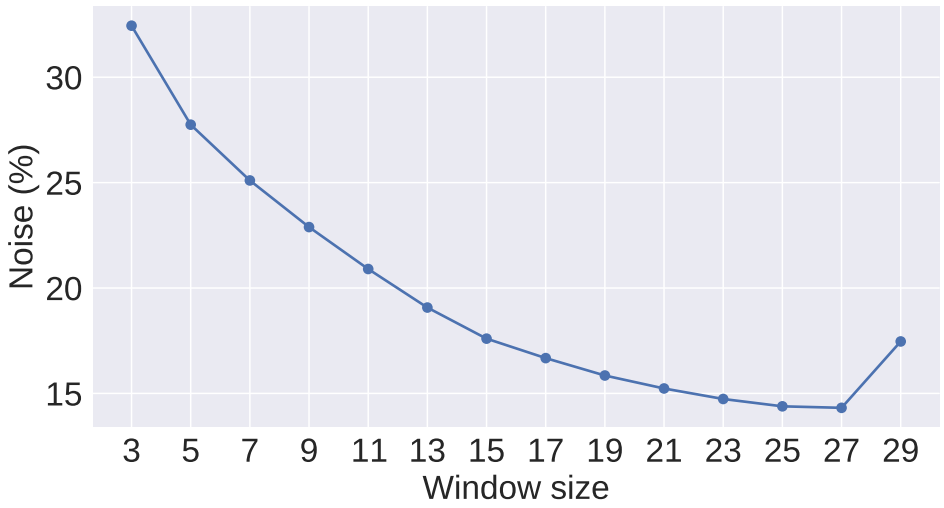


Figure A.6: StereoSGBM noise with varying window sizes.

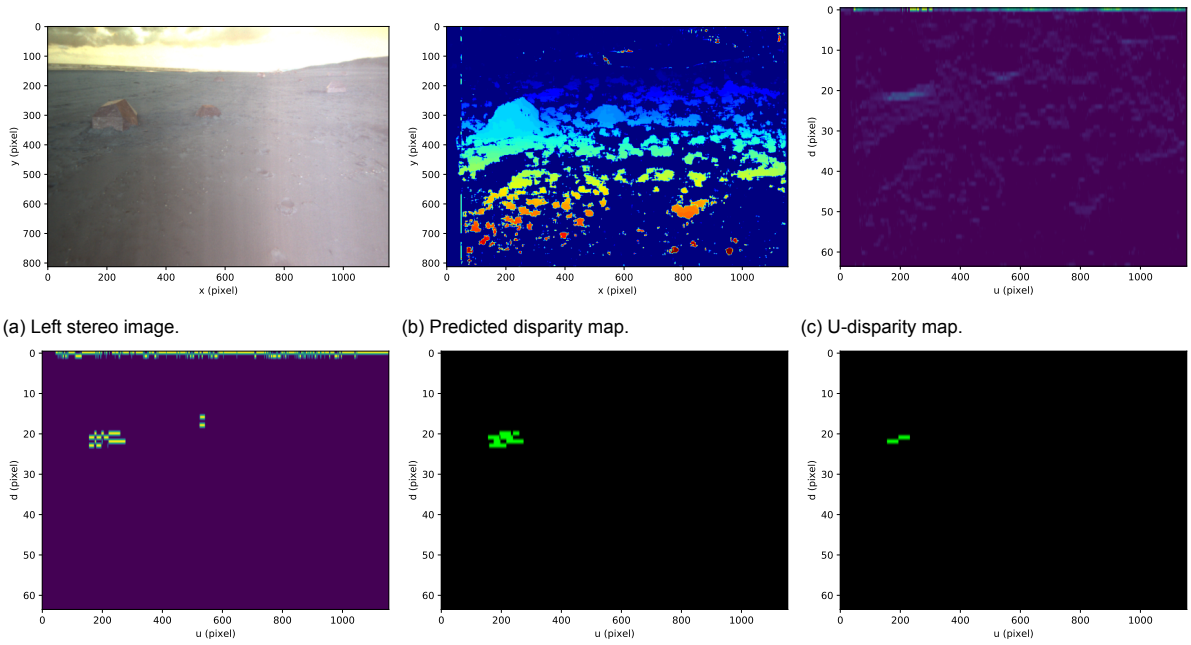
B

Predicted distance analysis

B.1. Experiment 1

B.1.1. StereoBM (SAD)

Small rocks



(a) Left stereo image.

(b) Predicted disparity map.

(c) U-disparity map.

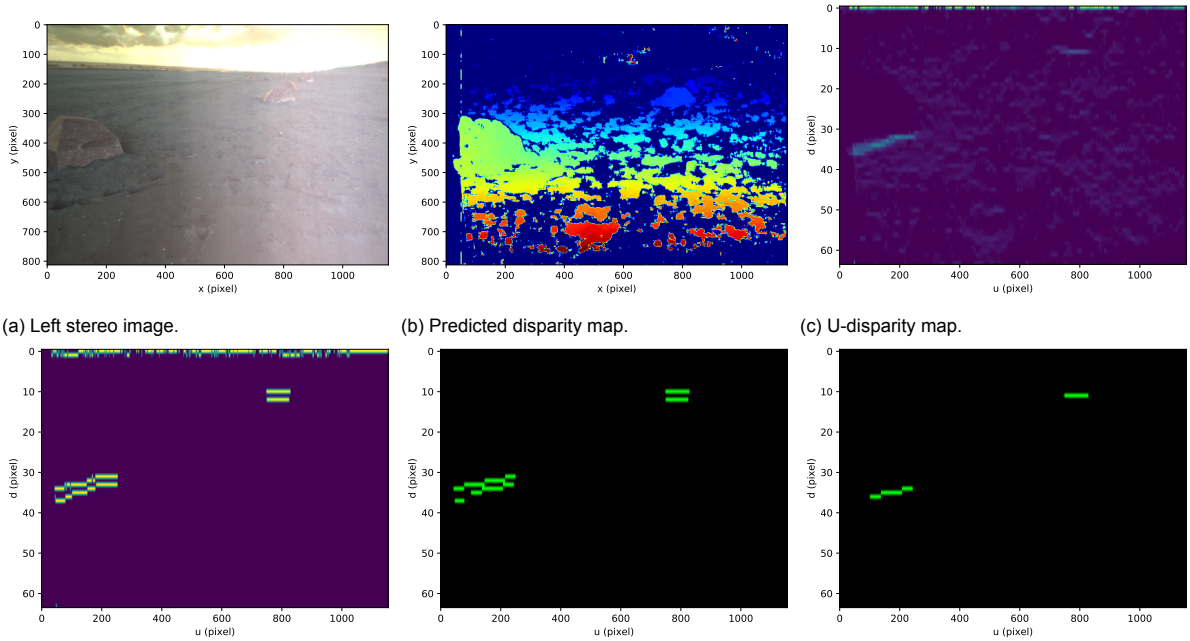
(d) Edge detection.

(e) Hough lines.

(f) Filtered Hough lines.

Figure B.1: Erroneous point analysis of the small rocks at time 12:56:19:924.

Medium rocks

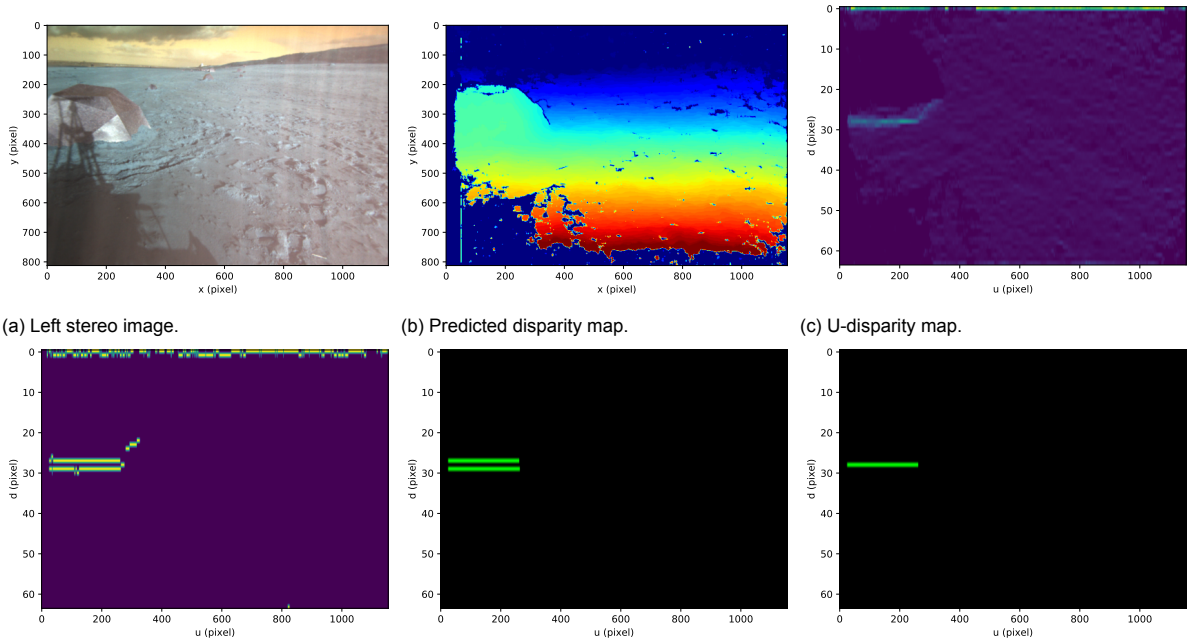


(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.2: Erroneous point analysis of the medium rocks at time 12:55:14:593.

Large rocks



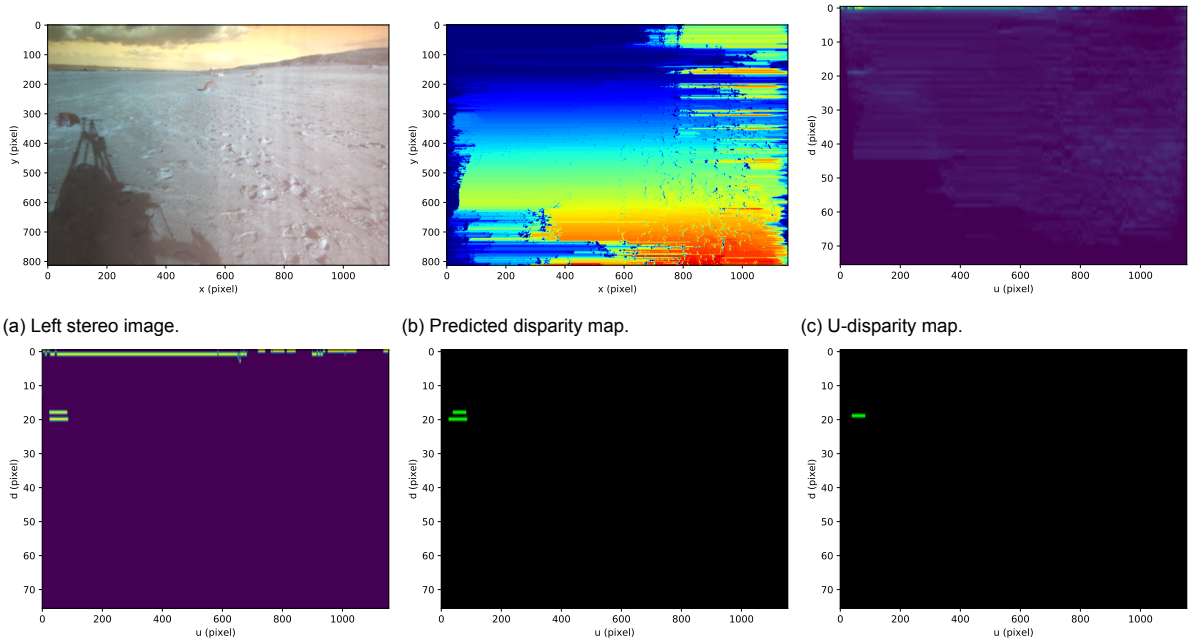
(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.3: Erroneous point analysis of the large rocks at time 12:57:50:379.

B.1.2. Hartley (DP)

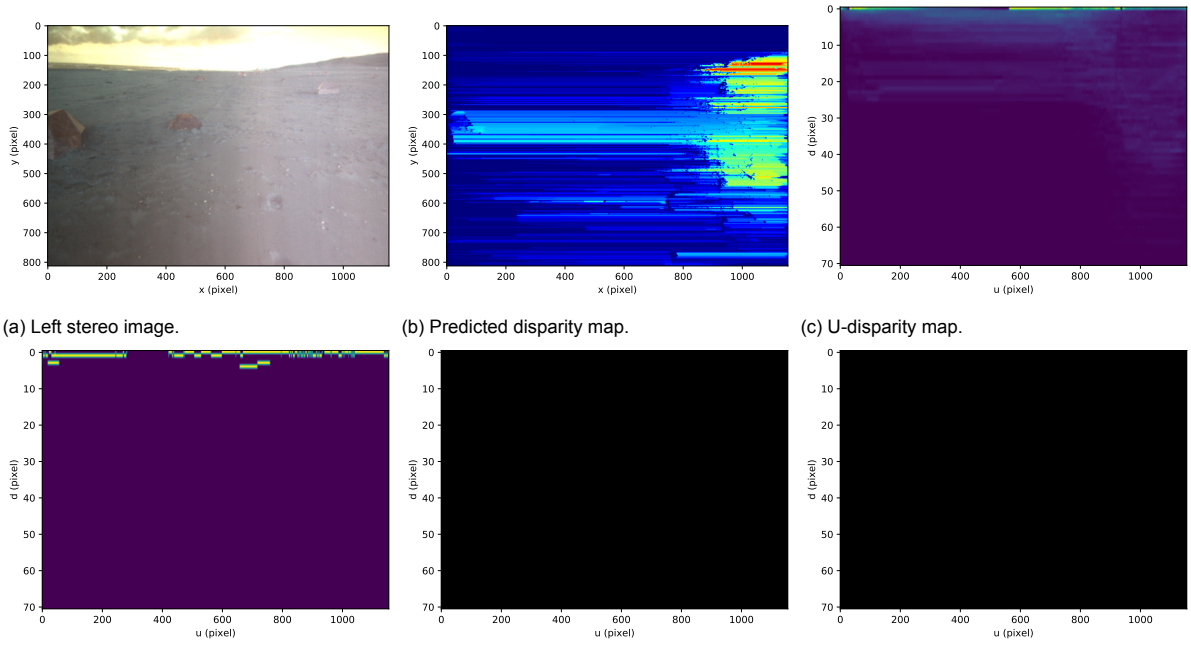
Small rocks



(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.4: Correct point analysis of the small rocks at time 12:58:06:543.

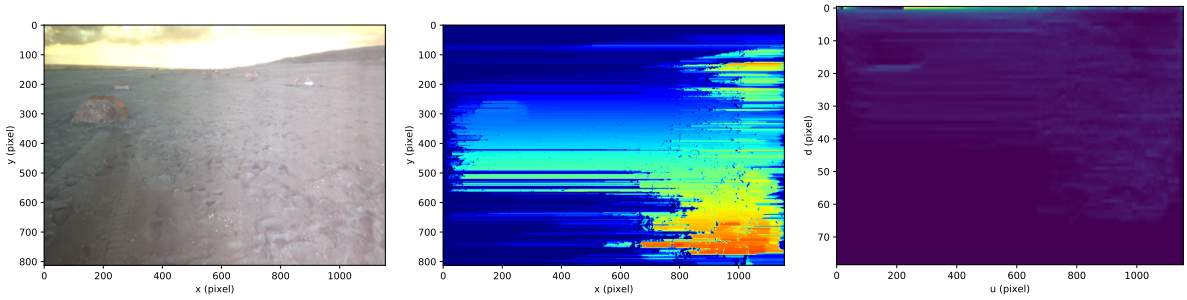


(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

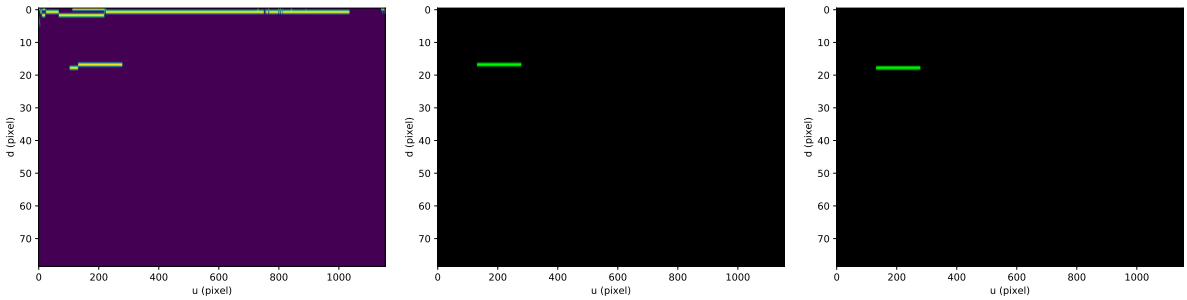
(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.5: Erroneous point analysis of the small rocks at time 12:56:22:711.

Medium rocks

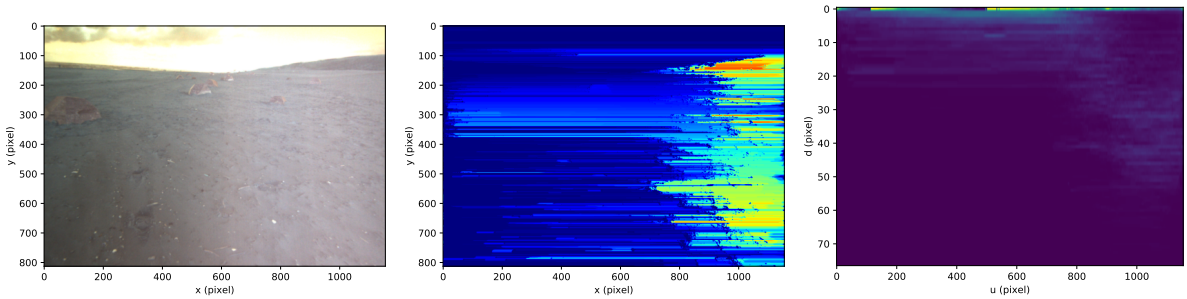


(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

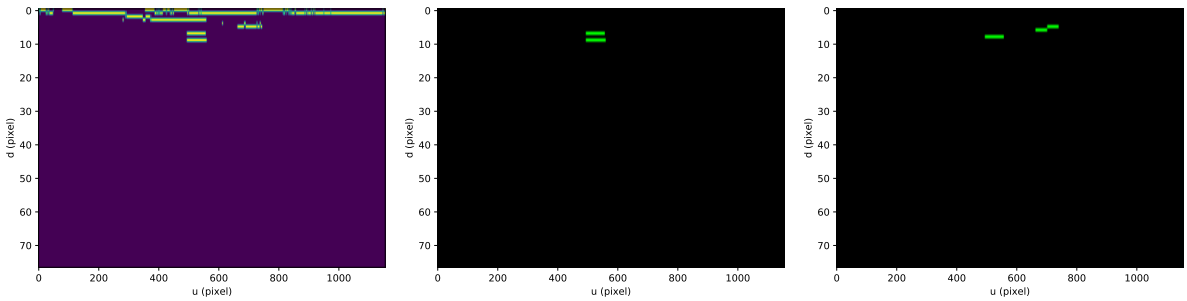


(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.6: Correct point analysis of the medium rocks at time 12:54:01:798.



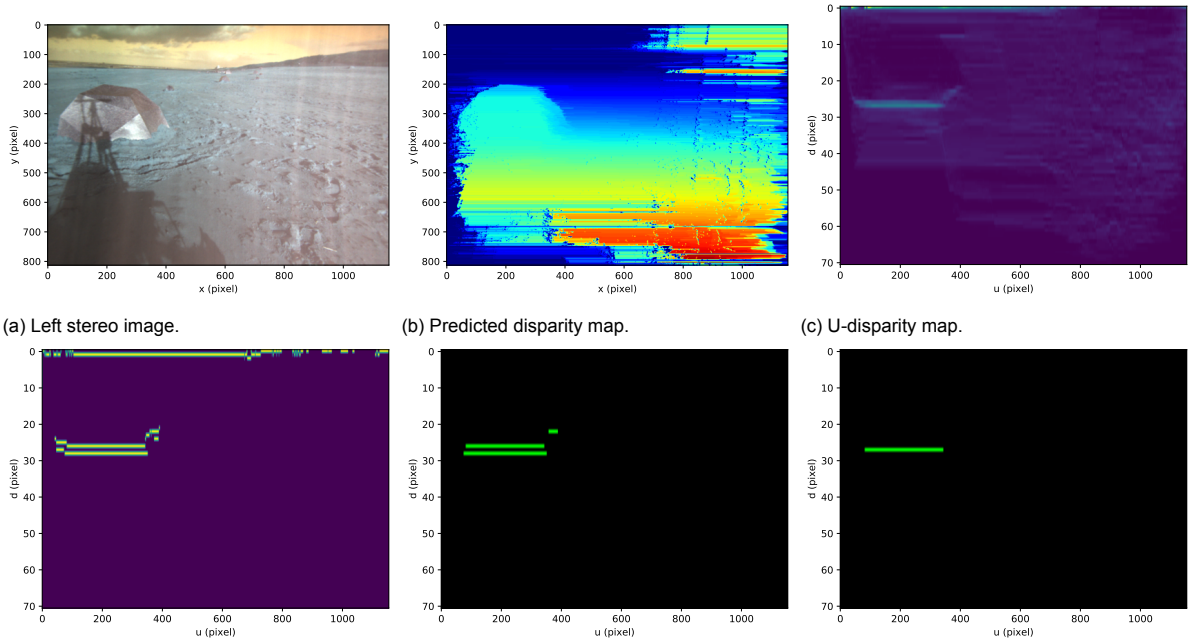
(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.



(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.7: Erroneous point analysis of the medium rocks at time 12:55:09:547.

Large rocks



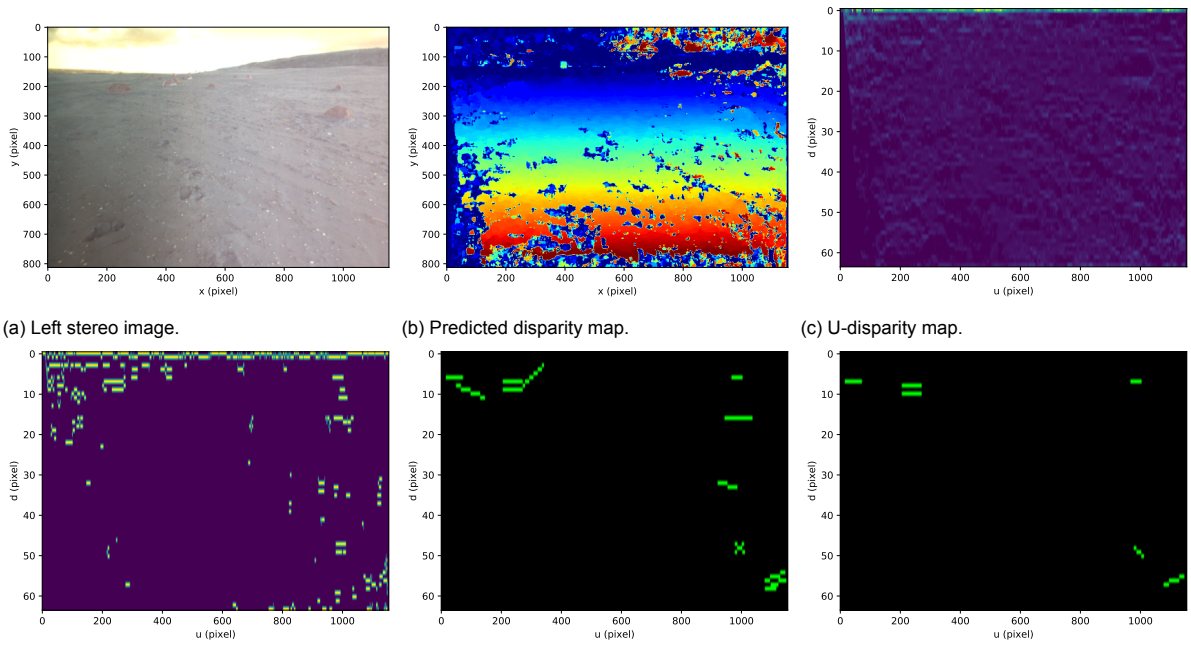
(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.8: Erroneous point analysis of the large rocks at time 12:57:50:123.

B.1.3. StereoSGBM (SGM)

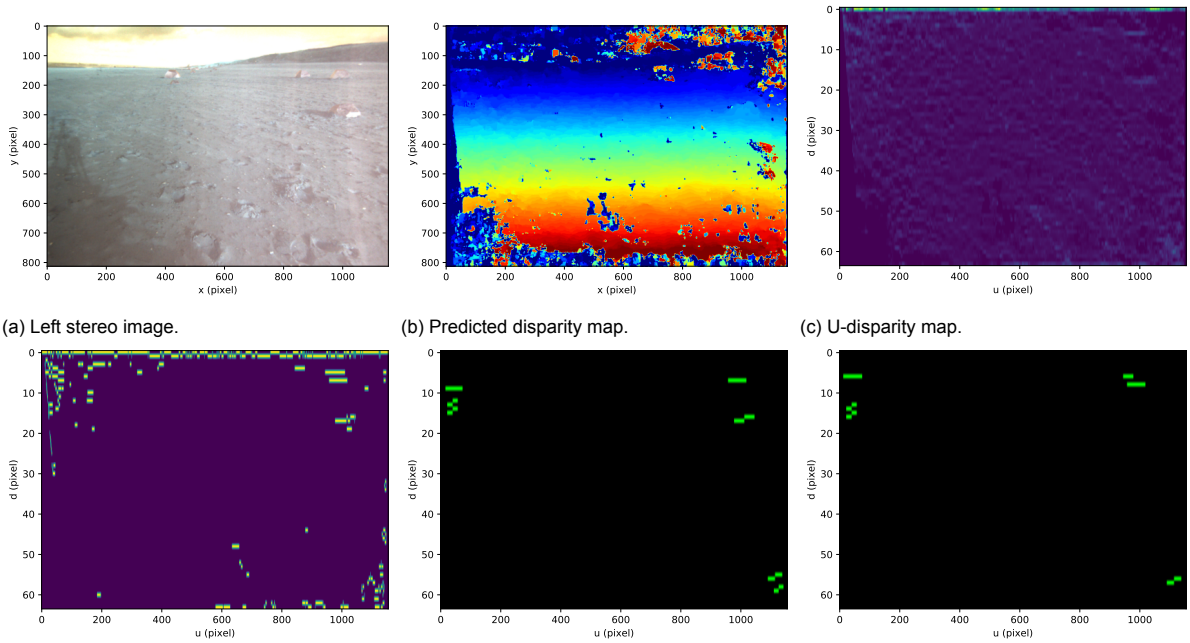
Small rocks



(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.9: Erroneous point analysis of the small rocks at time 12:54:54:894.

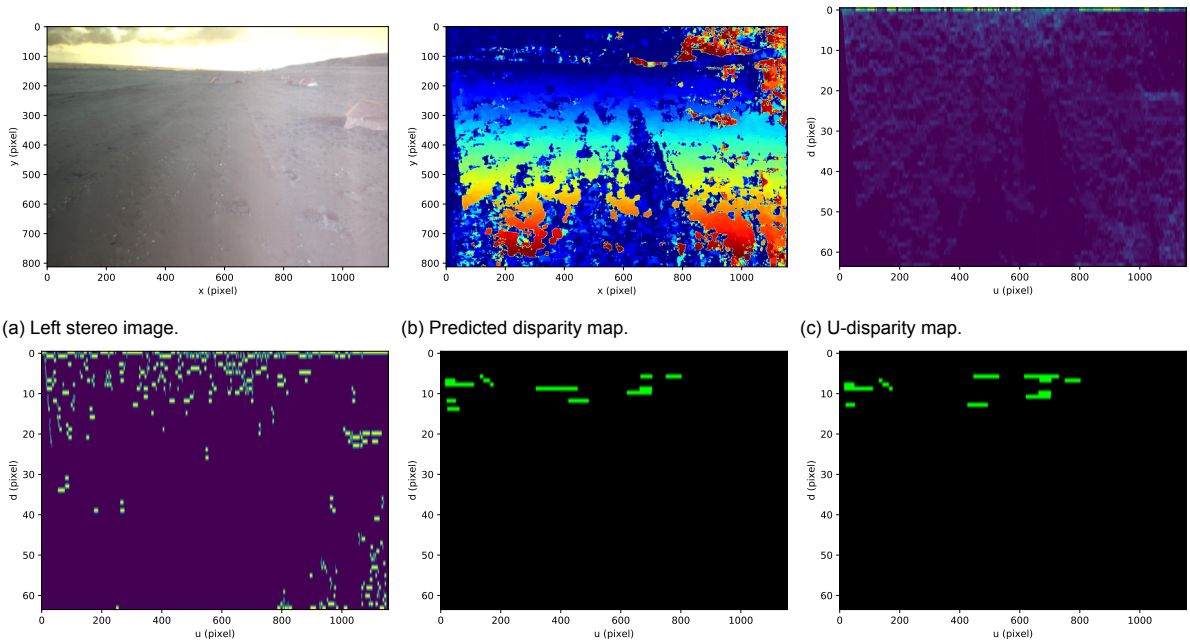


(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.10: Erroneous point analysis of the small rocks at time 12:56:59:469.

Medium rocks

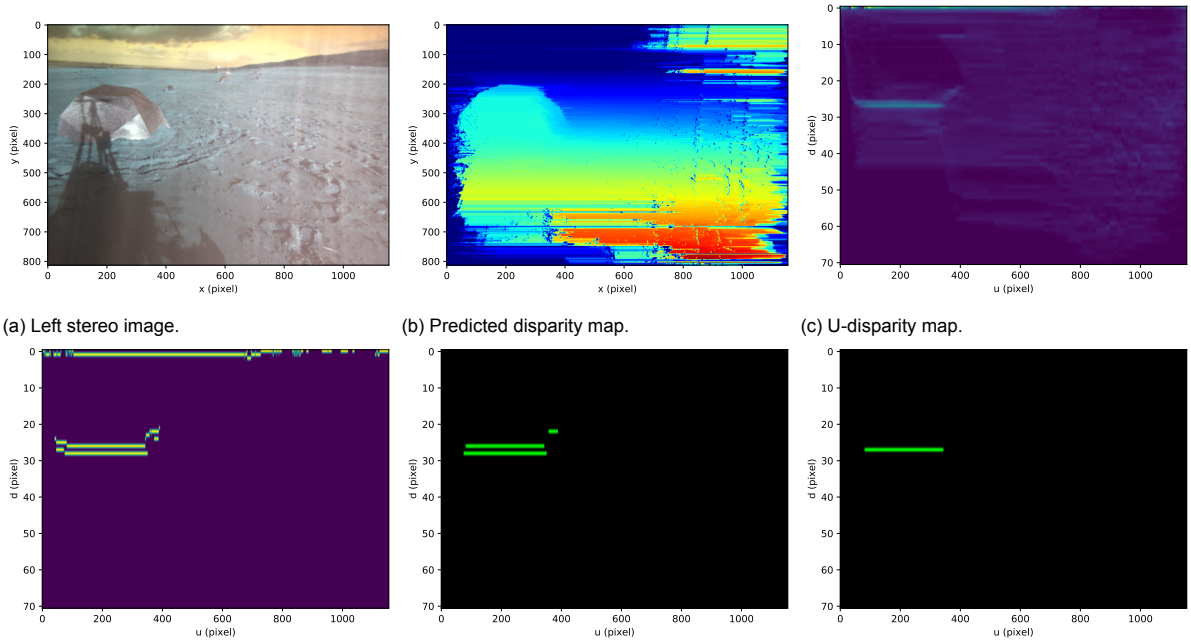


(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.11: Erroneous point analysis of the medium rocks at time 12:55:40:104.

Large rocks



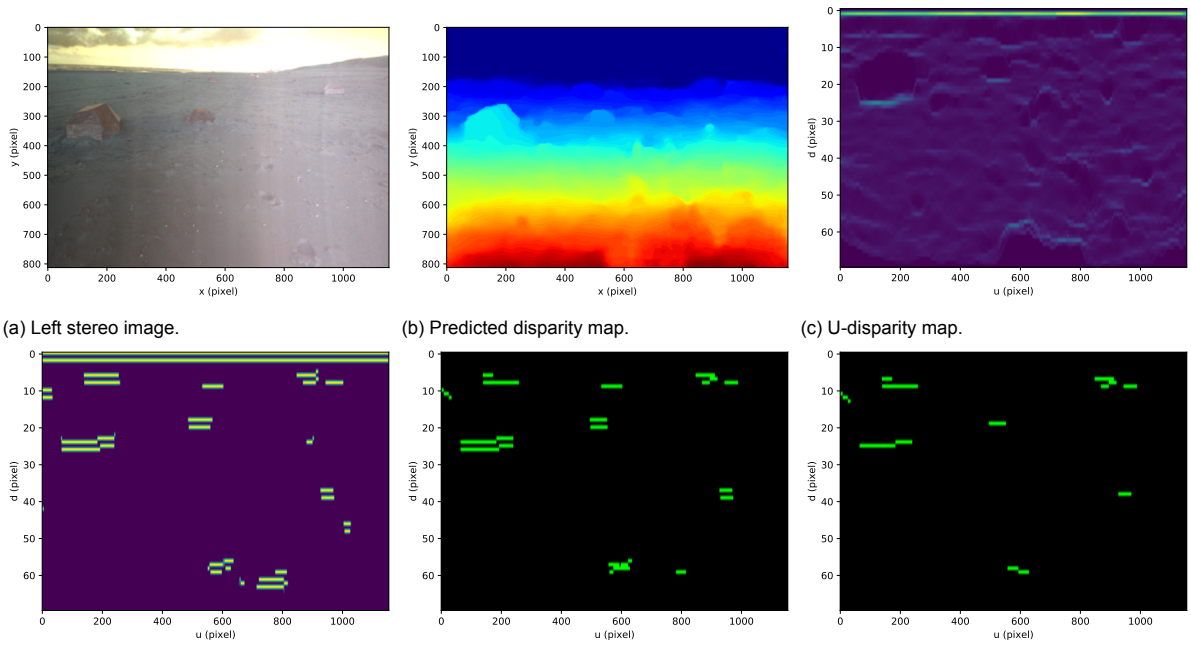
(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.12: Erroneous point analysis of the large rocks at time 12:57:50:379.

B.1.4. PWOC-3D

Small rocks

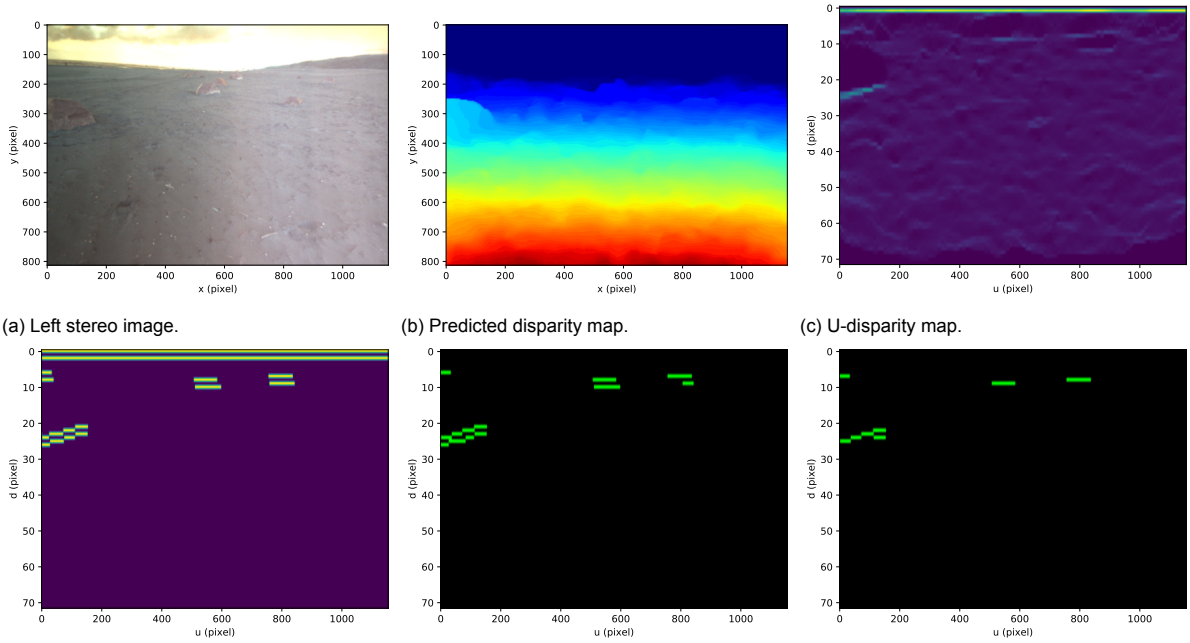


(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.13: Erroneous point analysis of the small rocks at time 12:56:21:167.

Medium rocks



(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

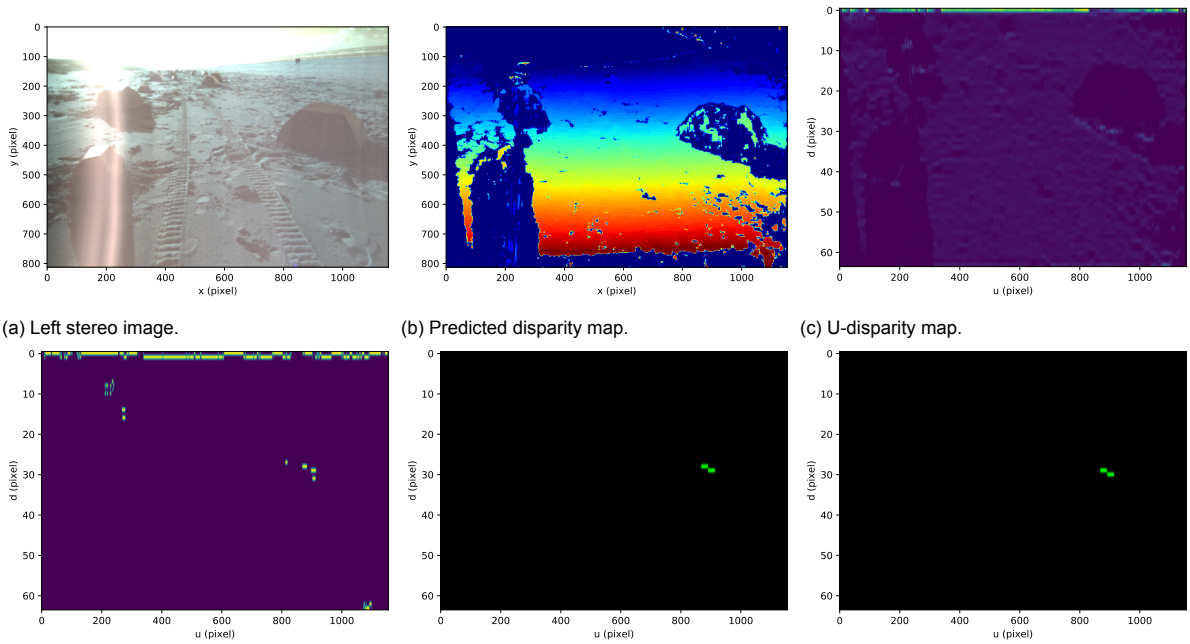
(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.14: Erroneous point analysis of the medium rocks at time 12:55:10:987.

B.2. Experiment 2

B.2.1. StereoBM (SAD)

Small rocks



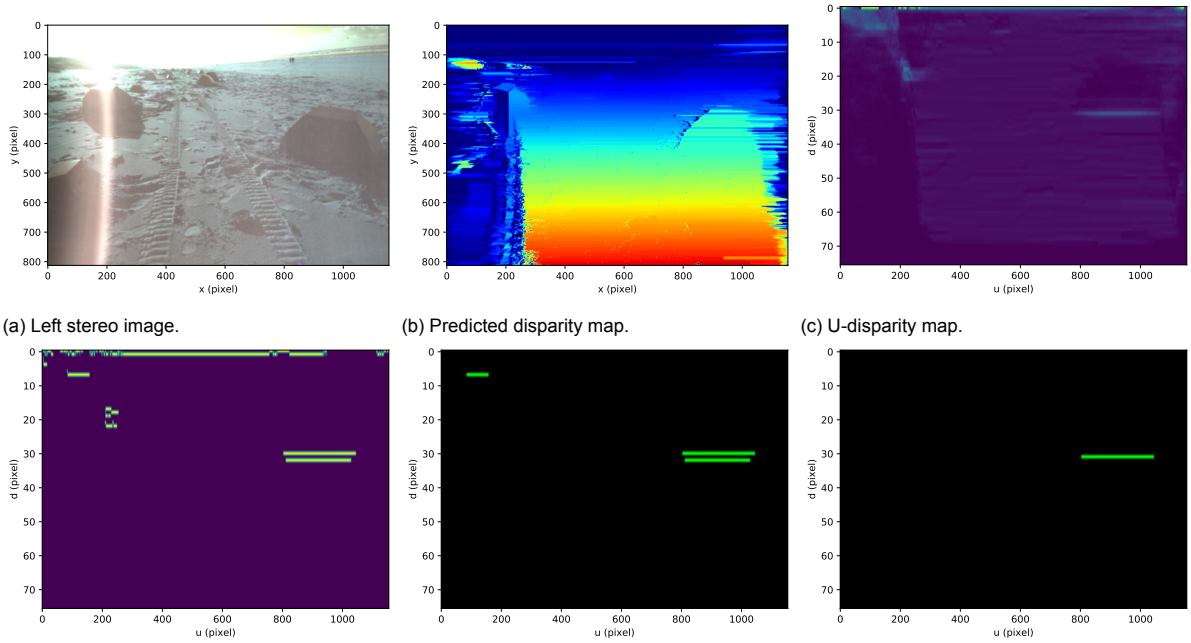
(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.15: Correct point analysis of the small rocks at time 13:39:56:704.

B.2.2. Hartley (DP)

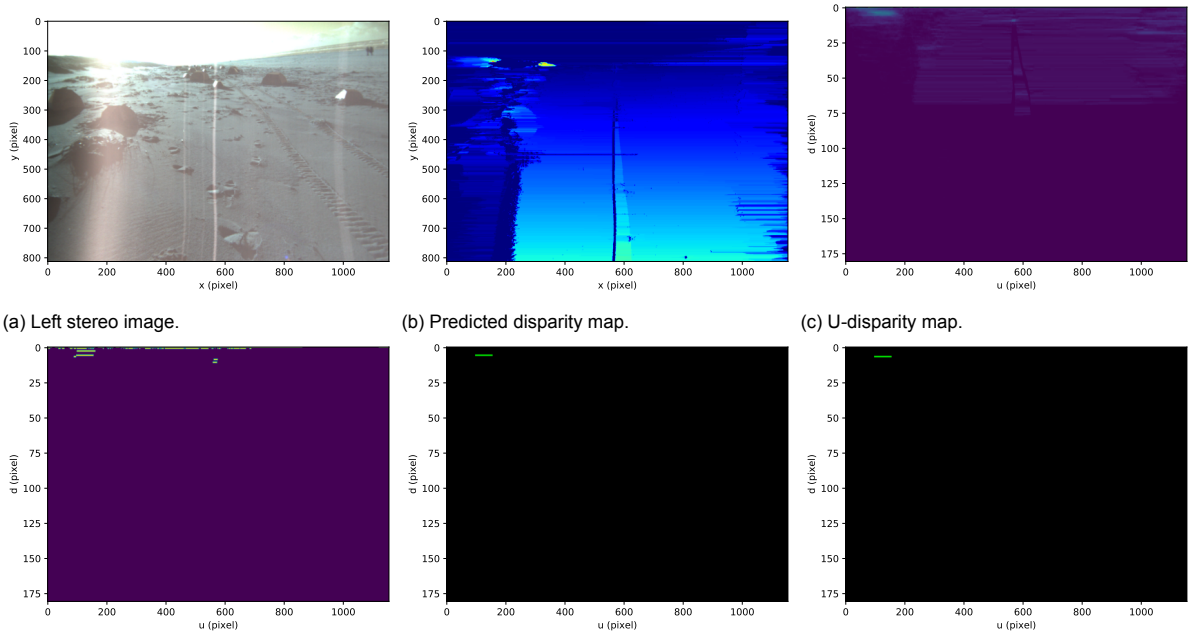
Small rocks



(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.16: Correct point analysis of the small rocks at time 13:39:57:770.



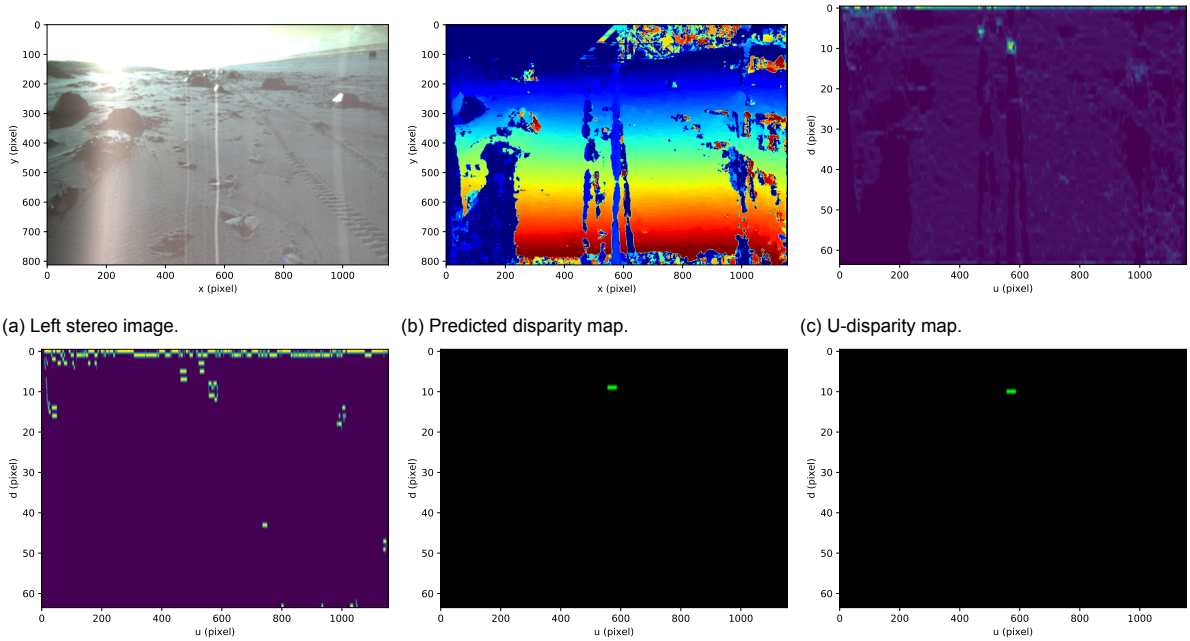
(a) Left stereo image. (b) Predicted disparity map. (c) U-disparity map.

(d) Edge detection. (e) Hough lines. (f) Filtered Hough lines.

Figure B.17: Erroneous point analysis of the small rocks at time 13:39:57:770.

B.2.3. StereoSGBM (SGM)

Small rocks



(a) Left stereo image.

(b) Predicted disparity map.

(c) U-disparity map.

(d) Edge detection.

(e) Hough lines.

(f) Filtered Hough lines.

Figure B.18: Erroneous point analysis of the small rocks at time 13:39:34:837.