Sparse Stereo by Edge-Based Search Using Dynamic Programming

Jonas Witt and Uwe Weltin

IZT at the Hamburg University of Technology
Eißendorfer Straße 40, 21073 Hamburg
{jonas.witt, weltin}@tuhh.de

Abstract

In this paper, a novel edge-based stereo matching technique is presented. Depth discontinuities are specifically accounted for in the choice of support regions. We employ dynamic programming along edge-segments to efficiently enforce inter-scanline consistency. Although not performing a global optimization over the whole image we show that our approach performs successfully on the Middlebury benchmark datasets[1] *while being computationally feasible in real-time.*

1. Introduction

Edge-based algorithms for both object detection and vision-based navigation are an actively researched topic. This is due to the absence of texture in many common man-made environments and objects for which point-based algorithms perform badly. Edges and shading are often the only reliable and abundantly available information. However, the evaluation of shading is costly. Edge-based systems have been shown to be capable alternatives. Tomono [12] proposed an edgepoint-based SLAM algorithm, while Chandraker et al. [3] proposed a robust structure-from-motion algorithm using straight lines. In [6] an edge-based object detection system was presented that can detect object instances in cluttered environments. All these algorithms require edges along with depth information.

Stereo edge matching poses a couple of challenges. For one, correlation-based matching costs can be problematic if the matching window is not carefully chosen, since edges often lie on object borders. Also, since the matching is only sparse, one can not gain confidence in disparities over larger surfaces. The matching of horizontal segments is particularly difficult due to the inherent ambiguities. Finally, corresponding edge pixels are not necessarily detected in both images. On the other hand, significantly less computational effort is required due to the simplified search space.

Previous papers present very different approaches to the problem of matching edges in two or three views. Different algorithms for straight lines have been proposed by [10], [9] and [1], the latter of which was also extended to parametric curves in [11]. A correlation-based method utilizing color information was presented in [8], while a more recent publication proposes a phase-based algorithm utilizing multi-scale gabor filters with a probabilistic model for matching [13]. In a previous paper of the authors, the best individual matches of a correlation-based matching cost are refined by enforcing smoothness along edge-segments, reducing false matches [14]. Here, we discuss another correlation-based approach which uses dynamic programming for optimization along edges (rather than scanlines) to increase the performance in the face of ambiguous matches.

2. Stereo Edge Matching

The matching of sparse features basically involves the same steps as for dense matching, namely preprocessing, cost aggregation, optimization and optionally refinement. For isolated point features such as corners or scale-space extrema the search space is usually unambiguous enough to simply assign the matches with the best costs. Also, connectivity information is not available and thus can not be utilized to improve the matching.

Edge points on the other hand are less discriminatory than point features, especially when an edge is parallel to the epipolar line. However, the connectivity information can be utilized to generate confident matches. A

[1] Thanks to Daniel Scharstein and Richard Szeliski for maintaining their vision homepage http://vision.middlebury.edu/stereo/ and providing many benchmark datasets.

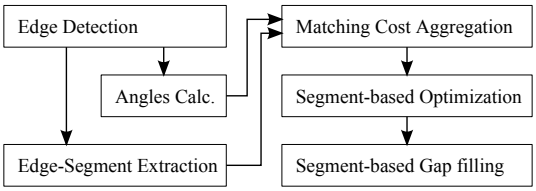

Figure 1. Algorithm structure

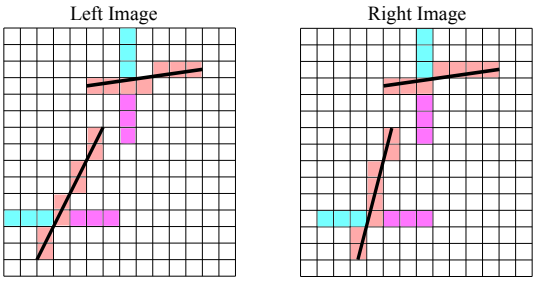

Figure 2. Asymmetric pixel-strip matching for two edges with different angles.

visualization of the proposed algorithm is shown in Figure 1. For the edge detection block we use the Canny detector [2] with subpixel refinement [4]. This way we gain subpixel accurate disparities for vertical and diagonal edges with little effort. Matching can still be done at pixel level. While the cost aggregation is discussed in the following Section, edge-segment extraction and optimization are explained in Section 2.2. The gap-filling technique is taken from [14], but with maximum confidence for each matched pixel. Effectively, gaps are filled if three consistent pixels on each side of the gap exist within the edge-segment and the disparity difference between both sides is not greater than 3. Intermediate pixels are linearly interpolated.

2.1. Cost Aggregation

For the cost aggregation of edge-based stereo matching, one has to consider the nature of disparities around edges. Edges either occur at ordinary high-contrast regions on a smooth surface or at object borders. Effectively, the change in disparities at depth discontinuities splits up the matching pixels of a symmetric support region in the other image of a stereo pair. This can be accounted for, as was shown in [7], but at a computational cost. Accordingly, a symmetric two-dimensional support region for correlation is not a good choice for edges.

In this work we choose asymmetric pixel-strips of length l_{match} on either side of a candidate edge as support region, as shown in Figure 2. For each side of the

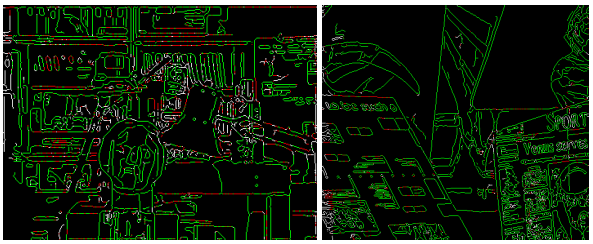

Figure 3. Disparity errors of WTA matching with a 15x1 pixel strip for Tsukuba (left) and Venus (right) image sets. Green: $e \leq 1\,\mathrm{px}$, red: $e > 1\,\mathrm{px}$, white: no match.

candidate edge, the sum of absolute differences (SAD) is computed and only the minimum of both is returned as the matching cost.

Consequently, if an edge lies on an object border, the foreground disparity is retrieved and the consistency of the support region is preserved. Increasing the width of the support regions (e.g. from one to three or five pixels) makes individual matches more robust against noise (which would be important for winner-takes-all matching). However, since support regions of adjacent edge pixels would overlap, essentially, no additional information is gained. Experiments showed that increasing the width of the pixel strip does not improve the matching performance for the proposed optimization-based method for the middlebury test images.

Depending on the edge angle, either a horizontal or a vertical pixel-strip is matched. This helps in disambiguating horizontal edge disparities. The cost is computed only if the edge angles in the left and right image differ by no more than α_{match}. Finally, if the mean SAD is smaller than t_{SAD}, the match is considered valid. However, as can be seen in Figure 3, a simple winner-takes-all (WTA) matching still shows insufficient performance on horizontal edge-segments since the connectivity information is not incorporated.

2.2. Segment-Based Optimization using dynamic programming

A more sophisticated method is proposed in this Section. After edges have been detected, edge segments are extracted by looking for edge pixels with only one neighbour (i.e. edge endpoints) and traversing the edge until a branch or no more (8-connected) neighbours are detected. At branches new segments are started. For each edge segment that was found, all disparity matches are computed according to the previous Section. Finally, we can build a graph for each segment as depicted

in Figure 4 and formulate an independent optimization problem for each such graph/segment. Due to this independence, the complexity is significantly reduced compared to a global optimization formulation. The graph can be laid out in two dimensions as a shortest path problem, in which each node $\mathbf{n}_{i,j}$ corresponds to an edge point index i and a disparity index j, which enumerates the matching possibilities/disparities of pixel i. The connectivity information between edge segments at branches is currently not used as it would complicate the optimization problem formulation significantly.

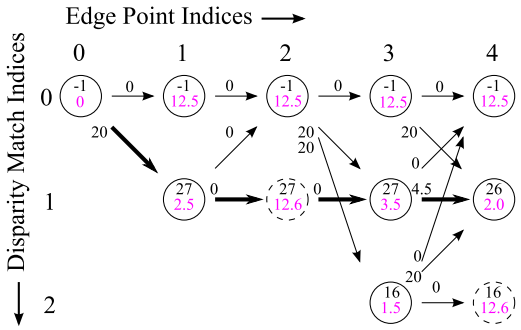

Figure 4. Graph of an edge segment with 4 pixels. The bold arrows are the graph edges that form the minimum cost path.

The circles in Figure 4 represent the graph nodes along with their disparity $d(\mathbf{n}_{i,j})$ (black) and matching cost $m(\mathbf{n}_{i,j})$ (magenta). The top row nodes $\mathbf{n}_{i,0}$ in the graph are "no match"-nodes, for which the disparity is not defined (and thus set to -1 in this case) and a tunable matching cost $m_{\text{nomatch}} = 12.5$ is assigned. The nodes with edge point indices 1 to 4 correspond to actual edge pixels, while node $\mathbf{n}_{0,0}$ is a virtual starting node for which the cost is set to zero. Dashed nodes are special "no match"-nodes with a defined disparity and matching cost $m_{\text{nomatch}} + \epsilon = 12.6$ which are introduced to fill gaps if no match with a disparity difference of one or zero is detected at the next edge pixel. The $\epsilon = 0.1$ is added to favor the undefined disparity "no-match"-nodes in the upper row in case of doubt. Effectively, this disallows graph edges from these nodes to all nodes with a disparity difference bigger than one, since the undefined disparity "no-match"-nodes in the upper row are always the cheaper path. One can see that for pixel 2 no match was found, as only "no match"-nodes exist.

The graph edges are visualized by arrows, along with their cost $p(\mathbf{n}_{i,j}, \mathbf{n}_{i+1,k})$. In fact, the actual graph edge cost $C(\mathbf{n}_{i,j}, \mathbf{n}_{i+1,k})$ in terms of graph algorithms is the sum of this value and the matching cost $m(\mathbf{n}_{i+1,k})$ of the destination node. We still chose this separate notation for visualization, as the matching cost does not depend on the previous node, while the penalty term $p(\mathbf{n}_{i,j}, \mathbf{n}_{i+1,k})$ does. The cost of a path $P = \{\mathbf{n}_0, \mathbf{n}_1, ..., \mathbf{n}_N\}$ in this graph accordingly is the sum of all matching costs and edge penalty terms that lie on its way:

$$C(P) = \sum_{i=1}^{N} m\left(\mathbf{n}_i\right) + p\left(\mathbf{n}_{i-1}, \mathbf{n}_i\right)$$

The penalty term $p(\mathbf{n}_{i-1}, \mathbf{n}_i)$ is used to favour paths that have smooth disparities along edge segments. Depending on the change in disparities between two nodes, different penalties are assigned:

$$p(\mathbf{n}_{i-1}, \mathbf{n}_i) = \begin{cases} 0, & \text{if } |d(\mathbf{n}_{i-1}) - d(\mathbf{n}_i)| = 0 \\ p_{\text{step}}, & \text{if } |d(\mathbf{n}_{i-1}) - d(\mathbf{n}_i)| = 1 \\ p_{\text{jump}}, & \text{if } |d(\mathbf{n}_{i-1}) - d(\mathbf{n}_i)| > 1 \end{cases}$$

The tuning penalty term p_{step} is added, when the disparity of adjacent edge pixels equals one, to reward constant disparities along horizontal edges. A larger penalty term p_{jump} is added for arbitrary disparity jumps or the transition from a node with no match to a matched node. The values that are used in Figure 4 are $p_{\text{step}} = 4.5$ and $p_{\text{jump}} = 20$.

Now that we have defined a graph with non-negative graph edge costs, we can apply standard techniques to find the minimum cost path along edge segments. In stereo algorithms the efficient optimal solution to this kind of problem is usually referred to as dynamic programming. In terms of graph algorithms, this is equivalent to Dijkstra's algorithm. We organize all nodes in two sets: the visited and the unvisited set. Initially, the visited set consists of only the $\mathbf{n}_{0,0}$ node while the rest is in the unvisited set. A brief outline of the algorithm can be given as follows:

1. Find the node in the unvisited set with minimal cumulative cost that is adjacent to a node in the visited set.

2. Remove this node from the unvisited set and add it to the visited set. The final minimum path to this node is now known.

3. Exit if the node belongs to the last pixel in the edge segment, otherwise go to step 1.

In Figure 4 we start by examining the adjacent nodes of $\mathbf{n}_{0,0}$ and find that the minimum cost node is $\mathbf{n}_{1,0}$. In the following, we denote the minimal cost from $\mathbf{n}_{0,0}$ to $\mathbf{n}$ with $\tilde{C}(\mathbf{n})$. As it is now added to the visited set, we take all its adjacent nodes into account when we search

for the next node with minimal cost in the unvisited set. Thus, our next options are $\tilde{C}(\mathbf{n}_{2,0}) = 25$ and $\tilde{C}(\mathbf{n}_{1,1}) = 22.5$. We repeat this process until $\mathbf{n}_{4,1}$ is added to the visited set, which signals that the minimum cost path for the edge segment has been found, since this is the first node of the last pixel that is added to the visited set (the bold arrows in Figure 4 denote the final minimum cost path). Note that the cost formulation is symmetric, so if we started at the other end of the edge we end up with the same minimum cost path. The virtual start node would move to the other side, though. Some of the "no match"-nodes that are introduced as gap-fillers possibly change positions, too. But due to the ϵ that is added to their match cost, one can see that any gap-filler "no match"-node has to be enclosed by at least two real match nodes. If this is not the case, the minimum path would rather lie on the cheaper "no match"-nodes.

If a lower bound for the remaining path cost can be calculated, the optimal path can be calculated in a more goal-oriented manner, effectively reducing the number of node evaluations [5]. In the Dijkstra algorithm only the cumulative cost $\tilde{C}(\mathbf{n}_{i,j})$ is considered as a measure in order to determine which node should be examined next. Thus, the search is undirected. The A* search, as proposed in [5], makes use of a heuristic to direct the search towards the goal. We use a simple heuristic $H(\mathbf{n}_{i,j})$ which depends on the edge-segment length M, the edge point index i and a tuning parameter m_{mincost}:

$$H(\mathbf{n}_{i,j}) = (M - i)m_{\mathrm{mincost}}$$

With this heuristic, step 1 of the algorithm now searches for the lowest goal-directed cost $\hat{C}(\mathbf{n}_{i,j}) = \tilde{C}(\mathbf{n}_{i,j}) + H(\mathbf{n}_{i,j})$. Theoretically it is possible to have zero matching costs, so in order to compute optimal paths in all instances m_{mincost} would have to be zero (which would be equivalent to the Dijkstra search). For real image sets the optimal matches were still found for nonzero heuristics, while reducing the computational effort notably.

3. Experimental Results

In this Section we examine the performance of edge-based dynamic programming (EBDP) with a selection of the Middlebury stereo sets and benchmark against two other sparse methods, namely edge matching by confidence-based refinement (EMCBR, [14]) and probabilistic phase-based sparse stereo (PPBSS, [13]). The parameterization of EBDP was found empirically and was not changed throughout the experiments. The parameter values are $l_{\mathrm{match}} = 15$, $m_{\mathrm{nomatch}} = 12.5$, $t_{\mathrm{SAD}} = 12.0$, $m_{\mathrm{mincost}} = 1.0$, $\alpha_{\mathrm{match}} = \pi/16$, $p_{\mathrm{step}} = 4.5$, $p_{\mathrm{jump}} = 20$ and the maximum disparity was set to 64. Table 1 lists the quantitative results for the Tsukuba (384×288), Teddy (450×375), Cones (450×375), Venus (434×383) and Sawtooth (434×380) image sets. The sparse ground truth disparities have been generated by first dilating the dense ground truth disparity images with a 3×3 structuring element to always retrieve the foreground disparity on object borders and then extracting these disparities at edge loci. The error percentage refers to disparity errors > 1 in relation to matched pixels. For EBDP and EMCBR the according error plots are given in Figure 5.

Since the overall matching performance of EMCBR is already very good, improvements can only be seen in the challenging Tsukuba image set. Here, EBDP is able to recover most horizontal edges, while EMCBR rejects many of these edges due to inconsistencies. Unfortunately, like many dense methods, EBDP produces an increased number of false matches in the lower part of the teddy image set due to the slanted ground plane. A comparison and discussion on dense vs. sparse algorithms in the context of edge matching is given in [14].

The experiments were run on an Intel i7-2640M CPU (2.8 GHz) using both cores. The overall computation times for EBDP were about 60-85ms and include 6-15ms for Gaussian filtering, Canny edge detection and angles computation. Approximately 5-13ms are used for the gap filling and edge segment extraction. For the Cones image set, about 55ms are spent on cost aggregation and optimization. This phase with WTA matching (as done in EMCBR) takes less than 10ms. Accordingly, EMCBR is usually about twice as fast as EBDP.

4. Discussion and Future Work

This paper presented an efficient optimization-based edge matching technique, that uses discontinuity penalties to find smooth disparities along edge-segments. The idea behind the support regions is simple, yet efficient and accounts for depth discontinuities and horizontal edge segments. This combination allows to retrieve significantly more disparities than EMCBR and PPBSS in challenging setups like the Tsukuba image set. However, the advantage over EMCBR vanishes for image sets with only few horizontal edge-segments. In further research we already observed that this difference can play an important role for object detection and navigation tasks, though (especially for man-made structures due to more straight and horizontal edges).

Potential for improvements can be identified in terms of speed: a segment-based coarse-to-fine approach might accelerate the optimization phase significantly. Furthermore, automatic rejection of ambiguous optimization results could significantly improve error rates in the case of the Teddy image set, which is left for

Table 1. Matching performance of three sparse algorithms (EBDP, EMCBR, PPBSS).

	Tsukuba		Teddy		Cones		Venus		Sawtooth	
	Matches	Errors	Matches	Errors	Matches	Errors	Matches	Errors	Matches	Errors
EBDP	9920	7.6%	11755	11.9%	15155	5.4%	11610	1.8%	14614	2.7%
EMCBR	8550	8.8%	10514	5.3%	16147	5.3%	11816	2.0%	14202	2.4%
PPBSS	2350	17.0%	-	-	-	-	1310	6.0%	3079	4.0%

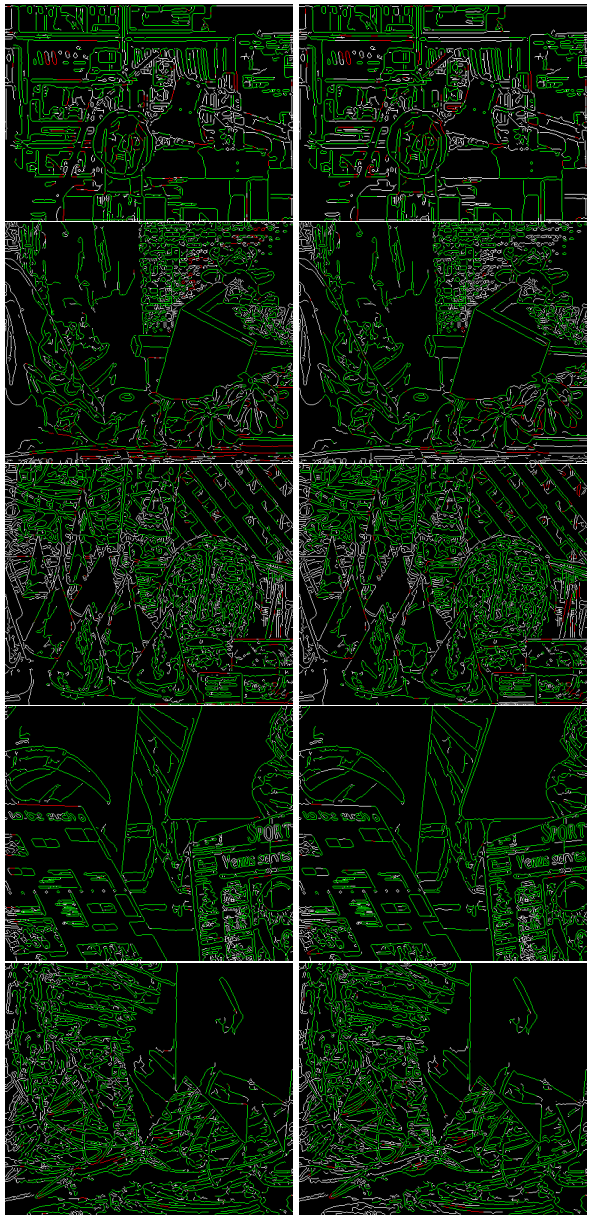

Figure 5. Disparity error plots of EBDP (left column) and EMCBR (right column). Green: $e \leq 1\,\mathrm{px}$, Red: $e > 1\,\mathrm{px}$, White: no match.

future research. Finally, higher quality edge detectors and maybe even color edge detectors likewise allow for performance improvements, at the cost of computation time, though.

References

[1] N. Ayache and B. Faverjon. Efficient registration of stereo images by matching graph descriptions of edge segments. *IJCV*, 1(2):107–131, 1987.

[2] J. Canny. A Computational Approach to Edge Detection. *PAMI*, 8(6):679 – 698, 1986.

[3] M. Chandraker, J. Lim, and D. Kriegman. Moving in stereo: Efficient structure and motion using lines. In *Proc. of ICCV*, pages 1741–1748. IEEE, 2009.

[4] F. Devernay. A Non-Maxima Suppression Method for Edge Detection with Sub-Pixel Accuracy. Technical report, INRIA, 1995.

[5] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *TSMC*, 4(2):100–107, 1968.

[6] S. Helmer and D. Lowe. Using stereo for object recognition. In *Proc. of ICRA*, pages 3121–3127. IEEE, 2010.

[7] H. Hirschmüller, P. R. Innocent, and J. Garibaldi. Real-Time Correlation-Based Stereo Vision with Reduced Border Errors. *IJCV*, 47(1):229–246, 2002.

[8] A. Koschan and V. Rodehorst. Towards real-time stereo employing parallel algorithms for edge-based and dense stereo matching. In *Proc. of IEEE Int. Workshop on Computer Architectures for Machine Perception*, pages 234–241. IEEE, 1995.

[9] Z. N. Li. Stereo correspondence based on line matching in Hough space using dynamic programming. *TSMC*, 24(1):144–152, 1994.

[10] G. Medioni and R. Nevatia. Segment-based stereo matching. *Computer Vision, Graphics, and Image Processing*, 31(1):2–18, 1985.

[11] L. Robert and O. D. Faugeras. Curve-based stereo: Figural continuity and curvature. In *Proc. of CVPR*, pages 57–62. IEEE, 1991.

[12] M. Tomono. Robust 3D SLAM with a stereo camera based on an edge-point ICP algorithm. In *Proc. of ICRA*, pages 4306–4311. IEEE, 2009.

[13] I. Ulusoy, U. Halici, and E. Hancock. Probabilistic phase based sparse stereo. In *Proc. of ICPR*, pages 84–87 Vol.4. IEEE, 2004.

[14] J. Witt and U. Weltin. Robust Real-Time Stereo Edge Matching by Confidence-based Refinement. In *Proc. of ICIRA*. IEEE, 2012.