Self-Supervised Finetuning of Stereo Matching Algorithms

R.U.J. Stikker

TU Delft

Self-Supervised Finetuning of Stereo Matching Algorithms

by

R.U.J. Stikker

in partial fulfilment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology,

to be defended publicly on Wednesday 5 October 2022 at 13:00.

Student number:	4476883	
Project duration:	3 June 2021 – 5 October 2022	
Thesis committee:	Prof.dr. G.C.H.E. de Croon	TU Delft, supervisor
	Dr.ir. E. Mooij	TU Delft
	Dr.ir. C. De Wagter	TU Delft
	Ir. S.U. Pfeiffer	TU Delft, supervisor

TU Delft

Preface

With this thesis, I am completing one of the main requirements for the degree of Master of Science in Aerospace Engineering. I am not quite there yet, as I still need to complete my internship before I can graduate, but finishing this thesis certainly feels like the end of a chapter. My life as a student is nearing its end, so I would like to take this opportunity to thank my friends for making this time so enjoyable. I wish to thank my family for their support during the entirety of my studies. And of course, I wish to thank my supervisors, Guido de Croon and Sven Pfeiffer, for their guidance and enthusiasm during the thesis.

R.U.J. Stikker
Delft, September 2022

Contents

Introduction vii

I Thesis Article 1

II Literature Study 17

1 Summary 19

2 Introduction 21

3 Vision 23
3.1 Stereo matching . . . 23
3.1.1 Matching cost computation . . . 24
3.1.2 Cost aggregation . . . 25
3.1.3 Disparity computation and optimization . . . 26
3.1.4 Refinement of disparities . . . 26
3.1.5 Global Methods . . . 26
3.2 Stereo matching algorithms . . . 26
3.2.1 Block Matching (BM) . . . 27
3.2.2 SemiGlobal Block Matching (SGBM) . . . 29
3.2.3 Efficient LArge-Scale Stereo matching (ELAS) . . . 30
3.3 Parameter tuning . . . 32
3.3.1 Supervised parameter optimization . . . 33
3.3.2 Self-supervised parameter optimization . . . 33

4 Navigation 35
4.1 Localization and mapping . . . 35
4.1.1 Mapping . . . 35
4.1.2 Visual (inertial) odometry . . . 36
4.1.3 Non-metric methods . . . 37
4.2 Motion planning . . . 38
4.2.1 Local planning . . . 38
4.2.2 Global planning . . . 39

5 Preliminary experiments 41
5.1 Parameter optimization . . . 41
5.1.1 Objective functions . . . 41
5.1.2 Parameter optimization . . . 44
5.2 Results . . . 44

6 Conclusion 49

Bibliography 51

Introduction

Unmanned Aerial Vehicles (UAVs) have become a popular tool in many applications, such as search and rescue, law enforcement, cinematography, and horticulture. For cheap and easy application these drones are becoming increasingly autonomous. Whereas in the early days the autonomy of drones was limited to automatic stabilization, current drones have more advanced capabilities, such as autonomously following the operator and avoiding obstacles. For any application, the UAV must pose little or no safety risks to people in the drone's vicinity. This can be achieved with safety measures such as detection and avoidance of humans. However, it is more reliable to use a UAV that is inherently safe due to its low weight. A lightweight drone poses less danger, even if the avoidance systems fail. A downside of lightweight drones is that they have simple sensors and low computational capabilities. Despite these limitations, the drone needs to obtain a 3D sense of the environment for tasks such as collision avoidance and mapping. To obtain this 3D information, stereo vision is a popular method, as the only sensors it requires are two simple cameras. The images from these cameras then need to be converted into depth maps using a stereo matching algorithm. The stereo matching algorithm needs to be as accurate as possible, while using little computational resources. However, the accuracy of a stereo matching algorithm in a specific environment is often difficult to quantify, because in most practical applications ground truth is not available. Therefore, this research investigates self-supervised finetuning of several lightweight stereo matching algorithms. More specifically, it will answer the following question.

> Can conventional stereo matching algorithms be improved through self-supervised optimization, and how do they compare to self-supervised deep learning?

This question will be answered in the thesis article in Part I. Subsequently, in Part II a literature review is given to provide background information to the article.It starts with a detailed explanation of stereo matching and stereo matching algorithms, followed by a chapter about navigation, and a chapter with preliminary experiments leading up to the main research.

I

Thesis Article

Self-Supervised Finetuning of Stereo Matching Algorithms

Roelof Stikker*, Sven Pfeiffer†, Guido de Croon†

***Abstract*— Stereo vision is a commonly applied method to achieve depth perception on Micro Air Vehicles (MAVs). Stereo matching algorithms are often optimized for specific environments and camera properties, using the ground truth error as a supervisor. However, in practical applications ground truth data is usually not available. Therefore, in this research, we finetune several conventional stereo matching algorithms (BM, SGBM, and ELAS) and a neural network (AnyNet) using self-supervision. The settings of the conventional algorithms are optimized with NSGA-II, using the reconstruction error and disparity density as objective functions. AnyNet is finetuned with the reconstruction error, as well as with the disparity map of conventional methods. We conclude that finetuning the parameters of conventional stereo algorithms using the reconstruction error can lead to a slight improvement in performance compared with the general settings, depending on the stereo algorithm. The performance of the conventional methods is comparable to that of AnyNet on a major portion of the image. However, removing the values with low confidence in the disparity map of ELAS and interpolating the missing disparities leads to an accuracy well above AnyNet.**

I. INTRODUCTION

MAVs need to have a 3D sense of the environment for successful autonomous flight. It is required for tasks such as collision avoidance, navigation, localization, and mapping. Stereo vision is often used on lightweight drones to obtain this 3D information because it only requires two simple cameras. The images from these cameras are used by a stereo matching algorithm to calculate a depth map. Lightweight drones do not have the computational capacity to process complicated stereo matching algorithms. For this reason, simple stereo matching algorithms such as Block Matching (BM) [1], SemiGlobal Block Matching (SGBM) [2], and Efficient LArge-scale Stereo (ELAS) [3] are popular methods in practical applications [4]–[6]. However, the performance of these methods is highly dependent on their parameter settings, the environment, and the camera properties. Furthermore, for each application, a different trade-off might be preferred between the density and the accuracy of the resulting disparity map. An NSGA-II optimization was performed by [7] to estimate the Pareto front of the trade-off between density and accuracy. The error with the ground truth was used to quantify the accuracy. However, in most practical scenarios, ground truth data is not available. Therefore, in this work, we will optimize the accuracy and density of BM, SGBM, and ELAS in a self-supervised way by using the reconstruction error to quantify the accuracy.

*MSc student, Faculty of Aerospace Engineering, Delft University of Technology

†Supervisor, Faculty of Aerospace Engineering, Delft University of Technology

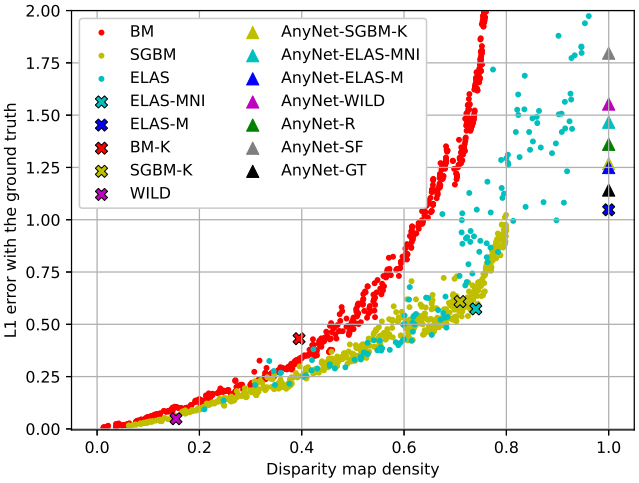


Fig. 1: We optimize the parameters of conventional stereo matching algorithms in a self-supervised way (dots) and finetune the neural network AnyNet in self-supervised and supervised ways (triangles). These are compared with the conventional algorithms with general settings (crosses). The legend is described in more detail in Table II.

Recent developments in stereo vision have been focused on the use of neural networks. Originally these were too computationally expensive to run on board a lightweight drone, but lately researchers have focused on designing neural networks specifically for application on resource-limited systems [8]–[12]. In this work, we compare the results of one of these networks, AnyNet [10], with the results of the conventional methods. AnyNet was designed to be trained on ground truth data only. We finetune AnyNet using self-supervision instead. Similar to the NSGA-II optimization, the reconstruction error will be used as a supervisor. In addition, conventional methods will be used as supervisors.

In short, this research investigates self-supervised parameter tuning of several conventional stereo matching algorithms, as well as self-supervised refinement of a deep stereo matching network.

The remainder of this paper is structured as follows. Section II shows an overview of previous research on traditional stereo matching and deep stereo matching, with a focus on self-supervised and lightweight methods. Then in section III the methodology of this research is explained in detail. The results are presented and discussed in section IV, followed by a conclusion in section V.

II. RELATED WORK

A. Conventional stereo matching

Efficient LArge-scale Stereo (ELAS) [3] and the OpenCV [13] implementations of Block Matching (BM) [1] and SemiGlobal Block Matching (SGBM) [2] are popular conventional stereo matching algorithms in practical applications because they produce good disparity maps at high framerates. BM is the simplest and fastest method and calculates the disparity by minimizing the sum of absolute differences between a block in the reference image and a block on the same row in the other image. SGBM expands on this by including a penalty for changing disparities between neighboring pixels. ELAS first matches a reliable set of support points and then fills the rest of the disparity map, where a penalty is given if the disparity deviates from the nearby support points.

The quality of the disparity maps obtained by these methods is dependent on their parameter settings. All three methods contain built-in confidence measures to remove unreliable values in the disparity map. By setting these parameters, a trade-off is made between the density and the quality of the disparity map. NSGA-II [14] was used by [7] to find the Pareto front for this trade-off. However, the objective function was based on the ground truth. This is not available in most practical applications. Instead, [15] and [16] optimized the settings of BM using the reconstruction error as objective function. However, they only optimized towards a single point on the Pareto front and did not provide a comparison of various methods.

B. Deep stereo matching

Self-supervised learning in deep stereo matching was first applied by [17]. They trained a monocular network to minimize the reprojection error and a disparity smoothness term. [18] expanded the loss function by including a structural similarity term [19] and a left-right consistency term. This loss function forms the basis for many self-supervised stereo models as well [20]–[24]. However, these models are too computationally demanding for real-time depth mapping on an embedded platform. Some models designed for real-time use are StereoNet [8], MADNet [9], AnyNet [10], FEStereo [11], and LWANet [12]. However, even some of these methods still have a rather low framerate when tested on an NVIDIA Jetson TX2.

Most deep stereo methods rely on the use of ground truth data to train and finetune the network. They can be modified to train in a self-supervised way instead. [25] finetunes a network using a confidence-guided loss and a smoothing term and tests it on the KITTI 2015 dataset [26]. Here the confidence-guided loss is defined as the difference between the output of the network and a set of points from the disparity map of a conventional algorithm that are deemed reliable. However, the classification of reliable versus unreliable is done with a neural network [27] that was trained on the KITTI 2012 dataset [28], using ground truth data. Although the weights were kept the same, the process is not fully self-supervised. Applying this method to a dataset different from KITTI could require retraining of the confidence network. Instead, [29] modified a network to be fully self-supervised. They train the network using disparity maps generated by a Monocular Completion Network. This is a network that takes as input one of the two images and a set of reliable disparities obtained from SGM or BM and gives as output the complete disparity map. The deep stereo model is then trained on this disparity map. The classification of reliable versus unreliable is done using the WILD strategy [30]. This is a machine-learned combination of conventional confidence measures. However, this method of training a stereo matching network is rather extensive as it requires training a monocular model before the stereo model can be trained.

III. METHODOLOGY

A. Conventional methods

The parameter settings of the conventional methods, BM, SGBM, and ELAS are optimized using NSGA-II. Instead of optimizing towards a single objective function consisting of both the accuracy and density of the result, NSGA-II allows optimizing for both objectives and finding the Pareto front.

All three models feature confidence filters, which can be used to invalidate uncertain disparity values. Uncertain disparities are for example those where the input images have low texture, so a texture filter can be used to invalidate these. Also, a match is only retained if it is sufficiently better than the second-best match (uniqueness ratio) and if it can consistently be matched left-to-right and right-to-left (left-right consistency filter). Lastly, a speckle filter removes small areas with disparity values dissimilar to their surroundings. These removed disparities can then be filled with interpolation. However, only ELAS has this option. Since allowing only ELAS to interpolate its missing disparities would give it an unfair advantage, it is switched off for the optimization.

The accuracy of the result is estimated without ground truth data. Instead, the reconstruction error is used. This works by reconstructing one image using the other image and a disparity map. Then by comparing the reconstruction with the actual image, the reconstruction error is found. In this work, the left image (I_L) is used as the reference image for which the disparity map (D_L) will be calculated. Once the left disparity map is calculated, a reconstruction of the left image ($\hat{I}_L$) is made by sampling from the right image (I_R) as stated below.

$$\hat{I}_L(i,j) = I_R(i - D_L(i,j), j) \tag{1}$$

Since the disparities are not necessarily integers, the pixel values are linearly interpolated. If a pixel in the reconstruction requires sampling from outside the right image, i.e. when $i - D_L(i,j) < 0$, the pixel is invalidated.

This method does not prevent double mappings of pixels from the right image to the reconstruction of the left image. Consider a grayscale stereo image of a faraway black cross and a nearby gray square, as is shown in Figure 2. Using the

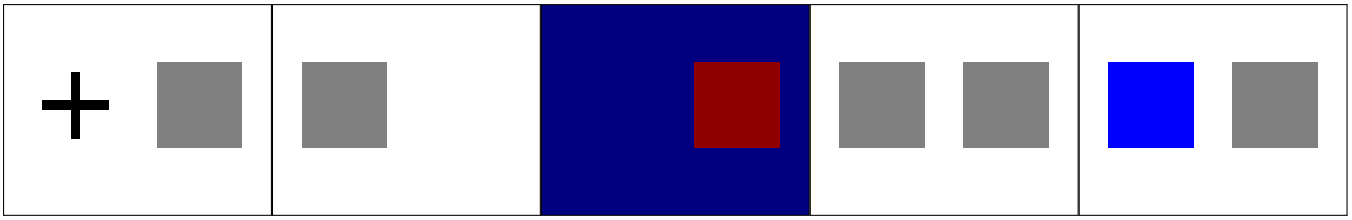


Fig. 2: Schematic images illustrating the occlusion filter that prevents double mappings in the reconstruction. From left to right: the left image, the right image, the left disparity map, the reconstruction of the left image, and the filtered reconstruction of the left image. In the last image, invalidated pixels are marked blue.

disparity map and the right image, the left image is reconstructed by sampling according to Equation 1. Consequently, two copies of the square appear in the reconstruction. Note that to reconstruct the cross, the right image is sampled at the correct location, but because it is occluded by the square, the square shows up instead. The reconstruction then has a large error, even though the disparity map is correct. To avoid these errors in the reconstruction, a pixel is invalidated if another pixel with a higher disparity originates from approximately the same location in the right image (plus or minus half a pixel). When the disparity map is a bit noisy this can happen often with nearby pixels, even on flat surfaces. Therefore, occluded pixels are only invalidated if the disparity difference with the occluding pixel is greater than or equal to 4. Thus, if for any $n \geq 4$, $|D(i+n,j)-n-D(i,j)| < 0.5$, we invalidate $\hat{I}_L(i,j)$. An example of this is visible in the reconstruction in Figure 3, where a double mapping of the traffic sign in the top right corner is prevented.

Now that the left image is reconstructed, it is compared with the actual left image to determine the accuracy of the disparity map. This is done similarly as in [18]. It consists of the average absolute difference in intensity (E_{L1}), an error based on the structural similarity (SSIM) [19] (E_{SSIM}), and a disparity smoothness component (E_{DS}). E_{L1} and E_{SSIM} are defined as in the equations below. In these equations, and in the remainder of this paper, the input images are assumed to be 8-bit grayscale, and thus have a range of [0, 255].

$$E_{\text{L1}} = \frac{1}{N_D} \sum_{p \in D} \left| \frac{I_p - \hat{I}_p}{255} \right| \tag{2}$$

$$E_{\text{SSIM}} = \frac{1}{N_B} \sum_{p \in B} \frac{1 - \mathcal{S}(I_p, \hat{I}_p)}{2} \tag{3}$$

Where D is the set of N_D pixels with a valid disparity. B is the set of N_B pixels for which all pixels in a surrounding 3×3 block have a valid disparity. This ensures that all pixels required for the calculation of the SSIM function $\mathcal{S}(\cdot)$ are valid. The SSIM function is defined as stated below.

$$\mathcal{S}(I_p, \hat{I}_p) = \frac{(2\mu_{I_p}\mu_{\hat{I}_p} + C_1)(2\sigma_{I_p\hat{I}_p} + C_2)}{(\mu_{I_p}{}^2 + \mu_{\hat{I}_p}{}^2 + C_1)(\sigma_{I_p}{}^2 + \sigma_{\hat{I}_p}{}^2 + C_2)} \tag{4}$$

Here, μ is the average of a 3×3 block surrounding pixel p and σ^2 is its variance. C_1 and C_2 are constants used to avoid instability when either $(\mu_{I_p}{}^2 + \mu_{\hat{I}_p}{}^2)$ or $(\sigma_{I_p}{}^2 + \sigma_{\hat{I}_p}{}^2)$ is close to zero. They have values of 0.0001 and 0.0009, respectively, as in [18].

The disparity smoothness error penalizes a gradient in the disparity map if this does not coincide with a gradient in the image. It is defined as follows.

$$E_{\text{DS}} = \frac{1}{N_G} \sum_{p \in G} \left(|\partial_x D_p| e^{-|\partial_x I_p|} + |\partial_y D_p| e^{-|\partial_y I_p|} \right) \tag{5}$$

Here G is the set of N_G pixels for which calculating the gradients requires no invalid pixels. The gradients are calculated as follows.

$$\partial_x D(i,j) = D(i,j) - D(i+1,j) \tag{6}$$

$$\partial_y D(i,j) = D(i,j) - D(i,j+1) \tag{7}$$

G is thus the set of pixels with a valid disparity with valid adjacent disparities to the right and down.

The reconstruction error E_R is then defined as the weighted combination of E_{L1}, E_{SSIM}, and E_{DS}. A visual representation of these components is shown in Figure 3.

$$E_{\text{R}} = \alpha_a \left((1-\alpha_s) E_{\text{L1}} + \alpha_s E_{\text{SSIM}} \right) + (1-\alpha_a) E_{\text{DS}} \tag{8}$$

Here, the weighting factors α_a and α_s were empirically set to 0.9 and 0.25, respectively.

B. Deep stereo matching

Current state-of-the-art deep stereo networks produce much better results than conventional methods. However, the computational requirements of these methods are much higher than conventional methods. To determine whether deep stereo matching is a viable alternative to conventional methods we select a neural network that was designed for real-time use on embedded platforms. Table I shows a list of networks that were tested on an NVIDIA Jetson TX2 and their performance.

Since there is neither a GPU implementation of BM, SGBM, nor ELAS, it is difficult to compare the computational time of the conventional methods with the neural networks. However, there is a GPU implementation of SGM,

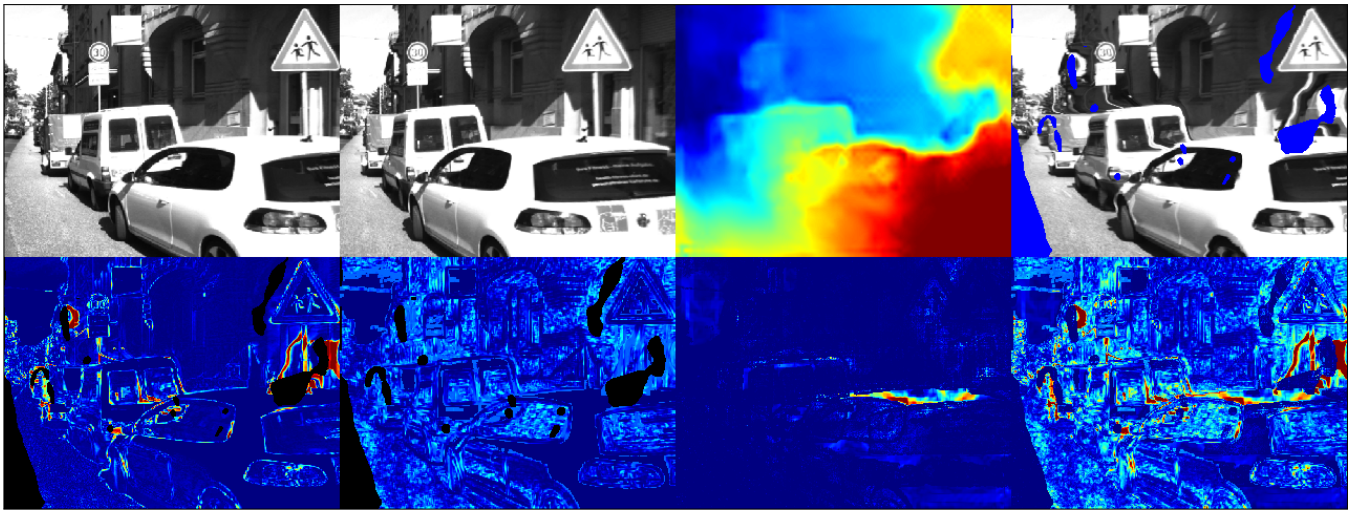


Fig. 3: Components of the reconstruction error. In the top row from left to right: the left image, the right image, the left disparity map (AnyNet-SGBM-K), and the left reconstruction. In the bottom row from left to right: the L1 error, the structural similarity error, the disparity smoothness error, and the total error

TABLE I: D1 error and computation time of compact stereo models with publicly available code, tested on the KITTI 2015 dataset on an NVIDIA Jetson TX2

Name	D1 [%]	Time [ms]
MADNet [9]	4.66 [12]	260 [12]
StereoNet [8]	4.83 [12]	950 [12]
LWANet [12]	4.94 [12]	200 [12]
AnyNet [10]	6.2 [10]	97.3 [10]
FEStereo [11]	11.2 [11]	14 [11]
SGMGPU [31]	8.7 [32]	34.5 [32]

called SGMGPU [31]. Considering that SGBM is a modification of SGM to reduce the computation time, a potential GPU implementation of SGBM is expected to have a lower inference time. As can be seen from Table I, FEStereo [11] is the only method that is faster than SGMGPU, but the disparity map is less accurate. Note that according to the KITTI 2015 benchmark, SGBM has a D1 error of 10.86%, which means that 10.86% of the disparities have both an absolute error greater than 3 and a relative error greater than 5%. Thus, to improve the disparity map, compared with SBGM, the fastest method is AnyNet [10]. Therefore, in this paper, AnyNet is compared with the conventional methods.

AnyNet performs disparity estimation at four stages. This allows the network to trade off computation time and accuracy. The D1 error and time range from 14% and 29.0 ms in the first stage to 6.2% and 97.3 ms in the final stage. AnyNet first uses a U-Net feature extractor [33] to build feature maps of the two images at three resolutions (1/16, 1/8, 1/4). In the first stage, the features at 1/16th resolution are passed through a disparity network to produce a disparity map. This disparity map is then upscaled to 1/8th resolution and used to warp the 1/8th right feature map onto a reconstructed left feature map. The reconstruction and the original left feature map are passed into the disparity network to produce a residual disparity map. The residual is then added to the upscaled disparity map of the previous stage to obtain the current disparity map. This process is repeated for stage 3. In the fourth and final stage, a spatial propagation network (SPNet) [34] is used to refine the disparity predictions. For ease of reference, the network configuration from [10] is reproduced in Table A.I.

The network is originally supervised, but in this work, we finetune it in a self-supervised way instead. We still use the pre-trained model, which was trained on the SceneFlow dataset [35], using the ground truth error. Two methods of self-supervised finetuning will be compared. The first uses the reconstruction error, as explained in section III-A. The second method uses the disparity maps of conventional methods as a supervisor, as was also done by [25] and [29]. Instead of quantifying the error of the disparity map using the reconstructed image, the error of the disparity map is quantified as the L1 error with the disparity map produced by a conventional method. Since the conventional methods are not perfect, they are filtered to retain only the high-confidence disparities. We follow four different versions of this approach. The first three versions use SGBM-K, ELAS-MNI, and ELAS-M, respectively. In the fourth version, WILD [30] is used as a supervisor, because it was found to be effective by [29]. It generates disparity maps using AD-CENSUS [36]. These disparity maps are filtered using a machine-learned combination of traditional confidence filters. The filtered disparity maps then supervise finetuning AnyNet. An example of a disparity map generated by WILD is shown in Figure A.1.

IV. EXPERIMENTAL RESULTS

A. Datasets

The purpose of lightweight stereo models is to be used on lightweight drones with simple cameras and little computational resources. Therefore, we test the stereo models

on a set of images recorded with a simple and lightweight stereo camera, made by connecting two JeVois cameras. The images are grayscale and have a resolution of 320×240 pixels. The set of images is recorded in a greenhouse, as this is considered an environment where a lightweight drone could be of good use. The dataset contains 478 images and is randomly split into a training and validation set in an 80/20 ratio.

Since this dataset contains no ground truth information, the only quantitative measure of the results is the reconstruction error, which is not a perfect measure. The performance on this dataset will therefore mainly be discussed qualitatively.

To obtain reliable quantitative results, the same methods are applied to a modified version of the KITTI 2015 dataset. The original KITTI 2015 dataset is converted to grayscale and downsampled to the same resolution as the JeVois camera. Since the images of the KITTI dataset are much wider, a random portion is cropped with the same aspect ratio as the JeVois image, and the same height as the KITTI image. This cropped image is then downscaled to a resolution of 320×240 pixels. This dataset contains 200 images and is also randomly split into a training and validation set in an 80/20 ratio.

B. Performance metrics

As stated above, the results will be discussed both qualitatively and quantitatively. The quantitative metrics that will be used are the reconstruction error and, for the KITTI dataset, the L1 and D1 errors with the ground truth. The L1 error with the ground truth is the average absolute difference between the disparity values in the disparity map produced by the model and the ground truth disparity map, averaged over all pixels with a valid disparity both in the ground truth and in the calculated disparity map. The D1 error is the percentage of the pixels with a valid disparity (in both the ground truth and the calculated disparity map), that have an absolute error greater than 3 and a relative error greater than 5%.

C. Conventional methods

1) Implementation details: For ELAS we use the implementation as published by the authors. For BM and SGBM the implementations by OpenCV are used. In the implementations by OpenCV, disparities are only calculated for pixels where the full disparity search range can be traversed. This means that no matter the parameter settings, the disparities are not calculated for the left part of the image, with a width equal to the maximum disparity. In addition, in the implementation of BM, disparities are not calculated if the surrounding block does not fully fit in the image. This means that the border of the disparity map will have no values. A visualization is shown in Figure A.1.

The parameters of the conventional methods are optimized using the Pymoo [37] implementation of NSGA-II [14]. In the optimization, the parameter limits are set as in Tables A.II, A.III, A.IV. The initial population is set to 1000 and the number of offspring to 100. The optimization is terminated when the change in the objective space drops below 0.0025 over the last 30 generations. This is evaluated every 5 generations. To reduce the computational time, the optimization does not use the full training set, but a random subset of 40 images. The same subset is used for all NSGA-II optimizations.

TABLE II: Descriptions of algorithm variations

Name	Description
BM-K	Block Matching, with manual settings based on the KITTI 2015 benchmark entry
SGBM-K	SemiGlobal Block Matching, with manual settings based on the KITTI 2015 benchmark entry
ELAS-M	Efficient LArge-scale Stereo, with manual settings based on the Middlebury benchmark entry
ELAS-MNI	ELAS-M, but without interpolation
WILD	AD-CENSUS disparity maps filtered with a machine-learned combination of traditional confidence filters
AnyNet-SF	AnyNet trained on SceneFlow
AnyNet-R	AnyNet-SF, finetuned with the reconstruction error
AnyNet-WILD	AnyNet-SF, finetuned with WILD
AnyNet-ELAS-M	AnyNet-SF, finetuned with ELAS-M
AnyNet-ELAS-MNI	AnyNet-SF, finetuned with ELAS-MNI
AnyNet-SGBM-K	AnyNet-SF, finetuned with SGBM-K
AnyNet-GT	AnyNet-SF, finetuned with the ground truth

2) Baselines: The results of the NSGA-II optimizations are compared with the results of general settings, which were based on submissions to online benchmarks. For BM and SGBM, the general settings were based on the submissions to the KITTI 2015 benchmark. They are manually adjusted to accommodate for the change in resolution. In the plots, they are indicated with BM-K and SGBM-K. For ELAS, two baselines are used. One is the parameter configuration that was used for the Middlebury benchmark (ELAS-M). In this configuration, all missing disparities are interpolated, resulting in a 100% dense disparity map.

Since interpolation is always switched off in the NSGA-II optimization of ELAS, the settings of ELAS-M can not be reached by the optimizer. It is still included to demonstrate the effect of interpolation. To make a fair comparison between general settings and the result of the optimization, a version of ELAS where no interpolation is done is included in the plots (ELAS-MNI). An overview of these algorithm variations is given in Table II. The specific settings are given in Tables A.II, A.III, and A.IV. In addition, an example scene with the disparity maps calculated by these algorithms is given in Figure A.1.

3) Discussion of the results: The Pareto fronts that were found using the NSGA-II optimization are shown in Figure 4. From this figure, we can see that in terms of the reconstruction error, the performance of BM is improved compared with the general settings. The performances of SGBM and ELAS are similar to the general settings, except for ELAS-M, but this configuration was not attainable by the optimizer. Figures 1 and 5 show that the ground truth is mostly in agreement with this. They both show an improvement for BM, while SGBM stays similar. From Figure 1 it looks as if there is a slight improvement for SGBM in terms of the L1 error, but around that density there are also solutions

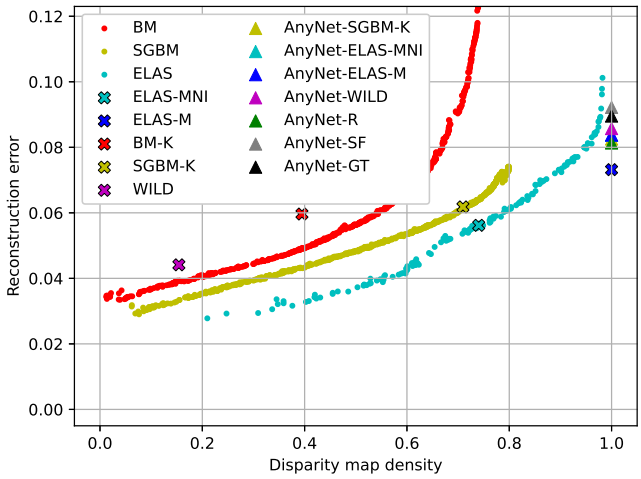


Fig. 4: Reconstruction error of solutions along the Pareto fronts. The crosses indicate the conventional methods with general settings. Triangles indicate AnyNet with varying finetuning methods.

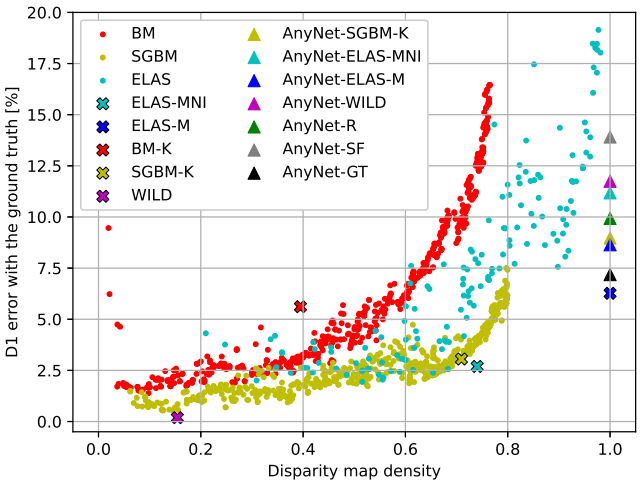


Fig. 5: D1 error of solutions along the Pareto fronts. The crosses indicate the conventional methods with general settings. Triangles indicate AnyNet with varying finetuning methods.

that perform worse. For ELAS on the other hand, there is a clear difference between the judgment by the reconstruction error (Figure 4) and by the ground truth (Figures 1 and 5). While according to the reconstruction error the optimization results are similar to ELAS-MNI, according to the ground truth the optimization results clearly perform worse. Also, the Pareto fronts in Figures 1 and 5 are much more scattered than the reconstruction error (Figure 4). Both of these symptoms indicate that settings with very similar reconstruction errors can have very different errors with the ground truth. In other words, a low reconstruction error does not guarantee a low error with the ground truth. An example of this is illustrated in Figure 6. In the upper image, large parts of the road have an incorrect disparity. This leads to an incorrect remapping in the reconstruction. However, since the road has a uniform color, these incorrect remappings go undetected. This means that a poor parameter configuration is not penalized, resulting in poor optimization results. To verify that the optimized versions of ELAS do not achieve the performance of ELAS-MNI because of shortcomings of the reconstruction error, rather than due to an incomplete optimization, the optimizations are also performed with the L1 error with the ground truth as supervisor. The results of these optimizations are shown in Figures A.2 and A.3. To visualize the completeness of the optimizations, the convergences of the hypervolumes are shown in Figure A.4. The figure shows that after around 9000 function evaluations all hypervolumes stop increasing.

Furthermore, we see that the results of ELAS are much more scattered and have much higher errors than the results of SGBM and BM. This is a result of the fact that ELAS performs its matching in two steps. First, a set of support points is matched, then these support points are used to perform dense matching. When the number of support points is too low due to excessive confidence filtering in this stage, an accurate dense disparity map cannot be achieved.

The fronts of BM and SGBM never reach the full 100% density. This is because in the OpenCV implementation, the left part of the disparity map, with a width of the maximum disparity, is set invalid because the full disparity search range cannot be traversed. The upper limit of BM is even lower because additionally, it invalidates pixels for which the surrounding block does not fully fit in the image. ELAS, on the other hand, can fill the entire disparity map, although the error rises steeply as the 100% density is approached. One could argue that when SGBM achieves a disparity density of 80%, it is equivalent to a 100% disparity since it only attempts to fill 80%. Technically there is no reason why SGBM should not be able to fill the remaining 20%. However, for computational reasons, it is not included in the OpenCV implementation. In a different implementation, SGBM could achieve 100% density. To estimate the performance of such an implementation, simply scaling the results up to 100% and comparing it to ELAS would not be fair, since the left part of the disparity map is more difficult to calculate because it is more likely to contain pixels that have no correct match. Therefore, we modify ELAS and SGBM to invalidate all pixels that are not calculated by BM either. This means that the pixels on the left with a width of the maximum disparity will be invalidated, and since the minimum block size of BM is 5, a border with a width of 2 will be invalidated as well. Now all three methods have a theoretical maximum density of 77.4%. These results are plotted in Figure 7 and Figure 8. Note that the algorithms and their settings are the same as in Figures 1, 4, and 5. The only difference is that now the left part and the borders are invalidated. They show that the accuracy of SGBM and ELAS is very similar in terms of the L1 error, while in terms of the D1 error SGBM is performing better.

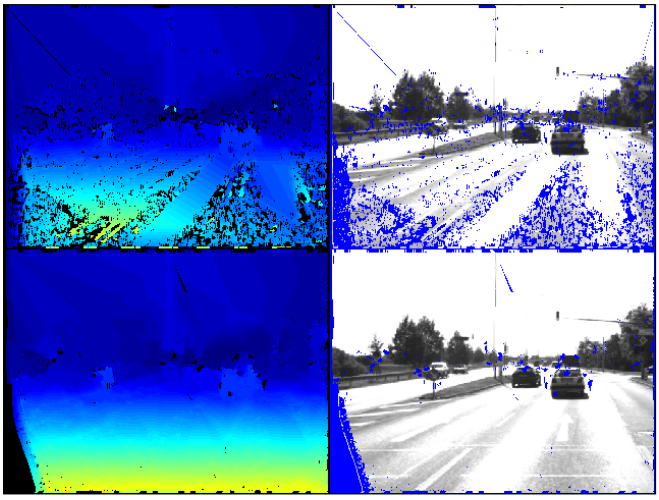


Fig. 6: Two solutions along the Pareto front of ELAS. On the left the disparity maps, with invalid pixels set to black, and on the right the reconstructed images, with invalid pixels set to blue. The upper and lower disparity maps have a reconstruction error of 0.053 and 0.059, respectively, while the D1 error with the ground truth is 36% and 2.7%, respectively.

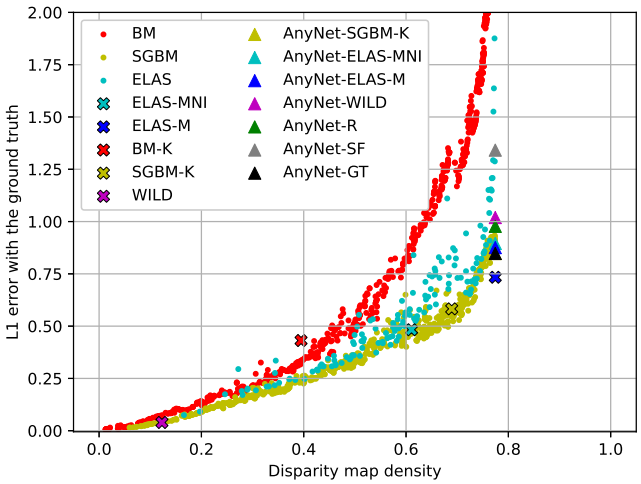


Fig. 7: L1 error of solutions along the Pareto fronts. The left part and border of all disparity maps are invalidated. The crosses indicate the conventional methods with general settings. Triangles indicate AnyNet with varying finetuning methods. The locations of AnyNet-GT and AnyNet-SGBM-K coincide.

D. AnyNet

1) Implementation details: The implementation of AnyNet is similar to the original work [10]. We use their published code, but modify the training loss function. The training parameters are kept the same, except that the maximum disparity is changed to 64. An overview of the training settings is given in Table A.V.

Instead of training on a portion of the image, as in [10], we train on the complete image, since it already has a much lower resolution than the original KITTI images. We start from the pre-trained model, which was trained on the SceneFlow dataset.

2) Baselines: The performance of the finetuned model is compared with the pre-trained model (AnyNet-SF). These results are presented to indicate the progress that is made after finetuning. In addition to this, we also finetune the model using the L1 error with the ground truth as the training loss (AnyNet-GT). This is expected to lead to the maximum attainable performance of the network and is used to show the effectiveness of the self-supervised training method. Figure A.1 shows an example scene with disparity maps calculated by AnyNet-SF and AnyNet-GT. Furthermore, the result of AnyNet will be compared with the performance of the conventional methods.

3) Discussion of the results: In this work, AnyNet was finetuned using various self-supervised methods. An example scene with the disparity maps calculated by these versions of AnyNet is shown in Figure A.1. The overall quantitative performances are plotted as triangles in Figures 1 and 5. As expected, these plots show that the performance of the model trained with ground truth performs best, followed by the self-supervised training using the conventional methods as supervisor. Both of these perform better than the model that uses the reconstruction error as training loss. This could be expected after seeing Figure 6. The reconstruction error does not always penalize incorrect disparity maps.

Furthermore, Figures 1 and 5 show that when AnyNet is trained on the sparse disparity maps of ELAS-MNI the performance is much lower than when it is trained on the fully dense disparity maps of ELAS-M. Note that the disparity maps are the same, except that in ELAS-M missing disparities are interpolated. This suggests that AnyNet benefits from denser disparity maps. This is also supported by the poor performance of AnyNet when it is trained using WILD since this produces disparity maps with very low density.

Interestingly, from Figure 7 and Figure 8 we see that the L1 error of AnyNet trained on SGBM is equal to the version trained on ground truth. The D1 error for the SGBM-trained version is even slightly lower than the version trained on ground truth. This could also be caused by the fact that the disparity maps from SGBM are much denser than the ground truth, as is shown in Figure A.1. Comparing these results with the results from Figure 1 and Figure 5 we can conclude that the error of AnyNet-SGBM-K is mainly located in the left part of the image and/or the borders. This is also confirmed by visual inspection of the disparity maps. An example scene where this is visible is shown in Figure A.1. This is likely a result of the fact that in all the training data, the left part of the disparity map has no valid value.

Furthermore, Figure 7 and Figure 8 show that the error of the best versions of AnyNet are comparable to the error of SGBM and ELAS. However, the version of ELAS that interpolates its missing disparities (ELAS-M) is still better than even the best version of AnyNet. The performance of ELAS near 100% density is much better when this density

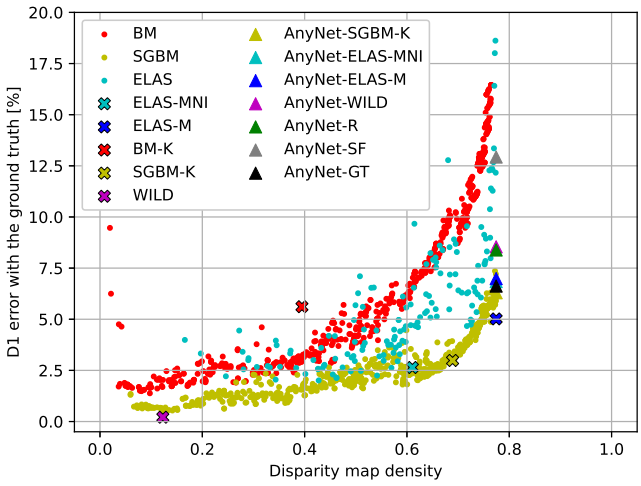


Fig. 8: D1 error of solutions along the Pareto fronts. The left part and border of all disparity maps are invalidated. The crosses indicate the conventional methods with general settings. Triangles indicate AnyNet with varying finetuning methods.

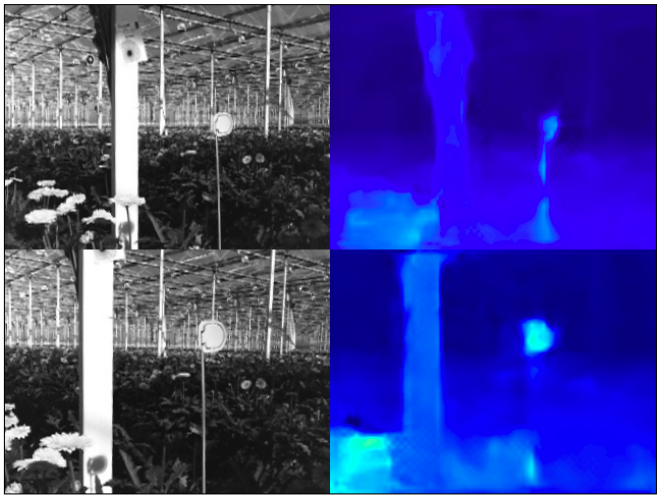


Fig. 9: Two subsequent disparity maps by AnyNet-ELAS-M

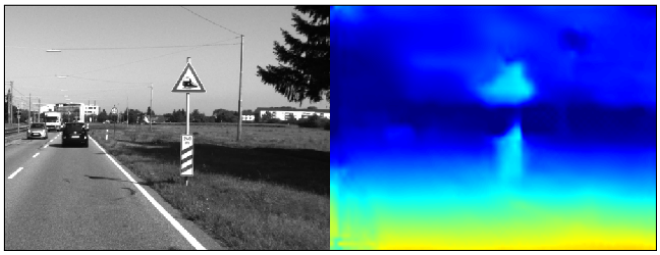


Fig. 10: A scene from the KITTI 2015 dataset with the disparity map calculated by AnyNet-SGBM-K

is achieved by interpolating the invalidated disparities, than when it is achieved by invalidating fewer disparities. This poses the question of whether AnyNet would benefit from this in the same way. There is no built-in confidence filter, but it is possible to apply the same confidence filters. AnyNet applies a series of convolutions to arrive at a cost volume, similar to conventional methods. It then computes the disparity through a weighted average. This means that if there are for example two matching candidates with an equally low cost, the average of the two disparities will be selected. The conventional methods, on the other hand, would in such a case classify the match as unreliable, because the uniqueness ratio is not sufficient. Besides a uniqueness ratio, a left-right consistency filter, and texture filter could perhaps be beneficial to AnyNet.

Besides training and testing on the KITTI 2015 dataset, we also trained AnyNet on the dataset from the greenhouse. Since the reconstruction error has proven to be an unreliable performance metric, the performance on this dataset will only be discussed qualitatively.

Figure 9 is an example that shows that AnyNet has difficulty detecting thin obstacles, especially if they are nearby. The thin stick is detected when it is far away, but when it gets closer it blends in with the background. This is because AnyNet works in four stages. The full disparity range is only traversed in the first stage at very low resolution. At such low resolution, the stick is not visible. In the subsequent stages, the disparity map is upscaled and only a small disparity range of ± 2 is traversed to finetune the disparity map, but because the stick has a higher disparity difference with the background, it goes undetected. This problem also occurs in the KITTI dataset, as is shown in Figure 10.

Figure 11 shows a challenging scene including a horizontal uniformly colored beam. This is fundamentally difficult to detect with stereo vision if the cameras are in the same plane as the beam. Unsurprisingly, all methods have trouble detecting the beam. The image also shows a small error near the entire left edge of the image. This is common among the disparity maps from AnyNet-SGBM-K and is likely to be caused by the fact that the disparity maps of SGBM have invalid values here.

V. CONCLUSIONS

In this work, we have optimized the settings of the conventional stereo matching algorithms BM, SGBM, and ELAS by minimizing the reconstruction error. Using NSGA-II optimization we estimated the Pareto front and compared this with the results obtained using general parameter configurations. These optimizations based on the reconstruction error lead to a slight improvement of BM, compared with the general settings. For SGBM and ELAS, the performance is similar to the general settings. However, if a different trade-off is required between density and accuracy, the NSGA-II optimization can be useful to find the parameter settings.

Besides optimizing the settings of conventional algorithms, we compared multiple self-supervised finetuning methods for AnyNet. Finetuning using the disparity maps of conventional methods was found to lead to better results than finetuning using the reconstruction error. Furthermore, AnyNet seems to benefit from training on denser disparity maps. When the left portion of the left disparity map is

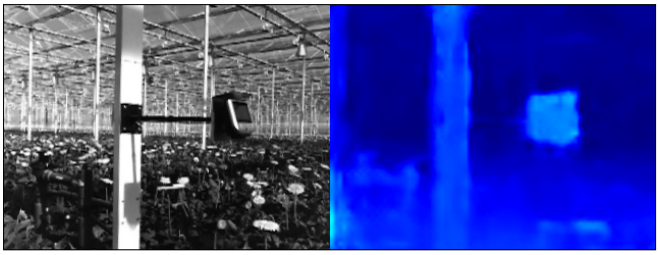


Fig. 11: A scene from the greenhouse dataset with a horizontal obstacle and the disparity map from AnyNet-SGBM-K

disregarded, finetuning using conventional disparity maps is comparable to finetuning using ground truth data. The good performance of finetuning using conventional methods could be a result of their high density compared with the ground truth. Perhaps training on ground truth can be further improved by filling the missing disparities using conventional methods.

Comparing the results of the traditional methods to the results of AnyNet we conclude that at similar density, the performance is similar. However, when ELAS interpolates low-confidence disparity values, the error drops well below that of AnyNet. Future work could investigate whether interpolation of low-confident disparity values is as beneficial to AnyNet as it is to ELAS.

References

[1] K. Konolige, "Small Vision Systems: Hardware and Implementation," in *Robotics Research*. Springer, London, 1998, pp. 203–212.

[2] H. Hirschmüller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008.

[3] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Computer Vision – ACCV 2010*. Springer Berlin Heidelberg, 2011, pp. 25–38.

[4] A. Ivanovas, A. Ostreika, R. Maskelinas, R. Damaševičius, D. Połap, and M. Woźniak, "Block matching based obstacle avoidance for unmanned aerial vehicle," in *Artificial Intelligence and Soft Computing*. Springer International Publishing, 2018, pp. 58–69.

[5] N. Krombach, D. Droeschel, and S. Behnke, "Evaluation of stereo algorithms for obstacle detection with fisheye lenses," in *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 2, no. 1W1, 2015, pp. 33–40.

[6] H. Madokoro, H. Woo, K. Sato, and N. Shimoi, "Development of Octo-Rotor UAV Prototype with Night-vision Stereo Camera System Used for Nighttime Visual Inspection," in *2019 19th International Conference on Control, Automation and Systems (ICCAS)*. IEEE Computer Society, 2019, pp. 998–1003.

[7] S. Selbek, "Multi-Objective Optimisation of Stereo Vision Algorithms," Master's Thesis, University of Oslo, 2015.

[8] S. Khamis, S. Fanello, C. Rhemann, A. Kowdle, J. Valentin, and S. Izadi, "StereoNet: Guided hierarchical refinement for real-time edge-aware depth prediction," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 573–590.

[9] A. Tonioni, F. Tosi, M. Poggi, S. Mattoccia, and L. di Stefano, "Real-time self-adaptive deep stereo," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 195–204.

[10] Y. Wang, Z. Lai, G. Huang, B. H. Wang, L. Van Der Maaten, M. Campbell, and K. Q. Weinberger, "Anytime stereo image depth estimation on mobile devices," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 5893–5900, 2019.

[11] C. A. Aguilera, C. Aguilera, C. A. Navarro, and A. D. Sappa, "Fast CNN stereo depth estimation through embedded GPU devices," *Sensors*, vol. 20, no. 11, jun 2020.

[12] W. Gan, P. K. Wong, G. Yu, R. Zhao, and C. M. Vong, "Light-weight network for real-time adaptive stereo depth estimation," *Neurocomputing*, vol. 441, pp. 118–127, jun 2021.

[13] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[15] S. N. Nasroddin, M. M. Mokji, T. Kok, A. Faiz, Z. Abidin, R. Amirulah, N. A. Nordin, S. H. Hasim, H. Zakaria, J. Hassan, and H. I. Jaafar, "Application of Particle Swarm Optimization in Optimizing Stereo Matching Algorithm's Parameters for Star Fruit Inspection System," in *Proceedings of International Conference on Recent Trends in Engineering and Technology*, 2014, pp. 11–15.

[16] E. C. Cheung, J. Wong, J. Chan, and J. Pan, "Optimization-based automatic parameter tuning for stereo vision," in *2015 IEEE International Conference on Automation Science and Engineering*. IEEE, 2015, pp. 855–861.

[17] R. Garg, V. Kumar B. G., G. Carneiro, and I. Reid, "Unsupervised CNN for single view depth estimation: Geometry to the rescue," in *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 740–756.

[18] 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 (CVPR)*, 2017, pp. 270–279.

[19] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[20] Y. Zhong, Y. Dai, and H. Li, "Self-Supervised Learning for Stereo Matching with Self-Improving Ability," *arXiv preprint arXiv:1709.00930:1709.00930*, 2017.

[21] G. Yang, H. Zhao, J. Shi, Z. Deng, and J. Jia, "SegStereo: Exploiting semantic information for disparity estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, vol. 11211 LNCS, 2018, pp. 636–651.

[22] Y. Zhong, H. Li, and Y. Dai, "Open-World Stereo Video Matching with Deep RNN," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 101–116.

[23] A. Li and Z. Yuan, "Occlusion Aware Stereo Matching via Cooperative Unsupervised Learning," in *Computer Vision – ACCV 2018*. Springer International Publishing, 2019, pp. 197–213.

[24] L. Wang, Y. Guo, Y. Wang, Z. Liang, Z. Lin, J. Yang, and W. An, "Parallax Attention for Unsupervised Stereo Correspondence Learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 4, pp. 2108–2125, sep 2022.

[25] A. Tonioni, M. Poggi, S. Mattoccia, and L. D. Stefano, "Unsupervised Adaptation for Deep Stereo," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1614–1622.

[26] M. Menze and A. Geiger, "Object Scene Flow for Autonomous Vehicles," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[27] M. Poggi and S. Mattoccia, "Learning from scratch a confidence measure," *Proceedings of the 27th British Conference on Machine Vision, BMVC*, 2016.

[28] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[29] F. Aleotti, F. Tosi, L. Zhang, M. Poggi, and S. Mattoccia, "Reversing the Cycle: Self-supervised Deep Stereo Through Enhanced Monocular Distillation," in *Computer Vision – ECCV 2020*. Springer International Publishing, 2020, pp. 614–632.

[30] F. Tosi, M. Poggi, A. Tonioni, and L. Di Stefano, "Learning confidence measures in the wild," in *British Machine Vision Conference 2017, BMVC 2017*, 2017.

[31] D. Hernandez-Juarez, A. Chacon, A. Espinosa, D. Vazquez, J. C. Moure, and A. M. Lopez, "Embedded real-time stereo estimation via Semi-Global Matching on the GPU," in *Procedia Computer Science*, vol. 80. Elsevier, 2016, pp. 143–153.

[32] Q. Chang, A. Zha, W. Wang, X. Liu, M. Onishi, and T. Maruyama, "Z2-ZNCC: ZigZag Scanning based Zero-means Normalized Cross Correlation for Fast and Accurate Stereo Matching on Embedded GPU," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, 2020, pp. 597–600.

[33] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, 2015, pp. 234–241.

[34] S. Liu, S. De Mello, J. Gu, G. Zhong, M. H. Yang, and J. Kautz, "Learning affinity via spatial propagation networks," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017, pp. 1521–1531.

[35] 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 (CVPR)*, 2016, pp. 4040–4048.

[36] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang, "On building an accurate stereo matching system on graphics hardware," in *Proceedings of the IEEE International Conference on Computer Vision*, 2011, pp. 467–474.

[37] J. Blank and K. Deb, "Pymoo: Multi-Objective Optimization in Python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

APPENDIX

TABLE A.I: AnyNet network configuration, reproduced from [10]. Note that 'conv' stands for a sequence of operations: batch normalization, rectified linear units (ReLU) and convolution. The default stride is 1.

0	Input image
2-D Unet features	
1	3×3 conv with 1 filter
2	3×3 conv with stride 2 and 1 filter
3	2×2 maxpooling with stride 2
4-5	3×3 conv with 2 filters
6	2×2 maxpooling with stride 2
7-8	3×3 conv with 4 filters
9	2×2 maxpooling with stride 2
10-11	3×3 conv with 8 filters
12	Bilinear upsample layer 11 (features) into 2x size
13	Concatenate layer 8 and 12
14-15	3×3 conv with 4 filters
16	Bilinear upsample layer 15 (features) into 2x size
17	Concatenate layer 5 and 16
18-19	3×3 conv with 2 filters
Cost volume	
20	Warp and build cost volume from layer 11
21	Warp and build cost volume from layer 15 and layer 29
22	Warp and build cost volume from layer 19 and layer 36
Regularization	
23-27	$3 \times 3 \times 3$ 3-D conv with 16 filters
28	$3 \times 3 \times 3$ 3-D conv with 1 filter
29	Disparity regression
30	Upsample layer 29 to image size: **stage 1 disparity output**
31-35	$3 \times 3 \times 3$ 3-D conv with 4 filters
36	Disparity regression: residual of stage 2
37	Upsample 36 it into image size
38	Add layer 37 and layer 30
39	Upsample layer 38 to image size: **stage 2 disparity output**
40-44	$3 \times 3 \times 3$ 3-D conv with 4 filters
45	Disparity regression: residual of stage 3
46	Add layer 44 and layer 38
47	Upsample layer 46 to image size: **stage 3 disparity output**
Spatial propagation network	
48-51	3×3 conv with 16 filters (on input image)
52	3×3 conv with 24 filters: affinity matrix
53	3×3 conv with 8 filters (on layer 47)
54	Spatial propagate layer 53 with layer 52 (affinity matrix)
55	3×3 conv with 1 filters: **stage 4 disparity output**

BM		
Parameter	**Optimization limits**	**BM-K**
`preFilterType`	[0, 1]	1
(`preFilterSize`-5)/2	[0, 42]	18
`preFilterCap`	[1, 63]	31
(`blockSize`-5)/2	[0, 10]	2
`minDisparity`	0	0
`numDisparities`	64	64
`textureThreshold`	[0, 200]	20
`uniquenessRatio`	[0, 200]	10
`disp12MaxDiff`	[0, 64]	1
`speckleRange`	[0, 5]	2
`speckleWindowSize`	[0, 100]	8

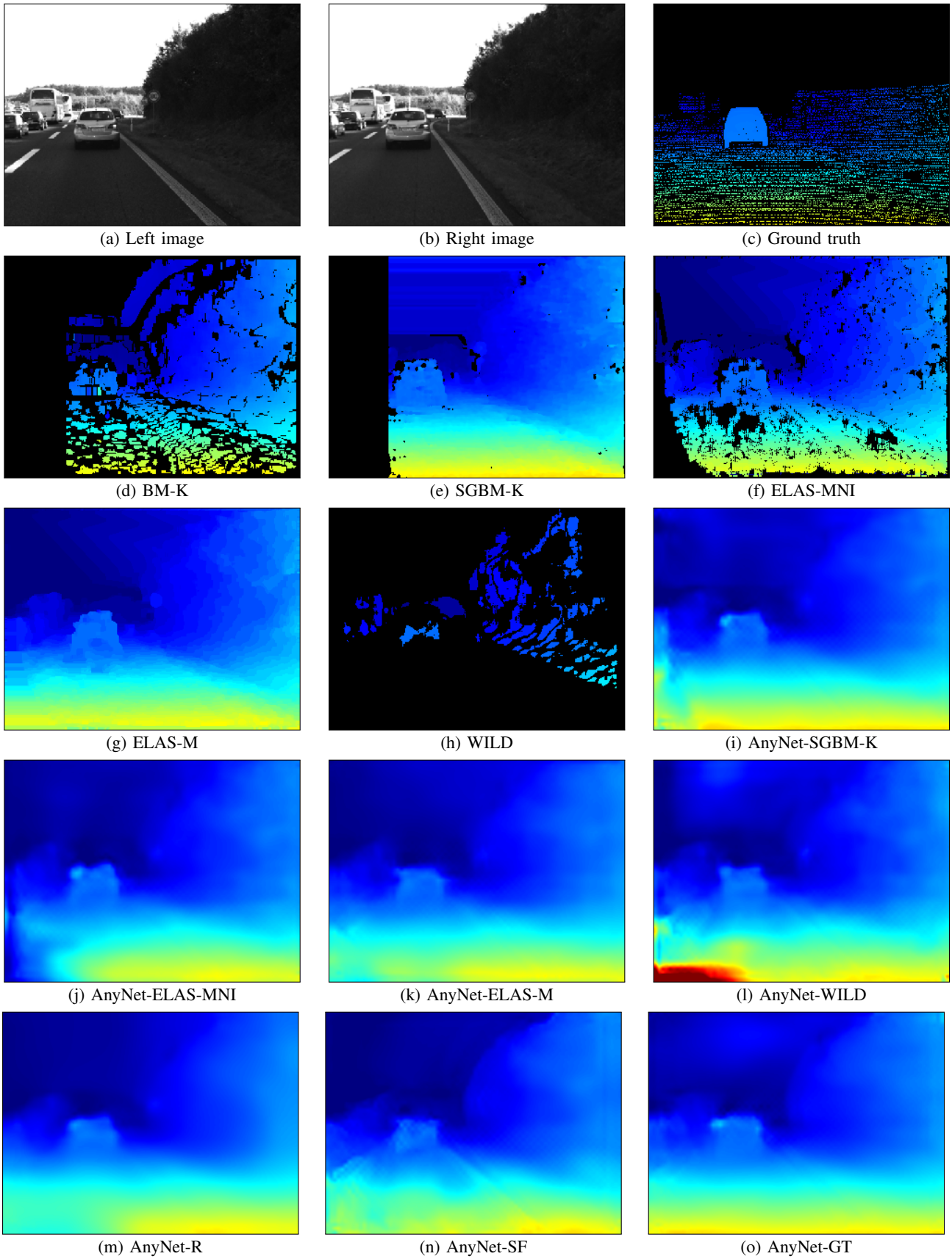
TABLE A.II: Parameter optimization limits of BM. The values between brackets indicate the lower and upper bound, respectively. Parameters for which only one value is given were fixed.

SGBM		
Parameter	**Optimization limits**	**SGBM-K**
`preFilterCap`	[15, 127]	63
`minDisparity`	0	0
`numDisparities`	64	64
(`blockSize`-1)/2	[0, 4]	1
`mode`	2	2
`P1`	[0, 500]	36
`P2`	[0, 5000]	288
`uniquenessRatio`	[0, 99]	10
`disp12MaxDiff`	[0, 64]	1
`speckleWindowSize`	[0, 200]	139
`speckleRange`	[0, 5]	1

TABLE A.III: Parameter optimization limits of SGBM. The values between brackets indicate the lower and upper bound, respectively. Parameters for which only one value is given were fixed.

ELAS		
Parameter	**Optimization limits**	**ELAS-M**
`disp_min`	0	0
`disp_max`	255	255
`support_threshold`	[50, 100]	95
`support_texture`	[0, 30]	10
`candidate_stepsize`	[1, 10]	5
`incon_window_size`	[1, 10]	5
`incon_threshold`	[0, 30]	5
`incon_min_support`	[0, 10]	5
`add_corners`	True	True
`grid_size`	[1, 50]	20
`beta`*1000	[0, 100]	20
`gamma`	[0, 30]	5
`sigma`	[1, 10]	1
`sradius`	[0, 10]	3
`match_texture`	[0, 30]	0
`lr_threshold`	[0, 64]	2
`speckle_sim_threshold`	[0, 5]	1
`speckle_size`	[0, 500]	200
`ipol_gap_width`	0	5000
`filter_median`	False	False
`filter_adaptive_mean`	False	False
`postprocess_only_left`	True	True
`subsampling`	False	False

TABLE A.IV: Parameter optimization limits of ELAS. The values between brackets indicate the lower and upper bound, respectively. Parameters for which only one value is given were fixed. The settings of ELAS-MNI are the same as ELAS-M, except that the interpolation gap width is set to 0.



(a) Left image (b) Right image (c) Ground truth

(d) BM-K (e) SGBM-K (f) ELAS-MNI

(g) ELAS-M (h) WILD (i) AnyNet-SGBM-K

(j) AnyNet-ELAS-MNI (k) AnyNet-ELAS-M (l) AnyNet-WILD

(m) AnyNet-R (n) AnyNet-SF (o) AnyNet-GT

Fig. A.1: A scene from the KITTI 2015 dataset and the disparity maps calculated by various algorithms

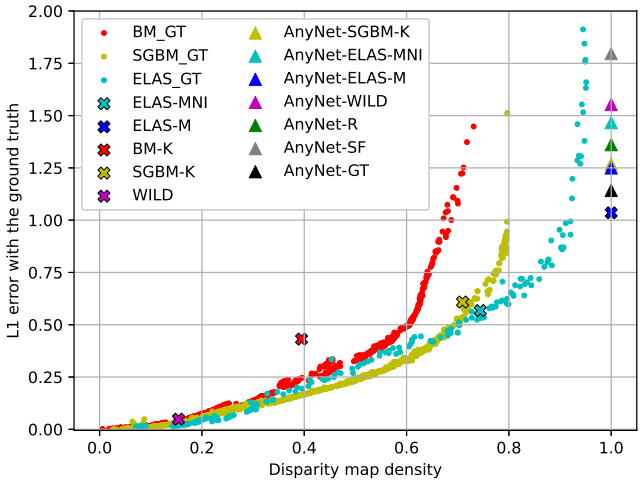


Fig. A.2: L1 error of solutions along the Pareto fronts. In these optimizations, the L1 error with the ground truth was used as a supervisor in the NSGA-II optimization.

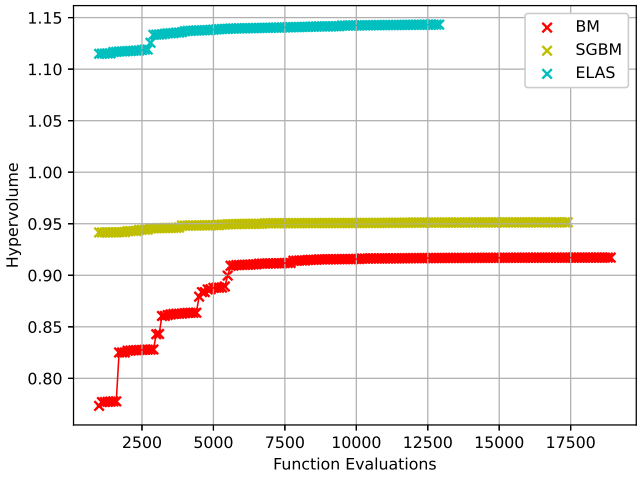


Fig. A.4: Convergence of the hypervolume when optimizing for the reconstruction error. The reference point of the hypervolume is (-0.1, 1.1) in Figure 4.

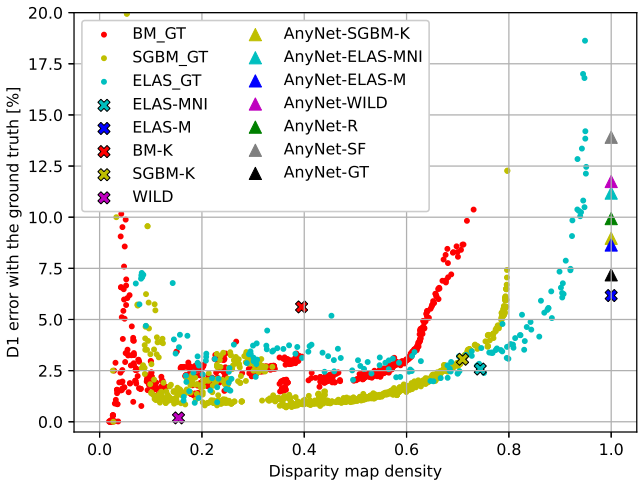


Fig. A.3: D1 error of solutions along the Pareto fronts. In these optimizations, the L1 error with the ground truth was used as a supervisor in the NSGA-II optimization.

TABLE A.V: Settings used to finetune AnyNet

Parameter	Value
maxdisp	64
loss_weights	(0.25, 0.5, 1.0, 1.0)
max_disp_list	(12, 3, 3)
epochs	300
train_batch_size	6
test_batch_size	8
with_spn	True
init_channels	1
nblocks	2
channels_3d	4
layers_3d	4
growth_rate	(4, 1, 1)
spn_init_channels	8
start_epoch_for_spn	121

II

Literature Study

1

Summary

Autonomous drones are used in many applications, such as search and rescue, law enforcement, cinematography, and agriculture. In many applications, a lightweight drone would be preferred because of its inherent safety. However, opting for a lightweight drone also has some disadvantages, such as reduced computational capabilities and reduced sensor quality and quantity. Stereo cameras are often used in lightweight drones because of their low sensor complexity, compared to active range sensors. This project uses the application of a drone in agriculture as a case study. More specifically, the design options are judged for their suitability on a drone operating in a greenhouse. However, autonomously navigating lightweight drones can be used in many other applications as well.

Scharstein and Szeliski [1] observed that stereo matching generally consists of matching cost computation, followed by cost aggregation and disparity optimization, and finalized with a disparity refinement step. The matching cost methods requiring the least computational effort are simple methods based on absolute differences. For more robustness to illumination differences, a census or rank transform [2] can be used, or methods based on normalized cross-correlation [3]. These matching costs are then aggregated. The simplest methods use a fixed window for aggregation. This is computationally lightweight but can lead to bad performance near object edges. For higher quality disparity maps, aggregation can be based on segmentation [4, 5], shiftable windows [6, 7], adaptive window sizes [8], or adaptive weights [9–11]. However, methods using fixed windows usually already produce satisfactory results, so the additional computational time for better results is often not worth it. The disparity optimization step is simple in local methods. It is based on a winner-takes-all approach, which means that the disparity with the lowest matching cost is selected. In the final step, the disparity map is refined. This includes detecting disparity values that are unreliable and removing them. The gaps can then be filled using interpolation or left empty. To make the disparity map smoother, a Gaussian blur or median filter can be applied.

The CPU-based stereo matching methods which are used most often are Block Matching (BM) [12], SemiGlobal Block Matching (SGBM) [13], and Efficient LArge-scale Stereo matching (ELAS) [14]. BM is the simplest and is based on a fixed window with the sum of absolute differences as cost computation method. SGBM expands on this by adding a penalty cost if the disparity changes between neighboring pixels. ELAS tries to reduce the computation time by first calculating disparity values for a subset of pixels. Then an initial guess is made for the remaining pixels by interpolation, and the disparity search range is set around this interpolated value.

These methods perform well only if their parameters are tuned well. Parameter tuning has been done based on ground truth data [15–17]. However, to optimize the settings for environments for which ground truth data is not available, self-supervised methods are used. These methods aim to improve the quality of the disparity map by minimizing the reprojection error. To the best of my knowledge, only Cheung et al. [18] and Nasroddin et al. [19] performed self-supervised optimization of traditional stereo matching algorithms.

Besides calculating a depth map to avoid nearby obstacles, the drone needs to track its position and

navigate towards the target location. Simultaneous Localization And Mapping (SLAM) is often used for localization and mapping. However, for a resource-limited system, it is too computationally expensive. Onboard tracking of the position can be done using Visual Inertial Odometry (VIO). Forster et al. [20, 21, 22] proposed a fast odometry system and tested it on an embedded platform with limited resources. However, according to Van Dijk [23], it suffers from too much drift. He concludes that the embedded Visual Odometer (eVO) [24] is a better option. As opposed to defining the position as a coordinate in a global frame, it is also possible to define the position relative to landmarks [25] or navigate based on route memory [26–29].

When the drone knows its own location and the location of the target, a route can be planned. This can be done based only on what is currently in sight, such as a Rapidly-exploring Random Tree (RRT) planner in the image space [30]. Unfortunately, this will not work effectively in complicated maze-like environments. A more general solution would be a 3D bug algorithm [31, 32]. However, this only works in simplified 3D environments. A route can also be planned based on memory of what the drone has seen before. A map then needs to be made and kept up to date. The route can be planned using a form of an RRT planner [33–37], or an A*-based planning algorithm [38, 39].

After the literature review, some preliminary experiments were performed on self-supervised optimization of BM, SGBM, and ELAS. The Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [40] was used to perform a multi-objective optimization. The two objectives were the reprojection error and the percentage of the disparity map with a valid disparity. The optimizations showed that SGBM performs best along the entire Pareto front. However, the optimized versions still have difficulty with repetitive and textureless areas. BM has the lowest computational cost, followed by SGBM. The computational cost of ELAS heavily depends on the parameter setting. The low computation time of SGBM and BM can be partly due to optimized CPU instructions. On a different machine, such as on board the drone, the difference in computation time could be very different, as stated by Selbek [17]. An improvement to the parameter optimization would be to take into account the computation time as well. In addition, further research can focus on improving the performance in repetitive and textureless areas.

2

Introduction

Unmanned Aerial Vehicles (UAVs) have become a popular tool in many applications, such as search and rescue, law enforcement, cinematography, and agriculture. For cheap and easy application these drones are becoming increasingly autonomous. Whereas the autonomy of early consumer drones was limited to automatic stabilization, current drones feature for example autonomous following of the operator and obstacle avoidance. However, this project will focus on the application of drones in agriculture. Here, the drone is used to monitor large areas of crops and detect diseases and pests at an early stage. This helps in preventing the further spread of the disease. Note that such a drone can be used for many different applications as well. For any application, the drone must pose little or no safety risks to people in the drone's vicinity. This can be achieved with safety measures such as the detection and avoidance of humans. However, it is more reliable to use a drone that is inherently safe due to its low weight. A lightweight drone poses less danger, even if the avoidance systems fail. Therefore, this project will focus on lightweight drones. Consequently, the drone will have low computational capabilities and simple sensors. This means that stereo vision is preferred over active range sensors such as LIDAR. The stereo camera used is a custom-made combination of two JeVois cameras and a single JeVois processor.[1] In addition to the stereo camera, the drone has an Inertial Measurement Unit (IMU).

This report aims to provide an overview of previous research related to the autonomous flight of lightweight drones. This includes a literature study on stereo vision and navigation, and a preliminary optimization of the three most promising stereo matching algorithms, SemiGlobal Block Matching (SGBM), Block Matching (BM), and Efficient LArge-scale Stereo matching (ELAS). Considering that the algorithms will run on a lightweight computer on board the drone, the methods must be efficient. The literature will therefore be evaluated with this in mind.

The drone's task package has been split into vision and navigation. The vision task is discussed in chapter 3. This chapter introduces the concept of stereo matching. It provides an overview of steps that need to be followed to arrive at a depth map. The various options for each step are discussed. The stereo matching algorithms BM, SGBM, and ELAS are explained, as well as tuning their parameters. Then, the navigation task is discussed in chapter 4. This chapter provides an overview of various methods of localization, mapping, and planning. In chapter 5 the results of some preliminary experiments on tuning the parameters of BM, SGBM, and ELAS are shown. Finally, the report ends with a conclusion in chapter 6.

[1] http://jevois.org/doc/Hardware.html

3

Vision

For safe autonomous flight, the drone needs to detect and avoid obstacles. The drone needs to have a sense of depth of the image to decide if an obstacle is nearby. This sense of depth is created through stereo matching. This chapter will first discuss some general concepts and approaches in stereo matching in section 3.1. Then in section 3.2 the most efficient CPU-based algorithms with publicly available code will be explained in detail. Finally, in section 3.3 the methods for tuning their parameters will be discussed.

3.1. Stereo matching

To obtain depth information from the two images, they need to be compared to find out which parts of the images correspond. Luckily, for finding the corresponding pixel of a pixel in one image one does not need to search the entire other image. This is explained using Figure 3.1. This image is a schematic representation of two pinhole cameras, located at O and O'. The plates represent the image planes. An object at any of the X locations will be captured on the image plane in x, thus the depth information is lost. However, on the second camera (the right one) all these locations X do not result in the same location on the image frame. They can be located at any of the x' locations. The line connecting the x' locations is called the epipolar line. An element that is photographed by the left camera and is located at x in the image, can be anywhere along the epipolar line in the right image. For each point in the left image, there is a corresponding epipolar line in the right image and vice versa. The pattern of epipolar lines is a direct result of the relative position and orientation of the two cameras. As long as the cameras stay fixed relative to each other, the pattern does not change. This is convenient in stereo matching because it means that the pattern of epipolar lines can be found in advance, and does not need to be recalculated for every image pair.

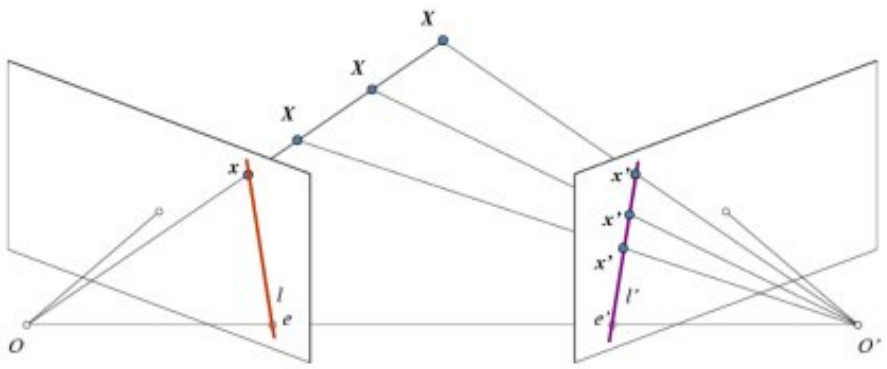


Figure 3.1: Epipolar lines, from [41]

To speed up the stereo matching algorithms, they only search along the epipolar lines for candidate matches. However, due to lens distortion, the epipolar lines are often curved. Also, the two cameras are usually not perfectly aligned, resulting in an epipolar line pattern that is shifted or rotated between the images. To correct for these effects and to align the epipolar lines, calibration and rectification need to be performed. Calibration refers to the correction of lens distortion and rectification refers to the alignment of the two cameras. The goal is to align the epipolar lines with the image rows because this makes it immediately clear that the epipolar line of a pixel in one image lies on the same image row in the other image.

The calibration and rectification process is commonly performed by taking a set of images of known geometry. Usually, this geometry is a flat chessboard pattern. The corners of the squares provide a set of points that is known to be on a rectangular grid in real life. This is used by the OpenCV [41] function `stereorectify` to build a map that specifies the translation per pixel.

After the images are calibrated and rectified the stereo correspondences can be found. Stereo matching algorithms can be categorized into local or global methods. Local methods try to optimize the matching choice for each pixel (or a subset of pixels) independently, whereas global methods try to optimize the entire disparity map using all pixels. Local methods are generally faster but less accurate than global methods. Because of the limited computational resources on the JeVois a local method is more feasible. Therefore these will be discussed in more detail. Global methods will be only be discussed briefly in subsection 3.1.5.

Scharstein and Szeliski [1] made a taxonomy of dense two-frame stereo matching algorithms. They observed that stereo matching algorithms generally consist of four steps.

1. matching cost computation
2. cost aggregation
3. disparity computation/optimization
4. disparity refinement

These steps will be further elaborated on below.

3.1.1. Matching cost computation

The first step is matching cost computation. In this step, the similarity of a pixel with its matching candidates is calculated. The matching candidates are the set of pixels within a specified disparity search range. This results in a 3-dimensional array of matching costs, the matching cost volume.

For calculating the matching cost, various methods have been tried. These are for example simple matching costs based on the pixel-wise intensity differences such as the squared intensity difference (SD), the absolute intensity difference (AD), or a truncated absolute difference (TAD). A sampling-insensitive intensity difference metric was given by Birchfield and Tomasi [42]. They compare the pixel values in one image with a linearly interpolated function of the pixel values in the other image. These methods are very common and require little computational effort, but they may not work well if there are illumination differences between the two images. As a result, corresponding pixels may have a higher matching cost than non-corresponding pixels. Illumination differences can be a result of using different cameras, using different camera settings, or due to the difference in view direction.

To make the matching cost less sensitive to illumination differences there are several options. A Census Transform (CT) [2] calculates for each pixel a bit vector describing whether or not neighboring pixels have a lower intensity than the center pixel. The similarity of pixels can then be quantified using the Hamming distance between their bit vectors. Similar to CT is the Rank Transform (RT) [2], which for each pixel assigns a value stating the number of pixels in the window that have a lower intensity than the center pixel. In other words, it is the sum of the bit vector given by the census transform, or the rank in the list of pixels in the window ordered by increasing intensity. Another option to mitigate the effect of illumination differences is Normalized Cross-Correlation (NCC) [3]. A disadvantage of NCC is that outliers in the window lead to high errors, resulting in a fattening effect. A possible solution to this is adaptive normalized cross-correlation (ANCC) [43], where the pixels in the window are weighted based on their intensity difference with the center pixel. This way, pixels across an edge are assigned

a lower weight. Hirschmüller [13] used mutual information [44] in SemiGlobal Matching (SGM) as a matching cost because of its insensitivity to recording and illumination changes.

A more extensive overview of matching cost computation methods is given in [45]. Hirschmüller and Scharstein [46] provided a comparison of cost functions when applied on stereo pairs with global intensity changes, local intensity changes, and noise. They concluded that the performance of the cost function depends on the stereo method that uses it. The Rank transform performed best for window-based stereo methods, whereas hierarchical mutual information performed best for pixel-based global and semiglobal stereo methods.

3.1.2. Cost aggregation

The next step is to aggregate the matching costs of neighboring pixels to obtain a better estimate of the similarity with the matching candidate. The purpose is to try to group pixels that belong to the same object because these are expected to have approximately the same disparity. This group of pixels is called the support window.

The support window can be constant, or it can be modified in terms of size, shape, and/or location to achieve better matching. The weight of the pixels in the support window can be modified as well. This weight can represent the confidence that a pixel has the same disparity.

The fixed window approach is the most common in fast stereo matching algorithms due to its simplicity. Summation of the absolute differences in this window results in the common Sum of Absolute Difference (SAD) metric. The advantage of a fixed window is that there is no need to calculate the optimal size and shape of the support window. A disadvantage, however, is that at the edges of objects, the support window covers both the object and the background. This means that the disparity calculation of this pixel on the edge will be disturbed by the background. In practice, this can lead to the disparities of the foreground being smeared over the background as well [47]. This is also referred to as foreground-fattening or bleeding.

A solution to bleeding would be to try to have all the pixels in the support window belong to the same object. This has been tried in several ways. A useful overview of cost aggregation methods has been provided by Tombari et al. [48]. They experimentally assess both the accuracy and computational requirements of the algorithms on a standard dataset. They divide the methods into the following categories: methods that are based on the selection of one window out of a set of windows, methods that allow for unconstrained support shapes, and methods that are based on adaptive weights.

Methods that are based on a selection of support windows out of a set of possible options are for example the shiftable window method by Arnold [6] or Bobick and Intille [7]. They shift the window such that the pixel to be matched is no longer in the middle, but on the side. This helps at the edges of objects because more of the support window's pixels will then belong to the object. Another method is the adaptive window size method proposed by Kang et al. [8]. This method ensures that larger regions with little texture get handled by larger windows. According to Tombari et al. [48] this way of cost aggregation takes a long time to compute.

Methods that allow unconstrained support shapes are for example segmentation-based methods [4, 5]. Although segmentation-based stereo matching can produce high-quality disparity maps, this often comes at a computational cost.

A method that uses adaptive weights is for example the information permeability approach by Cigla and Alatan [9, 10]. The weights of the support window are determined by the intensity similarity of the neighboring pixels and used to calculate the permeability towards that direction. This way, connected pixels with a similar color form the support window. Another method that uses adaptive weights is the one proposed by Hosni et al. [11]. Their method computes the geodesic distance from all pixels within a square support window to the center pixel. If a pixel has a path to the center pixel along which the color does not change it is given a high weight because it is then assumed to be part of the same object.

3.1.3. Disparity computation and optimization

In local matching, disparity computation is usually done on a winner-takes-all basis. The disparity with the lowest matching cost is selected. A limitation, as stated by Scharstein and Szeliski [1], is that if the left disparity map is being generated, pixels of the right image might be mapped to multiple points. In other words, uniqueness is only enforced in the reference image.

In global matching, disparity computation and optimization is a much bigger step. However, because global methods are too computationally expensive for the JeVois, the various optimization methods will not be discussed.

3.1.4. Refinement of disparities

Refinement of disparities consists of removing ambiguous disparity values, filling gaps in the disparity map, and smoothing the disparity map.

Several metrics can be used to classify a pixel as ambiguous. Common metrics are the uniqueness ratio, left-right consistency, the amount of texture, or the size of an area with dissimilar disparity. These will be explained in section 3.2. Apart from checking the left-right consistency, we can also check the temporal consistency. Cigla et al. [49] proposed a temporal fusion method based on Gaussian Mixture Models. By using temporal fusion, temporally consistent, flicker-free disparity maps with fewer errors can be obtained. It takes 0.4 seconds per frame on a 3.4 GHz i7 CPU on average on the KITTI stereo 2012 training set, which has an average resolution of 1250x350, which is still a long time considering that the entry for ELAS states 0.3 seconds on the KITTI benchmark.

Essentially, the classification of ambiguity is similar to generating a confidence map, where the pixels with low confidence are discarded. Hu and Mordohai [50] provided a quantitative evaluation of confidence measures.

The removed disparity values can either be left empty, or filled with a new estimation. This new estimation is usually an interpolation using neighboring pixels.

To make the disparity map smoother a Gaussian blur, a median filter, or an adaptive mean filter can be applied.

3.1.5. Global Methods

Global methods generally perform better than local methods but require considerably more computation time. Some examples of global optimization methods are dynamic programming, scanline optimization, graph cuts, and simulated annealing. Recently another global method is becoming more and more common. Almost all of the top-performing submissions on the Middlebury website[1] are based on neural networks [51–53]. Recent literature reviews on deep stereo matching are for example the ones by Laga et al. [54] and Zhou et al. [55]. Deep stereo matching algorithms can be run very efficiently on GPUs as they allow for parallel computing. The JeVois has a dual-core MALI-400 GPU, but this is unlikely to be powerful enough to run the large neural networks that obtain the best result. Perhaps a more compact deep learning method can be run on the JeVois. Some compact deep learning models are StereoNet [56], AnyNet [57], and MABNet_tiny [58]. AnyNet has 40K parameters and can process 1242×375 at 10-35 fps on an Nvidia Jetson TX2. MABNet_tiny has 47K parameters and a processing time of 0.11 s on a 256×512 stereo pair on four Nvidia RTX2080Ti's. StereoNet uses 1.77M parameters, has a runtime of 0.015s per frame on an Nvidia Titan X, and possibly more than twice as fast if the refinement steps are skipped.

3.2. Stereo matching algorithms

Due to the limited computational capacity on board the drone, it is relevant to ask which algorithms are lightweight enough to be run on board. This paragraph discusses some stereo vision algorithms that have been proven to work on a micro air vehicle.

[1] https://vision.middlebury.edu/stereo/eval3/

McGuire et al. [59] presented an algorithm for efficient determination of velocity and depth on an STM32F4 microprocessor. The STM32F4 has a speed of 168 MHz and 196 kB of RAM. The algorithm works by first calculating the image gradients. The gradients are summed along the image columns to obtain the edge distribution. The edge distributions of the left and the right image are matched to obtain the disparity distribution. However, this means that there is no dense depth mapping, making the method unsuitable for our application.

A modified version of the information permeability-based algorithm of Cigla and Alatan [10] was implemented by [60] on a 20g flapping wing MAV, the DelFly Explorer. The images, which have a resolution of 128x96 pixels, are processed on board a 168-MHz STM32F405 micro-controller, with only 196 kB of RAM, demonstrating the resource-efficiency of this method. Earlier, De Wagter et al. [61] introduced LongSeq, a stereo matching algorithm specifically designed to cope with the flapping motion and to run on the restricted computational resources. LongSeq performs optimization along one image line at a time. The fact that these methods can run on board a platform with such low processing speed makes them interesting options. However, higher-quality disparity maps are desired. Considering that the JeVois is capable of running for more advanced methods, this should be possible.

Barry et al. [62] achieved high-speed flight of a 664g fixed-wing aircraft using pushbroom stereo. They implemented a block matching algorithm that only searches at a single disparity, corresponding to a single distance d away from the drone. The system relies on continuous forward motion. It remembers what it has seen in previous frames, and propagates those detections to have a sense of obstacles closer than d. By only looking at a single disparity, the algorithm can be much faster, and high-speed flight is possible. Since they only look for matches at a single disparity, they cannot find the best match, instead, they take a threshold. Although high-speed flight is not an objective inside the greenhouse, a lightweight stereo matching algorithm is desired. Looking at a single disparity only may not be suitable in our low-speed application as it makes the drone blind for obstacles closer than d (and further than d). This would pose a problem when the drone changes heading while hovering, for example, to avoid an obstacle ahead. This exposes the camera to an environment that was not previously seen and obstacles closer than d cannot be detected. Therefore, the pushbroom stereo may not be suitable as-is, but limiting the search space can save time. By only looking for matches from a certain disparity and up, the drone only scans for obstacles up to a certain distance ahead, and will not spend time matching pixels corresponding to obstacles in the far distance.

Tippetts et al. [63] provided an overview of stereo vision algorithms and evaluated their suitability for resource-limited systems. However, it should be noted that it was completed in 2012, so it does not include more recent algorithms. Also, hardware has improved since then, so algorithms that were too computationally expensive then might be acceptable using today's hardware. Therefore, in 2020, Van Dijk [23] continued the work to include more recent algorithms. He compared the algorithms on the Middlebury and KITTI benchmarks for which the code is publicly available. He concludes that Efficient LArge-scale Stereo matching (ELAS) [14] and SemiGlobal Block-Matching (SGBM) [13] are still among the best performers. Therefore, this section will discuss these algorithms in detail. Block Matching (BM) [12] will be included as well, because of its simplicity and speed. In situations where ELAS and SGBM are too computationally expensive, BM could be an option. BM has previously been used by Goldberg and Matthies [64] on a system comparable to the JeVois and with the same image resolution. They report a framerate of 46 fps when using a disparity search range of 32, indicating that BM could be a feasible option on the JeVois. Depending on the computational requirements of other tasks it might be possible to use SGBM or ELAS instead. An overview of the difference in approach between these algorithms is shown in Table 3.1. The difference in performance is discussed in section 5.2.

3.2.1. Block Matching (BM)

The fastest and simplest stereo algorithm for a CPU is the SAD-based Block Matching (BM) algorithm in OpenCV, which was implemented by Konolige [12]. Kaehler and Bradski [65] provide a good explanation of how `StereoBM` works. Here follows a summary of their explanation. An overview of the input parameters is given in Table 3.2.

The algorithm works in three steps. The first step is prefiltering. In this step, the images are normalized for brightness and the texture is enhanced. Secondly, the matching costs are calculated

	BM	SGBM	ELAS
Prefiltering	Sobel filter in x-direction or normalized response filter	Sobel filter in x-direction	Sobel filter in x-direction and in y-direction
Matching cost	Absolute differences	Birchfield-Tomasi on fixed square window	Absolute differences
Cost aggregation	Summation over fixed square window	Penalty for changing disparity between neighboring pixels	Summation over fixed square window, and smoothness term based on difference with prior
Postfiltering	Uniqueness ratio Left-right consistency Speckle filter Texture threshold	Uniqueness ratio Left-right consistency Speckle filter	Uniqueness ratio Left-right consistency Speckle filter Texture threshold Gap interpolation Median filter Adaptive mean filter

Table 3.1: Overview of differences between BM, SGBM, and ELAS.

along the epipolar lines using a SAD window. The last step is postfiltering and aims to remove bad matches.

The prefiltering step can be done in one of two ways and is set by `preFilterType`. The default option `PREFILTER_NORMALIZED_RESPONSE` is to normalize the intensities in a window of size `preFilterSize`, and clip this value by [-`preFilterCap`, `preFilterCap`]. The other option, `PREFILTER_XSOBEL`, is to apply a Sobel filter in x-direction and clip its value by [-`preFilterCap`, `preFilterCap`]. Note that `preFilterSize` is not used when Sobel filtering is chosen.

Then in the second step, the matching costs are calculated by sliding a SAD window of size `blockSize` over the epipolar lines, which are assumed to be the image rows. The minimum disparity is set by `minDisparity` and is zero by default. If the axes of the cameras are parallel, this should be kept at zero. However, if the cameras are pointing slightly towards each other, distant objects might have a negative disparity. The disparity range is set by `numDisparities`. This means that the maximum disparity will be `numDisparities` + `minDisparity`.

After the matching costs have been calculated we go to the postprocessing step. If the texture of a block is below the `textureThreshold` no disparity value is given, because the block is assumed to lack distinct features, which makes matching unreliable.

To remove ambiguous matches, a match will only be accepted if the costs of the other disparities are sufficiently higher. If the match has matching cost $S(d^*)$ and the other disparities have cost $S(d)$, then the match will only be accepted if $S(d) \geq S(d^*)\left(1 + \frac{\texttt{uniquenessRatio}}{100}\right)$ for all $|d - d^*| > 1$ within the search range. In other words, the `uniquenessRatio` specifies the percentage by which the costs of all other disparities, except the adjacent, need to exceed the cost of the best disparity to accept the match. This parameter can thus have any positive value.

The next postprocessing step is to remove matches that cannot be consistently matched left-to-right and right-to-left. If the optimal disparity for a pixel at (x, y) in the left image equals d_l, and the optimal disparity of its corresponding pixel at $(x - d_l, y)$ in the right image equals d_r, then the disparity value in the left image will only be retained if $|d_r - d_l| <$ `disp12MaxDiff`.

The last step is to remove small areas with a different disparity than their surroundings. These areas are called speckles. The speckles are removed by first connecting each pixel in the image to neighbors with a similar disparity. If the disparity difference between neighboring pixels is smaller than `speckleRange`, they are connected. This means the disparity map is divided into segments of connected pixels. If a segment contains less than `speckleWindow` pixels, the disparity values in that segment are discarded. This can help to remove noise from the disparity map. However, it can have

adverse effects if the goal is to detect small obstacles.

The pixels that are removed from the disparity map are left empty in the OpenCV implementation of BM. However, it is also possible to fill the removed pixels by interpolation.

Prefiltering	
`preFilterType`	Option to choose between a normalized response filter or a Sobel filter in x-direction.
`preFilterSize`	Size of normalized response filter window
`preFilterCap`	Truncation value for prefilter response
Cost calculation	
`blockSize`	Matched block size
`minDisparity`	Minimum disparity
`numDisparities`	Disparity search range
Postprocessing	
`textureThreshold`	The disparity is only computed if the texture meets the threshold.
`uniquenessRatio`	The best match is only retained if it is sufficiently better than the second-best match.
`disp12MaxDiff`	Maximum allowed difference in the left-right consistency check
`speckleRange`	Maximum disparity variation across connected pixels in smooth disparity region
`speckleWindowSize`	Maximum size of a smooth disparity region to consider it a speckle and invalidate

Table 3.2: BM input parameters.

3.2.2. SemiGlobal Block Matching (SGBM)

The OpenCV function `StereoSGBM` is an implementation of the SemiGlobal Matching (SGM) approach by Hirschmüller [13]. The OpenCV implementation deviates from the original algorithm in a few ways. As the name implies, one of those differences is that the OpenCV implementation, SGBM, matches blocks, whereas the algorithm by Hirschmüller, SGM, matches individual pixels. `StereoSGBM` also includes some pre- and postprocessing steps from `StereoBM`. The process of `StereoSGBM` can be split into the same steps as for `StereoBM`. This section explains how these steps work, and how it differs from SGM. An overview of the input parameters of `StereoSGBM` is given in Table 3.3.

Unlike in `StereoBM`, the prefiltering step of `StereoSGBM` only offers one option: a Sobel filter in x-direction. This works the same way as in `StereoBM`. The output of the Sobel filter is clipped by [-`preFilterCap`, `preFilterCap`].

Whereas the cost calculation in [13] uses mutual information [44] as a correspondence measure, the OpenCV implementation uses the simpler Birchfield-Tomasi [42] measure. Furthermore, the original paper matched individual pixels, and the OpenCV implementation matches blocks of size `blockSize`. The algorithm calculates the full cost volume with a minimum disparity of `minDisparity` and a disparity range of `numDisparities`.

After the matching cost volume has been calculated, a penalty cost is added to each element in the cost volume if the disparity of neighboring pixels is different. To do this, the image is traversed in three (←→↑), five (←→↑↖↗), or eight directions (←→↑↖↗↓↙↘). If the disparity changes along that direction, a penalty cost is added. If the disparity changes by plus or minus 1, a penalty of P_1 is added, which equals the input parameter `P1`. If it changes by more than one, a penalty of P_2 is added. Since the disparity usually has a big change across object edges, the input parameter `P2` is adapted to the image intensity gradient, $P_2 = \frac{\texttt{P2}}{|I_\mathbf{p} - I_\mathbf{q}|}$. This ensures that the penalty is lower if a big disparity change is expected. The options with three or five directions can be done in a single pass, whereas traversing in eight directions requires a double pass, making the execution slower. The original paper by Hirschmüller [13] considered only two options, with 8 or 16 directions.

$C(\mathbf{p}, d)$ is the Birchfield-Tomasi cost value for pixel $\mathbf{p}$ with disparity d. The penalty cost along direction $\mathbf{r}$ is then defined recursively as

$$\begin{aligned} L_{\mathbf{r}}(\mathbf{p}, d) = C(\mathbf{p}, d) + \min(&L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d), \\ &L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d-1) + P_1, \\ &L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d+1) + P_1, \\ &\min_i L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, i) + P_2) - \min_k L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, k) \end{aligned} \tag{3.1}$$

The total aggregated cost $S(\mathbf{p}, d)$ is found by summing the L costs of each direction.

$$S(\mathbf{p}, d) = \sum_{\mathbf{r}} L_{\mathbf{r}}(\mathbf{p}, d) \tag{3.2}$$

Then the disparity map is made by choosing for each pixel the disparity with the lowest S cost.

Just like with `StereoBM` the match is only accepted if it is sufficiently better than the second-best match. However, the definition is slightly different. In `StereoSGBM` the best match is only accepted if its cost, $S(d^*)$, is sufficiently lower than the cost of the second-best match, $S(d)$. The match is accepted if $S(d^*) < S(d)\left(1 - \frac{\texttt{uniquenessRatio}}{100}\right)$. This means that the `uniquenessRatio` needs to be between 0 and 100.

Next, a pseudo left-right consistency check is done. The other disparity map is built using the same S cost volume. The pseudo left-right consistency check works the same as in `StereoBM`. It is called a *pseudo* left-right consistency check because the cost aggregation step is not symmetric, i.e. the S cost volume using the left image as a reference is different from the S cost volume when the right image is used as a reference. Slightly better results can be expected with a full left-right consistency check, according to [13] This means that for the right disparity map, the S cost volume is built again, but using the right image as reference. However, this takes a considerable amount of extra computation time.

Similar to `StereoBM`, speckles are again removed by setting `speckleWindowSize` and `speckleRange`.

Michael et al. [16] modified SGM to improve the quality of the result. Instead of using a constant P_1 and P_2 for each direction. They introduced $P_1(\mathbf{r})$ and $P_2(\mathbf{r})$. In other words, they made P_1 and P_2 dependent on the direction of aggregation. This makes it possible to give a different penalty if the disparity changes along some directions than along other directions. Furthermore, instead of scaling P_2 with the image gradient, they choose a different $\hat{P}_1(\mathbf{r})$ and $\hat{P}_2(\mathbf{r})$ if the image gradient exceeds a threshold. Furthermore, they introduce a scaling factor that defines the contribution of $L_{\mathbf{r}}$. These parameters are then optimized using an evolutionary algorithm. Using these modifications they achieved a reduction of the number of bad pixels by 27.5%.

3.2.3. Efficient LArge-Scale Stereo matching (ELAS)

The approach of Efficient LArge-Scale Stereo Matching (ELAS) [14] consists of three steps. First, a regular grid of support points is built and matched. Then dense matching is performed using interpolation of the disparities of surrounding support points as an initial estimate. After the dense disparity map is completed, postprocessing is done to improve the results.

The regular grid of support points is built with a step size of `candidate_stepsize` pixels. These points are matched, but only if their texture exceeds the threshold set by `support_texture`. Then, the horizontal and vertical Sobel filter response is calculated with a mask size of 3×3. For each point to be mapped, a feature vector is calculated. The feature vector is the concatenation of the horizontal and vertical Sobel filter responses on a 9×9 pixel window, resulting in a 2×9×9=162 dimensional feature vector. The matching cost between two pixels is then the sum of absolute differences of the feature vectors. For matching the support points, a disparity range of [`disp_min`, `disp_max`] is used. If the disparity of a support point is very different from the disparities of neighboring support

Prefiltering	
preFilterCap	Truncation value for prefiltered image pixels. The algorithm first computes the x-derivative at each pixel and clips its value by [-preFilterCap, preFilterCap]. The result values are passed to the Birchfield-Tomasi pixel cost function.
Cost calculation	
minDisparity	Minimum disparity
numDisparities	Maximum disparity minus minimum disparity
blockSize	Matched block size
Cost aggregation	
mode	Can be either MODE_SGBM, MODE_HH, MODE_SGBM_3WAY, or MODE_HH4. It specifies the number of aggregation directions.
P1	Penalty for disparity change of 1 along the aggregation direction
P2	Penalty for disparity change greater than one along aggregation direction
Postprocessing	
uniquenessRatio	Margin by which the cost of the best disparity should be below the cost of the other disparities
disp12MaxDiff	Maximum disparity difference in left-right consistency check
speckleWindowSize	Maximum size of smooth disparity regions to consider their noise speckles and invalidate
speckleRange	Maximum disparity variation within each connected component.

Table 3.3: StereoSGBM input parameters with descriptions from the documentation [41]

points, it is removed. This is done by counting the number of support points inside a window of size incon_window_size that have a similar disparity as the center support point. incon_threshold is the threshold for a support point to be considered similar. If the number of similar support points is below incon_min_support, it is invalidated. The setting add_corners enforces that support points are added at the image corners. The disparities of these are set to the disparities of their nearest neighbors.

The matched support points are then used to create a 2D mesh using Delaunay triangulation. Inside the triangles, a plane is fitted in the disparity space. In other words, a linear interpolation of the disparity is performed between the support points. The interpolated value at pixel $\mathbf{p}$ is $\mu(\mathbf{p})$. Based on the interpolated value, a disparity range is set for that specific pixel. The disparity search range is $\mu(\mathbf{p})$ plus or minus sradius×sigma. In addition, all disparities that occur on support points within a window of size grid_size are also added to the disparity search range.

The matching cost is a combination of the amount by which the disparity differs from the interpolated value and the sum of absolute differences between the feature vectors. The feature vectors are slightly different from the feature vectors in the support point matching. Here, the Sobel filter responses of a 5×5 window are included in the feature vector $\mathbf{f}$, resulting in a 2×5×5=50 dimensional feature vector. The matching cost $E(\mathbf{p},\mathbf{q})$ between pixel $\mathbf{p}$ in the left image and pixel $\mathbf{q}$ in the right image with disparity d is given below.

$$E(\mathbf{p},\mathbf{q}) = \texttt{beta}\,\|\mathbf{f_p} - \mathbf{f_q}\| - \log\left[\texttt{gamma} + \exp\left(-\frac{[d-\mu(\mathbf{p})]^2}{2\,\texttt{sigma}^2}\right)\right] \tag{3.3}$$

As we can see, the first term includes the sum of absolute difference of the feature vectors in the total matching cost, and its weight is given by beta. The second term includes the disparity similarity with the interpolated value. The disparity for pixel $\mathbf{p}$ is then set such that $E(\mathbf{p},\mathbf{q})$ is minimum.

This process is done for both images, and a left-right consistency check is performed, using the threshold set by lr_threshold. Speckles are again removed by setting speckle_size and speckle_sim_threshold.

Whereas ELAS uses a regular grid to select candidate support points, LS-ELAS [66, 67] improves on this by sampling them along edge segments. This results in a reduction of computation time because

Support point matching	
disp_min	Minimum disparity
disp_max	Maximum disparity
support_threshold	Maximum uniqueness ratio
support_texture	Minimum texture for support points
candidate_stepsize	Step size of regular grid on which support points are matched
incon_window_size	Window size of inconsistent support point check
incon_threshold	Disparity similarity threshold for support point to be considered consistent
incon_min_support	Minimum number of consistent support points
add_corners	Add support points at image corners with nearest neighbor disparities
grid_size	Size of neighborhood for additional support point extrapolation
Dense matching	
beta	Image likelihood parameter
gamma	Prior constant
sigma	Prior sigma
sradius	Prior sigma radius
match_texture	Min texture for dense matching
Postprocessing	
lr_threshold	Disparity threshold for left-right consistency check
speckle_sim_threshold	Similarity threshold for speckle segmentation
speckle_size	Maximum size of a speckle (small speckles get removed)
ipol_gap_width	Interpolate small gaps
filter_median	Optional median filter (approximated)
filter_adaptive_mean	Optional adaptive mean filter (approximated)
postprocess_only_left	Saves time by not postprocessing the right image
subsampling	Saves time by only computing disparities for each 2nd pixel

Table 3.4: ELAS input parameters with descriptions from the documentation [14]

a higher percentage of candidate support points can be robustly matched. They state a reduction of a factor three for the computation of support points. On the entire calculation, this results in a reduction of about 10%. This difference increases with higher resolution and becomes negligible at lower resolution.

Rahnama et al. [68] implemented ELAS on an FPGA. They use a Census Transform descriptor with Hamming Distance instead of SAD to achieve illumination invariant matching without the need for Sobel filtering in the pre-processing. This change reduces the number of computations and the memory requirements.

3.3. Parameter tuning

A prerequisite for obtaining high-quality disparity maps using the previously described stereo matching algorithms is that their parameters are tuned properly. The optimal parameter settings can depend on factors related to the imaging hardware, such as resolution, and factors related to the environment, such as lighting. Furthermore, the optimal settings are application-specific. Sometimes a different trade-off is preferred between computation time, accuracy, and the percentage of pixels with a valid disparity value.

Parameter tuning of a stereo algorithm is a high-dimensional nonlinear problem that is commonly solved either manually by trial-and-error or automatically using a grid search or more advanced methods like evolutionary algorithms. Needless to say, manual tuning is a tedious process and will often underperform automatic tuning. Therefore, this section discusses automatic parameter tuning. First, supervised parameter tuning will be discussed, followed by self-supervised parameter tuning.

3.3.1. Supervised parameter optimization

The simplest and most effective method of parameter optimization uses ground truth data to evaluate the performance of a combination of parameters. Nguyen and Ahn [15] proposed a Robust Evolutionary Algorithm (REAL) and applied it to optimize the parameters of SGBM, ELAS, and other stereo matching algorithms. The objective function is the average absolute difference between the produced disparity map and the ground truth disparity map. Compared with L-SHADE [69] and Particle Swarm Optimization (PSO) [70] they achieved faster convergence, mainly because the initial population of REAL is generated randomly around user-specified initial settings.

As mentioned in subsection 3.2.2 Michael et al. [16] modified SGM by introducing some additional parameters. They tuned these parameters using CMA-ES [71]. Again, the quality of the disparity map is quantified by comparing it with the ground truth data. More specifically, the objective function is the number of pixels in the disparity map whose difference with the ground truth exceeds 1.

Instead of optimizing a single objective, Selbek [17] performed a multi-objective optimization with NSGA-II [40] on SGBM, ELAS, and BM, and presented the Pareto fronts. The quality of the disparity maps is quantified by comparing them with ground truth data.

3.3.2. Self-supervised parameter optimization

Sometimes ground truth data is not available and an algorithm needs a self-supervised approach to judge the performance of a parameter combination. This is done by reprojecting one image onto the other image using the disparity map. The quality of the disparity map is then judged based on the similarity of the reprojection with the original image. To the best of my knowledge, this has only been done by Cheung et al. [18] and Nasroddin et al. [19].

Cheung et al. [18] applied CMA-ES to optimize the parameters of BM and update adaptively when the environment changes. First, they minimize the disparity search range while covering all important details of the scene in the depth map. Then they optimize the other parameters. They do this by minimizing a single objective function, which is a linear combination of the following sub-objectives.

- The number of pixels with a valid disparity
- The average pixel-to-pixel intensity difference between the original image and the reprojected image, normalized with the maximum intensity in the image
- The average difference between the maximum and minimum disparity within a sector of the image. This sub-objective is used to prevent noise in the disparity map.
- The computation time

They conclude that with automatic parameter tuning comparable or better depth maps can be found than with manual tuning and find that their method can quickly adapt to a changing environment.

Nasroddin et al. [19] applied PSO and Binary Particle Swarm Optimization (BPSO) to optimize the parameters of BM. The objective function is the sum of absolute differences between the original left image and the reconstructed left image. They compare the results of PSO and BPSO and conclude that PSO performs better. Unfortunately, they do not compare the result to a manually tuned version of BM.

Self-supervised optimization is not only used for parameter optimization of engineered stereo matching algorithms. It is also used for deep stereo matching algorithms. The loss function used in these algorithms can also be used as the objective function in evolutionary algorithms, used for parameter optimization.

Garg et al. [72] were the first to develop a neural network for depth estimation that was trained in a self-supervised way. They used a combination of the L1 error between the reconstruction and the original image and a disparity smoothness term penalizing gradients in the disparity map. Godard et al. [73] used a more extensive loss function. They combined the following.

- L1 error between the original image and the reconstructed image

- A structural similarity term (SSIM) [74] to reduce the effect of illumination differences. The SSIM term is a combination of the difference in luminance, contrast, and structure. The mathematical definition is given in subsection 5.1.1.
- A disparity smoothness term scaled with the image gradient. This allows high disparity gradients at locations where the image gradient is high.
- A left-right disparity consistency loss which attempts to make the left disparity map equal to the projected right disparity map.

Zhong et al. [75] had a similar approach but also included the L1 error between the gradient of the original image and the reconstructed gradient image. Furthermore, they introduce a loop consistency term, which is the L1 error between the original left image and a reconstruction of the left image, formed by reconstructing the right image first using the right disparity map and warping that image using the left disparity map to arrive back at a synthetic left image. They also add a maximum-depth heuristic loss, which minimizes the sum of all disparities. This is to train the model to favor low disparities in textureless areas.

Tonioni et al. [76] modified DispNet [77] to use self-supervised training instead of supervised training. The loss function consists of a confidence-guided loss and a smoothness term. The confidence-guided loss is the difference between DispNet's disparity map and a disparity map generated by a conventional stereo matching algorithm (SGM [13] or AD-CENSUS [2]). However, such a network would be trained to imitate the stereo algorithm's shortcomings as well. To avoid this, they use a different network to discriminate between reliable and unreliable pixels. Their smoothness term is based on the average absolute difference between the disparity of a pixel and pixels within a specified range of that pixel. Their disparity smoothness term does not consider the image gradient, and will therefore penalize a changing disparity even if it is correct.

4

Navigation

The navigation task is split into two subtasks. The first subtask is localization and mapping, i.e. to know the agent's position relative to the goal. Localization and mapping is considered one task because building a map inherently provides a position. The second subtask is motion planning. This can be done locally based on what is currently in sight, or globally by using a global map. Localization and mapping will first be discussed in section 4.1. Motion planning will then be discussed in section 4.2.

Similar to stereo matching, deep learning is also increasing in popularity in autonomous navigation. Due to the limited computational resources on the JeVois, deep learning is not considered a viable option and will therefore not be discussed any further. The interested reader is referred to Lee et al. [78] for a recent research overview of deep learning in autonomous drone navigation.

Previous literature studies on mapping and localization are for example those by Kanellakis and Nikolakopoulos [79] and Lu et al. [80].

4.1. Localization and mapping

Localization can be done in multiple ways. Pérez et al. [81] and Yuan et al. [82] presented overviews of various localization systems. For outdoor navigation, the most commonly used is GPS, but in indoor environments such as the greenhouse, the signal can be distorted, leading to inaccurate and unreliable positioning. A solution would be to build an indoor positioning system. Such a system can for example be vision-based motion capture systems such as Vicon[1] or Optitrack[2]. These use external cameras to follow the drone and triangulate its position, using images from cameras at different locations. Other positioning systems can be based on radio frequency, such as Ultrawideband (UWB), Radio Frequency Identification (RFID), Wireless Local Area Network (WLAN), or Bluetooth. These use either the signal strength or time of arrival measurements to calculate the position. However, the downside of these external systems is that a large investment is needed before a drone can be used. Therefore, this section only discusses localization methods that run on board. This allows the drone to be used in an unknown environment.

4.1.1. Mapping

The most complete form of onboard localization builds a map while navigating and determines the agent's position in that map. The advantage of this is that by remembering the surroundings, the agent can plan the shortest path to previously seen locations. Van Dijk [23] divides mapping into three classes: image-space maps, discretized space maps, and continuous space maps. The simplest form of a map would be an image-based map. This is essentially a map of the environment based solely on what is currently in sight. This can be obtained after converting the disparity map of a stereo algorithm to a

[1] https://www.vicon.com/
[2] https://optitrack.com/

depth map. The drone can then plan a route to avoid obstacles that are in sight. A more complete map would combine successive images into a single map. The benefit would be that the drone can then plan a route based on everything it has seen before, and thus plan a shorter path and avoid getting stuck in a loop. The observations can then be saved in a discretized space map or a continuous space map. The discretized space map is most common in applications that use SLAM. The surrounding is split into a collection of cells that can be free or occupied. Unfortunately, SLAM is still too computationally demanding for use on a lightweight drone. In a continuous space map, each observation is given a location coordinate. A point cloud is an example of this.

Another way of efficiently storing the observations would be using a topological map. This is used in topological SLAM [83]. The map then only saves certain locations and their relative positions. The benefit is that this is less computationally demanding. However, a disadvantage is that if two different locations are incorrectly recognized as the same, this leads to errors in the map.

4.1.2. Visual (inertial) odometry

Another common method is Visual Odometry (VO). The basic idea behind VO is to integrate the change in position and orientation. These changes are measured by comparing subsequent images from the camera stream. VO can be combined with inertial measurements from accelerometers and gyroscopes to estimate the change in position and orientation more accurately. The algorithm is then called Visual Inertial Odometry (VIO).

The problem with odometry, in general, is the error accumulation. Especially if the orientation estimation is inaccurate this can cause translations to be integrated along the wrong direction and lead to a large position error. A drift-free measurement of orientation can improve the accuracy greatly. Options for obtaining a drift-free measurement of orientation include sensing the Earth's magnetic field using magnetometers, or visual landmarks. In indoor environments the magnetic field is usually too disturbed to make it work, so a visual landmark-based orientation measurement could produce better results. Note that it does not necessarily need to be the same landmark during the entire mission. It can also be a successive selection of landmarks. Keeping track of the same landmark over multiple timesteps can at least mitigate the drift over a portion of the mission.

Forster et al. [20, 21, 22] proposed a Semi-direct Visual Odometry (SVO) system without loop-closure that is significantly faster than Simultaneous Localization And Mapping (SLAM) while achieving comparable accuracy. In [22] they present the first version, which uses a single camera. They implement it on an embedded platform (Odroid-U2, quadcore ARM Cortex A9) and achieve a framerate of 55 Hz, using two threads. Considering that the JeVois system features a comparable processor (quadcore ARM Cortex A7), similar performance can be expected. In [21] they show the extension using inertial measurements. They performed several indoor and outdoor experiments on a standard laptop (Intel i7, 2.4 GHz). On an outdoor trajectory of 300m around an office building with the end location the same as the start location they measure an end-to-end error of around 1 meter. In [20] they report results with several variations of monocular and stereo SVO without IMU. For stereo SVO the average processing time is around 4 ms per frame on a laptop with an Intel i7 processor (2.8 GHz), using two threads. This is similar to the time it takes to compute a single disparity map, as indicated in section 5.2, and could therefore be a feasible option. The tracked features could potentially form a basis for a map, allowing the drone to plan a trajectory towards a target location.

However, Van Dijk [23] tested SVO on a Parrot Bebop 2 with SLAMDunk and concluded that it suffers from too much drift. Better results were achieved with the embedded Visual Odometer (eVO) [24]. They perform stereo matching using Zero-mean Normalized Cross-Correlation (ZNCC) at low resolution first with a 3×3 window size, and then at higher resolution with a 9×9 window size. Then they track features across successive views with a KLT feature tracker [84]. The drone's pose relative to a keyframe is then calculated using a Perspective-3-Point algorithm [85, 86]. When the ratio of successfully tracked features drops below a threshold, a new keyframe is initialized. They report a processing time of 0.05s on the Kitti Odometry benchmark with an Intel Core2Duo. The implementation of Van Dijk [23] runs at around 0.15s on a SLAMDunk.

4.1.3. Non-metric methods

Whereas visual odometry provides a position coordinate in the global frame, it is also possible to calculate a position relative to landmarks seen along the route, or without a position coordinate at all. The latter would work through remembering the route and the scenes and navigate back based on what looks familiar.

Maravall et al. [87, 88] presented a navigation algorithm based on self-semantic localization and navigation based on entropic vision. The algorithm in [87] was implemented on a Parrot AR.Drone 2.0. The navigation relies on a provided visual topological map of the environment. The drone then uses the visual landmarks for localization, in order to find the target location. Although this will not work in new environments, it can be an effective method after building the topological map. Instead of using existing landmarks and adding them to a topological map, convenient landmarks can also be added to the environment to improve coverage and reliability. Effective landmarks are for example the tags presented by Olson [89].

Lambrinos et al. [25] presented the Average Landmark Vector (ALV) model. When the agent is in the home position it constructs several unit vectors to landmarks that are in sight. These are averaged to produce the average landmark vector. When the agent moves around in its environment and wants to return home it constructs the unit vectors to the landmarks and averages it. The vector pointing home is then calculated by subtracting the target ALV from the current ALV. Note that this method requires compass information (or any other reference direction) to give the vector meaning. In the indoor environment, a compass is not reliable enough, so the reference direction would have to be tracked with odometry. This makes the ALV model superfluous because odometry would already provide a position measurement and hence directly provide a homing vector. Furthermore, in the greenhouse the drone will need to navigate through corridor-like pathways where the scene frequently changes and previously seen landmarks quickly go out of sight, complicating the construction of the homing vector.

Denuelle and Srinivasan [26] presented a localization strategy based on the memorization of snapshots at regular intervals. The relative location of the snapshots is stored in memory as well. The drone can then retrace its previous path back home. Unfortunately, they do not specify the computational performance of the algorithm, but the drone is equipped with an Intel NUC computer (dual-core 2.6 GHz), which is more powerful than the JeVois computer.

A similar algorithm is visual homing based on scene familiarity [27, 28]. Baddeley et al. [27] propose an ant-inspired model for visual navigation and test it in simulation. Instead of saving all previously captured images, they train a two-layered neural network to perform familiarity discrimination. The images are only shown once to the network and the network's parameters change accordingly. This means that the memory load stays constant and does not scale with the length of the path. The input images were downscaled to 90×17 pixels, meaning that there are 1530 input parameters. The number of parameters in the second layer is equal, so it is a very lightweight network. They even state that a second layer with as little as 200 parameters can work well in many cases.

Stankiewicz and Webb [29] recently published a route-following approach that makes use of transverse oscillations to center the flight path on a learned trajectory. The image processing takes place on board an Odroid XU4. They demonstrate robust performance up to a travel distance of 30m. Furthermore, they find surprisingly robust descriptors of precise spatial locations in self-similar terrains such as a field of grass. To quantify the similarity of two views, they apply normalized correlation to the coefficients on the lowest level of the complex wavelet transform.

Although localization methods without position coordinate can be effective at returning to the base station, the lack of position coordinate can be a dealbreaker in some applications, such as when the drone is used to gather location-specific data. Location-specific data would be required when labeling an area in a greenhouse as infested by pests, for example. However, if the method is sufficiently lightweight it could be useful to assist visual odometry.

4.2. Motion planning

The second subtask of the navigation task is motion planning. Literature studies on motion planning algorithms are given in [90–92]. Motion planning can be performed with an image-space map or a map of the complete environment. Using a complete map will lead to the shortest path as a complete route can be planned, taking into account all obstacles between the current location and the destination. However, sometimes complete maps are not available and the route needs to be planned using only what is directly in sight. Minguez et al. [91] called the first global planning, and the latter local planning. Global planning typically requires a lot more memory and processing power to keep the map up-to-date than local planning. Therefore, local planning will likely be more suitable for lightweight drones with little computing capabilities.

4.2.1. Local planning

Droplet strategy

Tijmons et al. [93] demonstrated an obstacle avoidance strategy called the ”Droplet” strategy. The drone continuously checks if an obstacle is ahead and tries to keep an area ahead of the drone free of obstacles to allow for sufficient space to make a turn while maintaining the same speed. The area ahead of the drone that needs to be safe is shaped like a droplet, hence the name. The algorithm is very lightweight, but its purpose is only to avoid hitting obstacles. There is no destination that the drone is flying to, making it unsuitable for our purpose.

RRT

Matthies et al. [30] flew an Asctec Pelican in a grove of trees. They performed a C-space-like obstacle expansion on the disparity map, meaning that obstacles are expanded in the image space to take into account the size of the drone. An obstacle that is nearby is expanded more than an obstacle far away. This way, a single free pixel in the C-space expanded image represents a path just large enough for the drone to fly through. After the C-space expansion the drone checks for collision by projecting candidate 3D trajectories into the image space with a closed-loop RRT planner. A visualization they gave is shown in Figure 4.1.
In [94] they extended the field of regard to 180 by adding side-looking cameras. The depth data from all cameras is then merged into a cylindrical image space surrounding the drone, called the egocylinder. Similar to the previous work, C-space expansion is then performed on the egocylinder and a trajectory is planned. In [95] they further extended it by including full configuration flat dynamics.

The RRT planning directly onto the C-space expanded disparity map works well in environments with sparsely scattered obstacles, which take up only a small portion of the image space. If the space around the obstacle can be seen, it makes sense to directly plan a trajectory around it. In an environment such as the gerbera greenhouse (as seen on the cover image), where the main obstacles are support columns, this would be a feasible solution. In a greenhouse for tomato plants, however, the environment is much more maze-like. The drone then needs to navigate through the long aisles dividing the rows of plants, and a passage around a plant might not be immediately visible. An RRT planner based on a complete map might still be feasible, as long as storing and updating the map is feasible.

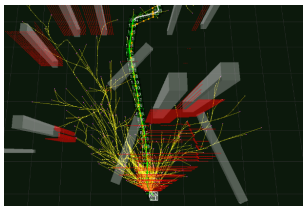


Figure 4.1: Visualization of the RRT planner in [30]

Bug algorithms

A solution that would work in a maze-like environment such as the tomato greenhouse is a Bug algorithm. Bug algorithms were first introduced by Lumelsky and Stepanov [96]. The agent starts by moving directly towards the target and reacts when it is immediately in front of a wall. The agent then starts following the wall until it can continue its path towards the target.

To shorten the path, the agent can already change its heading when it approaches a wall. This is

called a range sensor-based bug algorithm and was already mentioned in the original paper. Kamon et al. [97] further elaborated this and called their algorithm *TangentBug*. Based on *TangentBug* [97] Laubach and Burdick [98] presented the *WedgeBug* algorithm. They improved the algorithm in two ways; *TangentBug* assumes omnidirectional vision, whereas *WedgeBug* only uses a limited field of view. The algorithm was designed for planetary micro rovers. It bases its decisions on a limited field of view, which has the shape of a wedge. As is common in bug algorithms, *WedgeBug* is based upon two modes; motion-to-goal and boundary following. When in motion-to-goal mode the robot's distance to the target always decreases. If no obstacles are seen within the wedge, it moves directly towards the target ('direct mode'). If there is an obstacle it will change its heading to avoid the obstacle, while still decreasing its distance from the target ('sliding mode'). To improve efficiency, the robot can scan additional wedges to find a shortcut, if it assumes a shortcut is possible. When the robot cannot decrease its distance to the target a local minimum has been found and it switches to boundary following.

The previous algorithms were designed for rovers that avoid obstacles in 2D only. They move over the ground and can either go left or right around an obstacle. For our application, the drone has the benefit that it can utilize the third dimension as well and can avoid obstacles by going over or under the obstacles. However, this also makes the algorithm more complex. This is because now the obstacle's boundary is a 2D surface instead of a 1D curve, whereas the agent's path is still a 1D curve. In 2D, completing a loop around the obstacle will guarantee successful avoidance of the obstacle, unless the target is unreachable. In 3D, the entire surface may need to be explored to find a path towards the target. This is especially complex in concave boundaries, which are ubiquitous in indoor environments.

To solve the 3D avoidance problem, Kamon et al. [31, 32] extended their 2D *TangentBug* [97] to 3D and called it *3DBug*. It again has a motion-to-target mode and an obstacle-surface-traversal mode. In the motion-to-target mode, the robot moves directly towards the target if the target is visible. If the target is not visible, i.e. an obstacle is blocking the direct path to the target, the robot computes the shortest path to the target based on the current location and the locations of the points on the blocking contour. If none of the points on the blocking contour is closer to the target than the current position, the robot is in a local minimum and switches to obstacle-surface-traversal mode. During the traversal mode, the drone saves knowledge on the obstacle surface in the Convex Edges Graph (CEG). The robot saves the convex obstacle edges and the paths between the obstacle edges. The downside of this method is that it relies on the environment containing only polyhedral obstacles. This would be a problem in real-world applications because the obstacles are not perfect polygons.

McGuire et al. [99] presented a comparative study of bug algorithms, including the more complex methods mentioned above and simpler algorithms. They conclude that although the use of bug algorithms on resource-restricted platforms seems like an obvious choice considering the simplicity of the algorithm, many variants experience significant performance degradation when the robot is dependent on imperfect sensors generating noisy measurements. Keeping the algorithms simple is key for resiliency to odometry drift, and a robust loop detection system is crucial.

Imitation learning

Ross et al. [100] trained a controller using imitation learning based on a set of human pilot demonstrations. They implemented the system on a 420 g Parrot ARDrone and flew 3.4km with it, avoiding over 680 trees. The camera stream was sent to a separate computer for processing. The computer then sent the commands to the drone. Although they do not mention details on the type of computer they used, it seems like the algorithm is too expensive to be run on board.

4.2.2. Global planning

If the previous observations are saved in a global map, a path can be planned through this map towards a target location. This kind of path planning has been studied extensively. The algorithm can be search-based, such as A*, or sampling-based, such as RRT.

A Rapidly-exploring Random Tree (RRT) [33, 34] constructs a path to the goal by creating a tree that

is rooted at the starting location. The environment is then sampled, and if possible, connected to the tree. The downside is that if a path to a location is made, it will not be improved by adding more samples.

To improve this, RRT* was introduced by Karaman and Frazzoli [35]. When a new node is sampled in RRT*, it updates the tree within a specified radius to reduce the total distance. This ensures as long as enough samples are added, the path to the goal will converge to the optimal path.

Both RRT and RRT* generate paths to every location in the environment. However, we are only interested in the path towards the target location. Gammell et al. [36] presented Informed RRT*. After an initial solution has been found using RRT*, Informed RRT* only samples new points that have the potential to improve the solution. This makes convergence orders of magnitude faster.

Wilson et al. [37] recently developed a non-uniform sampling approach for fast and efficient path planning. The algorithm divides the environment into obstacle-free cells. The route-planner then only samples points along the edges of the obstacle-free cells to reduce the number of samplings compared to a traditional RRT* planner. The approach gives significantly better convergence compared with a traditional RRT* planner, as well as a smaller memory footprint. In a simulation on a map with several obstacles, their algorithm takes only 0.019s and needs a tree size of only 20 nodes, whereas the traditional RRT* planner needs 13.590s and 6888 nodes while generating a path of the same length after smoothing. The algorithm seems to be designed for a 2D environment, but it should be possible to extend this to 3D.

Besides RRT-based planning, A* is used as well. For example by Valenti et al. [101]. They used an A* planning algorithm to plan a route through a map generated using SLAM.

Heng et al. [102] presented a stereo-based obstacle avoidance and navigation system. They use the OpenCV implementation of the sum-of-differences block matching stereo algorithm to build a disparity map. This is then used to build a 3D occupancy map with an octree structure. The drone then plans a path towards the user-defined goal using an anytime replanning search algorithm called Anytime Dynamic A* (ADA*) [38].

An efficient path planning method called Local Tangent A* (LTA*) was proposed by [39]. The method works in three steps. First, it detects convex corners. Then only the local tangent corners are kept, the other ones are discarded. Finally a standard A* algorithm is applied. Selecting only locally tangent convex corners significantly reduces the number of samples, leading to faster convergence. It is guaranteed to find a solution if it exists. The algorithm is currently designed to work in 2D, but the writers intend to extend the approach to planning in 3D environments.

5

Preliminary experiments

To test which stereo matching algorithm works best in this application, they are tested on a dataset of images from a gerbera greenhouse. As can be seen on the cover image, the gerberas are all around the same height, with a lot of free space above. The most prevalent obstacles in this greenhouse are the supporting beams of the greenhouse. These are rather easily detected due to their size. More difficult obstacles are the spray booms because they have many thin hoses, electrical cables, and support cables. This chapter will show a comparison of stereo matching algorithms on the images taken in the greenhouse. Since the performance of the stereo matching algorithm is highly dependent on their parameter configuration, the parameters are optimized first.

5.1. Parameter optimization

When the stereo matcher is not able to find a sufficiently good match for a pixel, it is not given a disparity value. This produces areas in the disparity map where the disparity is unknown. Naturally, the number of unknown pixels needs to be minimized. At the same time, the error of the disparity map needs to be minimized. These two objectives cannot be optimized independently. Declaring fewer pixels in the disparity map as unknown can increase the error. If the number of invalid pixels is at its minimum, then the error can only be reduced by reducing the number of valid pixels. This presents a trade-off between the error and the fraction of the image with a valid disparity value. The line along which this trade-off can be made, the Pareto front, can be estimated with NSGA-II [40].

5.1.1. Objective functions

To run this parameter optimization, the objective functions need to be defined first. The first objective function is simply the fraction of the image with an invalid disparity. The second objective function aims to quantify the error of the disparity map and will be discussed below.

Since no ground truth data is available, the error of the disparity map is defined using the reconstruction error. This was also discussed in section 3.3. In this optimization, the measure of disparity error is a modified version of the loss function given by Godard et al. [73]. The error is a combination of the appearance error and the disparity smoothness error.

The appearance error is a combination of the structural similarity (SSIM) [74] and the absolute intensity difference and is defined for the pixel at (i, j) as the following.

$$E_{ap}(i,j) = \alpha_{ap}\frac{1 - \mathrm{SSIM}(i,j)}{2} + (1 - \alpha_{ap})\frac{\left|I(i,j) - \hat{I}(i,j)\right|}{255} \tag{5.1}$$

Here, $I(i,j)$ is the intensity of the pixel at (i,j) in the original left image and $\hat{I}(i,j)$ is the intensity in the reconstructed left image. α_{ap} is set at 0.25. The SSIM function is defined as follows.

$$\mathrm{SSIM}(i,j) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{5.2}$$

Here, x is the block centered at (i,j) in the original left image and y is the block centered at (i,j) in the reconstructed left image. C_1 and C_2 are constants to avoid instability when either $(\mu_x^2 + \mu_y^2)$ or $(\sigma_x^2 + \sigma_y^2)$ is close to zero. They have values of 0.0001 and 0.0009, respectively, as in [73]. μ_x is the average intensity value of the block, as stated below.

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{5.3}$$

Here N equals the number of pixels in the block. The definition of the variance σ_x^2 is the same as in the implementation by Godard et al. [73]. Note that it is slightly different from the definition in [74].

$$\sigma_x^2 = -\mu_x^2 + \frac{1}{N}\sum_{i=1}^{N} x_i^2 \qquad \sigma_{xy} = -\mu_x\mu_y + \frac{1}{N}\sum_{i=1}^{N} x_i y_i \tag{5.4}$$

Similar to Godard et al. [73], a block size of 3×3 is used.

The reconstructed left image contains some unknown pixels due to unknown disparities. The appearance error is undefined when the pixel value of an unknown pixel is needed for its calculation. This means that the appearance error is undefined if there is an unknown pixel within a 3×3 window neighborhood. The appearance loss is then defined as the average appearance error of all pixels with a defined appearance error.

The disparity smoothness error penalizes a high disparity gradient if it does not coincide with a high image gradient. The disparity smoothness error at pixel $p(i,j)$ is defined as follows.

$$E_{ds}(i,j) = |\partial_x D(i,j)| e^{-|\max(\partial_x I_l(i,j), \partial_x I_r(i,j))|} + |\partial_y D(i,j)| e^{-|\max(\partial_y I_l(i,j), \partial_y I_r(i,j))|} \tag{5.5}$$

Where D is the disparity map, I_l is the left image, and I_r is the right image. The gradient in x-direction of a pixel is defined as its intensity difference with its direct neighbor to the right. Similarly, the gradient in y-direction is defined as its intensity difference with the pixel directly below.

Sometimes a disparity change is correct even if the image gradient is low. This happens for example on slanted surfaces, where the disparity changes gradually, while the intensity stays more or less constant. To avoid penalizing this, the gradient of the disparity map is set to zero if it is below 20.

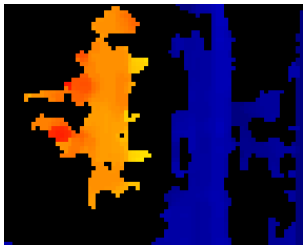


Figure 5.1: Local area of high disparity (orange), bordering with unfilled pixels (black)

Unlike the appearance error, the disparity smoothness error is defined even if its calculation includes a pixel with an unknown disparity. The reason for this is that sometimes a local area of high disparity borders an area of unknown disparity, as visualized in Figure 5.1. A disparity of 0 is used for pixels with unknown disparity. Inside areas of unknown disparity, the disparity smoothness area will always be zero, because the disparity is constant at zero. However, along the edges, the disparity smoothness error might have a positive value.

As a consequence, the disparity smoothness error penalizes a transition from a valid disparity value to an invalid disparity value if this does not coincide with a high image gradient. In general, the disparity smoothness error penalizes missing disparity values. However, occlusion is an example where a missing disparity value is actually desired. This is shown in Figure 5.2. The missing pixels in the disparity map to the left of the pole cannot be seen by both the left and right camera, and can not be matched.

Figure 5.2: An example of occlusion. The image shows the left image, the right image, and the left disparity map, respectively

BM		SGBM	
preFilterType	(0,1)	preFilterCap	(15,127)
(preFilterSize-5)/2	(0,10)	minDisparity	0
preFilterCap	(1,63)	numDisparities	64
(blockSize-5)/2	(0,10)	(blockSize-1)/2	(0,4)
minDisparity	0	mode	2
numDisparities	64	P1	(0,500)
textureThreshold	(0,200)	P2	(0,5000)
uniquenessRatio	(0,200)	uniquenessRatio	(0,99)
disp12MaxDiff	(0,64)	disp12MaxDiff	(0,64)
speckleRange	(0,5)	speckleWindowSize	(0,200)
speckleWindowSize	(0,100)	speckleRange	(0,5)

Table 5.1: Parameter optimization limits of BM and SGBM. The values between parentheses indicate the lower and upper bound, respectively. Parameters for which only one value is given were fixed.

Therefore, no disparity value is given. To avoid giving a disparity smoothness error at the transition of the background to the occluded background, the disparity smoothness error is also scaled with the image gradient of the right image, as indicated in Equation 5.5.

The disparity smoothness loss is then defined as the average disparity smoothness error over the entire image.

The error value that is used in the NSGA-II optimization is the weighted summation of the appearance loss and the disparity smoothness loss, where the weights are 0.9 and 0.1, respectively.

ELAS			
disp_min	0	sigma	(1,10)
disp_max	255	sradius	(0,10)
support_threshold	(50,100)	match_texture	(0,30)
support_texture	(0,30)	lr_threshold	(0,64)
candidate_stepsize	(1,10)	speckle_sim_threshold	(0,5)
incon_window_size	(1,10)	speckle_size	(0,500)
incon_threshold	(0,30)	ipol_gap_width	0
incon_min_support	(0,10)	filter_median	False
add_corners	True	filter_adaptive_mean	False
grid_size	(1,50)	postprocess_only_left	True
beta*1000	(0,100)	subsampling	False
gamma	(0,30)		

Table 5.2: Parameter optimization limits of ELAS. The values between parentheses indicate the lower and upper bound, respectively. Parameters for which only one value is given were fixed.

5.1.2. Parameter optimization

Out of the dataset of 478 images, 59 images were selected while the rest was discarded. These images contain challenging scenarios with thin objects, and large areas of little or repeating texture. Taking the full dataset would result in too much computation time. In addition, most images in the dataset contain no nearby obstacles. To train the algorithms to perform well in the presence of obstacles, a substantial number of images need to have nearby obstacles. Similarly, to draw a conclusion on the ability of the algorithms to detect obstacles, the test data needs to contain sufficient obstacles. Out of the 59 images, 40 images were used as training for the optimization, and 19 were used as test data. Selecting algorithms with nearby obstacles does not result in an underrepresentation of objects far away, because the images with obstacles still contain a background. This way, both nearby and faraway objects are represented in the dataset.

The training data consists of approaches to a beam, a cable, a wall, a device with horizontal support, and a spray boom. These are shown in Figure 5.3. The test data consists of approaches to a cable, a marker with a thin support pole, and a wall. These are shown in Figure 5.4. The approach to the wall is the same as for the training data. To avoid training and testing on the same images, the odd-numbered images are used for training, and the even-numbered images are used for testing. Ideally, the test data contains an approach to a different wall than the training data, but the dataset only contains one approach to a wall.

The NSGA-II optimization was run using the Pymoo implementation [103], with an initial population of 1000 and 100 offspring. The optimization is terminated when the change in the objective space drops below 0.0025 over the last 30 generations.This is evaluated every 5 generations.

The optimization only uses integer parameters, within a range specified in Table 5.1 and Table 5.2. Table 5.1 shows that the disparity range for BM and SGBM is kept fixed at (0,64). The lower limit of zero is set because objects in the far distance were found to have a disparity of zero. The maximum disparity is set constant at 64 because it specifies the operational limits in terms of how close the drone can fly to obstacles and still detect them. In other words, it is a design trade-off between the operational limits and the computation time of the disparity map, rather than a parameter affecting the quality of the disparity map. As can be seen in Table 5.2, the disparity search range for ELAS is set at (0,255), as in the default settings. It was not changed to (0,64), as with BM and SGBM, because it only has a very limited effect on the computation time, as opposed to BM and SGBM, where the computation time is very sensitive to the disparity search range. The mode of SGBM is set fixed at 2. This corresponds to the mode where the image is traversed in three directions. It is chosen because this is the fastest method. For ELAS, `add_corners` is set fixed to True, because it increases the number of pixels with a valid disparity. The postprocessing methods that are not included in BM and SGBM are kept off, to keep a fair comparison between the three algorithms. Subsampling is set off because it reduces the quality of the input images, which is not desired because it would result also result in an unfair comparison.

5.2. Results

The resulting Pareto fronts are shown in Figure 5.5. The crosses indicate the performance using manual tuning. The manual tuning is based on the settings used for the KITTI entry for BM and SGBM, and on the default settings of ELAS.

Since BM and SGBM exclude the left part of the left disparity map, a disparity map with only valid disparity values will never be achieved. The disparity is declared unknown if the full disparity range cannot be evaluated. This means that for a disparity search range of (0, 64) and an image width of 320, 20% of the image will always be declared unknown, and 80% is the maximum fraction of the image that will have a valid disparity value. The fraction valid disparity, as shown in the plots, is the fraction of valid disparity values in this part of the image. In other words, the fraction value is divided by 0.8.

Figure 5.7 and Figure 5.8 show the average time for the computation of the disparity map over the test dataset. The time was measured on a laptop with an Intel i7 processor, using a single core. To avoid giving BM and SGBM an unfair advantage compared to ELAS, since they only calculate 80% of the

Figure 5.3: Obstacles approached in the training data

Figure 5.4: Obstacles approached in the test data

disparity map, their time was divided by 0.8.

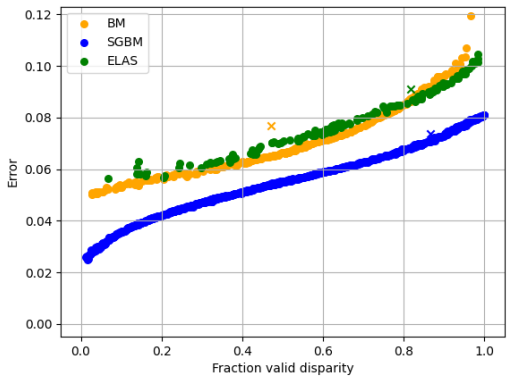


Figure 5.5: Error versus the fraction of the disparity map with a valid disparity value. The dots show the points along the Pareto front as found using NSGA-II. The crosses show the performance of the baseline configuration.

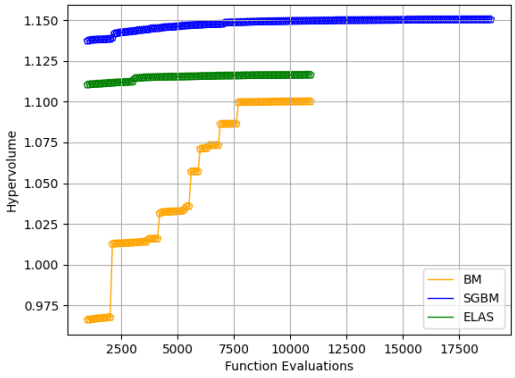


Figure 5.6: Convergence of the optimization, shown using the hypervolume of the objective space (Figure 5.5) with (-0.1,1.1) as the reference point. Since the reference point is arbitrarily chosen, the absolute value of the hypervolume is meaningless. However, the change and the difference in hypervolume are relevant.

As can be seen from Figure 5.5, SGBM outperforms ELAS and BM across the entire Pareto front. Furthermore, the optimizations slightly improve the performance of the manually tuned settings for ELAS and SGBM, and more considerably for BM.

Visual inspection of the disparity maps produced by the solutions along the Pareto front of SGBM shows that small details such as cables are generally detected well. However, repetitive areas with little texture, such as the walls, produce problems. Interestingly, the version of SGBM with hand-tuned settings performs better on these images, as visualized in Figure 5.9a. In images where the version with hand-tuned settings has an error, all the solutions along the Pareto front have the same error, or worse, as visualized in Figure 5.9b. This is likely because these errors are not recognized by the reprojection error metric, and thus not penalized. It is questionable whether a local method will ever consistently perform better on such challenging images. After all, the window frames are individually all extremely similar. Due to lighting differences between the left and right image, another window frame might appear more similar than the actual corresponding frame. This makes it extremely difficult for

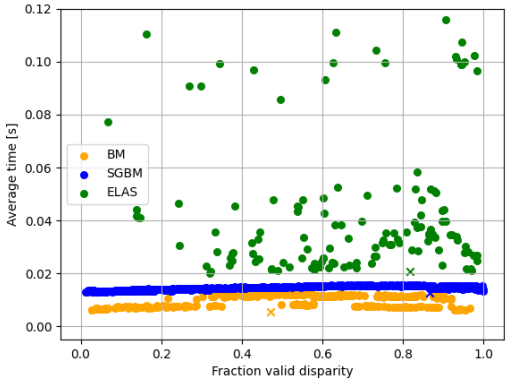


Figure 5.7: Average time for the computation of the disparity map on the test dataset, along the Pareto front. The crosses show the performance of the baseline configuration.

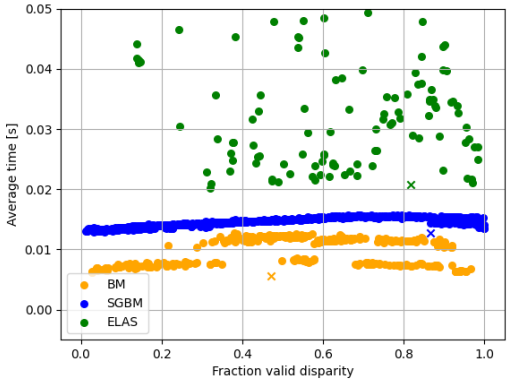
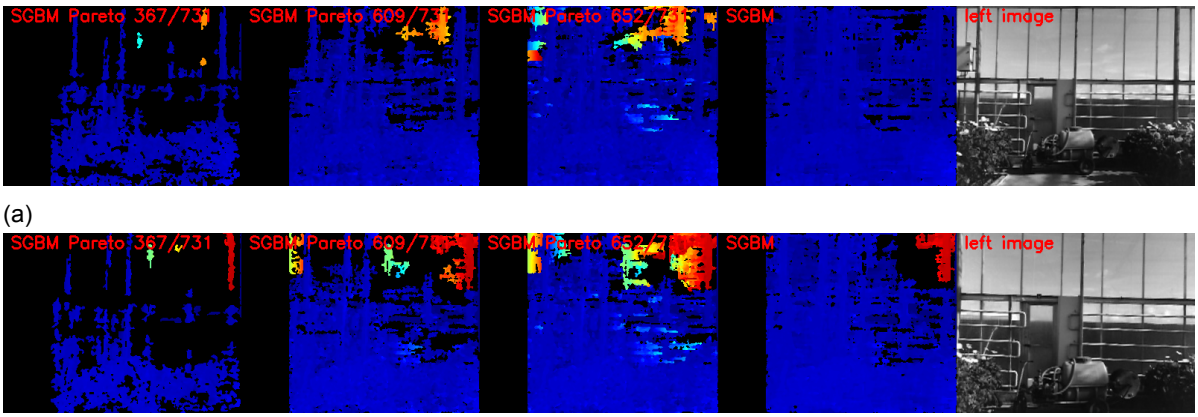


Figure 5.8: Zoomed-in version of Figure 5.7



(a)

(b)

Figure 5.9: Disparity maps of two challenging images of the wall. The three leftmost images are solutions of the Pareto front with an increasing percentage of valid disparity values. The fourth image is the disparity map generated using hand-tuned settings. The right image shows the image of the wall.

a local method to find the correct match. Perhaps this can only be improved upon by using a global method.

BM performs worse on these textureless areas. Also, the disparity maps are less detailed but it has the benefit that it has the lowest computational requirements.

The solutions along the Pareto front of ELAS still vary greatly. Both in terms of computational time and quality of the disparity map, although the latter is not visible from the plots. This is illustrated in Figure 5.10. As can be seen from the image, the quality of the disparity map can go up or down when traversing the Pareto front.

The average times to calculate a disparity map for the solutions along the Pareto front are shown in Figure 5.7 and Figure 5.8. The parameter configuration of ELAS has a much bigger impact on the computation time than BM and SGBM. As can be expected, BM is the fastest. It is remarkable, however, that even the fastest versions of ELAS are slower than SGBM. With the hand-tuned settings of ELAS, it takes about 21 ms to calculate the disparity map. Figure 5.11 shows a breakdown of the execution time of ELAS. The parameters `sigma` and `sradius` were set to 1 and 2, respectively. This means that for most pixels, the disparity search range was only plus or minus 2, centered around the interpolated value. Even after completing the interpolation, ELAS spends about 14 ms on calculating the disparity map, as can be seen in Figure 5.11. SGBM, on the other hand, has a disparity search range of 64 and needs only about 12 ms for the full calculation. This could in part be because BM and SGBM make

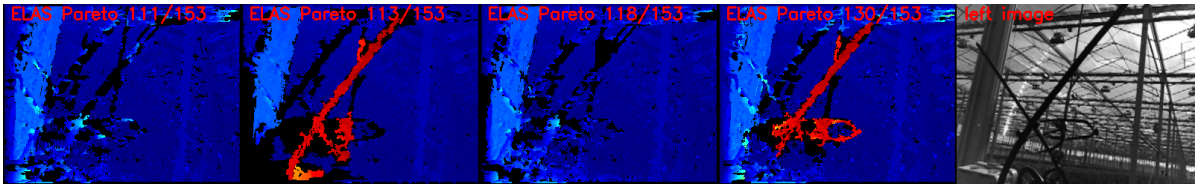


Figure 5.10: Disparity maps of four solutions along the Pareto front of ELAS, with an increasing percentage of valid disparity.

use of optimized CPU instructions. On a different CPU architecture, however, SGBM and BM could become a lot slower if the optimization is not effective, as was experienced by Selbek [17]. With the right CPU optimization, ELAS potentially becomes much faster.

Looking at the convergence of the hypervolume in Figure 5.6, it seems like BM could be further improved when letting the optimization run longer.

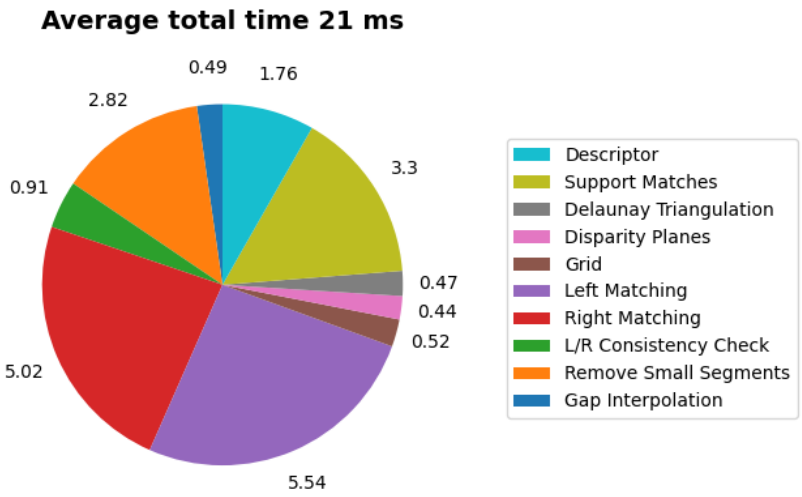


Figure 5.11: Breakdown of ELAS disparity map generation.

6

Conclusion

Many different approaches for stereo matching exist. Stereo matching algorithms have traditionally been designed for use on CPUs. However, recent developments in stereo matching have been more focused on deep learning methods on GPUs. For application on the JeVois system, a CPU-based method is more suitable. The fastest CPU-based methods are SGBM, BM, and ELAS. Their performance is highly dependent on their parameter settings. The parameter settings have previously been optimized in a supervised manner using ground truth data, or in a self-supervised manner using the reprojection error. Previous research on self-supervised optimization of stereo matching on resource-limited systems has been very limited, so this presents an opportunity for future research.

A lot of research has been performed on autonomous navigation on drones. SLAM is often used, but it is too computationally demanding for use on a JeVois. Positioning based on visual odometry might be an option, but its accuracy and computational requirements remain to be tested. As for path planning, simple image-based planning is the least computationally demanding but does not guarantee the fastest path. A bug algorithm would be a computationally lightweight solution, but these have so far only been designed for 2D environments or simplified 3D environments. Possible improvements to the current state of the art would be the development of a bug algorithm designed for use in realistic 3D environments. Another option is global path planning. This would lead to shorter paths, but it has the disadvantage that a map needs to be made and kept up to date. Further studies could focus on keeping a map up to date on a system with limited computational resources.

Preliminary experiments on self-supervised optimization of SGBM, BM, and ELAS indicate that SGBM is most suitable for stereo matching on a lightweight drone. No matter what trade-off is made between the percentage of valid disparity values in the disparity map and the error, SGBM always outperforms BM and ELAS. However, the experiments also showed that the problems with repetitive and textureless areas that were present in the manually tuned version persist after optimizing the parameters. Further research needs to be done to improve in these areas.

Bibliography

[1] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, volume 47(1-3):7–42, 2002.

[2] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In *Computer Vision — ECCV '94*, pages 151–158. Springer Berlin Heidelberg, 1994.

[3] Olivier Faugeras, Bernard Hotz, Herve Mathieu, Thierry Vieville, Zhengyou Zhang, Pascal Fua, Eric Theron, Laurent Moll, Gerard Berry, Jean Vuillemin, Patrice Bertin, and Catherine Proy. Real time correlation-based stereo: algorithm, implementations and applications. Technical report, 1993.

[4] Mark Gerrits and Philippe Bekaert. Local stereo matching with segmentation-based outlier rejection. *Third Canadian Conference on Computer and Robot Vision, CRV 2006*, volume 2006, 2006.

[5] Zhi Gang Zheng and Zeng Fu Wang. Region based stereo matching algorithm using cooperative optimization. *Zidonghua Xuebao/ Acta Automatica Sinica*, volume 35(5):469–477, 2009.

[6] R. David Arnold. Automated Stereo Perception. Technical Report Technical Report AIM-351, Artificial Intelligence Laboratory, Stanford University, 1983.

[7] Aaron F. Bobick and Stephen S. Intille. Large occlusion stereo. *International Journal of Computer Vision*, volume 33(3):181–200, 1999.

[8] Sing Bing Kang, Richard Szeliski, and Jinxiang Chai. Handling occlusions in dense multi-view stereo. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I. 2001.

[9] Cevahir Cigla and A. Aydin Alatan. Efficient Edge-Preserving Stereo Matching. pages 696–699, 2011.

[10] Cevahir Cigla and A. Aydin Alatan. Information permeability for stereo matching. *Signal Processing: Image Communication*, volume 28(9):1072–1088, 2013.

[11] Asmaa Hosni, Michael Bleyer, Margrit Gelautz, and Christoph Rhemann. Local Stereo Matching Using Geodesic Support Weights. *Science*, (1):2093–2096, 2009.

[12] Kurt Konolige. Small Vision Systems: Hardware and Implementation. In *Robotics Research*, pages 203–212. Springer, London, 1998.

[13] Heiko Hirschmüller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 30(2):328–341, 2008.

[14] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Computer Vision – ACCV 2010*, pages 25–38. Springer Berlin Heidelberg, 2011.

[15] Phuc Hong Nguyen and Chang Wook Ahn. Parameter selection framework for stereo correspondence. *Machine Vision and Applications*, volume 31(4), 2020.

[16] Matthias Michael, Jan Salmen, Johannes Stallkamp, and Marc Schlipsing. Real-time stereo vision: Optimizing Semi-Global Matching. In *IEEE Intelligent Vehicles Symposium, Proceedings*, Iv, pages 1197–1202. IEEE, 2013.

[17] Stian Selbek. *Multi-Objective Optimisation of Stereo Vision Algorithms*. Master's thesis, University of Oslo, 2015.

[18] Ernest C.H. Cheung, Jiali Wong, Joceyln Chan, and Jia Pan. Optimization-based automatic parameter tuning for stereo vision. In *2015 IEEE International Conference on Automation Science and Engineering*, pages 855–861. IEEE, 2015.

[19] Saidatul Nizan Nasroddin, Musa Mohd Mokji, Tan Kok, Amar Faiz, Zainal Abidin, Rahman Amirulah, Nur Anis Nordin, Saipol Hadi Hasim, Hamzah Zakaria, Jefery Hassan, and Hazriq Izzuan Jaafar. Application of Particle Swarm Optimization in Optimizing Stereo Matching Algorithm's Parameters for Star Fruit Inspection System. In *Proceedings of International Conference on Recent Trends in Engineering and Technology*, pages 11–15. 2014.

[20] Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems. *IEEE Transactions on Robotics*, volume 33(2):249–265, 2017.

[21] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-Manifold Preintegration for Real-Time Visual-Inertial Odometry. *IEEE Transactions on Robotics*, volume 33(1):1–21, 2017.

[22] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22. IEEE, 2014.

[23] Tom van Dijk. *Self-Supervised Learning for Visual Obstacle Avoidance*. Micro Air Vehicle Lab (MAVLab), TU Delft, 2020.

[24] Martial Sanfourche, Vincent Vittori, and Guy Le Besnerais. EVO: A realtime embedded stereo odometry for MAV applications. *IEEE International Conference on Intelligent Robots and Systems*, pages 2107–2114, 2013.

[25] Dimitrios Lambrinos, Ralf Möller, Thomas Labhart, Rolf Pfeifer, and Rüdiger Wehner. Mobile robot employing insect strategies for navigation. *Robotics and Autonomous Systems*, volume 30(1):39–64, 2000.

[26] Aymeric Denuelle and Mandyam V. Srinivasan. A sparse snapshot-based navigation strategy for UAS guidance in natural environments. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3455–3462, 2016.

[27] Bart Baddeley, Paul Graham, Philip Husbands, and Andrew Philippides. A model of ant route navigation driven by scene familiarity. *PLoS Computational Biology*, volume 8(1):1–16, 2012.

[28] Gerald J.J. Van Dalen, Kimberly N. McGuire, and Guido C.H.E. De Croon. Visual Homing for Micro Aerial Vehicles Using Scene Familiarity. *Unmanned Systems*, volume 6(2):119–130, 2018.

[29] J. Stankiewicz and B. Webb. Looking down: A model for visual route following in flying insects. *Bioinspiration and Biomimetics*, volume 16(5):055007, 2021.

[30] Larry Matthies, Roland Brockers, Yoshiaki Kuwata, and Stephan Weiss. Stereo vision-based obstacle avoidance for micro air vehicles using disparity space. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3242–3249. 2014.

[31] Ishay Kamon, Elon Rimon, and Ehud Rivlin. Range-sensor based navigation in three dimensions. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 1:163–169, 1999.

[32] Ishay Kamon, Elon Rimon, and Ehud Rivlin. Range-Sensor-Based Navigation in Three- Dimensional Polyhedral Environments. *The International Journal of Robotics Research*, volume 20(1):6–25, 2001.

[33] Steven M. LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. volume 129(98-11), 1998.

[34] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, volume 20(5):378–400, 2001.

[35] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. In *International Journal of Robotics Research*, volume 30, pages 846–894. SAGE PublicationsSage UK: London, England, 2011.

[36] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed RRT□: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2997–3004. Institute of Electrical and Electronics Engineers Inc., 2014.

[37] James P. Wilson, Zongyuan Shen, and Shalabh Gupta. A Non-uniform Sampling Approach for Fast and Efficient Path Planning. 2021.

[38] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *ICAPS 2005 - Proceedings of the 15th International Conference on Automated Planning and Scheduling*, pages 262–271. 2005.

[39] Muhammad Mateen Zafar, · Muhammad, Latif Anjum, and Wajahat Hussain. LTA*: Local tangent based A* for optimal path planning. *Autonomous Robots*, volume 45:209–227, 2021.

[40] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, volume 6(2):182–197, 2002.

[41] Gary Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[42] Stan Birchfield and Carlo Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 20(4):401–406, 1998.

[43] Seok Heo Yong, Mu Lee Kyoung, and Uk Lee Sang. Illumination and camera invariant stereo matching. *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008.

[44] Paul Viola and William M. Wells. Alignment by Maximization of Mutual Information. *International Journal of Computer Vision*, volume 24(2):137–154, 1997.

[45] Richard Szeliski. *Computer Vision: algorithms and applications*. Springer Science & Business Media, 2010.

[46] Heiko Hirschmüller and Daniel Scharstein. Evaluation of cost functions for stereo matching. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[47] Nils Einecke and Julian Eggert. A multi-block-matching approach for stereo. *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 585–592, 2015.

[48] Federico Tombari, Stefano Mattoccia, Luigi Di Stefano, and Elisa Addimanda. Classification and evaluation of cost aggregation methods for stereo correspondence. *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008.

[49] Cevahir Cigla, Roland Brockers, and Larry Matthies. Gaussian mixture models for temporal depth fusion. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 889–897, 2017.

[50] Xiaoyan Hu and Philippos Mordohai. A quantitative evaluation of confidence measures for stereo vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 34(11):2121–2133, 2012.

[51] Vladimir Tankovich, Christian Häne, Yinda Zhang, Adarsh Kowdle, Sean Fanello, and Sofien Bouaziz Google. HITNet: Hierarchical Iterative Tile Refinement Network for Real-time Stereo Matching. Technical report, 2021.

[52] Gengshan Yang, Joshua Manela, Michael Happold, and Deva Ramanan. Hierarchical deep stereo matching on high-resolution images. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2019-June, pages 5510–5519. 2019.

[53] Xiao Song, Xu Zhao, Liangji Fang, Hanwen Hu, and Yizhou Yu. EdgeStereo: An Effective Multi-task Learning Network for Stereo Matching and Edge Detection. *International Journal of Computer Vision*, volume 128(4):910–930, 2020.

[54] Hamid Laga, Laurent Valentin Jospin, F. Boussaid, and Mohammed Bennamoun. A Survey on Deep Learning Techniques for Stereo-based Depth Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 8828(c):1–1, 2020.

[55] Kun Zhou, Xiangxi Meng, and Bo Cheng. Review of Stereo Matching Algorithms Based on Deep Learning. *Computational Intelligence and Neuroscience*, volume 2020, 2020.

[56] Sameh Khamis, Sean Fanello, Christoph Rhemann, Adarsh Kowdle, Julien Valentin, and Shahram Izadi. StereoNet: Guided hierarchical refinement for real-time edge-aware depth prediction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 573–590. 2018.

[57] Yan Wang, Zihang Lai, Gao Huang, Brian H. Wang, Laurens Van Der Maaten, Mark Campbell, and Kilian Q. Weinberger. Anytime stereo image depth estimation on mobile devices. *2019 International Conference on Robotics and Automation (ICRA)*, pages 5893–5900, 2019.

[58] Jiabin Xing, Zhi Qi, Jiying Dong, Jiaxuan Cai, and Hao Liu. MABNet: A Lightweight Stereo Network Based on Multibranch Adjustable Bottleneck Module. In *Computer Vision – ECCV 2020*, pages 340–356. Springer International Publishing, 2020.

[59] Kimberly McGuire, Guido De Croon, Christophe De Wagter, Karl Tuyls, and Hilbert Kappen. Efficient Optical Flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomous Pocket Drone. *IEEE Robotics and Automation Letters*, volume 2(2):1070–1076, 2017.

[60] Sjoerd Tijmons, Christophe De Wagter, Bart Remes, and Guido de Croon. Autonomous door and corridor traversal with a 20-gram flapping wing MAV by Onboard Stereo Vision. *Aerospace*, volume 5(3):69, 2018.

[61] C. De Wagter, S. Tijmons, B. D.W. Remes, and G. C.H.E. De Croon. Autonomous flight of a 20-gram Flapping Wing MAV with a 4-gram onboard stereo vision system. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4982–4987, 2014.

[62] Andrew J. Barry, Peter R. Florence, and Russ Tedrake. High-speed autonomous obstacle avoidance with pushbroom stereo. *Journal of Field Robotics*, volume 35(1):52–68, 2018.

[63] Beau Tippetts, Dah Jye Lee, Kirt Lillywhite, and James Archibald. Review of stereo vision algorithms and their suitability for resource-limited systems. *Journal of Real-Time Image Processing*, volume 11(1):5–25, 2016.

[64] Steven B. Goldberg and Larry Matthies. Stereo and IMU assisted visual odometry on an OMAP3530 for small robots. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 169–176, 2011.

[65] Adrian Kaehler and Gary Bradski. *Learning OpenCV 3: computer vision in C++ with the OpenCV library*. " O'Reilly Media, Inc.", 2016.

[66] Radouane Ait Jellal, Manuel Lange, Benjamin Wassermann, Andreas Schilling, and Andreas Zell. LS-ELAS: Line segment based efficient large scale stereo matching. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 146–152, 2017.

[67] Radouane Ait Jellal. Stereo vision and mapping with aerial robots. 2020.

[68] Oscar Rahnama, Duncan Frost, Ondrej Miksik, and Philip H.S. Torr. Real-Time Dense Stereo Matching with ELAS on FPGA-Accelerated Embedded Devices. *IEEE Robotics and Automation Letters*, volume 3(3):2008–2015, 2018.

[69] Ryoji Tanabe and Alex S. Fukunaga. Improving the search performance of SHADE using linear population size reduction. *Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014*, pages 1658–1665, 2014.

[70] James Kennedy and Russell Eberhart. Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4:1942–1948, 1995.

[71] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, volume 9(2):159–195, 2001.

[72] Ravi Garg, Vijay Kumar B. G., Gustavo Carneiro, and Ian Reid. Unsupervised CNN for single view depth estimation: Geometry to the rescue. In *Computer Vision – ECCV 2016*, pages 740–756. Springer International Publishing, 2016.

[73] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 270–279. 2017.

[74] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, volume 13(4):600–612, 2004.

[75] Yiran Zhong, Yuchao Dai, and Hongdong Li. Self-Supervised Learning for Stereo Matching with Self-Improving Ability. *arXiv preprint arXiv:1709.00930:1709.00930*, 2017.

[76] Alessio Tonioni, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. Unsupervised Adaptation for Deep Stereo. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1614–1622. 2017.

[77] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas 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 (CVPR)*, pages 4040–4048. 2016.

[78] Thomas Lee, Susan McKeever, and Jane Courtney. Flying free: A research overview of deep learning in drone navigation autonomy. *Drones*, volume 5(2):52, 2021.

[79] Christoforos Kanellakis and George Nikolakopoulos. Survey on Computer Vision for UAVs: Current Developments and Trends. *Journal of Intelligent and Robotic Systems: Theory and Applications*, volume 87(1):141–168, 2017.

[80] Yuncheng Lu, Zhucun Xue, Gui Song Xia, and Liangpei Zhang. A survey on vision-based UAV navigation. *Geo-Spatial Information Science*, volume 21(1):21–32, 2018.

[81] M. C. Pérez, D. Gualda, J. De Vicente, J. M. Villadangos, and J. Ureña. Review of UAV positioning in indoor environments and new proposal based on US measurements. *CEUR Workshop Proceedings*, volume 2498:267–274, 2019.

[82] Shenghai Yuan, Han Wang, and Lihua Xie. Survey on Localization Systems and Algorithms for Unmanned Systems. *Unmanned Systems*, volume 9(2):129–163, 2021.

[83] Jaime Boal, Álvaro Sánchez-Miralles, and Álvaro Arranz. Topological simultaneous localization and mapping: A survey. *Robotica*, volume 32(5):803–821, 2014.

[84] Jianbo Shi and Carlo Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600. 1994.

[85] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, volume 24(6):381–395, 1981.

[86] Shinji Umeyama. Least-Squares Estimation of Transformation Parameters Between Two Point Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 13(4):376–380, 1991.

[87] Darío Maravall, Javier De Lope, and Juan P. Fuentes. Navigation and self-semantic location of drones in indoor environments by combining the visual bug algorithm and entropy-based vision. *Frontiers in Neurorobotics*, volume 11:46, 2017.

[88] Darío Maravall, Javier de Lope, and Juan Pablo Fuentes. Visual bug algorithm for simultaneous robot homing and obstacle avoidance using visual topological maps in an unmanned ground vehicle. In *Bioinspired Computation in Artificial Systems*, pages 301–310. Springer International Publishing, 2015.

[89] Edwin Olson. AprilTag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*, pages 3400–3407. 2011.

[90] C. Goerzen, Z. Kong, and B. Mettler. *A survey of motion planning algorithms from the perspective of autonomous UAV guidance*, volume 57. 2010.

[91] Javier Minguez, Florant Lamiraux, and Jean Paul Laumond. Motion planning and obstacle avoidance. *Springer Handbook of Robotics*, pages 1177–1201, 2016.

[92] Lun Quan, Luxin Han, Boyu Zhou, Shaojie Shen, and Fei Gao. Survey of UAV motion planning. *IET Cyber-Systems and Robotics*, volume 2(1):14–21, 2020.

[93] Sjoerd Tijmons, Guido C.H.E. De Croon, Bart D.W. Remes, Christophe De Wagter, and Max Mulder. Obstacle Avoidance Strategy using Onboard Stereo Vision on a Flapping Wing MAV. *IEEE Transactions on Robotics*, volume 33(4):858–874, 2017.

[94] Roland Brockers, Anthony Fragoso, Brandon Rothrock, Connor Lee, and Larry Matthies. Vision-Based Obstacle Avoidance for Micro Air Vehicles Using an Egocylindrical Depth Map. In *Springer Proceedings in Advanced Robotics*, volume 1, pages 505–514. Springer, Cham, 2017.

[95] Anthony T. Fragoso, Cevahir Cigla, Roland Brockers, and Larry H. Matthies. Dynamically Feasible Motion Planning for Micro Air Vehicles Using an Egocylinder. In *Field and Service Robotics*, volume 5, pages 433–447. Springer International Publishing, 2018.

[96] Vladimir J. Lumelsky and Alexander A. Stepanov. Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment. *IEEE Transactions on Automatic Control*, volume 31(11):1058–1063, 1986.

[97] Ishay Kamon, Ehud Rivlin, and Elon Rimon. New range-sensor based globally convergent navigation algorithm for mobile robots. *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1:429–435, 1996.

[98] S. L. Laubach and J. W. Burdick. An Autonomous Sensor-Based Path-Planner for Planetary Microrovers. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 1:347–354, 1999.

[99] K. N. McGuire, G. C.H.E. de Croon, and K. Tuyls. A comparative study of bug algorithms for robot navigation. *Robotics and Autonomous Systems*, volume 121:103261, 2019.

[100] Stephane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J. Andrew Bagnell, and Martial Hebert. Learning monocular reactive UAV control in cluttered natural environments. *2013 IEEE International Conference on Robotics and Automation*, pages 1765–1772, 2013.

[101] F. Valenti, D. Giaquinto, L. Musto, A. Zinelli, M. Bertozzi, and A. Broggi. Enabling Computer Vision-Based Autonomous Navigation for Unmanned Aerial Vehicles in Cluttered GPS-Denied Environments. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3886–3891, 2018.

[102] Lionel Heng, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided MAV using on-board processing. *2011 IEEE International Conference on Robotics and Automation*, pages 2472–2477, 2011.

[103] Julian Blank and Kalyanmoy Deb. Pymoo: Multi-Objective Optimization in Python. *IEEE Access*, volume 8:89497–89509, 2020.