NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE/CZ4003: Computer Vision

Lab 2 Report

Ingale Omkar
(U2020724H)

Table of Contents

3 Experiment 3

3.1 Edge Detection 3

3.1 (a) Macritchie.jpg 3

3.1 (b) Sobel masks 3

3.1 (c) Combined image 4

3.1 (d) Threshold 5

3.1 (e) Canny edge detection 7

3.2 Line Finding using Hough Transform 10

3.2 (a) Canny with sigma = 1 10

3.2 (b) Radon and Hough Transform 10

3.2 (c) Location of maximum pixel intensity 11

3.2 (d) Equation derivation 13

3.2 (e) Calculating yl and yr 14

3.2 (f) Superimposing estimated line on image 14

3.3 3D Stereo 15

3.3 (a) Disparity map algorithm 15

3.3 (b) Corridor images 15

3.3 (c) Disparity map of images 16

3.3 (d) Disparity map on triclops-jd.jpg 17

3 Experiment

3.1 Edge Detection

3.1 (a) Macritchie.jpg

The following piece of code was used to perform the input:

```
% 3.1 (a) Download macritchie.jpg

P = imread("/Users/omkaringale/Desktop/Computer Vision/Assignments/" + ...
    "Lab2 Edges, Hough Lines, and Disparity/macritchie.jpg");
P = rgb2gray(P);
imshow(P);
```

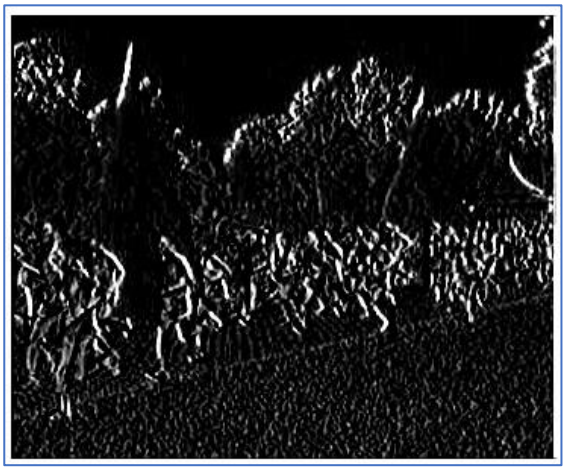
The resultant image:

3.1 (b) Sobel masks

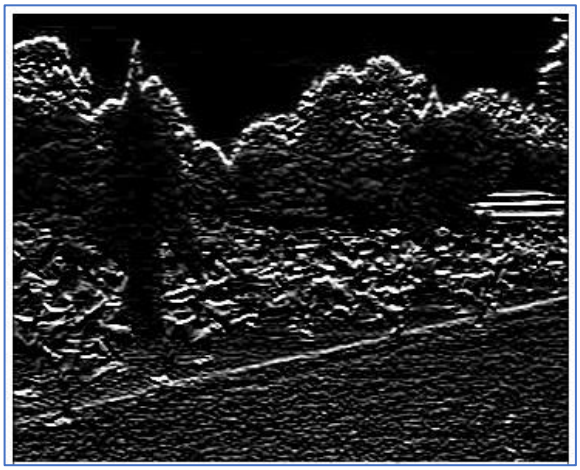
The Sobel masks were created as follows:

```
% 3.1 (b) Sobel Masks

sobel_1 = [-1 0 1;
           -2 0 2;
           -1 0 1];

sobel_2 = [-1 -2 -1;
            0  0  0;
            1  2  1];

P_sobel1 = conv2(P, sobel_1);
imshow(uint8(P_sobel1));

P_sobel2 = conv2(P, sobel_2);
imshow(uint8(P_sobel2));
```

The output images were as follows:



Sobel 1



Sobel 2

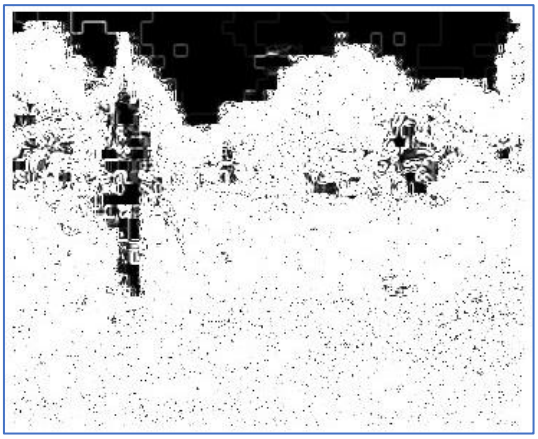
Observation: The two images highlight the edges in two different directions: vertical and horizontal. The first image filter detects the vertical edges because of the way the mask was created while the second filter was created to detect horizontal edges. Both filters fail to highlight diagonal edges effectively. Some diagonal edges are highlighted but I postulate that it is because these diagonals are elements of both vertical and horizontal edges when zoomed in.

3.1 (c) Combined image

The combined image was created as follows:

```
% 3.1 (c) Combined Image through squaring

P_sobel1 = conv2(P, sobel_1);
P_sobel2 = conv2(P, sobel_2);
P2 = P_sobel1.^2 + P_sobel2.^2;
imshow(uint8(P2));
imshow((P2),[]);
```

The resulting image was as follows:



The squared operation is carried out to calculate the magnitude of the gradient. This approach will allow us to obtain the horizontal, vertical, and diagonal edges in the image. Furthermore, the original filters might have yielded negative values for some parts of the image, hence, squaring corrects that leading to the exceptionally bright image where most pixels have a value.

3.1 (d) Threshold

Threshold of the image was calculated as follows:

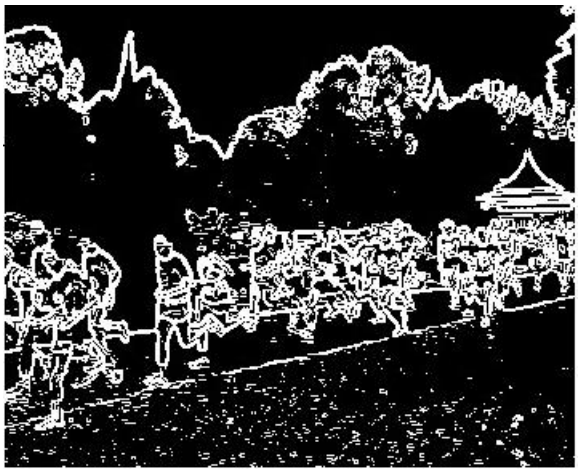
```
% 3.1 (d) Threshold at value t

% t = 5
P2_t5 = P2>5;
imshow(P2_t5);

% t = 10
P2_t10 = P2>10;
imshow(P2_t10);

% t = 20
P2_t20 = P2>20;
imshow(P2_t20);

% t = 50
P2_t50 = P2>50;
imshow(P2_t50);

% t = 100
P2_t100 = P2>100;
imshow(P2_t100);
```

However, the resulting images showed little to no difference. To test the threshold further exceptionally large values were chosen to see if they made any impact. The updated code is as follows:

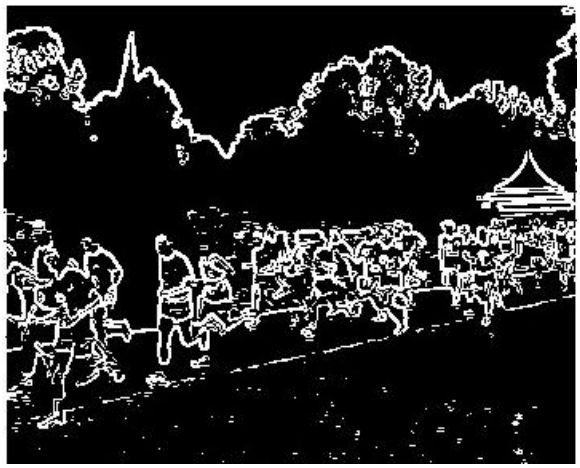
```
% 3.1 (d) Threshold at value t

% t = 10000
P2_t10k = P2>10000;
imshow(P2_t10k);

% t = 20000
P2_t20k = P2>20000;
imshow(P2_t20k);

% t = 40000
P2_t40k = P2>40000;
imshow(P2_t40k);

% t = 80000
P2_t80k = P2>80000;
imshow(P2_t80k);

% t = 100000
P2_t100k = P2>100000;
imshow(P2_t100k);
```

The resulting images are listed here:

T = 10000



T = 20000



T = 40000

T = 80000

T = 100000

From the images shown above, lower threshold values result in more noise in the resultant images. This was seen in action when threshold values like 5 and 10 were used initially and no difference was seen in the resultant images. Values of higher magnitude like 10000 show significant changes and are more effective. The advantages/disadvantages of using different threshold values is the number of edges detected. Lower thresholds can lead to more edges

being shown and hence more noise in the result. Whereas higher threshold values mean some edges may be filtered out because of their magnitudes. Choosing a right threshold may mean a compromise between having more noise (smaller edges in the result) and detecting lesser number of edges.

3.1 (e) Canny edge detection

The Canny edge detection was done as follows:

```
% 3.1 (e) Canny Edge Detection

tl=0.04;
th=0.1;
sigma=1.0;

E = edge(P,'canny' ,[tl th],sigma);
imshow(E);
```

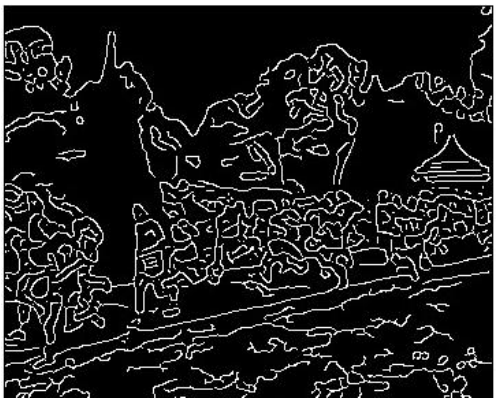
The initial image was as follows:

3.1 (e)(i) Varying values of sigma

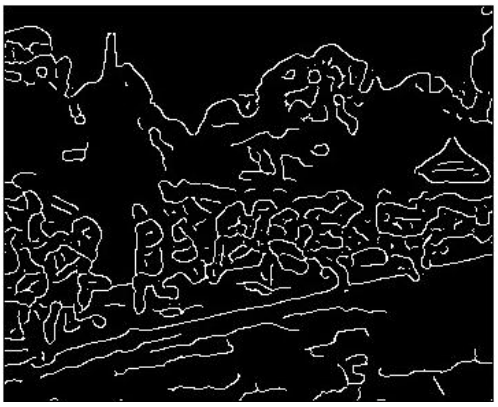
The values of sigma were altered, and the resultant images were as follows:

```
% 3.1 (e)(i)

sigmas = [1.0, 2.0, 3.0, 4.0, 5.0];

ES1 = edge(P,'canny' ,[tl th],sigmas(1));
imshow(ES1);

ES2 = edge(P,'canny' ,[tl th],sigmas(2));
imshow(ES2);

ES3 = edge(P,'canny' ,[tl th],sigmas(3));
imshow(ES3);

ES4 = edge(P,'canny' ,[tl th],sigmas(4));
imshow(ES4);

ES5 = edge(P,'canny' ,[tl th],sigmas(5));
imshow(ES5);
```

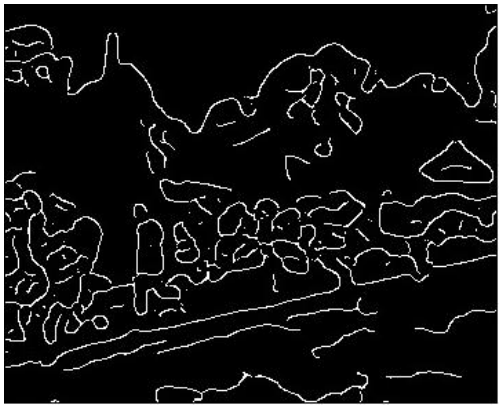
$\sigma = 1.0$



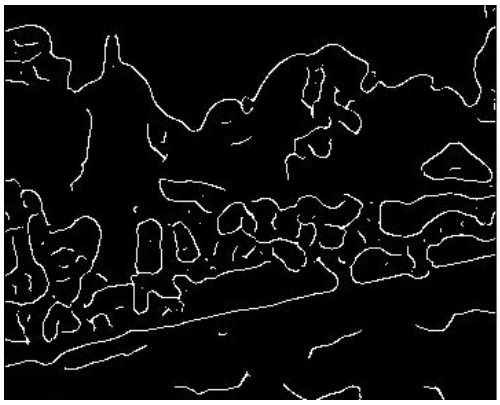
$\sigma = 2.0$



$\sigma = 3.0$



$\sigma = 4.0$



$\sigma = 5.0$

The relationship of sigma is that lower values lead to more noise (more edgels) but higher location accuracy. On the contrary, higher values lead to lower noise (less edgels) but low location accuracy as well. The reason can be that the algorithm uses gaussian filter. As shown in the last assignment, higher values of sigma smooth the image excessively and loses out on the information within it. Hence, this algorithm also misses out on location accuracy for higher values of sigma.

3.1 (e)(ii) Varying values of tl

The values of tl were altered using the code below and the resultant images have been appended below that. Note that the value of sigma used for this part is 1.0.

```
% 3.1 (e)(ii)

tls = [0.01, 0.02, 0.03, 0.04, 0.05];

ETL1 = edge(P,'canny' ,[tls(1) th], sigma);
imshow(ETL1);

ETL2 = edge(P,'canny' ,[tls(2) th], sigma);
imshow(ETL2);

ETL3 = edge(P,'canny' ,[tls(3) th], sigma);
imshow(ETL3);

ETL4 = edge(P,'canny' ,[tls(4) th], sigma);
imshow(ETL4);

ETL5 = edge(P,'canny' ,[tls(5) th], sigma);
imshow(ETL5);
```

tl = 0.01

tl = 0.02

tl = 0.03

tl = 0.04

tl = 0.05

Canny edge detection uses hysteresis thresholding and tl is the lower bound. Hence, edgels that have a magnitude smaller than tl will be filtered out. Hence, for smaller values of tl, more edgels will be detected and there will be more noise. Similarly, larger values of the threshold will lead to more noise being filtered out (lesser edgels).

3.2 Line Finding using Hough Transform

3.2 (a) Canny with sigma = 1

```
% 3.2 (a)

tl=0.04; th=0.1; sigma=1.0;

E = edge(P,'canny' ,[tl th],sigma);
imshow(E);
```

3.2 (b) Radon and Hough Transform

The Hough transform and the Radon transform are indeed very similar to each other, and their relation can be loosely defined as the former being a discretized form of the latter. The Radon transform is a mathematical integral transform, defined for continuous functions on $\mathbb{R}^n$ on hyperplanes in $\mathbb{R}^n$. The Hough transform, on the other hand, is inherently a discrete algorithm that detects lines in an image by polling and binning.

It is same in this case as image is applying discrete radon transform. Using this, both transforms will map every pixel into equivalent sinusoidal functions. However, if the image had applied continuous radon transform, the results would have been different.

The code for the transform was as follows:

```
% 3.2 (b)

[H, xp] = radon(E);
imshow(uint8(H));
```

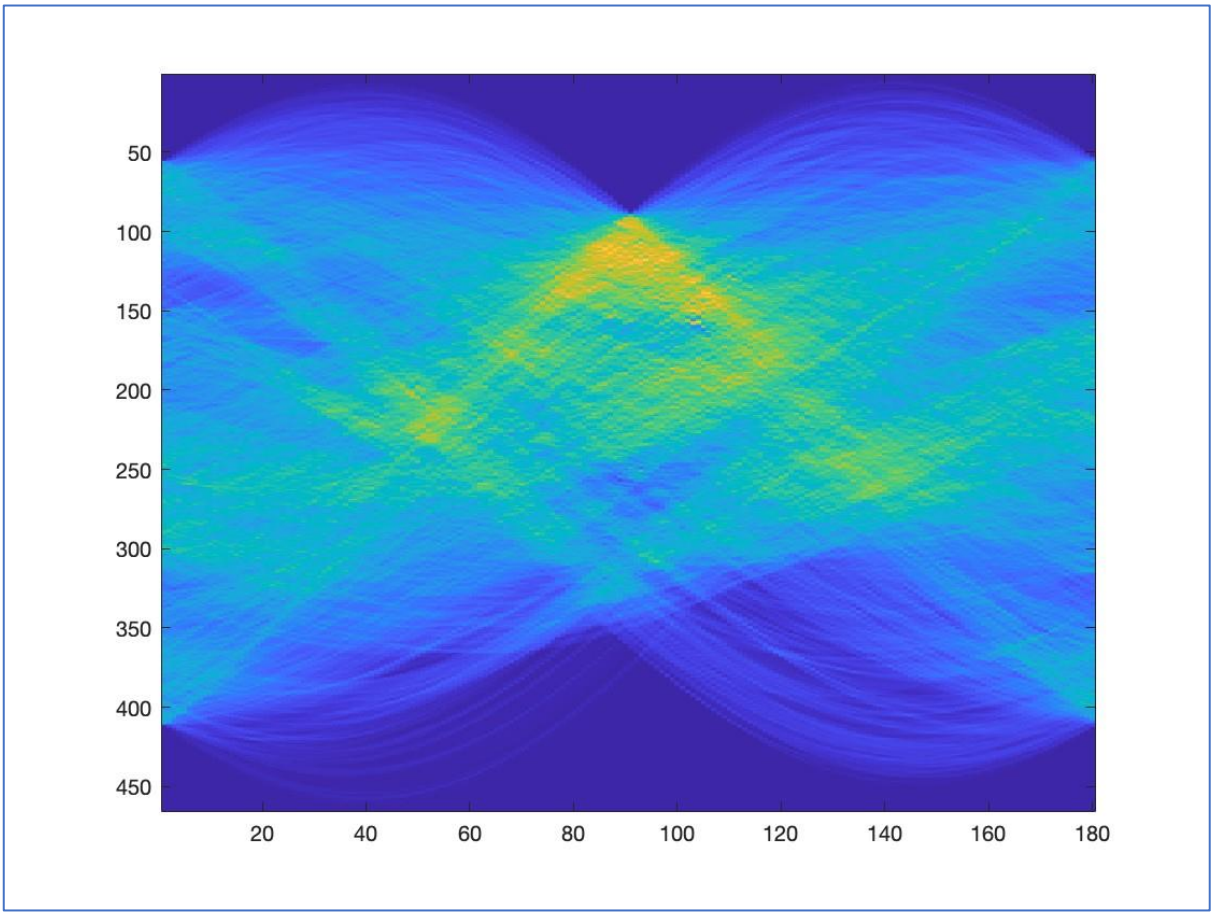
The result was as follows:

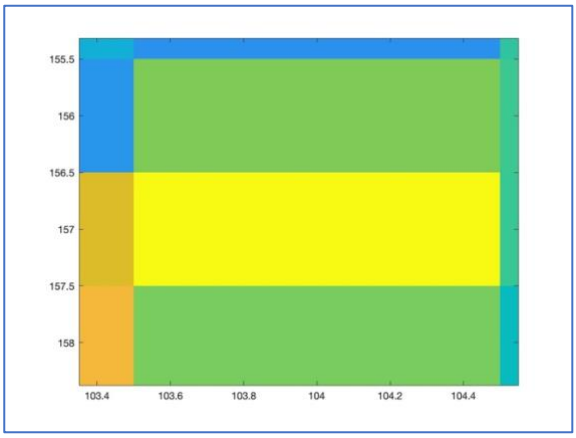
3.2 (c) Location of maximum pixel intensity

The location was found using a colormap. The code was as follows:

```
% 3.2 (c)

[H, xp] = radon(E);
imagesc(uint8(H));
colormap('default');
```

The results was as follows:





From the above images, it can be inferred that the peak is at theta = 103 and radius = 157.

3.2 (d) Equation derivation

```
>> help pol2cart
 pol2cart Transform polar to Cartesian coordinates.
    [X,Y] = pol2cart(TH,R) transforms corresponding elements of data stored
    in polar coordinates (angle TH, radius R) to Cartesian coordinates X,Y.
    The arrays TH and R must have compatible sizes. In the simplest cases,
    they can be the same size or one can be a scalar. Two inputs have
    compatible sizes if, for every dimension, the dimension sizes of the
    inputs are either the same or one of them is 1. TH must be in radians.

    [X,Y,Z] = pol2cart(TH,R,Z) transforms corresponding elements of data
    stored in cylindrical coordinates (angle TH, radius R, height Z) to
    Cartesian coordinates X,Y,Z. The arrays TH, R, and Z must have
    compatible sizes.  TH must be in radians.

    Class support for inputs TH,R,Z:
       float: double, single
```

As seen above, the function pol2cart convert polar coordinates to cartesian coordinates. Using the coordinates obtained in the last section, the values of A and B can be calculated as follows:

```
% 3.2 (d)

theta = 104;
radius = xp(157);
[A, B] = pol2cart(theta*pi/180, radius);
B = -B;
```

The center of the image in the cartesian plane is (145, 179). This can be obtained from the size of the image which is 290x358. This is how much it was translated by. Hence the new ‘x’ is (A + 179) and the new ‘y’ is (B + 145).

Therefore, the value of C, as defined by the equation is Ax + By. Which turns out to be:

$$C = A * (A + 179) + B * (B + 145)$$

The values of the variables resolve to the following:

```
>> A

A =

   18.3861

>> B

B =

   73.7425

>> C

C =

   1.9760e+04
```

3.2 (e) Calculating yl and yr

Based on the original equation of Ax + By = C.

$$yl = \frac{C - A * xl}{B}$$

and,

$$yr = \frac{C - A * xr}{B}$$

Where xl = 0 and xr = 358 – 1 = 357;

The values turn out to be:

```
>> yl

yl =

  267.9563

>> yr

yr =

  178.9463
```

3.2 (f) Superimposing estimated line on image

```
% 3.2 (f)
imshow(P);
line([xl xr], [yl yr]);
```

The result was as follows:

After observing the superimposed line, it can be determined that the line is almost perfectly aligned with the edge of the running path. However, it does deviate at certain parts. There can be some reasons that can be used to explain this deviation.

1. We used Radon transform which is supposed to work on continuous functions but we used it to work on discrete functions. Hence some precision is lost.
2. The path is not straight in reality and hence a non-linear function would be required to perfectly highlight it.

3.3 3D Stereo

3.3 (a) Disparity map algorithm

The algorithm is as follows:

```
function dmap = map(img_l, img_r)
% Function to calculate and return the disparity map

[x, y] = size(img_l); % Both images have same size

dmap = ones(x - 10, y - 10);

for i = 6 : x - 5 % 6: 1 + floor(11/2)
    for j = 6 : y - 5 % 5: floor(11/2)
        cr = img_l(i - 5 : i + 5, j - 5 : j + 5);
        cl = rot90(cr, 2);
        minimum = j;
        difference = inf;

        % searching for similar pattern in the second image
        for k = max(6 , j -14) : j
            T = img_r(i - 5 : i + 5, k - 5 : k + 5);
            cr = rot90(T, 2);
            cv1 = conv2(T, cr);
            cv2 = conv2(T, cl);
            ssd = cv1(11, 11) - 2 * cv2(11, 11);

            if ssd < difference
                difference = ssd;
                minimum = k;
            end
        end
        dmap(i - 5, j - 5) = j - minimum;
    end
end

end
```

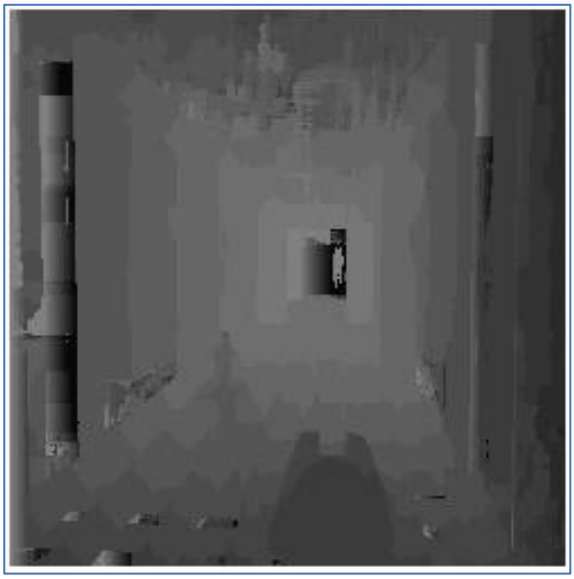
3.3 (b) Corridor images

```
% 3.3 (b)

corridor_l = imread("/Users/omkaringale/Desktop/Computer Vision/" + ...
    "Assignments/Lab2 Edges, Hough Lines, and Disparity/corridorl.jpg");
corridor_l = rgb2gray(corridor_l);
imshow(corridor_l);

corridor_r = imread("/Users/omkaringale/Desktop/Computer Vision/" + ...
    "Assignments/Lab2 Edges, Hough Lines, and Disparity/corridorr.jpg");
corridor_r = rgb2gray(corridor_r);
imshow(corridor_r);
```

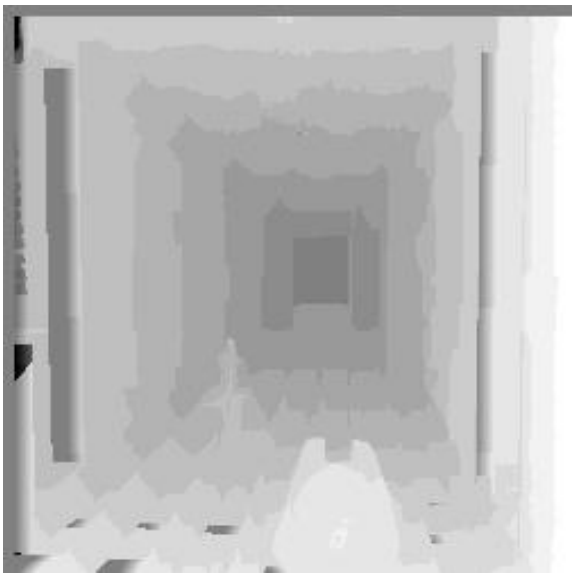
corridor_l

corridor_r

3.3 (c) Disparity map of images



Disparity map



Corridor_disp.jpg

As seen above, the disparity map is darker as we move towards the end of the corridor. We can see that there are some differences compared to corridor_disp as the comparison was only on the same scanline and the search was only limited to 15 pixels. However, the sphere and the cone seem to be correctly mapped.

3.3 (d) Disparity map on triclops-jd.jpg

```
% 3.3 (d)

triclopsi2_l = imread("/Users/omkaringale/Desktop/Computer Vision/" + ...
    "Assignments/Lab2 Edges, Hough Lines, and Disparity/triclopsi2l.jpg");
triclopsi2_l = rgb2gray(triclopsi2_l);
imshow(triclopsi2_l);

triclopsi2_r = imread("/Users/omkaringale/Desktop/Computer Vision/" + ...
    "Assignments/Lab2 Edges, Hough Lines, and Disparity/triclopsi2r.jpg");
triclopsi2_r = rgb2gray(triclopsi2_r);
imshow(triclopsi2_r);

D2 = map(triclopsi2_l, triclopsi2_r);
imshow(-D2,[-15 15]);
```

The result:

The output does not seem to be as good as the first image. The reason can be that the images are very similar. This makes the disparity maps less accurate.