



Master in Science (Artificial Intelligence)
(MSAI)

AI-6121

Assignment 2: Disparity map

Name of Student:
Teo Lim Fong

Matriculation No: G2101964G

1. Describe the procedure of disparity computing given a pair of rectified images of the same scene captured from two different viewpoints.

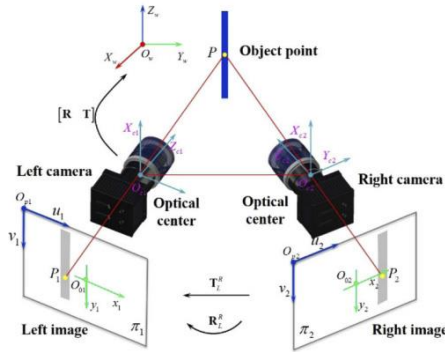


Figure 1: A diagram demonstrating two images facing at the same object in different viewpoints (left)

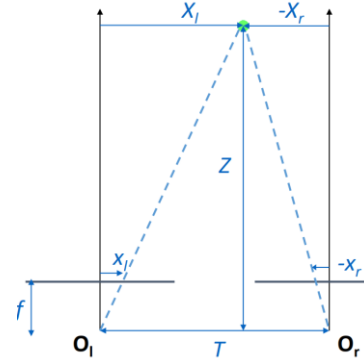


Figure 2: A simple diagram to demonstrate the parameters of depth (Z), focal length (f), translation (T) and the disparity (x_l-x_r)

Disparity only can be computed when there are two images lying along the x-axis. The pair of images can create a triangle when facing at the same point. The distance between the two images is called the translation (T) and the distance perpendicular from the object point to the translation is called the depth (Z). The formula for 2D triangulation (left image) is given by: $x_l = \frac{fX}{Z}$ and $x_r = \frac{fX - T}{Z}$ (right image).

Disparity can be computed by simply subtracting $d = x_l - x_r = \frac{fT}{Z}$.

In order to find the disparity, we need to find the corresponding pixel from the right image to match the pixel from the left image. Since there is only changes in x-axis, the pixels in the right image will only search that particular row for the corresponding pixel. The distance difference between the similar pixel from the left and right image is called the disparity.

For example, a point (20,35) in the left image is trying to find the corresponding pixel in the right image. It will search from (0,35) to (W, 35) in the right image. Then we use point matching to find the least error between the two pixels from the left and right image. To find the disparity, assume the best corresponding point in the right image is (25,35), then subtract it from the left image (25-20). Therefore, the disparity between the point (20,35) and (25,35) is 5.

There are two methods to compute point matching. There are the appearance-based matching and feature-based matching. For appearance-based matching, the algorithm is called the Sum-of-Squared Difference (SSD).

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

Figure 3: Formula for Sum-of-Squared Difference (SSD)

The images must be in the same size and converted to grayscale. The pixel in the left images is computed by searching the corresponding pixel in the right images. It uses a small window patch in both left and right images. For the right image, it is subtracted by an additional disparity (d). The best matching disparity is the minimum value of SSD. If the translation between the two images is small then the correspondence is accurate. However, the downside is that 3D estimation required the translation to be large to obtain a better accuracy.

For feature-based matching, the algorithm is called Scale invariant feature transform (SIFT). SIFT divide the image into 16 small patches and within the small patch is further divided into another 16 smaller patches. Every small patch consists of its gradient direction and SIFT used histogram to compute 128 dimensions of feature vector. The feature vector is then used for the matching.

2. Write a computer algorithm that computes the disparity of two images. The programming language can be Matlab, python, or other languages.

I will be using Sum-of-Squared Difference (SSD) method to find the disparity map.

Step 1:

The given pair images are in RGB channels. Since we are finding the disparity and the depth, the color information is not important. Therefore, it is better to convert it to grayscale to minimum the number of parameters.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

path1 = 'C:/Users/LimFong/Desktop/MSAI/computer vision/2d/corridor1.JPG'
path2 = 'C:/Users/LimFong/Desktop/MSAI/computer vision/2d/corridor2.JPG'

left = cv2.imread(path1)
right = cv2.imread(path2)

gray_left = cv2.cvtColor(left, cv2.COLOR_BGR2GRAY)
gray_right = cv2.cvtColor(right, cv2.COLOR_BGR2GRAY)

w,h=gray_left.shape
```

Step 2:

In order to plot the disparity map, I need to create a zero array with dimension (h, w). h and w refer to the image input's height and weight. Then I create a list to append all the minimum values for disparity in (Step 3). Since I am using SSD method, this mean that I will need to create a window for each image (left and right). N will represent half the size of the image patch/window. N_pixel_search refer to the number of pixels to search for the corresponding point.

```
N = 3
N_pixel_search = 30
d_map = np.zeros([h, w])
disp_list = []
```

Step 3: First, loop the all the pixels in the images in x axis/ row. Without the window, the range for the row is 0 to w. Since I am using the window, I need to readjust the range for the looping. Therefore, the range will start at the size of the half window to the width- half window. The next few lines are to create empty array for searching the disparity.

```
for x in range(N, w-N):
    disp_diff = x-N
    min_disp = min(disp_diff, N_pixel_searchs)
    disp_list.append(min_disp)
    disp_array = np.unique(disp_list)
    len_disp_array = len(disp_array)
    diff_array = np.zeros([len_disp_array])
```

Step 4: Similarly, loop all the pixels in y-axis(column). The purpose of looping the disp_array is to subtract the disparity from the right images.

Next, I will be creating window by using a range (minimum column and maximum column) to determine the height and another range (minimum row and maximum row) to determine the width.

```
for y in range(N, h-N):
    for d in disp_array:
        min_col = y-N
        max_col = y+N
        min_row = x-N
        max_row = x+N
```

Step 5: In this portion, I will be applying SSD shown in Figure 3. First, minimum column and maximum column will be applied to the left gray image and minimum row and maximum row will be applied to the right gray image. However, for right image we need to subtract disparity.

Then convert the images into integer. Finally, we apply the formula to get the difference (SSD).

```
left_image = gray_left[min_col:max_col, min_row:max_row]
right_image = gray_right[min_col:max_col, min_row-d:max_row-d]
left_image = left_image.astype(int)
right_image = right_image.astype(int)
diff_array[d] = np.sum((left_image - right_image)**2)
```

Step 6: The final step is to find the best matching SSD and apply it to the d_map empty array.

```
mini_disp = np.argmin(diff_array)
d_map[y, x] = mini_disp
```

3. Apply your developed algorithm to the two provided image pairs and derive the corresponding disparity maps. Discuss your observation of the obtained disparity maps.

These are the disparity maps obtained by using cmap 'gray'. The brighter pixel mean that the object is closer can darker region mean that it is further away. There is a parameter called the N_pixel_searchs in my coding. This parameter can be adjustable and the default value is 30. This parameter refers to number of pixels to search for matching.

I will rename N_pixel_searchs parameters as **D**. The larger the value of D, the least accurate is the disparity map.

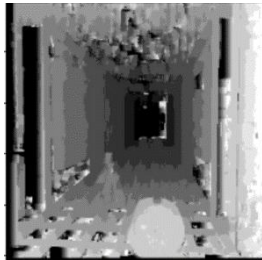


	D =10	D = 30	D = 50
Corridor			

Table 1: Applying disparity map using Corridor image with different value of D

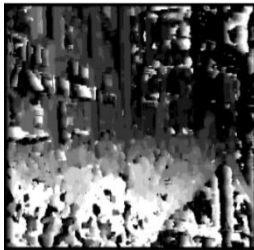


	D =10	D = 30	D =50
Triclopsi2			

Table 2: : Applying disparity map using Triclopsi2 image with different value of D

To plot the SSD, I just used back the above coding and substitute (x,y) with a random point. For demonstration, I will be using point (200,40) to plot a graph. In the graph, the x-axis represent the number of pixel to match the corresponding pixel and the y-axis represent the SSD. The red dot shows the minimum point among the 30 pixels because previously my default N_pixel_searchs value is 30. Thefeore, the red dot represent the particular pixel in the right image for best matching at point (200,40) in the left image.



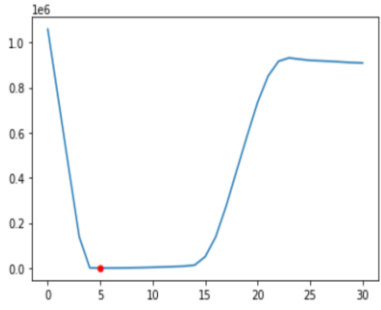
	Left image	Right image	SSD graph
Corridor			

Table 3: Showing SSD graph by using a point (200,64) on the left image

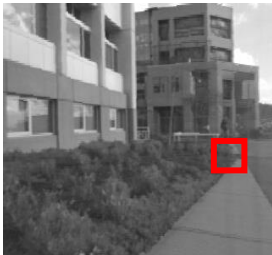

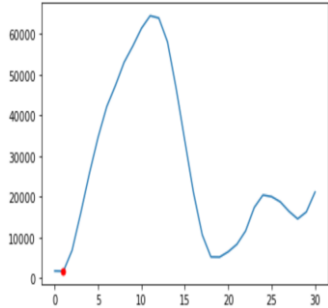
	Left image	Right image	SSD graph
Triclopsi2			

Table 4: Showing SSD graph by using a point (200,64) on the left image

4. Discuss what affects the disparity map computation, and any possible improvements of your developed algorithm. Implement and verify your ideas over the provided test images. This subtask is **optional**, and there will be credit marks for good addressing of this subtask.

The cons of disparity map is that if the pair of image is too bright or dark, then it is hard to find the corresponding point. Every point has a similar pixel intensity. If every point on that row has similar intensity, then by computing point matching (SSD) it will become meaningless as most of the pixels error will equal to 0. Therefore, if we can do a preprocessing method to balance the intensity value before computing the disparity map then the darker or brighter region will be equally distributed. The preprocessing method is called the Histogram Equalization (HE).

I will be using the existing HE coding from the previous assignment to incorporate with my disparity map. The incorporated code will be located in Annex.

Below shows the images after applying HE and also plotting the SSD graph at point (200,40).

	Left image	Right image	SSD graph
Triclopsi2			

Table 5: Original triclopis images (top) and HE-applied triclopis images (below)



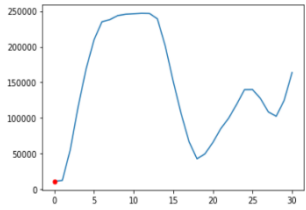
	Left image	Right image	SSD graph
Triclopsi2 HE			

Table 5: Original triclopis images (top) and HE-applied triclopis images (below)

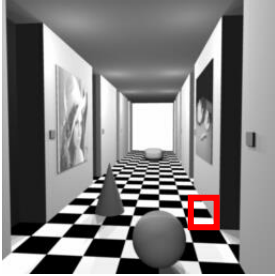

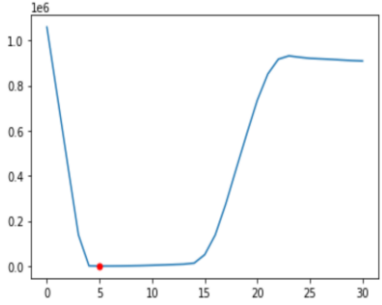
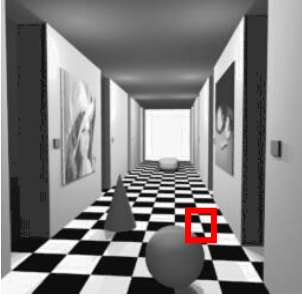

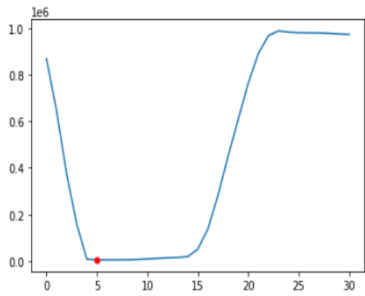
	Left image	Right image	SSD graph
Corridor			
	Left image	Right image	SSD graph
Corridor			

Table 6: Original triclopis images (top) and HE-applied corridor images (below)

We can see that the intensity is well distributed. Next, I will apply the disparity map on the HE verison of the images.

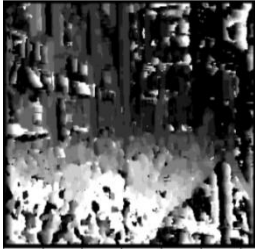


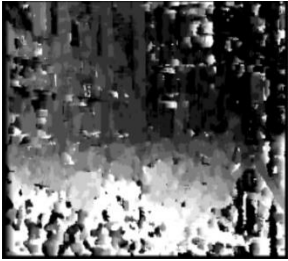


	D =10	D = 30	D =50
Triclopsi2			
	D =10	D = 30	D = 50
Triclopsi2 HE			

Table 7: Applying disparity map using HE Triclopso2 image with different value of D

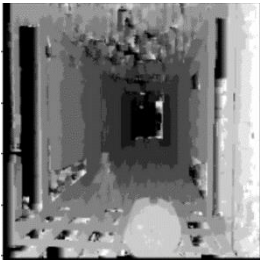

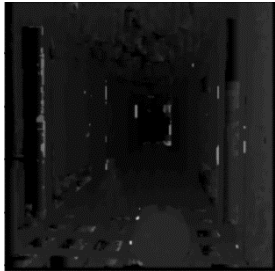



	D =10	D = 30	D = 50
Corridor			
	D =10	D = 30	D = 50
Corridor HE			

Table 8: Applying disparity map using HE Corridor image with different value of D

In conclusion, in order to find the the disparity between the two images, we need to first use a small image patch. Then using this image patch pixel to find the best matching along the same row in the right image. However, if the image is too bright or dark then the matching will not be accurate. Therefore, histogram equalization is applied to balance the intensity to avoid any bright or dark region. The results that I have experimented in Table 7 and 8 shows a better result when Histogram Equalization is applied.

Annex

Original Disparity Coding

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

path1 = 'C:/Users/LimFong/Desktop/MSAI/computer vision/2d/corridorl.JPG'
path2 = 'C:/Users/LimFong/Desktop/MSAI/computer vision/2d/corridorr.JPG'

left = cv2.imread(path1)
right = cv2.imread(path2)

gray_left = cv2.cvtColor(left, cv2.COLOR_BGR2GRAY)
gray_right = cv2.cvtColor(right, cv2.COLOR_BGR2GRAY)

w,h=gray_left.shape
N =3
N_pixel_search= 30
d_map = np.zeros([h, w])
disp_list =[]

for x in range(N, w-N):
    disp_diff = x-N
    min_disp = min(disp_diff,N_pixel_searchs)
    disp_list.append(min_disp)
    disp_array = np.unique(disp_list)
    len_disp_array =len(disp_array)
    diff_array = np.zeros([len_disp_array])

    for y in range(N, h-N):
        for d in disp_array:
            min_col = y-N
            max_col = y+N
            min_row = x-N
            max_row = x+N
            left_image = gray_left[min_col:max_col, min_row:max_row]
            right_image = gray_right[min_col:max_col,min_row-d:max_row-d]
            left_image = left_image.astype(int)
            right_image = right_image.astype(int)
            diff_array[d] = np.sum((left_image - right_image)**2)

    disp_ind = np.argmin(diff_array)

    d_map[y, x] = disp_ind

plt.imshow(d_map, cmap='gray')
```

Histogram Equalization Disparity coding

There are two steps to compute Histogram Equalization Disparity Map. First, I applied Histogram Equalization on an image individually and saved it as a HE version. Next, I use disparity coding to compute disparity map.

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import pywt

img = cv2.imread('C:/Users/LimFong/Desktop/MSAI/computer vision/2d/corridor.JPG')
filename = ('C:/Users/LimFong/Desktop/MSAI/computer vision/2d/corridor_HE.JPG')
ycbcr = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
y,cb,cr = cv2.split(ycbcr)

(n,m)= y.shape

k= 0
kk=0
L= 255
H_array = np.zeros((256), dtype=int)

while k<256:
    H_array[k] = np.count_nonzero(y==k)
    k+=1

res = n*m
H_normalise = H_array/res

new_list=[]
j=0
for i in range(0,len(H_normalise)):
    j+=H_normalise[i]
    new_list.append(j)
    |
new_array = np.array(new_list)

sk = np.uint8(L * new_array) #

new_y = np.zeros_like(y)

for ii in range(0, n):
    for jj in range(0, m):
        new_y[ii, jj] = sk[y[ii, jj]]

new_ycbcr = cv2.merge([new_y, cb, cr])
new_rgb = cv2.cvtColor(new_ycbcr, cv2.COLOR_YUV2RGB)
```