Real-time Stereo Vision: Optimizing Semi-Global Matching

Matthias Michael*, Jan Salmen*, Johannes Stallkamp*, and Marc Schlipsing*

***Abstract*— Semi-Global Matching (SGM) is arguably one of the most popular algorithms for real-time stereo vision. It is already employed in mass production vehicles today. Thinking of applications in intelligent vehicles (and fully autonomous vehicles in the long term), we aim at further improving SGM regarding its accuracy. In this study, we propose a straight-forward extension of the algorithm's parametrization. We consider individual penalties for different path orientations, weighted integration of paths, and penalties depending on intensity gradients. In order to tune all parameters, we applied evolutionary optimization. For a more efficient offline optimization and evaluation, we implemented SGM on graphics hardware. We describe the implementation using CUDA in detail. For our experiments, we consider two publicly available datasets: the popular Middlebury benchmark as well as a synthetic sequence from the .enpeda. project. The proposed extensions significantly improve the performance of SGM. The number of incorrect disparities was reduced by up to 27.5 % compared to the original approach, while the runtime was not increased.**

I. INTRODUCTION

A stereo camera system allows to estimate the depth of the scene by calculating disparities for the two input images. While stereo vision is among the most vivid topics in computer vision for decades still many research is done, driven by a wide range of possible fields of application like robotics, surveillance, and automotive systems. Stereo vision has recently reached mass production vehicles where it is employed for object detection and collision avoidance, among others. We give an overview of related work in Sec. II.

A popular approach for real-time systems given limited hardware resources is Semi-Global Matching (SGM) proposed by Hirschmüller [1]. The original approach makes use of intensity differences, mutual information, and an optimization method integrating different paths through the image. We explain SGM in more detail in Sec. III.

In this study, we investigate different ways to improve the algorithm's performance. We propose to extend the parametrization of the algorithm. Originally, SGM relies on two penalty parameters. We extend this in a straight-forward manner to up to 20. The extended parametrization allows for more flexibility, as described in Sec. IV.

While parameters for a task at hand are typically tweaked by hand, we propose to tune parameters using automatic optimization. This has several advantages compared to manual tuning. We employ a state-of-the-art evolutionary optimization algorithm. The applied optimization procedure is explained in Sec. V.

Optimization and evaluation of the SGM algorithm is done offline. We propose to make use of an implementation using graphics hardware for that purpose. We describe how to implement SGM using *CUDA*[1] in Sec. VI.

For our experiments, we consider the well-known Middlebury benchmark [2] and a long synthetic sequence provided by Vaudrey et al. [3]. We systematically compare the performance of differently extended SGM variants and evaluate the achieved improvements. The experiments and their results are described in Sec. VII followed by a conclusion in Sec. VIII.

II. RELATED WORK

Many different stereo vision algorithms have been proposed. The Middlebury Stereo site[2] established by Scharstein and Szeliski [2][4] gives an overview of state-of-the-art methods. We focus on approaches that are suitable for real-time applications.

The simplest algorithm, Winner-Takes-All (WTA), selects an optimal disparity for each pixel independently – disregarding its neighborhood. Using pixel-wise intensity differences as matching cost, results are not sufficient for real-world applications. Van der Mark and Gavrila [5] proposed new variants of WTA, considering more complex matching cost calculation and post-processing. They evaluated the performance on synthetic sequences and conclude that "simple WTA techniques for dense stereo, combined with robust error rejection schemes such as the left-right check, are more suitable for intelligent vehicle applications" than other algorithms considered in this study.

Scanline optimization (SO) considers horizontal image rows instead of computing results for single pixels independently [2]. It is related to another very popular real-time approach, dynamic programming (DP). DP was initially used for edge-based approaches [6], later based on pixel-wise intensity differences [7][8]. Many extensions of the basic DP algorithm have been proposed. Morales et al. study the influence of different types of noise to stereo vision algorithms [9]. They compare popular algorithms in terms of robustness, also considering temporal integration which is seldom used in stereo vision. They conclude that DP employing temporal integration "produces images that are visually better quality, for a wider range of noise types".

*M. Michael, J. Salmen, J. Stallkamp, and M. Schlipsing are with the Institut für Neuroinformatik, Ruhr-Universität Bochum, 44780 Bochum, Germany firstName.lastName@ini.rub.de

[1]Compute Unified Device Architecture by *Nvidia*, for further information see www.nvidia.com/cuda

[2]http://vision.middlebury.edu/stereo

In addition to algorithms considering single pixels or scanlines, approaches that perform *global* optimization have been proposed. They consider measures across the whole image. Such algorithms, e. g., belief propagation (BP), allow for high accuracy on the one hand. On the other hand, they are computationally more demanding than local methods.

Semi-global matching (SGM) approximates a global optimization by combining several local optimization steps. We describe SGM in detail in the next section. Klette et al. [10] recently published an extensive comparison of different stereo algorithms including variants of DP, BP, and SGM. In their summary, they state that "SGM can potentially deal with scenes of high-depth complexities". The algorithm performed en par with DP and BP in their overall results.

A few extensions and modifications to the original SGM algorithm have already been proposed: In [11] improvements for structured environments are achieved by employing an additional segmentation step. Hermann et al. [12] show that the runtime of SGM can be significantly reduced while the accuracy is only slightly affected. Implementing SGM on graphics hardware has been discussed, we refer to related work in Sec. VI where we describe our CUDA implementation.

III. SEMI-GLOBAL MATCHING

As already introduced in the last section, current stereo algorithms can be divided into two major groups: local and global methods. Local methods try to find optimal disparities for small image regions – e. g., a single row – which can lead to discontinuities between different regions (for instance, the well-known streaking effects in DP). Global approaches optimize all disparities at once and, thus, achieve an overall better performance. However, the required computation time is considerably higher.

The SGM – as a *semi-global* method – incorporates the advantages of both groups, achieving relatively low complexity and relatively high quality. It can be broken down into three distinct phases which will be briefly explained below.

Given the maximum disparity d_{max} and rectified image pairs, for each pixel at position (x, y) in the base image, the corresponding pixel in the match image is bound to lie between (x, y) and $(x-d_{\text{max}}, y)$ (assuming that the left image is chosen as base image). The chosen disparities d are rated by a cost function like

$$C(d) = |I_b(x, y) - I_m(x - d, y)|, \tag{1}$$

which just computes the absolute difference of gray levels given a certain pixel at position (x, y) in the base image I_b and pixel $(x-d, y)$ in the match image I_m. There are several reasonable cost functions for the matching. Beside taking the absolute difference of gray levels, Hirschmüller proposes a method based on mutual information to incorporate illumination changes between both input images [1].

The objective function $E(D)$ from Eq. (2), which assigns costs to a disparity image D, has to be minimized in order to achieve optimal disparities.

$$E(D) = \sum_{d \in D} \left(C(d) + \sum_{d' \in N(d)} P_1 T[|d - d'| = 1] + \sum_{d'' \in N(d)} P_2 T[|d - d''| > 1] \right), \tag{2}$$

where $T[\cdot] = 1$ if its argument is true and 0 otherwise, and $N(d)$ denotes d's neighborhood. The function sums the initial costs $C(d)$ for the chosen disparities and two additional penalties which depend on the difference to the neighborhood disparities. If they differ by 1, a small penalty P_1 is applied. For larger differences, P_2 is added. The choice of a smaller P_1 allows to adapt to slanted or tilted surfaces. Bigger jumps in disparity are only possible if $C(d'') - C(d) > P_2$.

Minimizing $E(D)$ in a two-dimensional manner would be very costly. Therefore, SGM simplifies the optimization by traversing one-dimensional paths and ensures the constraints with respect to these explicit directions. This approach requires a second phase known as *cost aggregation*. Equation (3) describes this procedure for a horizontal path from the left to the right in an arbitrary image row y.

$$\begin{aligned} E(x, y, d) = & C(x, y, d) + \min [\\ & E(x - 1, y, d), \\ & E(x - 1, y, d - 1) + P_1, \\ & E(x - 1, y, d + 1) + P_1, \\ & \min_i (E(x - 1, y, i) + P_2)] \end{aligned} \tag{3}$$

$E(x, y, d)$, which is recursive, defines a second cost measure assigned to each pixel at a certain position (x, y) with disparity value d. Therefore, the cost for a single pixel requires all cost values of preceding pixels. This allows to compute the costs via path traversal from the left to the right.

This traversal effectively emulates the constraints introduced by the original objective function $E(D)$. A pixel may be assigned a lower cost if it adapts its disparity to its neighbors. However, this only holds for one direction and, thus, several differently oriented paths have to be executed. According to [1], the minimum number of paths should be at least 8 (i. e., two paths for the horizontal, vertical and both diagonal orientations respectively).

The final (smoothed) costs for each pixel and each disparity $S(x, y, d)$ are obtained by summing the costs $E_{\mathbf{r}}(x, y, d)$ of paths in all directions $\mathbf{r}$:

$$S(x, y, d) = \sum_{\mathbf{r}} E_{\mathbf{r}}(x, y, d) \tag{4}$$

The final step finds the disparity that minimizes the cost for each pixel given S. With the described method, a disparity is assigned to every pixel of the base image – regardless of occlusions. However, there exist certain methods to refine the disparity values.

Occlusion detection is achieved by additionally computing disparities for the match image followed by pixel-wise con-

sistency check. If disparities differ, the corresponding value in the original image is regarded invalid.

A rather simple refinement of the disparities can be obtained by filtering the disparity image with a small median filter. Thereby single outliers are removed and the edges in the image may be enhanced. Lastly, a sub-pixel estimation can be realized by fitting a quadratic curve through the lowest cost value and its two neighboring disparities. This may shift the actual minimum by a value smaller 1 and, therefore, increase disparity resolution.

IV. EXTENDED PARAMETRIZATION

As stated in Sec. III, SGM originally requires only two parameters – the matching penalties that are used for every path. It is possible to calculate a disparity image based on a single path. Comparing the results of different paths, one is able to observe major differences in quality, depending on the global structure of the input images.

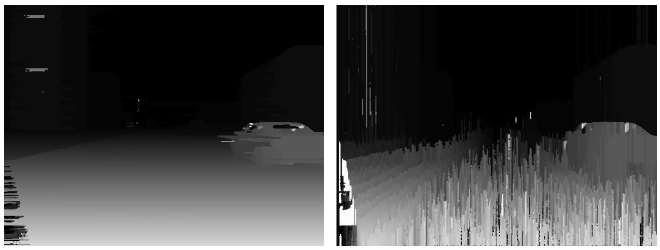

Fig. 1. Disparity images based on the horizontally and vertically oriented paths with $P_1 = 15$ and $P_2 = 35$.

One example of this behavior are images containing street surfaces as shown in Fig. 1. The horizontally oriented paths allow to compute a good disparity image. The vertical orientations, however, show a large amount of streaking. This is mainly caused by the ground plane featuring disparities that monotonously change in vertical directions. The algorithm favors regions of constant disparities (due to the penalties in Eq. (2)) and, therefore, causes a large amount of errors.

In the scenario shown in Fig. 1, the penalties are well chosen for the horizontal paths, whereas the vertical paths would benefit from smaller penalties supporting a larger deviation in neighboring disparities. Therefore, we allow for individual $P_1(\mathbf{r})$ and $P_2(\mathbf{r})$ for each path depending on its direction $\mathbf{r}$.

Regardless of the choice of penalties, the results obtained from a certain path orientation may still lead to better estimates than the results from other orientations. Hence, it should be possible to alter its contribution to the final disparity image, which is achieved by introducing weights for each path orientation. We extend Eq. (4) in the following way, introducing $w_{\mathbf{r}}$:

$$S(x, y, d) = \sum_{\mathbf{r}} w_{\mathbf{r}} \cdot E_{\mathbf{r}}(x, y, d). \tag{5}$$

Note that for both extensions proposed above, only the path orientation (e. g., horizontal, vertical, etc.) should have impact on the parameters. We disregard the particular sequence (e. g., left to right or right to left) and assign the same penalties and weights to paths sharing the same orientation. For instance, all parameters for horizontal paths from left to right and right to left are set identically although the paths have different directions $\mathbf{r}$ given the notation above.

Finally, it is well-known that stereo vision can benefit from adapting penalties for disparity discontinuities depending on the actual image content. For this reason, Hirschmüller proposes to adapt P_2 depending on the intensity gradient $I_\Delta(x, y)$ for each pixel considered at position (x, y). We follow this suggestion in a more general way by introducing additional penalties $\hat{P}_1(\mathbf{r})$ and $\hat{P}_2(\mathbf{r})$ that are used at (x, y) if $I_\Delta(x, y)$ exceeds a fixed threshold instead of $P_1(\mathbf{r})$ and $P_2(\mathbf{r})$, respectively.

The presented SGM extensions incorporate a number of up to five parameters for each considered orientation: four penalties and one weight. Given that the minimum number of orientations is four, this yields 20 parameters for our fully extended SGM. Tuning such an amount of parameters should not be done manually. An automated optimization is favorable and is explained in the following section.

V. AUTOMATIC OPTIMIZATION

The SGM variants proposed here rely on up to 20 parameters. For tuning such high numbers of parameters, a large amount of data has to be considered. These are common circumstances in computer vision and optimizing parameters is often done manually in a more or less systematic way.

We argue that the task of finding optimal parameters for given training data should be automated. There are powerful and well established methods available, for instance evolutionary algorithms. Automatic optimization, compared to manual tuning, has several benefits: It can consider a large amount of data, handle many parameters and in particular their complex interplay. It is time-saving, is unbiased and impartial, and it assures to find (near-)optimal solutions.

For the problem at hand, we can interpret all parameters as real-valued. This allows to use the covariance matrix adaptation evolution strategy (CMA-ES), which represents "the state-of-the-art in evolutionary optimization in real-valued $\mathbb{R}^n$ search spaces" [13]. Like other evolutionary algorithms, CMA-ES performs optimization by iteratively adapting solutions. However, in contrast to other approaches, it also adapts its search distribution during the evolution, allowing for faster convergence. For more details, see [14].

In order to apply CMA-ES, one has to assign a quality measure (fitness) to each generated parameter combination (individual). For the given problem, this can be done by computing disparity images for the proposed parameters and evaluating their correctness with respect to ground-truth.

VI. CUDA IMPLEMENTATION

Implementing SGM using graphics hardware has already been discusses. Rosenberg et al. [15] make use of Cg shaders. Ernst and Hirschmüller [16] describe an implementation using OpenCL, but they state that "since CUDA offers more flexibility and higher abstraction from the graphics hardware, we are going to implement SGM in CUDA". In [17] Zhu

et al. described their CUDA implementation resulting in a reasonable speed-up. Here, we present our approach.

Since the required operations are very time-consuming, classical CPU implementations are not real-time capable. Moreover, in order to optimize parameters with an evolutionary algorithm several thousand iterations are necessary. Thus, a highly efficient implementation of SGM would speed-up optimization and support real-time execution.

Many of the involved tasks are completely independent of each other, e. g., the intensity-based calculation of the initial costs of the original SGM. Also, the path traversal – in which each step depends on the previous one – can be parallelized to a certain degree, by executing several paths at the same time. Therefore, the overall performance of the algorithm can profit from parallel execution. Due to a recent popularity of programmable GPGPUs (General Purpose Graphics Processing Units) in off-the-shelf consumer hardware, we chose CUDA technology for our implementation, which allows programming graphics hardware in *C* with extensions for parallel functions, data transfer, and explicit caching (cf.[18]).

CUDA follows a SIMD (Single Instruction Multiple Data) execution model enabling the programmer to frame a computational task (*kernel*) that is performed in the same manner on small chunks of data. These tasks are assigned to lightweight threads executed on the graphics hardware in parallel. The threads of a kernel function are organized in a two-level hierarchical structure with up to three dimensions. The *grid* is divided into several blocks holding the threads. Generally, each thread is assigned data (e. g., a single pixel of an image) depending on its location within the grid / block.

Another important aspect of CUDA is the memory management. The use of different kinds of memory is mostly necessary to achieve high performance. In the following, the structural design of our implementation is described.

To manage access to the CUDA part of the code the host functions are encapsulated in traditional *C++* classes that execute calls to the CUDA-API. Since SGM can be divided into three distinct phases, initial cost calculation, cost accumulation, and disparity estimation, one kernel function implements the main work of the particular task.

Initial cost calculation: The calculation of pixel-wise initial matching costs can be parallelized in the most straight forward manner. The rectified left and right image are copied to GPU memory and serve as input to this stage. The designated output is a $w \times h \times d_{\text{max}}$ cost matrix C, where w and h denote width and height of the input images, and d_{max} is the chosen maximal disparity. Each matrix entry can be computed independently according to Eq. 1.

Thus, for each entry of C, a single thread is launched. Those perform the lookup in both input images and calculate the absolute difference of grey levels and thereby eliminate the need for synchronization between different threads mentioned in [17]. As mentioned above the created kernel grid has the same dimensionality as C. Both images are bound to texture memory, which offers implicit two-dimensional caching for read accesses and, thus, increases performance. Moreover, textures offer very efficient border handling for accesses outside the defined coordinates.

Path traversal: Based on C, Eq. (3) is implemented by traversing paths in eight directions across the disparity image and accumulating the results in a path matrix E of the same size. Since one path step depends on its predecessors, parallelization is constrained. However, different paths of the same direction can be executed at the same time. The straightforward implementation would invoke a single thread to calculate an entire path considering all possible disparities. This is improved by separating a path into d_{max} threads, each of them responsible for a single disparity value. Since the minimum of all preceding disparities is determined, all calculations of the previous path step have to be completed. This is assured by CUDAs kernel synchronization mechanism (`__syncthreads()`), which stops further execution until all threads of a block have reached this statement. Thus – and in contrast to [17] – all threads responsible for the same path are grouped in a block. At the same time, this allows for using shared memory, which is shared across a block, for handing over intermediate results of the previous path step. Thereby no information has to be transmitted between different blocks. The minimum is calculated using a reduction technique which is popular in parallel computing [18].

Is is noteworthy that the calculation of differently oriented paths is achieved by modifying starting point and step direction, i. e., coordinate in- or decrement which is done by utilizing device function pointers. To increase code maintainability, a single kernel function was implemented for that purpose. Depending on the parameters this kernel can efficiently traverse arbitrary paths across the image.

Disparity Estimation: The final step is to determine the optimal disparity d of from E for each pixel (x, y). Again this can be done separately for each pixel by applying an extension of the before-mentioned reduction technique. Naturally reduction is used to calculate the overall minimal value. Here it is necessary to determine the disparity that corresponds to this minimum. Therefore, instead of one, two arrays in the shared memory are required from which one holds the computed minima while the other contains the original disparity of the value. Both arrays are updated simultaneously.

This implementation only requires $\log_2(d_{\text{max}})$ steps to compute the optimal disparity for each pixel. It is noteworthy, that at this point some mismatch can arise between our implementation and an otherwise equivalent single-threaded one. If several potential minima with the same cost value exist, the reduction is prone to select one in the middle of the array, while a typical iterative approach most likely selects the first or the last minimum it encounters.

Extensions: Since most of the here proposed extensions consist of additional parameters the general implementation described above does not need to be changed. The additional penalties as well as the weights can be simply transferred to the GPU and used as input to the path kernel. The gradient image was calculated beforehand and copied to the GPU as well. During each step of the path traversal a lookup to this

TABLE I

SGM ALGORITHM VARIANTS CONSIDERED FOR OUR EXPERIMENTS.

Algorithm	orientation-dependent penalties	paths weights	gradient-dependent penalties	#param
1. SGM	–	–	–	2
2. SGM$_{\Delta}$	–	–	+	4
3. SGM$_{\vec{w}}$	–	+	–	6
4. SGM$_{\angle}$	+	–	–	8
5. SGM$_{\Delta,\vec{w}}$	–	+	+	8
6. SGM$_{\vec{w},\angle}$	+	+	–	12
7. SGM$_{\Delta,\angle}$	+	–	+	16
8. SGM$_{\Delta,\vec{w},\angle}$	+	+	+	20

image is necessary to determine which penalty (e. g., P_1 or $\hat{P}_1$) has to be applied. However, all threads in a block will branch the same way in a certain path step and, therefore, do not suffer from a slow down.

VII. EXPERIMENTS

In order to evaluate the accuracy improvements provided by the proposed modifications of SGM, we compare different variants on two datasets. The experimental setup (Sec. VII-A) and results (Sec. VII-B) are documented in this section.

A. Setup

We evaluated SGM variants using eight paths (i. e., four orientations) which is the minimum according to [1]. Individual penalties can be applied for each path orientation, denoted with an additional symbol $\vec{w}$. We considered variants using weights for integration of path results, denoted as $\angle$. Finally, we investigated the influence of penalties depending on image gradients, denoted as Δ. See Tab. I for an overview of the eight different algorithm variants considered. The total number of parameters differs from 2 (baseline algorithm) to 20 (applying all extensions proposed here).

All SGM variants were implemented using CUDA (see Sec. VI). Experiments were conducted on a four-core *Intel Xeon W3520* PC equipped with a *Nvidia Geforce GTX 480*.

Fig. 2. Images from the datasets considered: Middlebury benchmark (left) and synthetic sequence (right).

We performed parameter optimization for all approaches independently. The popular Middlebury benchmark dataset as well as a synthetic sequence made publicly available by Vaudrey et al. [3] were used. Figure 2 shows examples from both data sources. For the Middlebury benchmark, training and evaluation was performed on all four image pairs. From the simulated sequence, the first 100 frames were used for training and the remaining 296 frames for evaluation. Since consecutive images show little variance we only considered every tenth frame.

As cost function, we employed pixel-wise intensity differences, see Eq. (1). Mutual information was not considered, as the used images do not have any brightness differences. We did not apply post-processing in order to evaluate the performance of the raw algorithms.

In order to evaluate the quality of disparity images compared to ground-truth, we considered the root mean squared error (RMSE) as well as the number of bad pixels. The latter (using a threshold of 1) was used for fitness assignment to solutions during evolutionary optimization (see Sec. V). However, depending on the target application, one might choose another fitness to ensure correct results especially for objects close to the viewer or for object borders.

Five optimization trials were conducted for each algorithm variant. Parameters were initialized randomly for each run. We assured $P_1 < P_2$ for all starting points. Nevertheless, we did not enforce this constraint during optimization. We used the CMA-ES implementation from the *Shark*[3] open-source machine learning library [19].

B. Results

Table II shows the results of different SGM variants on the Middlebury dataset. The best performance was achieved with the SGM$_{\Delta,\vec{w},\angle}$ resulting in 4.1 % bad pixels, which is 27.5 % less than for the original SGM.

Table III shows results for the synthetic sequence. Note that testing was performed on a different part than training here, so the results allow to evaluate the generalization error. Our fully extended SGM performed best on the training set again. However, for the test set, the best result was achieved by an other new variant which applied individual weights and penalties for each orientation. Here an improvement of 15.1 % was obtained compared to the original approach.

VIII. CONCLUSION

Semi-global matching (SGM) is a popular algorithm for real-time stereo vision, for instance, in the context of intelligent vehicles. In this study, we investigated three approaches to extend the algorithm's parameter set in order to improve the accuracy of SGM.

Firstly, SGM can benefit from discriminating the considered paths by introducing individual penalties. Secondly, we proposed to introduce weights for the integration of paths. Thirdly, we investigated penalties that adapt to discontinuities detected in the input images.

In order to tune the algorithm's parameters, evolutionary optimization was employed. Among other advantages, this assures an unbiased evaluation. We provided results for the popular Middlebury benchmark dataset as well as for a synthetic sequence.

In order to allow for a more efficient optimization, we implemented SGM using CUDA. Our implementation runs

[3] `http://shark-project.sourceforge.net`

TABLE II

RESULTS ON THE MIDDLEBURY BENCHMARK.

Algorithm	P_1^1	P_2^1	$\hat{P}_1^1$	$\hat{P}_2^1$	P_1^2	P_2^2	$\hat{P}_1^2$	$\hat{P}_2^2$	P_1^3	P_2^3	$\hat{P}_1^3$	$\hat{P}_2^3$	P_1^4	P_2^4	$\hat{P}_1^4$	$\hat{P}_2^4$	$\vec{w}_1$	$\vec{w}_2$	$\vec{w}_3$	$\vec{w}_4$	**Error**
1. SGM	17.41	54.13																			5.63%
2. SGM$_\Delta$	21.99	129.4	14.86	34.57																	4.71%
3. SGM$_{\vec{w}}$	17.98	62.47															0.68	0.57	0.22	0.12	5.05%
4. SGM$_\angle$	22.02	82.79			17.75	80.87			14.93	23.30			10.67	28.87							4.99%
5. SGM$_{\Delta,\vec{w}}$	23.15	167.01	14.35	36.84													2.65	1.45	0.80	0.58	4.16%
6. SGM$_{\vec{w},\angle}$	21.19	84.10			14.99	44.79			44.22	73.94			27.22	48.58			0.96	0.98	0.06	0.27	4.65%
7. SGM$_{\Delta,\angle}$	36.68	122.47	27.38	78.51	34.64	125.2	27.62	98.03	79.19	159.4	1.12	81.16	26.8	172.1	25.54	29.31					5.28%
8. SGM$_{\Delta,\vec{w},\angle}$	20.25	111.8	14.13	40.14	23.39	182.1	22.25	42.76	31.49	163.5	7.92	15.38	53.24	110.0	2.07	113.22	0.91	0.48	0.4	0.08	**4.08**%

TABLE III

RESULTS ON THE *.ENPEDA.* BENCHMARK.

Algorithm	P_1^1	P_2^1	$\hat{P}_1^1$	$\hat{P}_2^1$	P_1^2	P_2^2	$\hat{P}_1^2$	$\hat{P}_2^2$	P_1^3	P_2^3	$\hat{P}_1^3$	$\hat{P}_2^3$	P_1^4	P_2^4	$\hat{P}_1^4$	$\hat{P}_2^4$	$\vec{w}_1$	$\vec{w}_2$	$\vec{w}_3$	$\vec{w}_4$	**Training**	**Test**
1. SGM	7.98	73.69																			3.99%	3.51%
2. SGM$_\Delta$	19.88	99.36	10.88	33.10																	3.83%	3.95%
3. SGM$_{\vec{w}}$	3.56	108.9															4.31	2.37	1.18	0.13	3.77%	3.67%
4. SGM$_\angle$	23.41	84.57			6.63	142.3			18.83	38.93			5.79	169.1							3.68%	3.79%
5. SGM$_{\Delta,\vec{w}}$	31.67	109.8	10.73	39.37													4.76	3.06	1.75	1.24	3.69%	4.07%
6. SGM$_{\vec{w},\angle}$	86.07	100.2			8.53	131.8			10.79	42.14			14.76	206.4			1.41	0.66	1.70	0.09	3.29%	**2.98**%
7. SGM$_{\Delta,\angle}$	42.07	105.95	1166	14.77	55.00	44.03	2953	19.05	3.86	132.61	871.9	38.48	5.44	84.90	1.43	188.3					3.39%	3.73%
8. SGM$_{\Delta,\vec{w},\angle}$	21.52	108.5	20.04	11.66	10.73	127.4	7.13	245.8	104.3	90.06	179.85	107.6	20.99	41.39	19.08	7.74	6.23	6.12	0.73	3.22	**3.25**%	3.66%

at 11.7 fps (using VGA resolution and 64 disparities) and, therefore, can also be employed in real-time applications. It is noteworthy that the runtime is not affected by the extended parametrization.

Applying each of the modifications proposed here individually allowed to reduce the amount of erroneous pixels for both data sets. If the modifications are combined, these results can significantly be improved. The number of bad pixels was reduced by up to 27.5 %.

For the *.enpeda.* benchmark, results differ slightly on both parts of the sequence (training and test). This might have various reasons, one of them is overfitting in the evolutionary optimization and should be further investigated. Future work could also incorporate optimization on real sequences.

Nevertheless, applying separate penalties and weights for each orientation a significant enhancement could be achieved even for the test data.

REFERENCES

[1] H. Hirschmüller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005, pp. 807–814.

[2] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, pp. 7–42, 2002.

[3] T. Vaudrey, C. Rabe, R. Klette, and J. Milburn, "Differences between stereo and motion behaviour on synthetic and real-world stereo sequences," in *Proceedings of the Conference on Image and Vision Computing New Zealand*. IEEE Press, 2008, pp. 1–6.

[4] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2003, pp. 195–202.

[5] W. van der Mark and D. M. Gavrila, "Real-time dense stereo for intelligent vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 1, pp. 38–50, 2006.

[6] Y. Ohta and T. Kanade, "Stereo by intra- and inter-scanline search using dynamic programming," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, pp. 139–154, 1985.

[7] D. Geiger, B. Ladendorf, and A. L. Yuille, "Occlusions and binocular stereo," in *Proceedings of the European Conference on Computer Vision*, 1992, pp. 425–433.

[8] S. Birchfield and C. Tomasi, "Depth discontinuities by pixel-to-pixel stereo," *International Journal of Computer Vision*, vol. 35, pp. 1073–1080, 1999.

[9] S. Morales, T. Vaudrey, and R. Klette, "Robustness evaluation of stereo algorithms on long stereo sequences," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2009, pp. 347–352.

[10] R. Klette, N. Kruger, T. Vaudrey, K. Pauwels, M. van Hulle, S. Morales, F. Kandil, R. Haeusler, N. Pugeault, C. Rabe, and M. Lappe, "Performance of correspondence algorithms in vision-based driver assistance using an online image sequence database," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 5, pp. 2012–2026, 2011.

[11] H. Hirschmüller, "Stereo vision in structured environments by consistent semi-global matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2006, pp. 2386–2393.

[12] S. Hermann, S. Morales, and R. Klette, "Half-resolution semi-global stereo matching," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2011, pp. 201–206.

[13] H.-G. Beyer, "Evolution strategies," *Scholarpedia*, vol. 2, no. 8, p. 1965, 2007.

[14] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[15] I. D. Rosenberg, P. L. Davidson, C. Muller, and J. Han, "Real-time stereo vision using semi-global matching on programmable graphics hardware," in *Proceedings of the ACM SIGGRAPH Sketches*, 2006, pp. 89–89.

[16] I. Ernst and H. Hirschmüller, "Mutual information based semi-global stereo matching on the gpu," in *Proceedings of the International Symposium on Advances in Visual Computing*, 2008, pp. 228–239.

[17] K. Zhu, M. Butenuth, and P. d'Angelo, "Comparison of dense stereo using CUDA," in *Proceedings of the European Conference on Computer Vision, Workshop, Computer Vision on GPUs*, 2010.

[18] R. Farber, *CUDA Application Design and Development*. Morgan Kaufmann, 2011.

[19] C. Igel, T. Glasmachers, and V. Heidrich-Meisner, "Shark," *Journal of Machine Learning Research*, vol. 9, pp. 993–996, 2008.