Stereo Disparity Report
COMP90086 Computer Vision 2022 Semester2

Yixuan Liu (980551)
School of Computing and Information Systems
The University of Melbourne
yihwang3@student.unimelb.edu.au

Yihan Wang (1056614)
School of Computing and Information Systems
The University of Melbourne
yixliu1@student.unimelb.edu.au

I. Introduction

This report focuses on the stereo matching problem. Using a non-deep learning approach, find the correspondence between two stereo images taken simultaneously by two parallel mounted cameras on board a moving vehicle, and output a disparity image. Its performance is also examined by comparing the disparity map with ground truth using different similarity measurements(i.e.SSD, ZSSD and ZNCC) and constraints(ex: smooth constraint). Images of 2 views and the corresponding ground truth disparity data are from [1].

II. Methodology

Our first task was to match all pixels of the left image to the corresponding right perspective stereo graph. Moreover, based on the assumption stated in [1] that the two cameras are placed in parallel planes, and the epipolar lines are therefore parallel, which reduces the search to a one-dimensional line. In other words, instead of comparing each pixel in the right view with the pixel in the left view, it is sufficient to search each corresponding line, i.e. search for the same x-coordinate value. That is, for a left view with coordinates (x, y), the coordinates must be (x', y) as long as the right view has a matching point. The matching point x' is select by dissimilarity values, which give a detailed explanation in the *Basic Stereo matching functions* subsection. Some constraints(i.e. smooth constraint) and optimization approaches are also included to improve the algorithm.

The second task is to create a disparity map between the 2 views, which is simply the coordinate difference between the best match point x' and test pixel x, namely $|x - x'|$. Then, using root mean square error, the fraction of pixels within some error range and the runtime compared our disparity map and the ground truth [1] to examine the performance of the algorithm.

A. Basic Stereo Matching Functions

To find out the matching point of a given point, counting the difference between these two pixel points is not sufficient enough. The basic idea is assuming the matching point will have a similar surrounding as the given point. Therefore, a squared window is formed on both left and right images taking the points to be compared as centers. The pixel in the right image with the most similar neighbors is served as the matching point of the given pixel. To ensure the testing pixel is placed in the center, the window size is restricted to odd. Different windows size is chosen and compared in the performance evaluation part. Due to the restriction of the initial settings, the windows are moved inside the image. The border image (with border w//2 size) would have no matching point and thus be treated as the missing value. In order to still make effective use of that information, padding is added around the image with a value of 0 and size of w//2 to guarantee each pixel in the left image will be treated as "center" for the selected window size.

Each pixel with coordinate (x,y) in the left image, as the center, is compared with all pixels in the right image which have the same height (i.e. y-axis value) in window formation. The best match points were selected using different criteria (i.e. SSD, ZSSD and ZNCC) and the x-coordinate of the best match point on the right image was denoted as x'.

1) SSD (Sum of square difference):

$$SSD = \sum_{i,j} (w_{i,j} - w'_{i,j})^2, \tag{1}$$

where w_{ij} is the window on the left image and $w'_{i,j}$ is the window in the right image. SSD evaluates how different the two given windows are by the sum of square error. Pixel windows with the minimum error thus less dissimilar will be selected along the sliding window.

2) ZSSD (Zero Mean Sum of square difference):

$$ZSSD = \sum_{i,j} \left((w_{i,j} - \overline{w_{i,j}}) - (w'_{i,j} - \overline{w'_{i,j}}) \right)^2 \tag{2}$$

Based on SSD, ZSSD scales the window information with mean. The mean filter removes the local intensity offset thus improving accuracy. It also selects the one with the minimum error along the sliding window.

3) ZNCC (Zero Mean Normalised Cross Correlation):

$$ZNCC = \frac{\sum_{i,j} (w_{i,j} - \overline{w_{i,j}}) \cdot \left(w'_{i,j} - \overline{(w'_{i,j}} \right)}{\sqrt{\sum_{i,j} (w_{i,j} - \overline{w_{i,j}})^2 \cdot \sum_{i,j} \left(w'_{i,j} - \overline{w'_{i,j}} \right)^2}} \tag{3}$$

NCC calculates the cross normalised correlation between two given windows in order to find similarity. What is better than SSD is that it is robust to some changes in lighting.

Like ZSSD, ZNCC scales the window by mean. It selects the maximum value of the sliding window.

What needs to be mentioned is that we use colored images instead of grey-scale ones to gain more similar information. For the calculation of the mean value, we use the mean of each window to serve as the window mean instead of the mean of the r,g,b values in each window separately.

B. Window size

Different window sizes will be tried in order to get better performance. We need to make a trade-off between window sizes, as smaller sizes are computationally faster and provide better detail but more error matching, while larger sizes are computationally slower and provide smoother results but lack detail.

C. Weight constraint

With the assumption that more closer the neighbor to the given center pixel more important the neighbor is, we give different weight to neighbors according to their distance to center. Therefore, we apply Gaussian kernel on each window [2].

D. Smooth Constraints

$$E(D) = E_d(D) + \lambda E_s(D) \tag{4}$$

$E_d(D)$ is the match quality used by the matching function and $E_s(D) = \sum_{\text{Neighbors i,j}} \rho(D(i) - D(j))$. Function ρ represents the criteria for smooth value of disparity map, which can take L1 norm, L2 norm or potts model. We use the L1 norm which is $\rho(D(i) - D(j)) = |D(i) - D(j)|$, where $D(i)$ and $D(j)$ are the value in the disparity map to guarantee adjacent pixels move about the same amount. In our case, we only choose the 4 neighbours, namely the up, down, left and right pixel. λ is a tuning parameter to make a trade-off between match quality and smoothness term. In our function, λ is served as a parameter, with default value 0.5.

E. Sub-pixel Accuracy

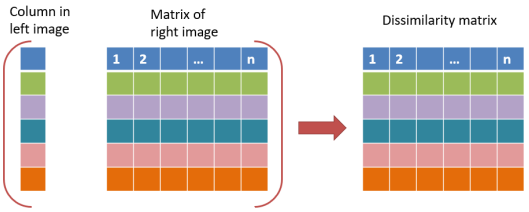
In previous calculations, matching accuracy was measured in units of 1 pixel. Sub-pixel accuracy is a way to improve the matching precision by modulating it to 0.5 or even less. After finding matching point x' by the above methods of comparing windows, we resize the window (with size m) with center x' to n times its size in order to form a new compared matrix M. We re-compare the matched point x with M by original window size m to find a more precise matching point. As the purpose is to find a more precise center coordinate, there is no need to compare x with border pixels. Therefore, we don't use padding here. The calculation of new coordinates is:
1). find the coordinates of the upper left corner (a,b) of x' as anchor points
2). find the new matching point (c,d) in M
3). recalculate the matching point (e,f) by

$$\begin{aligned} e &= c/n + a \\ f &= d/n + b \end{aligned} \tag{5}$$

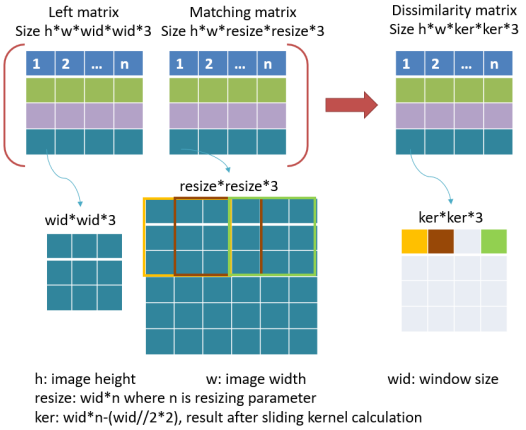
The resizing size n determines the preciseness (to 1/n pixel level). In our function, it is a parameter with default value = 5 thus 0.2 pixel level.

F. Acceleration

In our basic algorithm, we firstly pick the matched point x by two for loops, and then using one for loop to compare x with each x' with same height of x. Therefore, the time complexity is $O(m * n^2)$ where m is the height of image and n is the width. To accelerate it, we use tensor operation in numpy. For each time, we pick the whole column in left image (in window formation) and compare with the whole matrix of right image to find the dissimilarity values of the corresponding row for each value in this column. The time complexity decreases to $O(n)$



Also, for sub-pixel algorithm, we directly compare left image matrix with the corresponding matching point matrix after resizing (both in window formation) and do kernel calculation by sliding window. The time complexity is $O(ker^2)$ where the definition of ker is shown below.



In addition, according to physics, for a fix object point, its corresponding point on right image is appeared on the left of its on left image. Therefore, as we assume the phase difference between two images will not be far away, for each pixel on left image (x,y), we just search 100 pixels before x, i.e. we just search area [x-100, x] on the right image.

III. Test Statistics

A. The root mean square error

$$\text{rms} = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2}{N}} \tag{6}$$

We use the root means square error to measure the difference between the disparity map generate of the select algorithm and the ground truth, which is the general accuracy.

B. The fraction of pixels with error

Then also used the fraction of pixel with error to check the small value accuracy. In which we compared the fraction of error (the disparity map we generate an the ground truth) within 4 pixels, 2 pixels, 1 pixel, 0.5 pixel and 0.25 pixel respectively.

IV. Examination Result

Due to the page limit of the report, we only use a representative 12 of the 25 pairs of graphs, which is specified in the coding files. Moreover, for the single graph comparison, we have chosen a good performing pairs, namely pair taken at 2018-07-09-16-29-12-297,with the left view image as figure 1.

Fig. 1. The sample left view graph for testing performance

A. Runtime

The basic matching point found time without any acceleration techniques was 127.8 seconds with SSD and window size 7, without any constraints (i.e. weight constraints, smooth constraints and subpixel accuracy). For accelerated SSD is 5.8 seconds, which is more than ten times faster. Like what we mention in acceleration part, the execution time of the basic algorithm is exponentially longer than that of the accelerated algorithm. While to compare the execution time of different matching algorithms with different constraints, Table I shows the time to compute the disparity map with uniformly constrained parameters, i.e. a window size of 7, a Gaussian weights, a sub-pixel precision of 0.2 and a smoothing parameter of 0.5.

TABLE I
Runtime Comparison with window size 7

Settings in (sec)	Dissimilarity Method		
	SSD	***ZSSD***	***ZNCC***
None	6.4	10.7	13.3
weight	7.1	11.1	15.2
smooth	86.3	85.8	89.9
subpixel	82.2	86.7	249.1
weight + smooth + subpixel	163.1	167.4	328.2

Based on the statistics shown in table I, the SSD is the fastest, next come to the ZSSD and ZNCC. The extra time ZSSD spends over SSD is on calculating the mean value. While for ZNCC, except calculating mean value, it also spends much time on calculating its complex correlation calculation method. The execution time of adding weight is not much longer than when it is not added as it just multiplies a kernel weight on each window. However, the time consuming for smooth and subpixel are much longer, especially for subpixel. When adding smooth, we need to adding smooth constraint after getting the similarity differences and disparity map. Therefore, after the normal matching algorithm, we need to process on each pixel thus adding $O(m * n)$ time complexity (m, n are image size). While for subpixel, we firstly get the matching points, then resizing them and doing matching again. Even though we use multiplications between matrices, for each pixel, as we mentioned before, the time complexity is $O(ker^2)$, thus much time consuming.

B. Matching function performance

In most of the cases, ZSSD performs best, followed by SSD and ZNCC. Table II shown the value for sample figure (figure 1) for consistency random pick window size 7. The testing criteria has already been stated in *Test Statistics* section, where ε represent the fraction of pixel with some specific error between disparity map computed and the ground truth.

TABLE II
Basic Stereo Matching Functions Comparison (w = 7)

Method	Test Statistics					
	rms	$\varepsilon < 4$	$\varepsilon < 2$	$\varepsilon < 1$	$\varepsilon < 0.5$	$\varepsilon < 0.25$
SSD	**13.7**	**76.9%**	66.9%	60.2%	56.5%	54.5%
ZSSD	13.8	71.1%	60.3%	53.2%	49.5%	47.4%
ZNCC	18.3	66.6%	55.9%	48.8%	45.1%	43.2%

As we mentioned, ZSSD works better as it eliminates the local intensity by mean. However, before experiment, we expected ZNCC works better as it solves the lighting sensitivity issue of SSD, while in fact it is not the case. In our perspective, it is firstly due to the fact of the photo conditions. The corresponding photos were taken at the same time. Also, the two camera are placed horizontally and not far apart with no special obscurants. Therefore, there is no significant light changes. Additionally, we use colored images instead of grey-scale ones. ZNCC works better on images with obvious edges. In some extent, grey-scale images has more obvious edges than colored ones. Therefore, ZNCC works not well in our case.

As ZNCC did not have very good performance compared with the other matching functions as well as the longer processing time, in the next few section, we mainly focused on the comparison with SSD and ZSSD.

C. Window size

As we mentioned before, smaller window focused more on the detail, it would have more noise. In our cases, backgrounds such as skies, trees and roads take up many spaces in the graph. These large amount of pixels with pretty similar colors may lead to same matching point (ex: all ”blue” sky pixels match to the most similar ”blue” pixel in another image). Larger window size, as involving more surroundings, may solve the problem of multi-matching (ex: the surroundings of blue sky maybe trees or others). However, larger window size would result in a less detailed but more smooth disparity map, as compared by Figure2 and Figure3.

Fig. 2. The disparity heat map of the sample pair with simple SSD of window sized 3

Fig. 3. The disparity heat map of the sample pair with simple SSD of window sized 30

Therefore, window size need to be carefully selected. For each graph, we select the window size given the minimum rms from 7,15,31,51 and 71, and the best one for each algorithm is stated as in table III.

TABLE III
THE CHOSEN WINDOW SIZE WITH CORRESPONDING RMS

Figure Number	Window Size		RMS value	
	SSD	*ZSSD*	*SSD*	*ZSSD*
Figure1	15	15	10.2	10.9
Figure2	31	51	9.4	9.6
Figure3	71	51	10.3	11.2
Figure4	51	31	9.3	8.1
Figure5	71	31	11.8	10.1
Figure6	15	31	12.4	12.6
Figure7	31	31	9.6	9.4
Figure8	71	71	8.6	9.0
Figure9	71	71	9.9	9.0
Figure10	15	15	16.3	15.8
Figure11	51	71	9.6	9.7
Figure12	71	71	10.7	10.5

Most of the figure has lower rms with higher window size. For images with higher optimized window size, they generally have extensive fractions of background sectors (i.e. trees, roads, sky) thus needing more surrounding information. Otherwise, it can lead to multiple matches, resulting in inaccuracies. While if more objects (e.g. vehicles) are detected, window size would chosen to be smaller as it is enough to distinguish differences by surrounding. In such situation, large window size may introduce more noise.

However, the lowest rms does not represent the lowest error fraction. In most cases, ZSSD has better performance in RMS than SSD, but poorer performance in error fraction. It suggests that although ZSSD has lower average error, SSD is more "precise" and has more pixels that are close to the ground truth value.

D. Optimization constraints performance

Considering the runtime and performance, we generally select window size 7 in the following sections.

TABLE IV
SINGLE CONSTRAINT PERFORMANCE WITH WINDOW SIZE 7

Method and Constraint	Test Statistics					
	rms	$\varepsilon < 4$	$\varepsilon < 2$	$\varepsilon < 1$	$\varepsilon < 0.5$	$\varepsilon < 0.25$
SSD	13.7	76.9%	66.9%	60.2%	56.5%	54.5%
SSD+weight	15.1	75.7%	66.1%	59.3%	55.6%	53.7%
SSD+smooth	13.1	79.1%	71.5%	66.6%	64.0%	62.3%
SSD+subpixel	13.9	77.5%	68.0%	61.9%	58.7%	56.9%
SSD+all	13.2	79.1%	71.5%	66.6%	64.0%	62.6%
ZSSD	13.8	71.1%	60.3%	53.2%	49.5%	47.4%
ZSSD+weight	15.7	69.5%	59.2%	52.3%	48.7%	46.8%
ZSSD+smooth	12.7	76.9%	68.8%	63.7%	61.1%	59.7%
ZSSD+subpixel	13.8	72.2%	62.0%	55.7%	52.3%	50.5%
ZSSD+all	12.7	77.0%	68.9%	63.7%	61.1%	59.8%

1) Weight Constraint performance: The performance of the matching algorithm after adding Gaussian weight decreases for both RMS and fraction error. When matching, we use surroundings to detect if two points are matched. Also, in our case, the percentage of background is large. Therefore, for a background pixel center, we may use surroundings which is at the border of the window (thus out of the background) to find matching. The importance of surroundings should not be determined by distance (as the Gaussian kernel is a distance-based technique). It may be the reason why adding weight does not work in this case.

2) Smooth Constraint performance: There is a slight improvement for RMS and a considerable improvement in fraction error when adding smooth constraints. This is particularly evident when the window size is not properly selected. It implies that the constraint of adjacent pixels moving about the same amount is effective in matching. In other words, a partial sacrifice in similarity to maintain the same offset as much as possible is a good strategy for matching.

3) Subpixel Precision: The performance of subpixel is slightly improved for both RMS and fraction error. It indicates the effectiveness of improving precision. The offset between the left camera and the right camera is not necessarily 1-pixel-unit. Therefore, finer scoping can yield more accurate disparity.

V. CONCLUSION

With the consideration of both time consumption and performance, SSD is our prior choice among the three stereo matching functions. Moreover, if considering the constraints, the smooth constraint is preferable. It is the most cost-effective option in terms of the trade-off between performance and computational efficiency. However, window sizes need to be carefully selected for different pairs of graphs.

REFERENCES

[1] G. Yang, X. Song, C. Huang, Z. Deng, J. Shi, and B. Zhou, "Drivingstereo: A large-scale dataset for stereo matching in autonomous driving scenarios," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[2] "OpenCV: Image Filtering."