Image Understanding – Assignment #2

Stereo Matching

1. Part I-Disparity Map Calculation

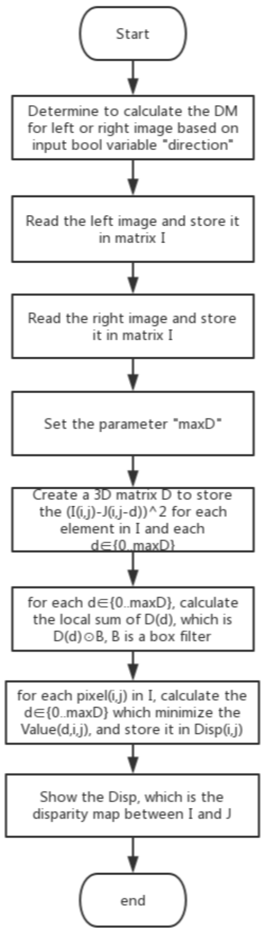
1.1 Organization of the code

The main function is DisparityMap.m. The information of this function is shown in Table 1.

Table 1. Description of DisparityMap Function

Name	DisparityMap		
Description	The function takes two input images and a window size to calculate the disparity map for the one input image based on another input image.		
	Name	Type	Description
Input	*pathLeft*	string	the path and name of the given left image
	pathRight	string	the path and name of the given right image
	windowSize	int	the window size of the box filter
	direction (optional)	bool	The default value is true. If the input is false, the output is the disparity map of right image based on the left image
Output	*Disp*	matrix	the disparity map of one image based on another image

The flow chart of how to calculate the disparity map is shown is Fig 1.1



Start
Determine to calculate the DM for left or right image based on input bool variable "direction"
Read the left image and store it in matrix I
Read the right image and store it in matrix I
Set the parameter "maxD"
Create a 3D matrix D to store the (I(i,j)-J(i,j-d))^2 for each element in I and each d∈{0..maxD}
for each d∈{0..maxD}, calculate the local sum of D(d), which is D(d)⊙B, B is a box filter
for each pixel(i,j) in I, calculate the d∈{0..maxD} which minimize the Value(d,i,j), and store it in Disp(i,j)
Show the Disp, which is the disparity map between I and J
end

Fig 1.1 Flow chart for disparity map calculation

Also, there are several helper functions which are called by the main function. Below is the description of those functions.

Table 2. Description of FastBoxFilter Function

Name	FastBoxFilter
Description	Calculate the convolution of the given matrix and a box filter. This function doesn`t actually calculate the convolution result and can run in time independent of the window size.

Input	Name	Type	Description
	I	matrix	the image needed to be convoluted
	s	int	s is a parameter of windowsize, s = (windowsize-1)/2
Output	*J*	matrix	the result image after convoluted by the box filter

Table 3. Description of IntegralImage Function

Name	IntegralImage		
Description	Calculate the integral image of the input image		
Input	Name	Type	Description
	I	matrix	the input image
Output	*J*	matrix	the integral image of input image

Table 4. Description of BoxFilterSum Function

Name	BoxFilterSum		
Description	Calculate the sum of elements within a given window based on the integral image		
Input	Name	Type	Description
	J	matrix	the integral image
	x1	int	x coordinate of upper left corner of the window
	y1	int	y coordinate of upper left corner of the window
	x2	int	x coordinate of bottom right corner of the window
	y2	int	y coordinate of bottom right corner of the window
Output	*sum*	matrix	the sum of elements within the given window

1.2 Fast Convolution method for box filter

To calculate the convolution between a matrix *"I"* and a box filter, which runs in time independent of the window size, several steps are used:

Step 1: Generate an integral matrix *"J"* to store the sum of all elements in *"I"* in range of $(1:i, 1:j)$ in $J(i,j)$.

First, calculate the size of the first matrix and save in variable "*rowNum_I*" and "*colNum_I*".

Second, copy the element of matrix *"I"* to *"J"*.

Third, calculate the first column and first row of *J* to deal with boundary.

Last, calculate the sum of all the elements with subscripts smaller or equal to *i* and *j*, and store it in *J(i,j).*

$$J(i, j) = J(i, j)+J(i-1, j)+J(i, j-1)-J(i-1, j-1)$$

Step 2: Calculate the local sum of each element in "*I*" and store it in matrix *J*

In order to reuse the function BoxFilterSum, the upper left corner coordinate and bottom right corner coordinate of the window is needed as parameters. Because it may be used under the condition that the window is not centered at *(i,j)*

To deal with the boundary condition, if the upper left corner is out of upper or left boundary, the value will be set by 0. If the bottom corner is out of bottom or right boundary, the element at the corresponding boundary of integral image *J* will be used instead.

Since in the integral matrix *J*, *J(i,j)* stores the sum of all elements with subscripts smaller or equal to *i* and *j*. The sum of the window can be calculated by:

$$\mathbf{Sum} = J(y2, x2) + J(y1-1, x1-1)-J(y2, x1-1)-J(y1-1, x2)$$

Apparently, this implement can run in time independent of the window because it only needs to calculate the sum of the given matrix *I*.

1.3 Evaluation of the disparity maps

Usually, the calculated disparity maps are evaluated by comparing with the given ground truth map. Since the ground truth map is not available, the evaluation needs to focus on the disparity maps themselves.

Different window size, namely 3, 9, 15, 21, are selected to test the effect of the window size.

Below are the test results.

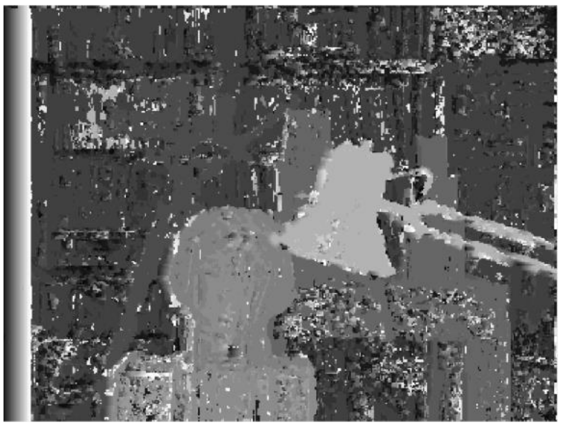


Fig 1.2 Disparity Map of "tsukuba"(window size =3)

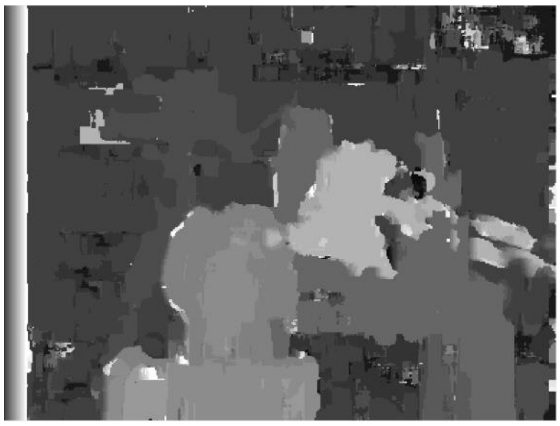


Fig 1.3 Disparity Map of "tsukuba"(window size =9)

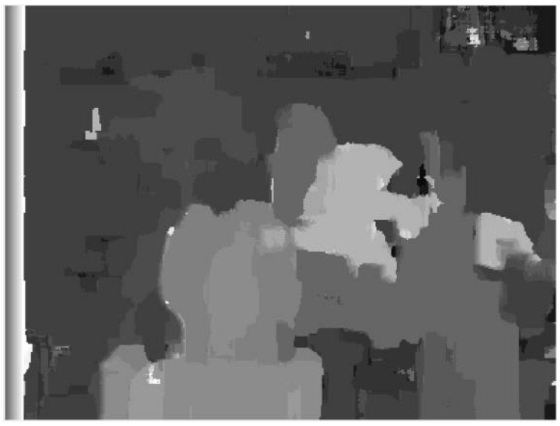


Fig 1.4 Disparity Map of "tsukuba"(window size =15)

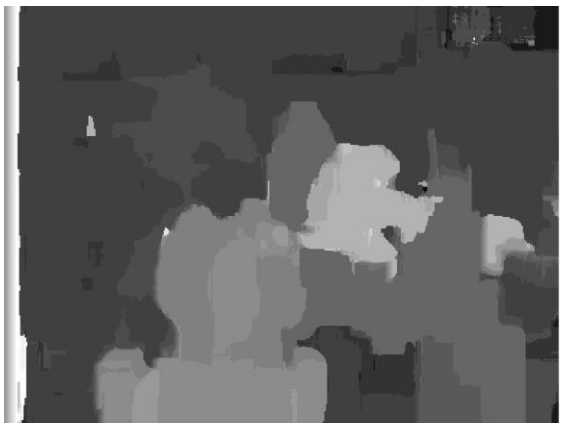


Fig 1.5 Disparity Map of "tsukuba"(window size =21)

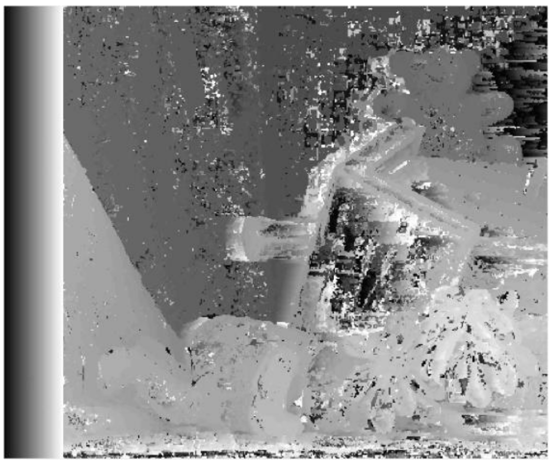


Fig 1.6 Disparity Map of "teddy"(window size =3)

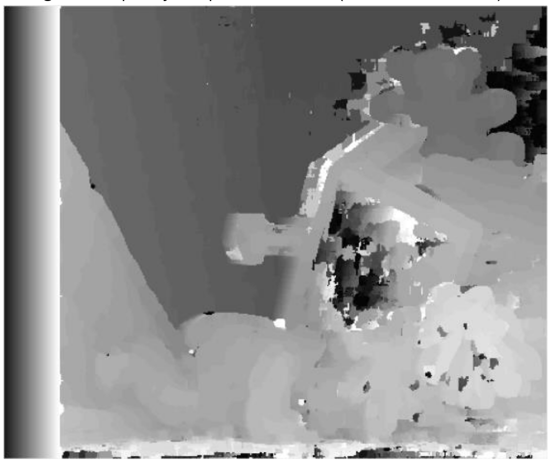


Fig 1.7 Disparity Map of "teddy"(window size =9)

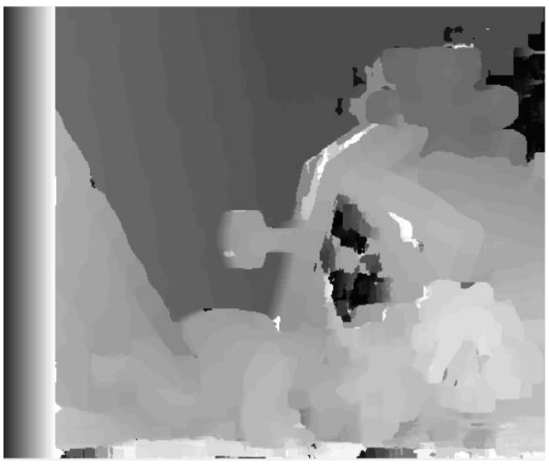


Fig 1.8 Disparity Map of "teddy"(window size =15)

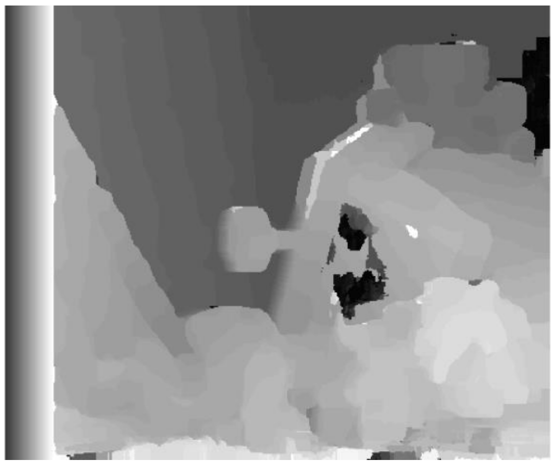


Fig 1.9 Disparity Map of "teddy"(window size =21)

According to the test results, as window size increase, the disparity map becomes more and more smooth, however, more detail is lost. For example, in the disparity map of "tsukubai", the post of table lamp is visible when the window size is small. If the window size increases, the post can be no longer found.

It`s hard for me to give an evaluation standard. I tried to calculate the stand deviation the disparity maps generated by different window sizes. Apparently, as the window size increases, the disparity map becomes much smoother and has better consistency. I tried to compute the disparity map of right image based on left image, however I can hardly draw some conclusion by comparing two disparity maps. In my opinion, there isn`t an optimal size.

2.Parts 2 – Optimization of the Disparity Map Calculation Method

2.1 Introduction of the optimization method

According the test results in Part 1, there are several disadvantages of the origin disparity map calculation method:

1. There is always some "useless data" in the left side of the disparity map, making the boundary quite ugly;
2. Since the window size is fixed, the total quality of the disparity map is not very satisfying. If the window size is big, detail is lost and the results near the boundary are not accurate enough. If the window size is small, there is a lot of noise in the disparity map.

In order to optimize the disparity map calculation, several methods are used:

1. Extend the boundary of the input image by copying the pixels on the boundary.
2. Set a threshold for evaluating the "texture" degree within a window. If in the window there is little texture, in other words, within the window the consistency of the image is quite good, different window size and location will be tested to reach the threshold. If the texture in the window is good enough, smaller window size will be used to preserve more detail.
3. In the boundary of the image, smaller window size is preferred to get an accurate result.
4. If a window matches quite well, all the elements in the window will share the same local sum of this window.

2.2 Implement of selecting window size automatically

The information of SelectWindow function is shown in Table 5.

Table 5. Description of SelectWindow Function

Name	SelectWindow		
Description	Calculate the suitable window for each pixel in input image		
Input	Name	Type	Description
	I	matrix	the input image
	s	Int	the given window size
Output	*D*	3Dmatrix	the suitable window information for each pixel in the input image I.

The basic idea is the calculate the standard deviation of the elements within the whole window to evaluate how much texture there is in the window. A threshold is used to evaluate the degree of texture. If the calculated standard deviation is above threshold, it means within the window there is enough texture and the window may need to be shrined to preserve more detail. If the calculated standard deviation is smaller than the threshold, it means within the window there is not enough texture and the window should be expanded to improve the accuracy of the stereo matching.

Besides, the window is separated into 4 parts, namely the upper left part, upper right part, bottom left part and bottom right part. Each part is calculated for the standard deviation to evaluate the degree of texture. Also, which part has most texture and which part has least part are recorded.

If the window needs to be expanded, it will be expanded in the direction where most texture exists. If the window needs to be shrined, it will be shrined in the direction where least texture exists.

The flow chart of SelectWindow function is shown in Fig.2.1.

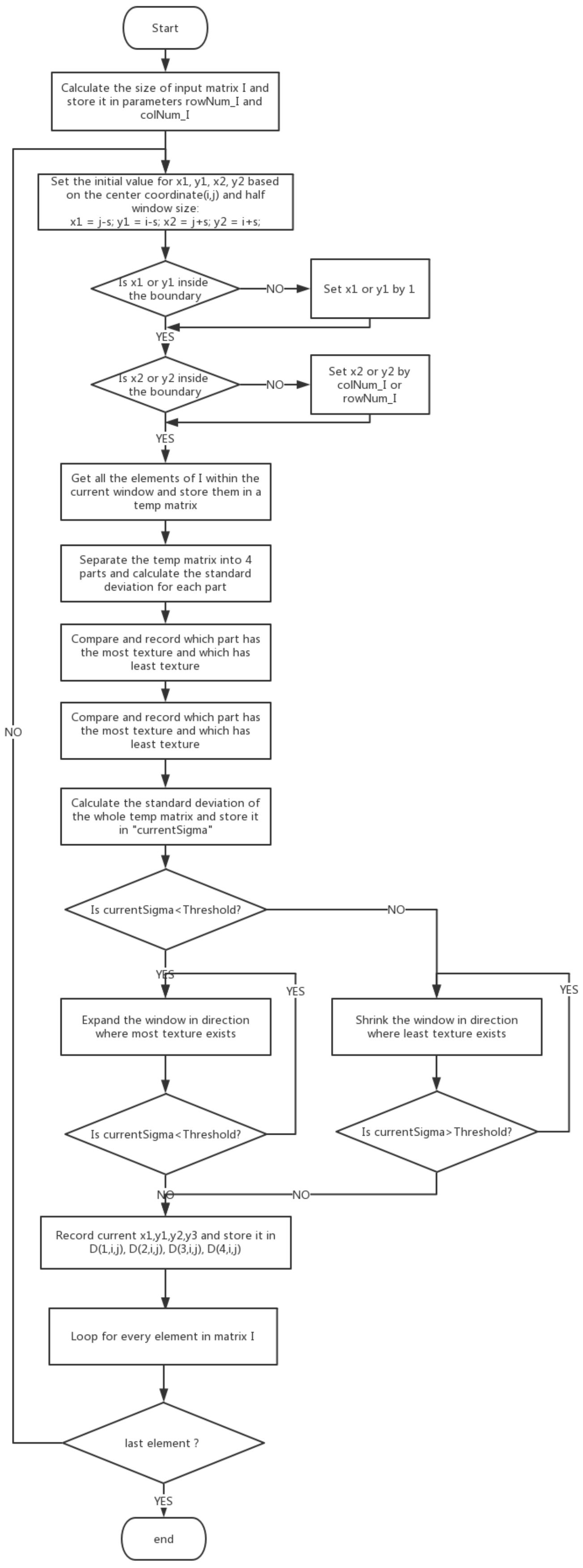

Start
Calculate the size of input matrix I and store it in parameters rowNum_I and colNum_I
Set the initial value for x1, y1, x2, y2 based on the center coordinate(i,j) and half window size:
x1 = j-s; y1 = i-s; x2 = j+s; y2 = i+s;
Is x1 or y1 inside the boundary
NO
Set x1 or y1 by 1
YES
Is x2 or y2 inside the boundary
NO
Set x2 or y2 by colNum_I or rowNum_I
YES
Get all the elements of I within the current window and store them in a temp matrix
Separate the temp matrix into 4 parts and calculate the standard deviation for each part
Compare and record which part has the most texture and which has least texture
Compare and record which part has the most texture and which has least texture
Calculate the standard deviation of the whole temp matrix and store it in "currentSigma"
Is currentSigma<Threshold?
NO
YES
Expand the window in direction where most texture exists
YES
Is currentSigma<Threshold?
Shrink the window in direction where least texture exists
YES
Is currentSigma>Threshold?
NO
NO
Record current x1,y1,y2,y3 and store it in D(1,i,j), D(2,i,j), D(3,i,j), D(4,i,j)
Loop for every element in matrix I
last element ?
NO
YES
end

Fig 2.1 Flow chart for selecting suitable window

2.3 Other modification

First, the new version of main function is “DisparityMap_Optimized.m”, and it is modified to call the SelectWindow function to calculate the suitable window information for each pixel in matrix *“I”*. Then main function uses the calculated windowInformation as a parameter to call FastBoxFilter_Optimized function to calculate the local sum within the suitable window for each element.

Second, to deal with the boundary problem mentioned in chapter 2.1, the image *“I”* is expanded by copying the left boundary.

Third, FastBoxFilter_Optimized function is also updated. A threshold value is added to evaluate the matching degree of a window. If the window matches quite well, the local sum of this window will be relatively low. If the local sum of the window is smaller than threshold, then all the elements in this window will be assigned by this low local sum.

2.4 Optimization result

Below are results of the optimized disparity maps.

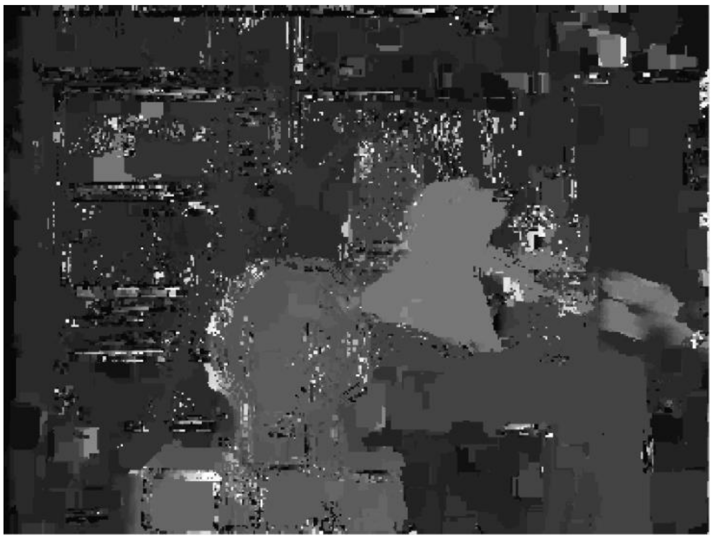


Fig 2.2 Optimized Disparity Map of “tsukuba”(window size =9)

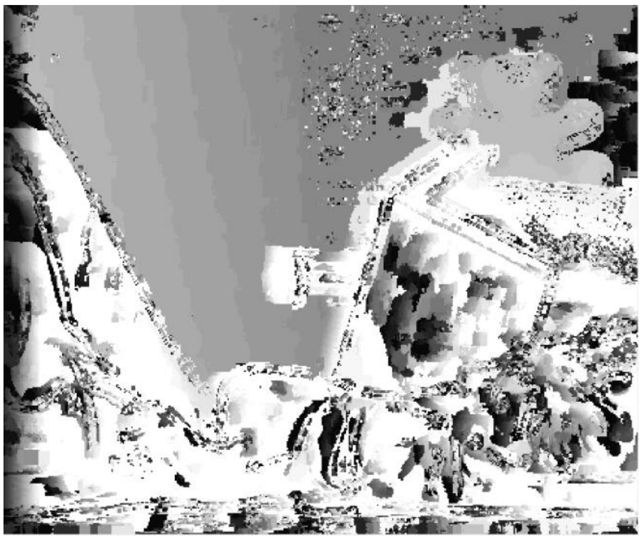


Fig 2. 3 Disparity Map of “teddy”(window size =9)

Compared with the old disparity maps calculation, the optimized disparity map can preserve more detail. Meanwhile, compared with Fig.1.2 and Fig.1.6, there is less noise in the disparity map. Basically, the optimization improves the old algorithm by reducing noise and preserving detail.