AI6121 Assignment 2 Disparity and Depth

YU YUE, G2202151A
JU XILAI, G2202544B
LUO HAO, G2202279H

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2022

Contents

Abstract

In the field of stereo vision, stereoscopic ranging techniques are widely used to estimate a 3D model of a scene, in which a popular ranging technique named triangulation is employed to compute the depth from 2D images. Disparity is a key parameter in this technology, which is inversely proportional to the depth. Disparity can be computed from the correspondence points of two images of the same scene that are captured from two different viewpoints. In our assignment 2 *Disparity and Depth*, we mainly use an Appearance-based Matching technology called Sum-of-Squares Difference (SSD) to compute disparity given a pair of rectified images of the same scene captured from two different viewpoints. All the code can be download from github

Keywords: Stereo Vision, Disparity and Depth.

List of Figures

Chapter 1

Disparity Computing

In stereo vision field, for each pixel in an image captured, a disparity value and a depth value are associated with it. Simply put, the depth of an object is the distance of the object being captured from the camera. To compute the depth of objects in the image, we usually compute the disparity of each object on a pair of rectified images of the same scene captured from two different viewpoints, and then with the known focal length and the distance between two viewpoints, we can compute the depth of the objects indirectly. The figure 1.1 shows the simple 2D triangulation expression.

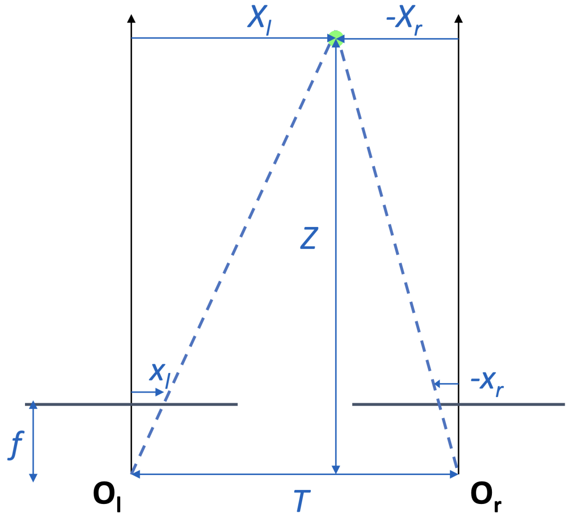


Figure 1.1: Simple 2D triangulation

In the figure 1.1, we assume a camera translates by T along x-axis in camera frame, O_l and O_r are two viewpoints and Z is the depth we want to estimate. The focal length is f and x_l, $-x_r$ are displacement distances of the object relative to the center of the image. By triangulation, we know that $x_l = \frac{fX_l}{Z}$ and $x_r = \frac{fX_r}{Z}$, then $x_l - x_r = \frac{f(X_l - X_r)}{Z} = \frac{fT}{Z}$, then the depth:

$$Z = \frac{fT}{x_l - x_r}$$

, in which the disparity $d = x_l - x_r$. With the known focal length f and the distance T, if we can compute the disparity d, then the depth can be computed correspondingly. Another way of interpretation is that translating camera to the right by T is equivalent to leaving the camera stationary and translating the world to the left by T. We know that the disparity are smaller the farther an object is from the camera when it moves the same distance. In summary, we can compute disparity to estimate the depth of an object captured.

Chapter 2

Sum of Absolute Difference

2.1 Introduction to SAD

Let's say we have a pair of rectified images of the same scene captured from two different viewpoints. From two points in the left and right images, we can reconstruct the 3D point which assumes that the two 2D points are image of the same 3D point. Then the problem becomes the correspondence problem: for each important image point in the first image, we need to find the corresponding image point in the second image, and then compute the disparity between these points.

We can assume that the corresponding points in the two images have image patches that look almost identical. Then the first correspondence problem solution is Sum of Absolute Difference (SAD), which is an Appearance-based Matching algorithm to measure the similarity between image blocks (show in figure 2.1). SAD is calculated by taking the absolute difference between each pixel in the original block and the corresponding pixel in the block being used for comparison [2]. Let's say, for a small image patch $g[(2N+1)\times(2N+1)]$ in the first image, we can find the corresponding image patch centered at (x,y) with the lowest Sum of Absolute Difference in the second image I:

$$\arg\min_{(x,y)} \sum_{u=-N}^{N} \sum_{v=-N}^{N} |I(x+u,y+v)-g(u,v)|$$

We assume that the corresponding patches in the two images only have horizontal displacement. Then the horizontal distance between the matched patches in the two images is just the disparity.

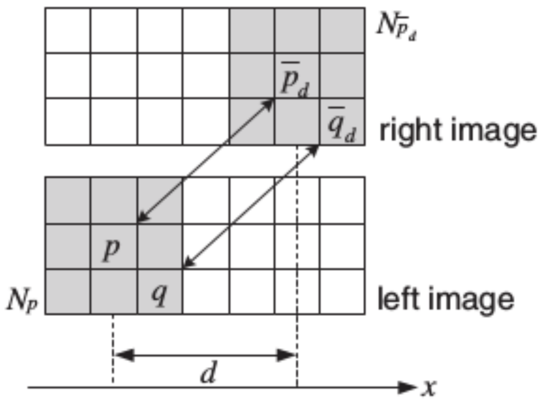


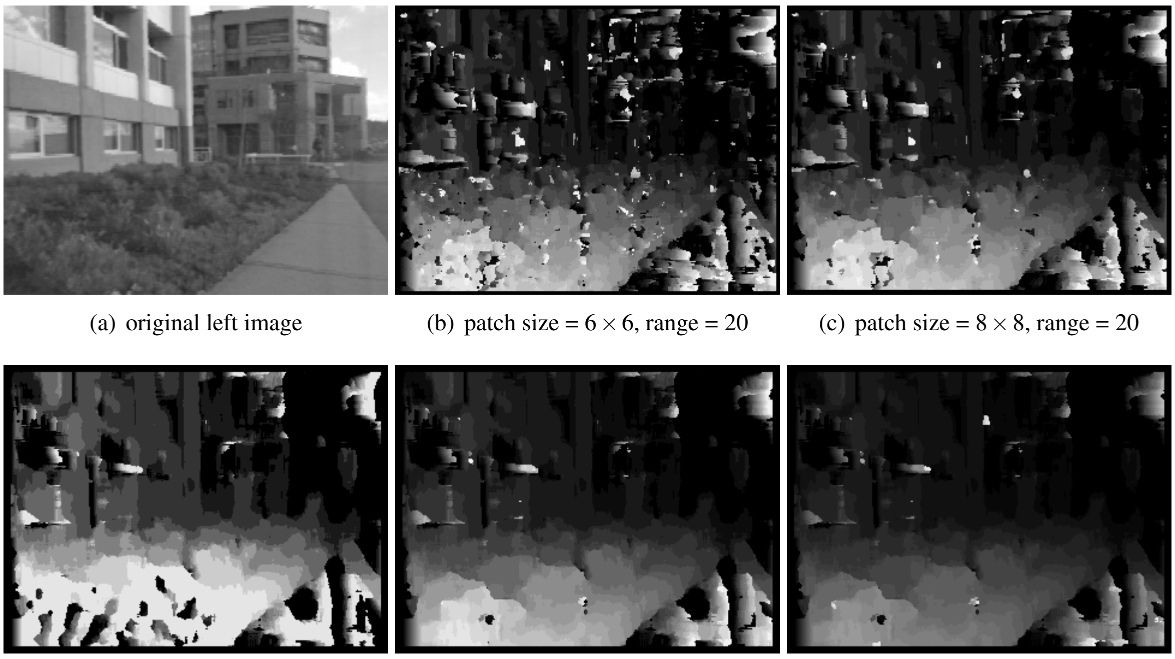
Figure 2.1: Image patch matching algorithm

SAD execution times			
Image pair	Patch size	Search range	Time (s)
triclopsi2	6×6	20	29.50
	8×8	20	49.71
	12×12	10	55.74
	12×12	20	111.06
	12×12	30	170.06
corridor	6×6	20	25.27
	8×8	20	42.82
	12×12	10	45.51
	12×12	20	90.79
	12×12	30	143.50

Table 2.1: SAD execution time

2.2 Implementation and Results

In our implementation, we set the patch size to 6, 8 and 12, and the search range as 10, 20 and 30 respectively, to explore the effects of these parameters. We set the first image captured at the left viewpoint, and we always find the matching patches from the image captured at the right viewpoint. We assume that the center of the matching patch in the second image always appears on the left of the center of the first patch, because the objects are always moving from right to left when we observe the right image first and then the left. Therefore, for the search range, we search the patches with the center on the left of the first patch. Besides, we assume that the corresponding patches in the two images only have horizontal displacement, so we only search patches at the same horizontal line. The table 2.1 shows the running time and the figures 2.2 and 2.3 shows the corresponding disparity maps.

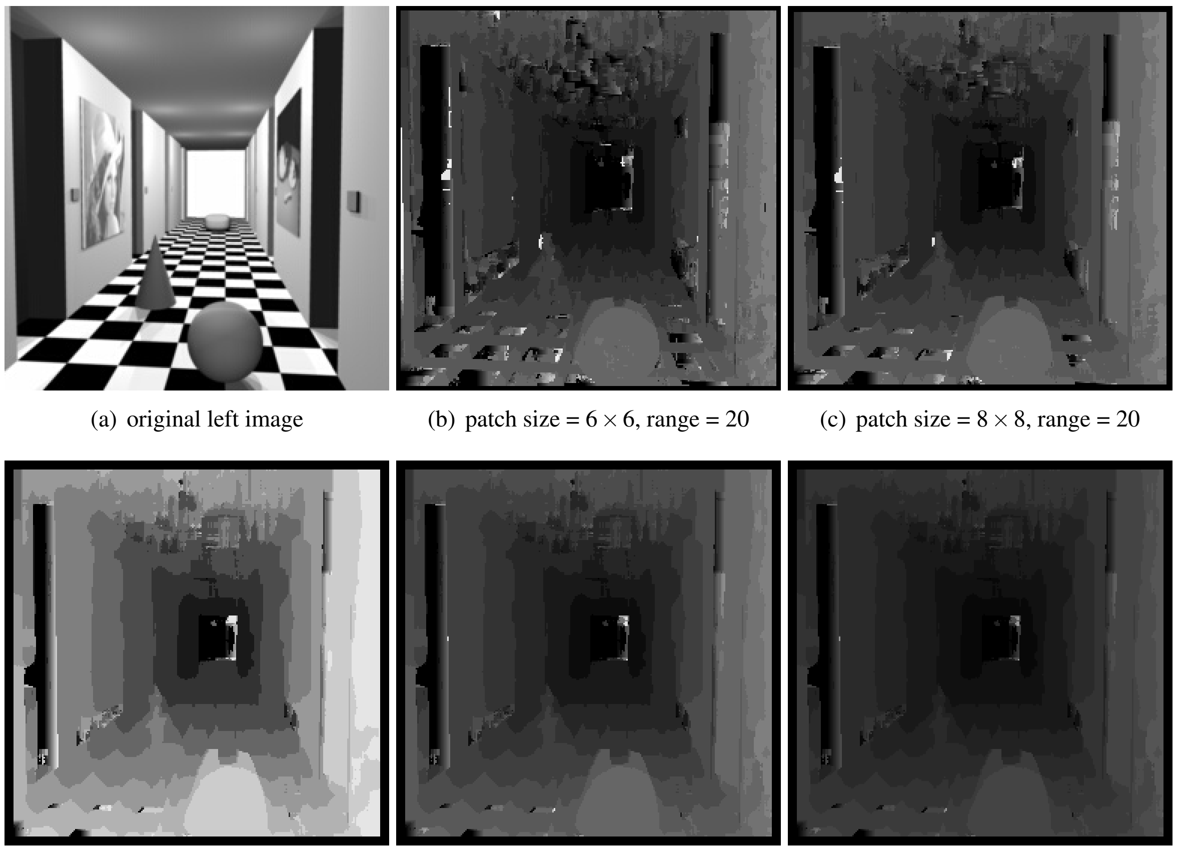


(a) original left image (b) patch size = 6×6, range = 20 (c) patch size = 8×8, range = 20

(d) patch size = 12×12, range = 10 (e) patch size = 12×12, range = 20 (f) patch size = 12×12, range = 30

Figure 2.2: SAD disparity maps for triclopsi2 images

As we can see from the results, the execution time increases with the increasing patch size and the search range. If we set the patch size too small, let's say 6×6, then the disparity maps have a lot of


(a) original left image (b) patch size = 6×6, range = 20 (c) patch size = 8×8, range = 20

(d) patch size = 12×12, range = 10 (e) patch size = 12×12, range = 20 (f) patch size = 12×12, range = 30

Figure 2.3: SAD disparity maps for corridor images

noise, And the objects within the same area don't have a relatively uniform and similar disparities. In other words, the disparities of that same area are not smooth enough. With the increasing patch size, the disparities of each area become smooth and uniform.

At the same time, due to the dyeing gray levels for different search ranges are the same (0 - 255), if the search range is small, i.e., range = 10, then the number of disparity levels are only 10 so that the transitions of disparities are not smooth, and the black and white contrast is larger; if the search range is large, i.e., range = 30, then the transitions of disparities are relatively smooth but it costs too much time. As we can see from the result disparity maps, consider the time consuming, when the patch size is 12×12 and the searching range is 20, the disparity maps is relatively clear, reasonable and smooth, and the execution time is also not too long.

However, the limitations of the algorithm are also obvious:

- Low texture: Some non-detailed parts, such as the road and the sky, will generate noise because the surrounding matching degrees of the patches are all very similar.
- Repetitive/ambiguous patterns: Some recurring textures, such as checkerboard texture, can cause noise.
- Smooth specular reflection of a surface: Smooth surfaces, such as the sphere surface on the surface of the highlights lack texture details, no feature points.

Chapter 3

Sum of Squares Difference

3.1 Introduction to SSD

Similar to the SAD algorithm, the Sum of Squares Difference also measures the similarity between image patches. But the difference is that, SSD is calculated by taking the sum of squares difference between each pixel in the original block and the corresponding pixel in the block being used for comparison. Let's say, for a small image patch $g[(2N+1)\times(2N+1)]$ in the first image, we can find the corresponding image patch centered at (x,y) with the lowest Sum of Squares Difference in the second image I:

$$\arg\min_{(x,y)} \sum_{u=-N}^{N} \sum_{v=-N}^{N} [I(x+u,y+v)-g(u,v)]^2$$

We assume that the corresponding patches in the two images only have horizontal displacement. Then the horizontal distance between the matched patches in the two images is just the disparity.

3.2 Implementation and Results

In our implementation, similarly, we also set the patch size to 6, 8 and 12, and the search range as 10, 20 and 30 respectively, to explore the effects of these parameters. The table 3.1 shows the running time and the figures 3.1 and 3.2 shows the corresponding disparity maps.

As we can see from the results, similar to SAD, the execution time increases with the increasing patch size and the search range. With the increasing patch size, the disparities of each area become smooth and uniform but costs more time. If the search range is small, then the transitions of disparities are not smooth; but if the search range is large, then the transitions of disparities are relatively smooth but it costs too much time. As we can see from the result disparity maps, consider the time consuming, when the patch size is 12×12 and the searching range is 20, the disparity maps is relatively clear, reasonable and smooth, and the execution time is also not too long.

Actually, the disparity maps generated by SSD is very similar to SAD, because the algorithms are very close to each other. And the time complexity of the two algorithms are also very close. The only difference is that SAD uses absolute but SSD uses squares. Sometimes SSD is relatively accurate, for example, lets say a patch with pixel values $g[[2,2],[2,2]]$, then compare with the patches $I_1[[1,3],[1,3]]$ and $I_2[[2,4],[2,4]]$, if we use SAD, then the difference between g and I_1 is the same as g and I_1 (4). But by using SSD, the difference between g and I_1 is 4 while the difference between g and I_2 is 8. Therefore, sometimes SSD is relatively more efficient than SAD.

However, similar to SAD, the limitations of the algorithm are also obvious:

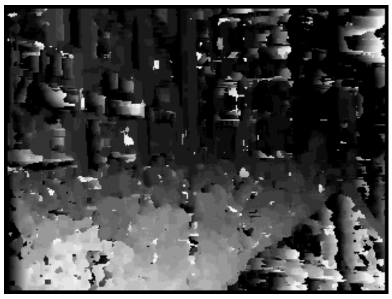
- Low texture: Some non-detailed parts, such as the road and the sky, will generate noise because the surrounding matching degrees of the patches are all very similar.

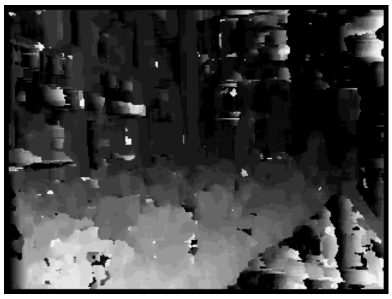
SAD execution times			
Image pair	Patch size	Search range	Time (s)
triclopsi2	6×6	20	31.29
	8×8	20	53.62
	12×12	10	57.91
	12×12	20	114.14
	12×12	30	172.08
corridor	6×6	20	26.36
	8×8	20	44.72
	12×12	10	46.66
	12×12	20	94.34
	12×12	30	141.75

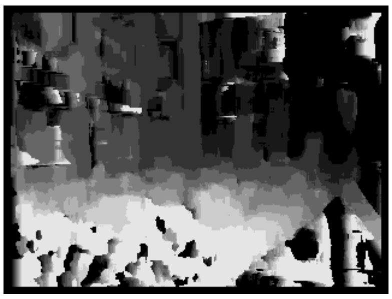
Table 3.1: SSD execution time

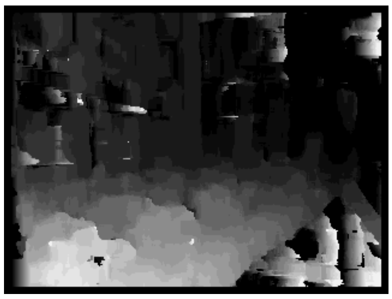
- Repetitive/ambiguous patterns: Some recurring textures, such as checkerboard texture, can cause noise.
- Smooth specular reflection of a surface: Smooth surfaces, such as the sphere surface on the surface of the highlights lack texture details, no feature points.

(a) original left image


(b) patch size = 6 × 6, range = 20


(c) patch size = 8 × 8, range = 20


(d) patch size = 12 × 12, range = 10


(e) patch size = 12 × 12, range = 20

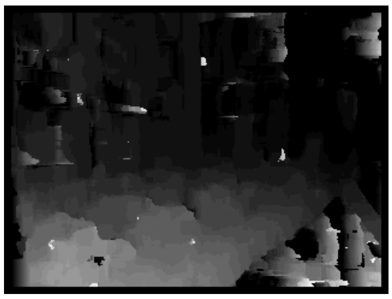
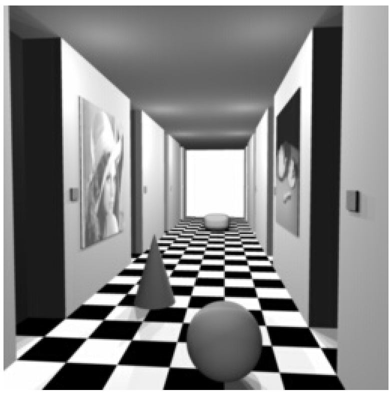
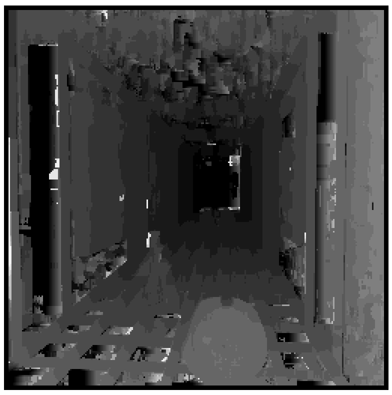
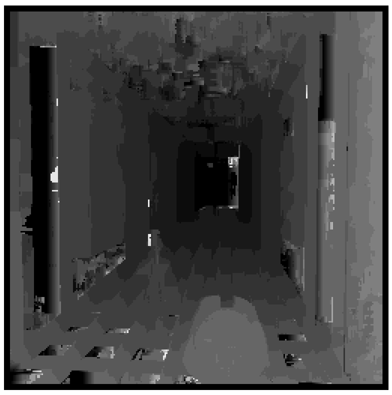
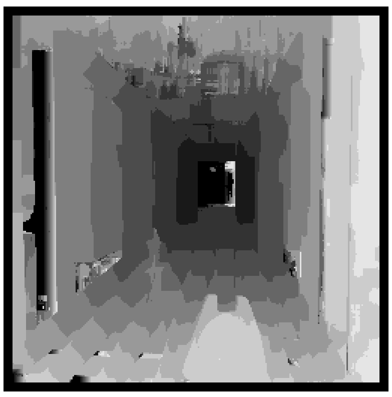

(f) patch size = 12 × 12, range = 30

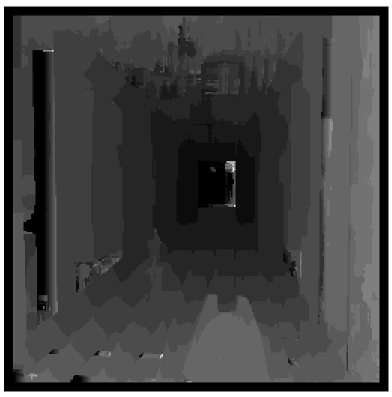
Figure 3.1: SSD disparity maps for triclopsi2 images


(a) original left image


(b) patch size = 6 × 6, range = 20


(c) patch size = 8 × 8, range = 20


(d) patch size = 12 × 12, range = 10


(e) patch size = 12 × 12, range = 20

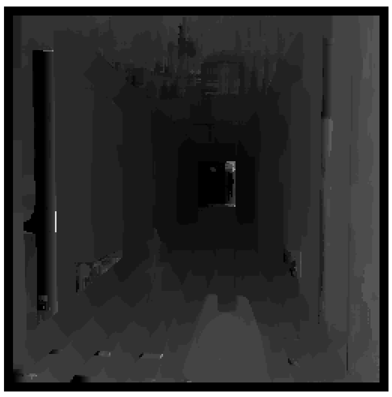

(f) patch size = 12 × 12, range = 30

Figure 3.2: SSD disparity maps for corridor images

Chapter 4

Normalized Cross Correlation

4.1 Introduction to NCC

We also tried normalized cross correlation (NCC) algorithm, which is a common cross correlation calculation method in patch matching. It describes the correlation between two vectors, windows or samples, and the value ranges from -1 to 1, where -1 means the two vectors are uncorrelated and 1 means the two vectors are correlated. For example, the NCC score between the vector $[1,1]$ and $[-1,-1]$ is -1. Therefore, the larger the value of NCC, the more similar the two vectors are.

Let's say, for a small image patch $g[(2N+1)\times(2N+1)]$ in the first image, we can find the corresponding image patch centered at (x,y) with the lowest NCC score in the second image I:

$$\underset{(x,y)}{\arg\min} \frac{\sum_{u=-N}^{N}\sum_{v=-N}^{N} I'(x+u,y+v)g'(u,v)}{\sqrt{\sum_{u=-N}^{N}\sum_{v=-N}^{N} I'(x+u,y+v)^2 \sum_{u=-N}^{N}\sum_{v=-N}^{N} g'(u,v)^2}}$$

We assume that the corresponding patches in the two images only have horizontal displacement. Then the horizontal distance between the matched patches in the two images is just the disparity. In the equation, I', g' are the normalized images: $I' = I - \overline{I}$ and $g' = g - \overline{g}$.

We use the NCC algorithm for the matching task. In the right image, the patch with the highest correlation on the same horizontal level as the patch in the left image is considered as the optimal matching.

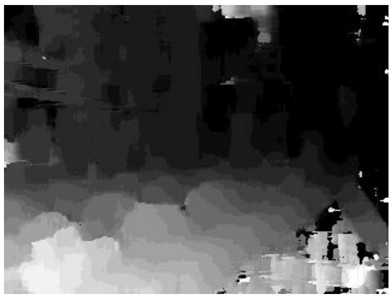
4.2 Implementation

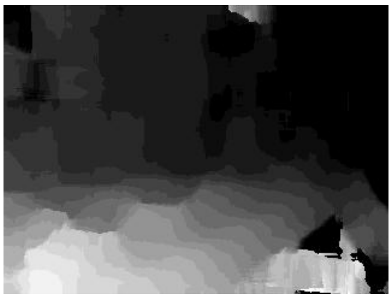
Our implementation refers to [3]. We use*uniform_filter* in the package *scipy.ndimage* to calculate the average of the image blocks by uniform filtering. We tried patch size 15×15, 21×21 and 27×27 and the search range 10, 20 and 30. The results are shown in the figures 4.1 and 4.2.

As we can see from the results, with the increasing patch size, the disparity maps becomes much more smooth and there is less noise. With the reasonable search range, i.e., search range = 20, the transitions of different disparity levels are more obvious comparing to large search range (search range = 30), and there are more disparity levels and more depth information can be expressed comparing to small search range (search range = 10). However, this comparison is subjective, and we should choose the appropriate parameters according to the situation. In short, NCC can also efficiently do patch matching for drawing disparity maps.

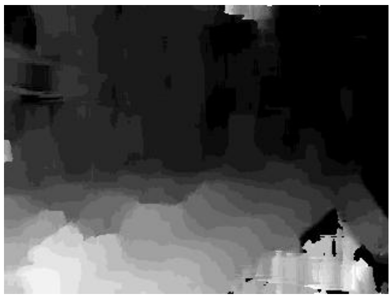
In essence, similar to SAD and SSD, NCC is also looking for matching patches, but the difference is the formula it uses. Therefore, The problems such as low texture and smooth specular reflection still exist.

(a) original left image


(b) patch size = 15 × 15, range = 20

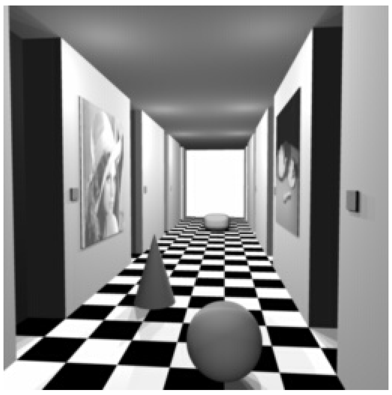

(c) patch size = 27 × 27, range = 20

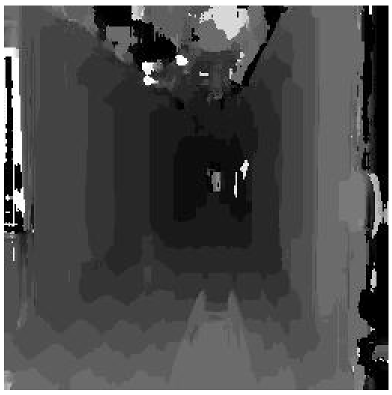
(d) patch size = 21 × 21, range = 10

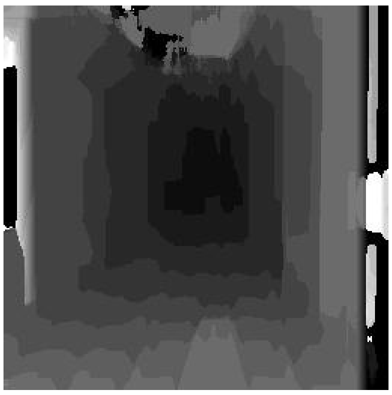

(e) patch size = 21 × 21, range = 20

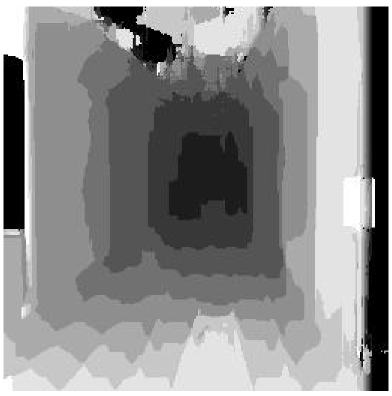
(f) patch size = 21 × 21, range = 30

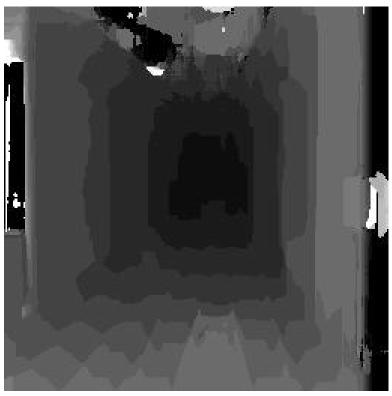
Figure 4.1: NCC disparity maps for triclopsi2 images


(a) original left image


(b) patch size = 15 × 15, range = 20


(c) patch size = 27 × 27, range = 20


(d) patch size = 21 × 21, range = 10


(e) patch size = 21 × 21, range = 20

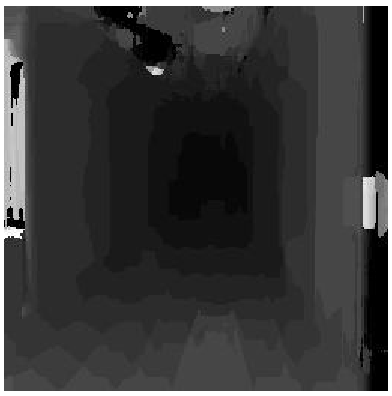

(f) patch size = 21 × 21, range = 30

Figure 4.2: NCC disparity maps for corridor images

Chapter 5

Block Matching and Semi-Global Block Matching

5.1 Idea of Semi-Global Block Matching

There are some basic steps for SGBM [1]:

1. Calculate BT cost after SobelX processing of input image
2. Input image to calculate BT value directly
3. Integrate the values of the above two steps
4. Block the value obtained by the above steps

The figure 5.1 shows the basic steps.

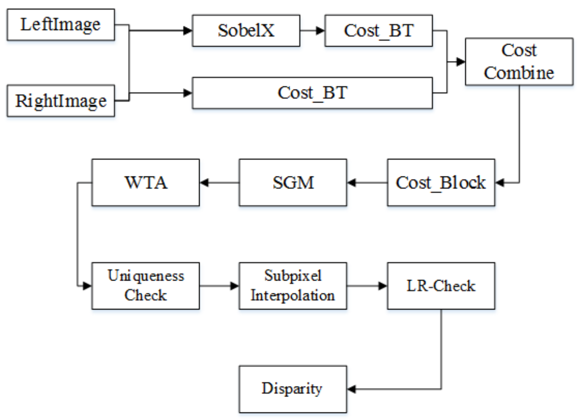


Figure 5.1: SGBM process [1]

The core of the algorithm is that, a global energy function related to the disparity map is set, we want to minimize the energy function so as to solve the optimal disparity for each pixel. The energy funtion is

$$E(D) = \sum_{p}(C(p,D_p) + \sum_{q\in N_p} P_1 \mathbb{1}(|D_p - D_q| = 1) + \sum_{q\in N_p} P_2 \mathbb{1}(|D_p - D_q| > 1)$$

, where D is disparity map and $E(D)$ is the energy function of D. p and q are the pixels in the image, and N_p means the 8 neighbor pixels of the pixel p. $C(p,D_p)$ means the cost of p when the disparity of p is D_p. P_1 and P_2 are penalty coefficients, P_1 applies to pixels whose disparity value differs by 1 from the disparity value of p in neighboring pixels of pixel p, and P_2 applies to pixels whose disparity value differs from p's disparity value by more than 1 in neighboring pixels of pixel p.

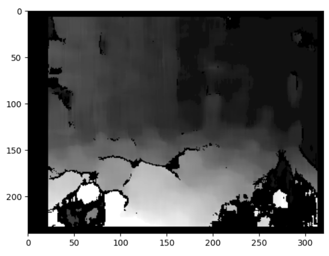
5.2 Implementation

In the OpenCV implementation, a block matching algorithm based on Konolige's Block Matching Algorithm [4] is implemented. We directly use the OpenCV implementation for comparison experiments. The parameter *numDisparities* represents the maximum search disparity number, we set as 16, and we set *blockSize* as 15. The disparity maps are shown in the figure 5.2.

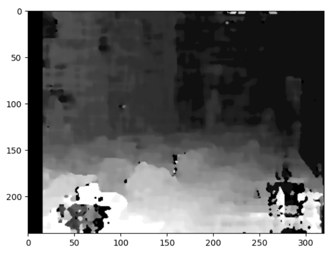
We directly use the OpenCV implementation of SGBM. The implementation is based on *Stereo Processing by Semi-global Matching and Mutual Information* written by Heiko Hirschmuller [5], but the matching cost calculation based on mutual information in the original text is not realized, but adopted the BT algorithm proposed by S. Birchfield and C. Tomasi [6], also, some BM algorithms in the pre- and post-processing procedures are added. We set the minimum disparity parameter *minDisparity* as 0 by default, disparity search range length *numDisparities* as 16, the size of the cost calculation window of SAD *blockSize* as 5 by default, the parameters of energy function $P1 = 8 \times channel_number \times sgbm.SADWindowSize^2$ and $P2 = 32 \times channel_number \times sgbm.SADWindowSize^2$, and the SAD window size as 5. The disparity maps are shown in the figure 5.2.

As we can see from the result disparity maps, by using SGBM, the disparity maps are relatively reasonable and clear than only BM algorithm, and there are less noise. The two algorithms are both effecient to generate disparity maps.

(a) original left triclopsi2 image

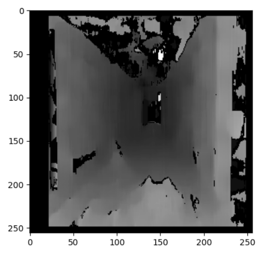


(b) triclopsi2 MB

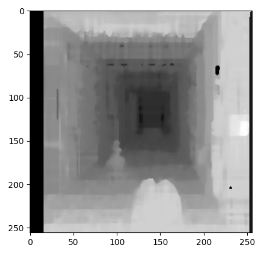


(c) triclopsi2 SGMB

(d) original left corridor image



(e) corridor MB



(f) corridor SGMB

Figure 5.2: BM and SGBM implemented in OpenCV

Chapter 6

Summary

In this assignment, we implemented SAD, SSD and NCC algorithms, and tried BM and SGBM algorithms implemented in OpenCV to draw the disparity maps of given image pairs. We explored the performance of SAD, SSD algorithms and compared the result disparity maps of all the algorithms. Some limitations of each algorithm are also pointed out. Our conclusion is that, for the given two pairs of the images, all these algorithm can be applied to obtain the general disparity maps, but the performance really depends on the parameters, the characteristics of given image pairs and so on. And the comparison discussion is also very subjective. For the real application, it really requires a unique solution for each situation, such as different matching algorithms, different parameters and different pre-processing and post-processing. Besides, some common difficulties faced by the matching algorithm are also pointed out, such as low texture, repetitive/ambiguous patterns and smooth specular reflection of a surface. We might need more advanced algorithms to determine the disparity of these areas.

Bibliography

[1] OrdinaryW. Sgbm algorithm detailed solution (translated from chinese), 2019. [Online; accessed 29-October-2022].

[2] Wikipedia contributors. Sum of absolute differences — Wikipedia, the free encyclopedia, 2021. [Online; accessed 28-October-2022].

[3] qq_41934490. Computer vision python - computing disparity maps (translated from chinese), 2020. [Online; accessed 29-October-2022].

[4] Kurt Konolige. Small vision systems: Hardware and implementation. In *Robotics research*, pages 203–212. Springer, 1998.

[5] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2007.

[6] Stan Birchfield and Carlo Tomasi. Depth discontinuities by pixel-to-pixel stereo. *International Journal of Computer Vision*, 35(3):269–293, 1999.

Appendix

Here are the functions we implemented in our programs:

SAD.py

```
# SAD.py

# command line:
# python SAD.py
# disparity maps generated under the folder "results"

def SAD(left_img, right_img, kernel, max_offset, img_name)
    # input: left image, right image, patch size, search range and image name
    # output: disparity maps of the left and right images
```

SSD.py

```
# SSD.py

# command line:
# python SSD.py
# disparity maps generated under the folder "results"

def SSD(left_img, right_img, kernel, max_offset, img_name)
    # input: left image, right image, patch size, search range and image name
    # output: disparity maps of the left and right images
```

NCC.py

```
# NCC.py

# command line:
# python NCC.py
# disparity maps generated under the folder "results"

def NCC(im_l, im_r, start, steps, wid)
    # input: left image, right image, start offset, search range and patch size
    # output: disparity maps of the left and right images
```

BM.py

```
# BM.py

# command line:
# python BM.py
# disparity maps generated under the folder "results"
```

```
def BM(imgL, imgR, result_name):
    # input: left image, right image and result image name
    # output: disparity maps of the left and right images
```

SGBM.py

```
# SGBM.py

# command line:
# python SGBM.py
# disparity maps generated under the folder "results"

def SGBM(imgL, imgR, result_name):
    # input: left image, right image and result image name
    # output: disparity maps of the left and right images
```