COMP90086 Final Project: Stereo Disparity

David Chin
University of Melbourne
912668

Philipp Habicht
University of Melbourne
1372069

***Abstract*— Stereo Vision is a highly researched topic within the field of Computer Vision, as it enables applications such as autonomous driving vehicles. In this project we calculate disparity maps, comparing different free parameters in a simple-block matching algorithm. We found that SSD, with a window size of 15 presents the best results, with an average RMSE of 18.35.**

Keywords—Computer Vision, Stereo Matching, Disparity

I. INTRODUCTION

Stereo vision aims to provide insights into a 3D model of a world from a pair of two 2D images from the same scene taken from two different viewpoints at the same time. This has been a long-studied area of computer vision, as the challenge of generating a 3D world model has many interesting applications, with probably the most notable being self-driving vehicles or robotic automation. One outcome of many stereo algorithms is a disparity map, from which a depth map can be calculated which completes the mapping from a 2D space to 3D space. Depth is inversely proportional to disparity, and can be calculated as follows [2]:

$$z = \frac{fB}{x - x'}$$

The general process of calculating disparity is as follows: for a given pair of points, (x_l, y_l), (x_r, y_r) which represents a pixel in the left and image respectively, find the distance between the two points such that they match the same point in the projected 3D world space. In the dataset provided, the images have been *rectified*, which means this distance is simply the horizontal distance, and we can come up with the following formula:

$$x_l = x_r + d$$

The difficulty in calculating the disparity lies in matching the correct pair of points, as occlusions, texture-less surfaces and ordering constraints can all lead to match failures. Reference [1] Attempts to build a taxonomy of approaches and categorizes algorithms that solve this problem into four areas.

- Matching Cost
- Cost Aggregation
- Disparity computation / optimization
- Disparity Refinement

This report will provide insight in our attempt to develop a matching cost algorithm, as well as a disparity computation / optimization algorithm. Our approach, experimental design and results will be discussed.

II. APPROACH

A. Dataset

The given dataset contains 25 pairs of simultaneously taken images (8-bit) from left and right viewpoints taken from a driving vehicle. They are parallel and separated along their x-axis. Additionally, 25 ground truth pictures (16-bit) of LiDAR generated disparity maps are given for validation. All images have a shape of 400x881.

Figure 1: Example of Data Set Images

In image pre-processing, because of easier handling, the pictures were transformed into grayscale because they only use one channel, whereas coloured images use three channels.

B. Algorithms

Our evolution of algorithms is as follows:

1. Simple Block Matching
2. Simple Block Matching with Search Window
3. Simple Block Matching with vectorized Scan Line Matching
4. Graph Cuts

1) Simple Block Matching

This was our first attempt at block template matching. The algorithm took around 30-40 minutes to run per image, and the disparity map generated was not feasible.

2) Simple Block Matching with Search Window

Following approach 1, the decision to add a search window was made to localize the template matching to a smaller window than across the entire Scan Line, reducing the number of computations. This dramatically speeds up performance, bringing the algorithm down to around ~2 minutes for a pair of images, and manages to produce a feasible looking disparity map.

The key parameters are:

1. Window Size
2. Scan Line Search Size
3. Loss Function

3) Simple Block Matching with vectorized ScanLine matching

This approach is like approach 2 but using a "vectorized" approach for matching the template in the search window. The performance of the algorithm improved from ~2 minutes to ~15 seconds, which was a speed up of around 8 times compared to approach 2, but outputs the same disparity map.

This is the approach used in running the experiments, which are discussed in the next section.

4) Kolmogorov and Zabih's Graph Cuts Stereo Matching Algorithm

The motivation behind using Graph Cuts is that it solves the hyper-local problem that simple block template matching suffers from. Rather than assign a pixel a disparity based on just the "data" term, that is, the differences in value intensities between the left and right image, Kolmogorov and Zabih's graph cut algorithm adds a smoothness, uniqueness, and occlusion term to form a global minimizable energy function, from which a disparity labelling can be extracted.

The energy function is minimized through finding the best labelling (which we found useful to think of as a function that maps pixels to disparity labels) through a series of "alpha-expansion" moves.

An attempt was made to re-implement the alpha-expansion graph cut algorithm from "Kolmogorov and Zabih's Graph Cuts Stereo Matching Algorithm". The approach taken was to read the paper, understand the different elements and using the provided C++ code from the paper, attempt to reimplement the algorithm.

Unfortunately, due to running out of time this was unable to be fully completed. Key considerations made were:

1. The original algorithm handles both grayscale and colour images. Ours was to focus on grayscale for simplicity.
2. KZ implements their own graph structure and maxflow algorithms, we use the PyMaxFlow library instead.
3. C++ Implementation may run a lot faster than our python implementation, and no guarantees of correctness are ensured as there was some ambiguity while translating from C++

III. EXPERIMENTS

A. Design

All experiments were run on a computer with Windows 10, 3.2GHz and 16GB of memory. Experiment results are summarized in the tables and charts below. Our choice for the free parameters were to test window sizes of 5, 15 and 25, while the Scan Line Search Window was set to 50 pixels. We also tested to see if there was any difference between using SSD and SAD loss functions. Only the results for the first 10 images in the dataset, sorted by name, are shown for brevity.

Because the ground truth images were 16bit, we down sampled the images (by dividing all pixel values by 256), which allows for sub-pixel disparity precision when the values are subtracted from the 8-bit left and right images.

When comparing to the ground truth images for error calculations, we first applied a Boolean mask to our calculated disparity map so as to only compare errors in places where the ground truth actually had values.

B. Experiment Results

1) Charts

The charts below show RMSE plotted against each ground truth images, with a mean showed in the label for each Loss function.

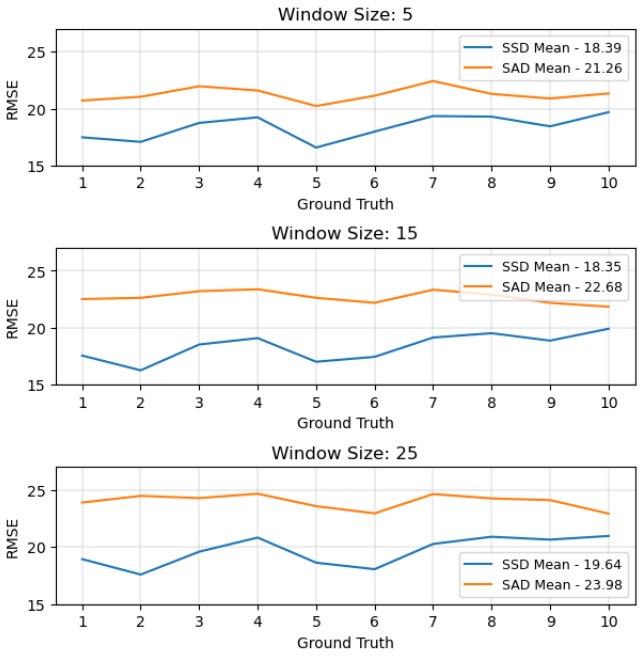

Figure 2: SSD/SAD RMSE Comparison

2) Sub Pixel Accuracy Tables

The fractions of pixels with errors less than 4, 2, 1, 0.5 and 0.25 pixels are shown in the tables below. The higher the number, the better, as this represents more accuracy in the generated disparity map.

TABLE I. SSD SUB PIXEL ACCURACY - WINDOW SIZE 5

Ground Truth Image	Sub Pixel Accuracy				
	< 4	**< 2**	**< 1**	**< 0.5**	**< 0.25**
1	0.47	0.37	0.26	0.16	0.09
2	0.47	0.38	0.27	0.16	0.07
3	0.45	0.37	0.27	0.16	0.09
4	0.44	0.35	0.24	0.15	0.08
5	0.47	0.38	0.28	0.18	0.09
6	0.44	0.36	0.27	0.17	0.09
7	0.43	0.35	0.25	0.15	0.08
8	0.46	0.37	0.27	0.18	0.10
9	0.49	0.39	0.27	0.16	0.09
10	0.43	0.35	0.26	0.16	0.08
Average	0.45	0.37	0.26	0.16	0.09

TABLE II. SSD SUB PIXEL ACCURACY - WINDOW SIZE 15

Ground Truth Image	Sub Pixel Accuracy				
	< 4	**< 2**	**< 1**	**< 0.5**	**< 0.25**
1	0.56	0.41	0.29	0.16	0.09
2	0.61	0.46	0.33	0.19	0.09
3	0.57	0.46	0.32	0.18	0.10
4	0.56	0.40	0.26	0.16	0.09
5	0.52	0.39	0.28	0.17	0.08
6	0.56	0.47	0.35	0.21	0.10
7	0.55	0.43	0.30	0.18	0.09
8	0.57	0.42	0.29	0.19	0.11
9	0.60	0.43	0.28	0.16	0.09
10	0.51	0.42	0.29	0.17	0.08
Average	0.56	0.43	0.30	0.18	0.09

TABLE III. SSD SUB PIXEL ACCURACY - WINDOW SIZE 25

Ground Truth Image	Sub Pixel Accuracy				
	< 4	**< 2**	**< 1**	**< 0.5**	**< 0.25**
1	0.51	0.36	0.24	0.12	0.06
2	0.58	0.43	0.29	0.17	0.08
3	0.53	0.43	0.28	0.15	0.08
4	0.52	0.36	0.21	0.11	0.06
5	0.46	0.33	0.21	0.12	0.06
6	0.56	0.45	0.32	0.18	0.09
7	0.51	0.39	0.26	0.15	0.07
8	0.51	0.36	0.23	0.14	0.08
9	0.52	0.36	0.23	0.12	0.07
10	0.48	0.39	0.25	0.14	0.07
Average	0.52	0.39	0.25	0.14	0.07

TABLE IV. SAD SUB PIXEL ACCURACY - WINDOW SIZE 5

Ground Truth Image	Sub Pixel Accuracy				
	< 4	**< 2**	**< 1**	**< 0.5**	**< 0.25**
1	0.24	0.14	0.08	0.04	0.02
2	0.23	0.13	0.07	0.04	0.02
3	0.19	0.11	0.06	0.03	0.01
4	0.27	0.17	0.10	0.05	0.03
5	0.24	0.14	0.07	0.04	0.02
6	0.19	0.10	0.05	0.02	0.01
7	0.18	0.10	0.05	0.03	0.01
8	0.23	0.13	0.07	0.03	0.02
9	0.25	0.15	0.08	0.04	0.02
10	0.19	0.10	0.05	0.02	0.01
Average	0.22	0.13	0.07	0.03	0.02

TABLE V. SAD SUB PIXEL ACCURACY - WINDOW SIZE 15

Ground Truth Image	Sub Pixel Accuracy				
	< 4	**< 2**	**< 1**	**< 0.5**	**< 0.25**
1	0.23	0.13	0.08	0.04	0.02
2	0.28	0.16	0.08	0.04	0.02
3	0.19	0.1	0.06	0.03	0.01
4	0.29	0.17	0.09	0.04	0.02
5	0.2	0.12	0.07	0.04	0.02
6	0.2	0.1	0.05	0.02	0.01
7	0.19	0.11	0.06	0.03	0.02
8	0.24	0.13	0.07	0.04	0.02
9	0.24	0.13	0.07	0.04	0.02
10	0.19	0.1	0.05	0.02	0.01
Average	0.23	0.13	0.07	0.03	0.02

TABLE VI. SAD SUB PIXEL ACCURACY - WINDOW SIZE 25

Ground Truth Image	Sub Pixel Accuracy				
	< 4	**< 2**	**< 1**	**< 0.5**	**< 0.25**
1	0.19	0.12	0.06	0.03	0.01
2	0.2	0.11	0.06	0.03	0.01
3	0.17	0.1	0.06	0.03	0.02
4	0.28	0.16	0.09	0.04	0.02
5	0.18	0.1	0.06	0.03	0.01
6	0.19	0.09	0.05	0.02	0.01
7	0.16	0.09	0.05	0.02	0.01
8	0.23	0.12	0.06	0.03	0.02
9	0.22	0.13	0.07	0.03	0.02
10	0.18	0.09	0.05	0.02	0.01
Average	0.20	0.11	0.06	0.03	0.01

3) Runtime

TABLE VII. AVERAGE RUNTIME

Window Size	Average Runtime Seconds	
	SSD	SAD
5	12.31	12.11
15	22.85	22.37
25	34.62	33.33

IV. RESULTS AND ANALYSIS

The key findings are:

1. SSD performs better than SAD for RMSE
2. Out of the tested sizes, the best window size is 15
3. Increasing the window size increases runtime

Our justification for SSD performing better than SAD is that SSD penalizes differences between windows in the two images, which may lead to less match failures and thus a lower RMSE. This is shown clearly in figure 2. SSD is marginally slower, by a fraction of a second, and is expected as it involves slightly more computation than SAD.

As shown in table 3, SSD with window size 15 had the highest average percentage of pixels with errors under all levels of error thresholds, with 56% of pixels being under a 4-pixel threshold, and the lowest mean RMSE as shown in figure 2.

According to [5], too small of a window size can introduce noise and wrong matches, while too large of a window size reduces matches but blurs over object boundaries. Our experiments support this argument, and as shown in figure 5, window size 5 produces quite a "grainy" image, while window size 25 is clearly more smoothed out. This may be a contributing factor to why our best performing window was 15.

From figure 2, we see that our algorithm (using SSD) performed best on image 2 and performed the worst on image 10. Looking at the two images, image 2 (shown in figure 3) has a lot more texture than image 10 (shown in figure 4), as in image 10 almost the entire half bottom of the image appears to be the same texture. Occlusions are appearing in both images, which would cause errors in both images. However, the lack of texture caused uniqueness match failures and thus it is probably this factor that contributes to the higher error rates in image 10.

The reason for the overall high error values is that the calculations for the disparity are independent for each scanline horizontally, and as such the algorithm has no sense of "vertical" disparity. No uniqueness constraint is enforced on matches along the scanline, nor is there any handling of occlusions at all - the algorithm will still find a match even if an occlusion exists. We would expect techniques such as [4] graph cuts or [3] dynamic programming to enforce some additional constraints on matches, which is a further improvement that could be made on our algorithms.

Figure 3: Image #2

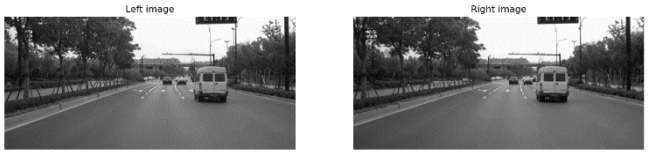

Figure 4: Image #10

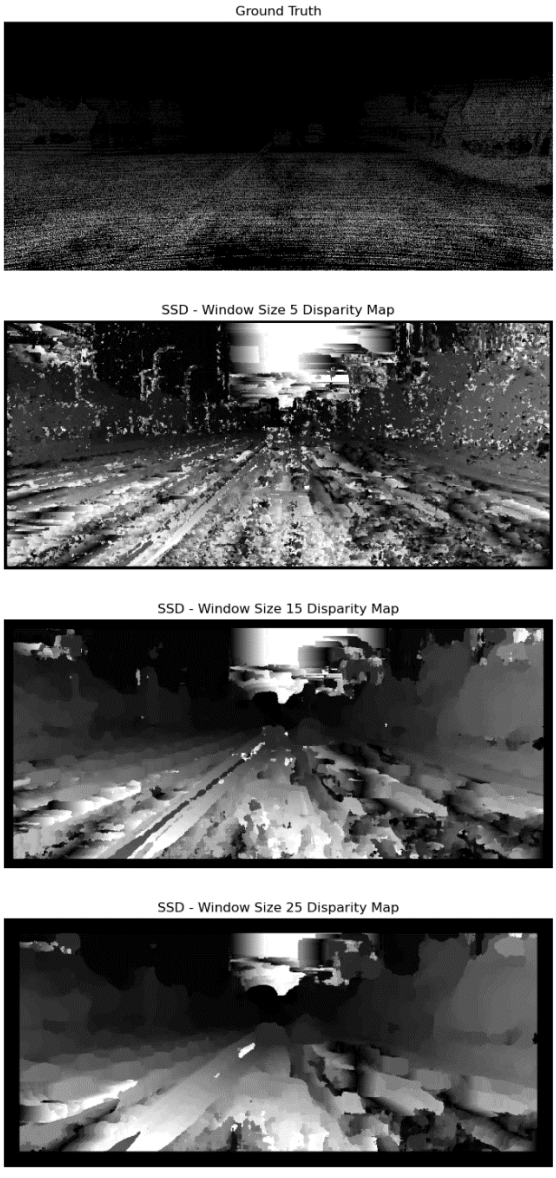

Figure 5: Example of Calculated Disparity Map – SSD

V. CONCLUSION

In this report, we attempted to calculate the disparity maps for some images in the DrivingStereo public dataset using block template matching techniques. We present our results along with some critical analysis. Future improvements include different loss functions, smoothing techniques, and different algorithms.

VI. REFERENCES

[1] D. Scharstein and R. Szeliski, “A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms.” [Online]. Available: https://vision.middlebury.edu/stereo/taxonomy-IJCV.pdf

[2] R. A. Hamzah, S. F. A. Ghani, A. F. Kadmin, M. S. Hamid, S. Salam and T. M. F. T. Wook, "Disparity map estimation uses block matching algorithm and bilateral filter," *2017 International Conference on Information Technology Systems and Innovation (ICITSI)*, 2017, pp. 151-154, doi: 10.1109/ICITSI.2017.8267934.

[3] Y. Ohta and T. Kanade, "Stereo by Intra- and Inter-Scanline Search Using Dynamic Programming," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-7, no. 2, pp. 139-154, March 1985, doi: 10.1109/TPAMI.1985.4767639.

[4] V. Kolmogorov, P. Monasse, and P. Tan, “Kolmogorov and Zabih’s Graph Cuts Stereo Matching Algorithm,” Image Processing On Line, vol. 4, pp. 220–251, Oct. 2014, doi: 10.5201/ipol.2014.97.

[5] Mariş, R. & Brad, Remus. (2017). A comparative study of Block Matching Optical Flow algorithms. TEM Journal. 6. 760-770. 10.18421/TEM64-16.