COMP90086 Project: Stereo Disparity

Haonan Lyu (1252105)
The University of Melbourne
Melbourne, Austrilia
halyu1@student.unimelb.edu.au

Jiachen Huo (1293736)
The University of Melbourne
Melbourne, Austrilia
jihuo@ student.unimelb.edu.au

***Abstract*—Stereo Matching task is a key component in many applications and fields such as Object Detecting, Autonomous Driving and Mobile Robotics. This matching task requires to calculate the stereo disparity among pictures and converting these 2D pixel differences into 3D depth. These pictures can be taken from multiple horizontally shifted positions, so they are parallel and separated along the x-axis. Common ways of solving such a disparity map could be adopting traditional matching cost and targeting for pixels' similarity matching, which is the main purpose of this research project of evaluating different classical approaches' performance. Take a step further than simple pixel's matching, a more robust matching algorithm named block matching is desired in case of noise and outliers. Another commonly used technique adopts deep learning framework which requires more computational resources and heavily depends on sufficient ground-truth training data, but on the other hand gives outstanding performance compared with simple classical matching cost approach. Among three matching functions we examined, strided ZNCC with smooth constraint turns out to have the best rms loss compared to the ground truth disparity map result of 11.74.**

Keywords—Stereo Matching, Disparity Map, 3D Depth, sub-pixel accuracy, rms

1 Introduction

How to implement the classical stereo block matching algorithm in a way that it's both computationally efficient and accuracy remains as a challenge. A stereo matching task is designed for recover the 3D depth from batch of 2D multi-view image sets or pairs, given that 2D images contain various information for depth perception. For a best match between pixel x in left image and x' in right image, the amount of horizontal distance in absolute value (x-x') is the *disparity* we are trying to recover. The 3D depth z is then inversely proportional to disparity once we have camera focal length f and baseline B, where B is distance between two cameras. Classical ways of computing disparity map depend on the choice of matching functions.

This report is structured as follow: Section 2 introduce some related work and background information about stereo matching problem. We propose our high-level framework pipeline and main methodologies in section 3. Detailed experimental settings and result analysis is then summarized in section 4. Lastly, we propose and discuss some future directions in section5.

2 Related works

Previous attempts for solving disparity map in a stereo matching task can be classified into three algorithms: the local, global and semi-global methods. Local method is when calculating disparity of a pixel based on the corresponding positioned pixel or neighboring pixels on the other image that together forms a stereo image pair. The local method evolved from pixel-based matching to block-based (or window-based) matching [1]. Considering of common sampling noise and varying lighting, block-based matching technique is able to overcome these input noise by using a similarity window from its surrounding neighbor pixels. Common pixel-based matching function could be absolute differences and squared differences. Common block-based matching function could be the summed version, SSD and SAD which stands for summed squared difference and summed absolute difference. However general resultant from SSD and SAD tends to have excess noise and low precision compared to a later matching function NCC, which stands for normalized-cross-correlation. Also, their zero-mean version ZSSD, ZSAD and ZNCC were established for compensation of a constant offset which would be the window mean.

3 Methodology

There are two working stages of this project, the first stage we adopt different matching functions from the local method mentioned above for constructing initial baseline version of computing disparity map. Then multiple optimizations are developed by reconstruct data structure for enabling GPU, adding weight kernel and smooth constraints. Then second stage we establish multiple evaluation matric including rms (root mean square) from ground truth disparity map, pixel-accuracy and sub-pixel accuracy.

A. Baseline Model

Using classical method to compute a disparity map for a two-view stereo image pair, with camera positioned parallel on x-axis and only horizontal shifts, includes a grid search that for every pixel x in the left image, we loop through all the column at same row with x and computing the similarity by using different matching functions. For a square-sized image pair, the run time is estimated to be $O(n^3)$ which means this matching process is extremely slow for large images as it has three loop iterations indicated in Algorithm 1. Generally an

odd window size is desired since we want to place the pixel x on the central of the window. Notice given this "convolutional-like" matching algorithm with stride =1, padding is required of original input image in order to have same-sized output. As indicated padding = window size //2 and common choices of padding are also provided by the open source *OpenCV* (e.g. constant padding, replicate padding etc.).

Algorithm 1 Baseline Model for Block Matching (e.g.SSD)

```
1: padding ← windowsize//2
2: for i in left image row do
3:     for j in left image column do
4:         l ← W(i, j)                    ▷ left window with pixel [i,j] at center
5:         for k in right image column do
6:             r ← W(i, k)                ▷ right window with pixel [i,k] at center
7:             compute SSD_{i,k}(l,r)
8:             index ← argmin_k SSD_{i,k}
9:             disparity_{i,j} ← abs(j − index)
10:        end for
11:    end for
12: end for
13: return disparity
```

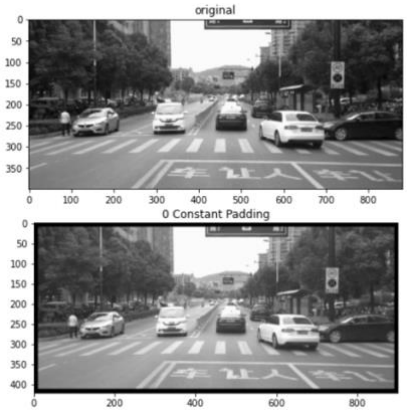

Fig 1. Constant (0) Padding Effect

The main structure of Baseline Model for Block-Matching remains the same except for the choices of matching functions at line 7. Thus, in this project we mainly conducted the implementation using SSD, NCC and ZNCC which using all pixels within the user-defined window size as fig 2.

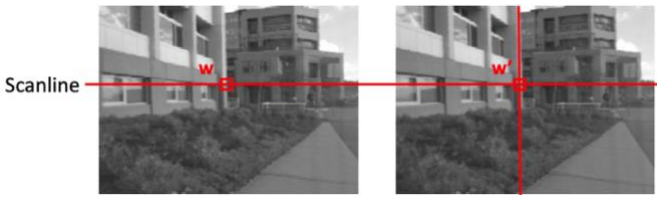

Fig 2. Block-Matching on Through Horizonal Axis

1) SSD (Sum of Squared Difference)

$$SSD = \sum_{i,j}(W_{i,j} - W'_{i,j})^2$$

2) ZSSD(Zero Mean Sum of Squared Difference)

$$ZSSD = \sum_{i,j}((W_{i,j} - \overline{W}_{i,j}) - (W'_{i,j} - \overline{W'}_{i,j}))^2$$

3) NCC (Normalized Cross-Correlation)

$$NCC = \frac{\sum_{i,j} W_{i,j}.W'_{i,j}}{||W_{i,j}||.||W'_{i,j}||}$$

4) ZNCC (Zero Mean Normalized Cross-Correlation)

$$ZNCC = \frac{\sum_{i,j}(W_{i,j} - \overline{W}_{i,j}).(W'_{i,j} - \overline{W'}_{i,j})}{||W_{i,j} - \overline{W}_{i,j}||.||W'_{i,j} - \overline{W'}_{i,j}||}$$

Where $W_{i,j}$ indicates matching window in left image with pixel [i,j] in center and $W'_{i,j}$ indicates matching window in right image with pixel [i,j] in center. For each of the zero mean version, $W'_{i,j}$_bar indicates the pixel mean within each window. Theoretically the zero-mean version should perform better than original version, and NCC should outstanding SSD since SSD is not good at handling plain texture, edge detection and occlusion. [2]

B. Gaussian Kernel

When we are dynamically altering the window size, we want make difference between more closer neighbor pixels and less close neighbor pixels so that local block-matching method can make the best out of it. This can be done by carrying out a Gaussian Kernel or convolve the input window with 2D gaussian kernel from *OpenCV*. This is equivalent to giving central pixels more weight than outer pixels so that not all pixels within a window are out of same importance, and theoretically would improve matching accuracy but take more time to finish.

C. Smooth Constraints

Generally pixels near each other have similar disparities values when using SSD or SAD, except for special cases of discontinued depth caused by object boundaries or occlusion. In this case the smoothing algorithm penalize sharp disparity changes by adding extra smoothing term to original SSD costs when SSD fails to find most suitable match.

$$E(D) = \sum_i (W_{i,j} - W'_{i,j})^2 + \lambda \sum_{heighborsk} \rho(D(i,j) - D(k))$$

E(D) referred to disparity value after smoothing constraint which we still want to minimize, and first term in the equation is how we get SSD values originally. The strength of smoothness is determined by λ with larger λ has larger penalization. ρ here means we are free to choose between different regularizations L1 norm or L2 norm. Choice of neighbors k is also worth of testing during experiments where in this project we simply use up, down, left and right pixels around pixel at [i,j]. Also, the matching accuracy is expected to improve after adding the smooth constraints.

D. Window Size

Window size also affects the result disparity map. As mentioned before an odd window size is desired where a trade-off exists between matching accuracy, granularity and runtime. Since the output disparity map remains the same, this

means the number of times we do matrix computation remains the same, thus larger window size requires more computational resource and runs slower but gives smoother output by omitting some of details. Where smaller window runs faster and gives finer detail but receives more noise.

E. Runtime Optimization

Without any runtime optimization the baseline model takes xxx to finish one image pairs result which is not a desired runtime so far, thus three optimization approach we carried are:

a) Reform Data Structure: We adopted the as_strided() function from numpy.lib.stride_tricks to accelerate the matrix computation process.

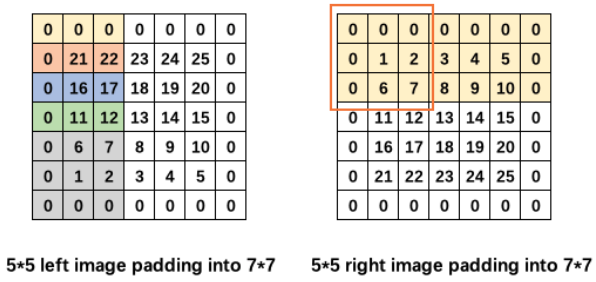

Fig 3. Input Image Pair to be as_strided ()

Suppose the input image is 5*5 with window size 3*3, then padding is required to be 3//2 = 1 as fig 3 indicated. We then first apply as_strided () function to right image as fig 4, which turns the original 7*7 sized input into 5*5*3*3 sized strided matrix. Notice each 3*3 window has it's center colored red, this is the original pixel [i,j] on input image and this "element-wise" matching result is then of same size as input 5*5. The runtime is then effectively reduced from $O(n^3)$ to O(n) by eliminating two for loops and only left with looping through left image's column.

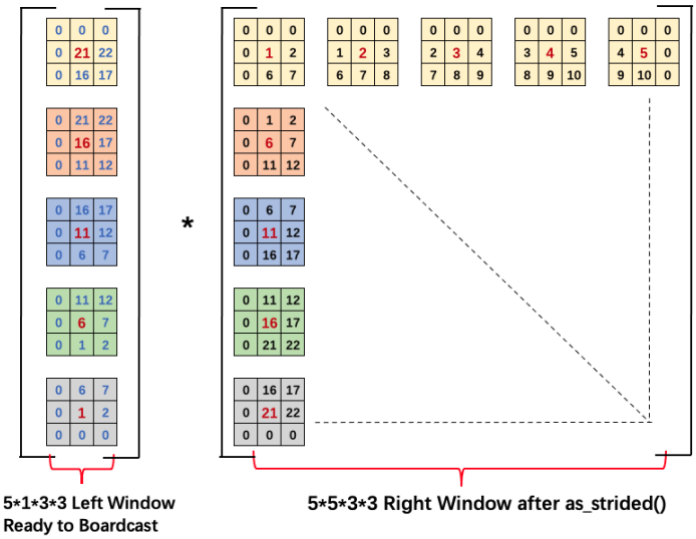

Fig 4. Broadcast of left image column to right strided image

b) Enabling GPU: Further optimization on runtime is developed by turning numpy object into torch tensor once we successfully reform the data structure as part a). The main logical coding remains almost the same except for special declaireation by adding .cuda().

c) Search Trick: Since we know the existence of disparity d >0, this means for every pixel [i,j] on the left image, the best-matched pixel must appear on some j'-th column before j-th column, i.e. |j-j'| >0 (E.g. imagine several left-most columns on left image would be "cut-off" and disappear from right image). Thus we can reduce our matching range by eliminating columns appear after j-th column.

F. Sub-Pixel Accuracy

Currently we are representing disparity as minimum of unit 1 since index of pixels are taken as whole numbers, but generally disparity should be a continuous value where decimal values are allowed. To allow such a sub-pixel accuracy we do pixel interpolations when we already have a best match point x' on the right image. This is supported by *OpenCV* using *resize ()* with a declaration of interpolation flag.

For example an original 3*3 pixel-sized window with resize factor of 5 would result in a still 3*3 pixel-sized window but 15 knots each axis we name this new 15*15 knots-sized window X. After interpolation we slide a 3*3 knots-sized window within X to find new best matching point x'', normally the coordinates of x'' is now a float number [i',j'], so we use abs(j-j') to represent new disparity value, that's how we get sub-pixel accuracy.

4 Results and Evaluation

The evaluation of previous methodologies and models are mainly from these dimensions: runtime, matching accuracy with different models, matching accuracy with varying window size.

A. Runtime

Table 1 indicates runtime criteria of different model settings based on one set of image pair. The device for running following result is CPU i7-12700H, and GPU 3060. Several important notices are:

- Only SSD has been experimented with baseline model (The 3-for loop version), without any runtime optimization strategy and GPU, this takes more than 9 minutes to finish.
- Comparing different matching functions, on average their runtime from slow to fast as: ZNCC > ZSSD>SSD. As ZNCC has two more norm calculations than ZSSD and SSD, thus ZNCC generally takes more time for all ZNCC-related models.
- Comparing different model setting between strided, strided +Gaussian and strided + smooth, average time needed to complete running orders as: strided + smooth > strided + Gaussian > strided. Especially with strided + smooth is 20~40 times slower than single strided model. When adding smooth constraints, we have to first generate a disparity map

then loop through all pixels to adding the neighbor-smooth-constraints, that's why models with smooth exceeds 300s. Also Strided model with Gaussian only takes extra 1~2 s which falls in acceptable range.

Model Choice		Avg run time (s)
SSD	baseline	561.25
	strided	7.75
	strided + Gaussian	8.50
	strided+smooth	351.00
ZSSD	strided	16.00
	strided + Gaussian	17.25
	strided+smooth	319.00
ZNCC	strided	18.00
	strided + Gaussian	19.75
	strided+smooth	311.00

Table 1. Runtime of Different Model

B. *Matching Accuracy with Different Window Size*

As fig 5 indicated, the rms decrease as window size increase where we choose 4 set of image pairs to test, which justified our assumption that larger window size gives smoother output and better accuracy. Also notice from the error fraction results, generally fraction of pixels with error less than 0.25 is the hardest to achieve ranges within 25%, where fraction of error < 4 is a easier goal ranges from 40% to more than 80% as window size increases. The parameter tuning of window size and these graphic result gives a best window size of 13. Notice due to page limit only two set of image pairs' graphical fraction accuracy results are presented here, but generally all image pairs follow the same trend.

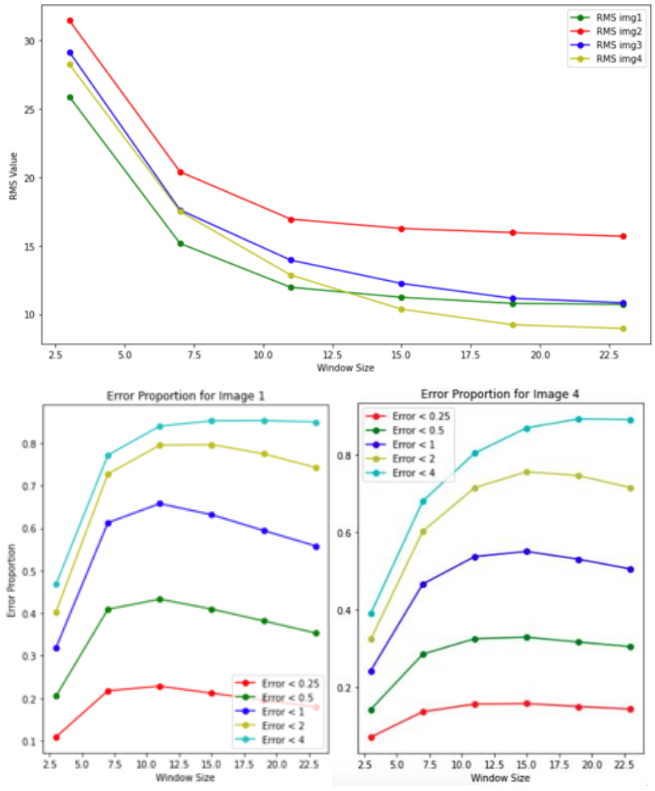

Fig 5. Matching Accuracy with Different Window Size

C. *Matching Accuracy with Different Models*

Table 2 is a result of matching accuracy with different models, several important conclusions can be drawn from here:

Model Choice		Test Statistics					
		rms	<0.25	<0.5	<1	<2	<4
SSD	baseline	100.32	16.53%	32.07%	50.17%	63.68%	69.52%
	strided	16.89	16.61%	32.24%	50.58%	64.48%	70.75%
	strided + Gaussian	19.28	14.36%	27.80%	43.90%	55.08%	61.35%
	strided+smooth	**16.81**	**16.67%**	**32.40%**	**50.81%**	**64.72%**	**70.98%**
ZSSD	strided	13.03	19.67%	38.06%	59.86%	77.43%	84.18%
	strided + Gaussian	18.68	15.24%	29.44%	46.18%	57.39%	63.36%
	strided+smooth	**12.76**	**19.82%**	**38.34%**	**60.28%**	**77.94%**	**84.66%**
ZNCC	strided	12.51	19.80%	38.31%	60.31%	77.71%	84.44%
	strided + Gaussian	17.64	17.94%	34.27%	53.32%	65.80%	71.40%
	strided+smooth	**11.74**	**19.92%**	**38.48%**	**60.43%**	**77.69%**	**84.49%**

Table 2. Matching Accuracy with Different Models

- The baseline model using SSD has rms of around 100 which is 5 times higher than strided version but has similar error fractions. This is because we didn't apply the search trick as mentioned in optimization methodology part which gives chance of matching to noisy pixels or outliers. SSD is easily affected by any outliers and noise, if one valid ground truth is different from the disparity map by say 10, then this easily caused a rms of 100.
- Comparing between three matching functions using rms criteria, the matching quality decreases as ZNCC > ZSSD > SSD which is insist with previous theoretical conclusion that the zero-mean version works better than original, and ZNCC works better than ZSSD. Also the bolded row represents the best model within each matching function which turns to be strided + smooth.
- Comparing between five fractions of error pixels, more finer accuracy is more harder to achieve as the best fraction of error less than 0.25 is 19.92% achieved by ZNCC's strided + smooth, and the best fraction of error less than 4 pixels is 84.66% achieved by ZSAD's strided + smooth.

5 Error Analysis and Further Improvements

- Notice from table 2 that within each matching functions, the smooth constraints didn't actually improve the matching quality but rather worsen it. This is because we used the default setting of gaussian kernel supported by *OpenCV* which makes pixels near the window border diminishing approach to zero. For example a 9*9 window size may be weaken to only central 5*5 useful pixels. This is typically equal to we offset the effect of initial padding purpose which is not a desired optimization effect we are expecting. That's the main reason we have rms for Gaussian kernel optimization increased. However, the idea of adding weight to pixels within the search window for highlighting closer pixels is still valid. Future potential directions can be to replace Gaussian Kernel with mean filter, or a smoothed second derivate of Gaussian kernel (i.e., Laplacian).[1]
- Another further improvement could be carried out is during the implementation of smooth constraints, where current smooth penalty term is not transformed into matrix calculation instead of for-loops.

Reference

[1] Hirschmuller, H. and Scharstein, D. (2009) “Evaluation of Stereo Matching Costs on Images with Radiometric Differences”. IEEE Transactions on Pattern Analysis and Machine Intelligence, 31, 1582-1599.

[2] Dematteis, N., & Giordan, D. (2021). Comparison of digital image correlation methods and the impact of noise in geoscience applications. Remote Sensing, 13(2), 327. https://doi.org/10.3390/rs13020327