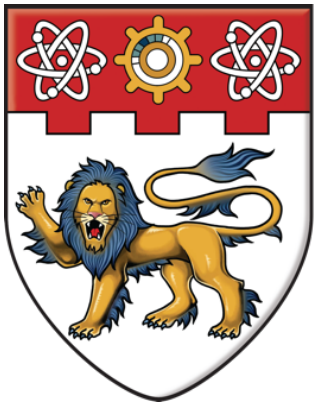


NANYANG TECHNOLOGICAL UNIVERSITY

CZ4003 Computer Vision Laboratory 2

DAI JUNWEI

N902106B

1. Introduction

1.1 Objectives

This laboratory aims to introduce image processing in MATLAB context. In this laboratory you will:

a) Explore different edge detection methods;
b) Employ the Hough Transform to recover strong lines in the image;
c) Experiment with pixel sum-of-squares difference (SSD) to find a template match within a larger image. Estimate disparity maps via SSD computation;
d) Optionally, compare the bag-of-words method with spatial pyramid matching (SPM) on the benchmark Caltech-101 dataset.

2. Experiments

2.1 Edge Detection

a) **Download `macritchie.jpg' from edveNTUre and convert the image to grayscale. Display the image.**


```
>> p = imread('macritchie.jpg');
>> p_gray = rgb2gray(p);
>> imshow (p_gray);
```

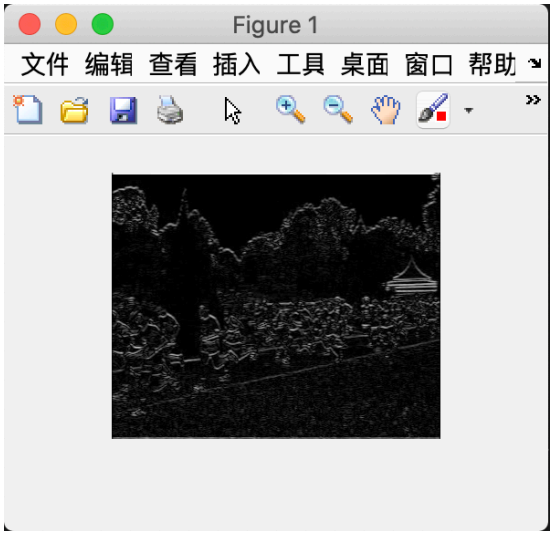
b) **Create 3x3 horizontal and vertical Sobel masks and filter the image using conv2. Display the edge-filtered images. What happens to edges which are not strictly vertical nor horizontal, i.e. diagonal?**

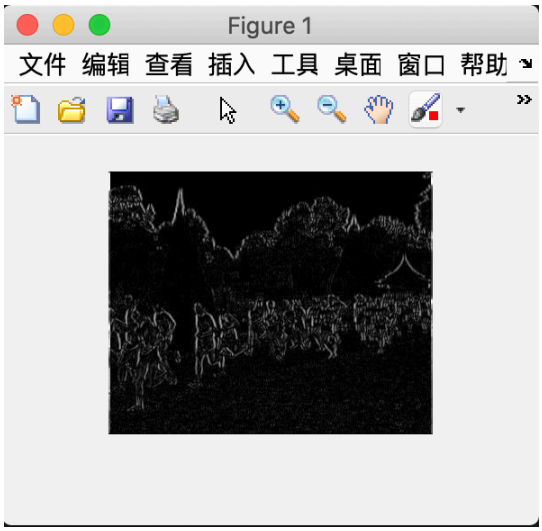
$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Vertical Jumps

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Horizontal Jumps



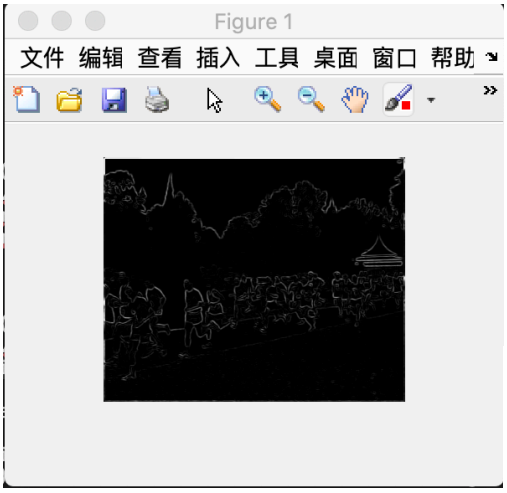



```
>> vertical_matrix= [-1 -2 -1; 0 0 0; 1 2 1];
>> horizontal_matrix= [-1 0 1; -2 0 2; -1 0 1];
>> p2 = conv2(double(p_gray),double(vertical_matrix));
>> p2=abs(p2);
>> p2 = p2*255/996;
>> imshow(unit8(p2));

>> p3 = conv2(double(p_gray),double(horizontal_matrix));
>> p3=abs(p3);
>> p3=p3*255/993;
>> imshow(uint8(P3))
```

The picture on the left is detecting vertical edges， The picture on the right is detecting horizontal edges.

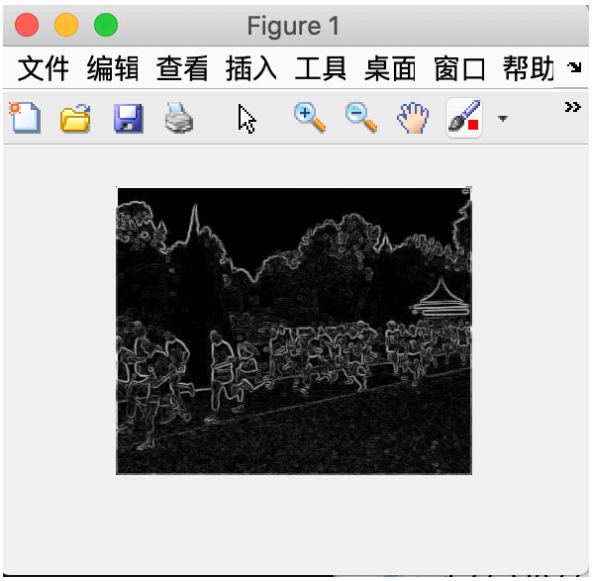
The edges like diagonal disappear from the edge images.

c) **Generate a combined edge image by squaring (i.e. .^2) the horizontal and vertical edge images and adding the squared images. Suggest a reason why a squaring operation is carried out.**




```
>> combine_p=p2.^2 + p3.^2;
>> imshow(combine_p);
>> imshow((combine_p),[]);
```

The squaring operation is done to get the gradient magnitude. Like in the lecture note.

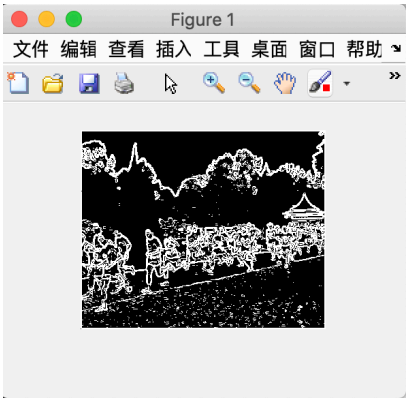



```
>> imshow(uint8(sqrt(combine_p)));
```

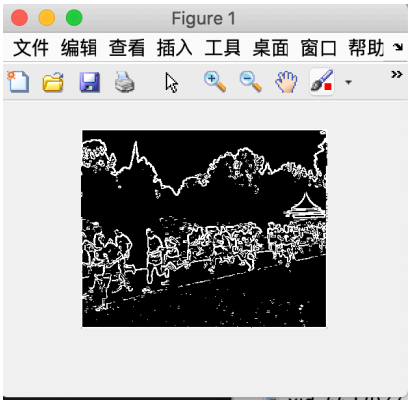
d) **Threshold the edge image E at value t by >> Et = E>t; This creates a binary image. Try different threshold values and display the binary edge images. What are the advantages and disadvantages of using different thresholds?**


```
>> min = min(combine_p(:));
max=max(combine_p(:));
E = 255*(combine_p - min)/(max-min);
```

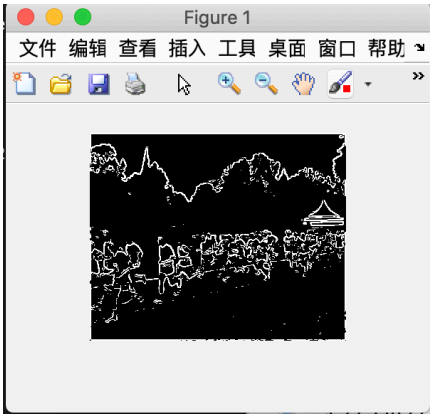
When threshold t = 5;



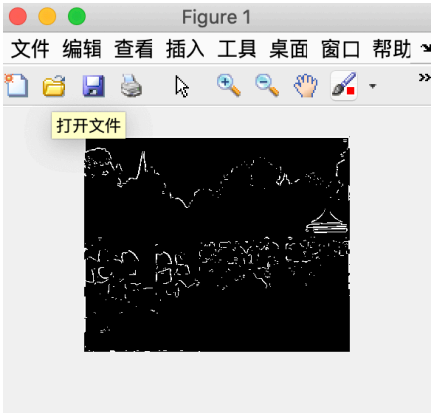
When threshold t = 10;



When threshold t = 20;

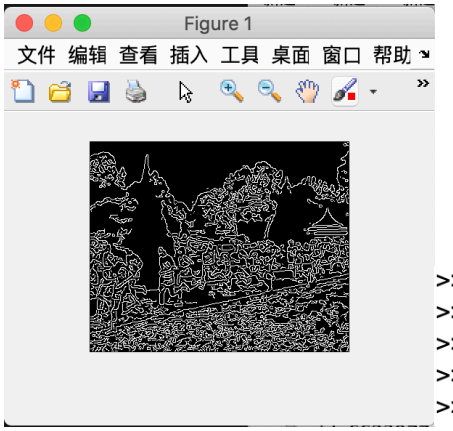


When threshold t = 100;



The threshold value increases, the edge slowly disappear. If there is no threshold, the noise will disturb us and the computer, so it is beneficial for us to find a balance between clear edges and less noise.

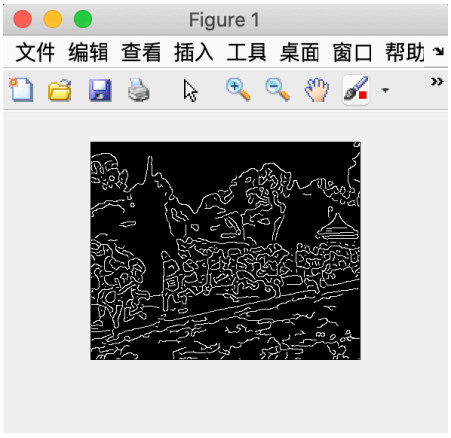
e) **Recompute the edge image using the more advanced Canny edge detection algorithm with tl=0.04, th=0.1, sigma=1.0 >> E = edge(I,'canny',[tl th],sigma);**



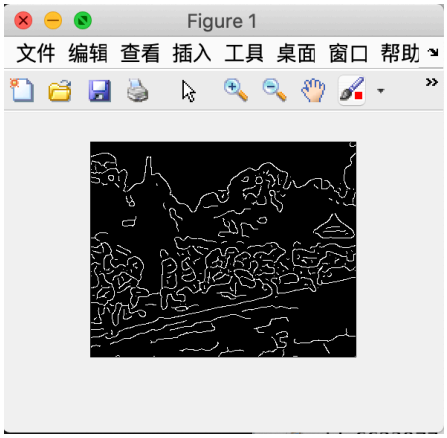

```
>> tl = 0.04;
>> th = 0.1;
>> sigma = 1.0;
>> canny_edge=edge(p_gary,'canny',[tl th],sigma);
>> imshow(canny_edge);
```

(i) Try different values of sigma ranging from 1.0 to 5.0 and determine the effect on the edge images. What do you see and can you give an explanation for why this occurs? Discuss how different sigma are suitable for (a) noisy edgel removal, and (b) location accuracy of edgels.

When sigma = 2.0;

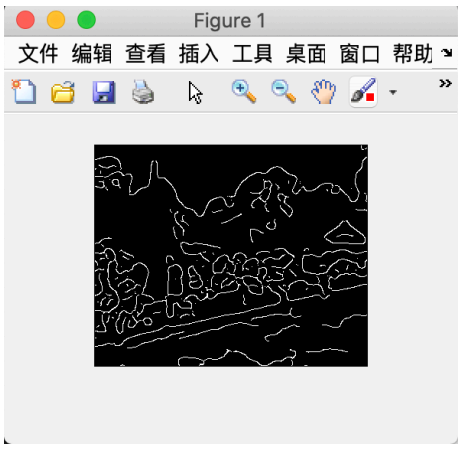


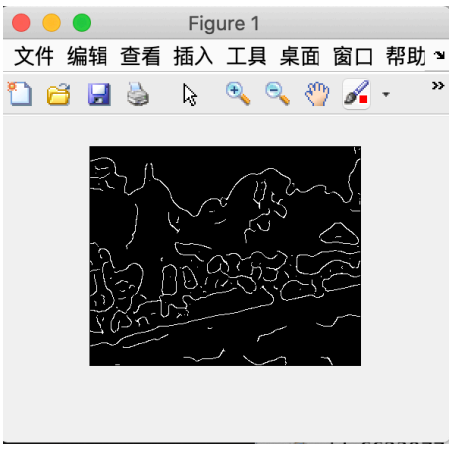
When sigma = 3.0;



When sigma = 4.0;

When sigma = 5.0;





When sigma increases, there are less edge details, and less noise.

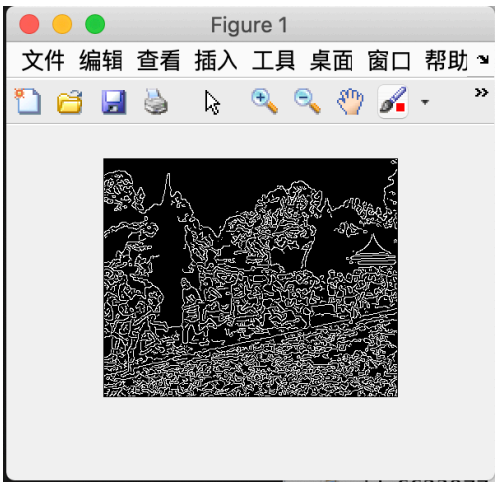
```
tl = 0.04;
th = 0.1;
sigma = 2.0;
canny_edge=edge(p_gary,'canny',[tl th],sigma);
imshow(canny_edge);
>>  p_gary=pic_gray;
tl = 0.04;
th = 0.1;
sigma = 3.0;
canny_edge=edge(p_gary,'canny',[tl th],sigma);
imshow(canny_edge);
>>  p_gary=pic_gray;
tl = 0.04;
th = 0.1;
sigma = 4.0;
canny_edge=edge(p_gary,'canny',[tl th],sigma);
imshow(canny_edge);
>>  p_gary=pic_gray;
tl = 0.04;
th = 0.1;
sigma = 5.0;
canny_edge=edge(p_gary,'canny',[tl th],sigma);
imshow(canny_edge);
```

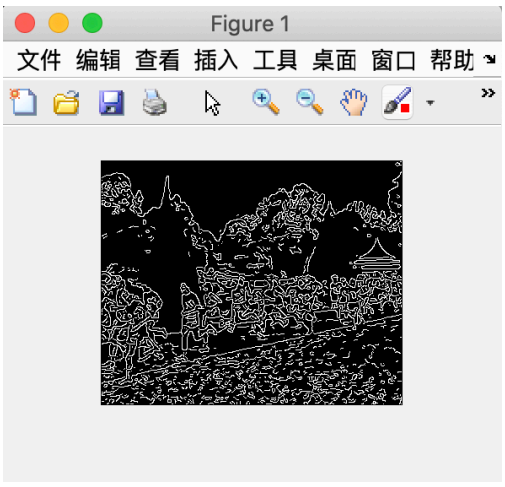
(ii) Try raising and lowering the value of tl. What does this do? How does this relate to your knowledge of the Canny algorithm?

```
tl = 0.02;
th = 0.1;
sigma = 1.0;
canny_edge=edge(p_gary,'canny',[tl th],sigma);
imshow(canny_edge);
>>  p_gary=pic_gray;
tl = 0.08;
th = 0.1;
sigma = 1.0;
canny_edge=edge(p_gary,'canny',[tl th],sigma);
imshow(canny_edge);
```

When tl = 0.02;

When tl = 0.08;



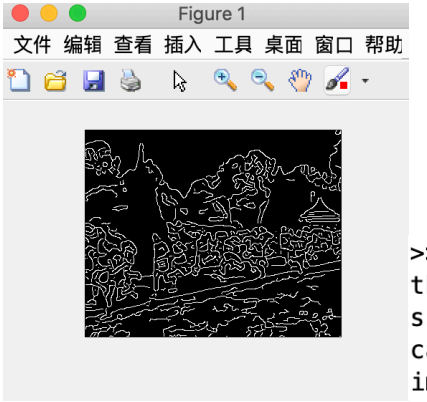


When tl has a lower value, more pixel will be set to one. So among the above images, the first one has more complete edges and more noise.

2.2 Line Finding using Hough Transform

In the section, the goal is to extract the long edge of the path in the `macritchie.jpg' image as a consistent line using the Hough transform.

a) **Reuse the edge image computed via the Canny algorithm with sigma=1.0.**

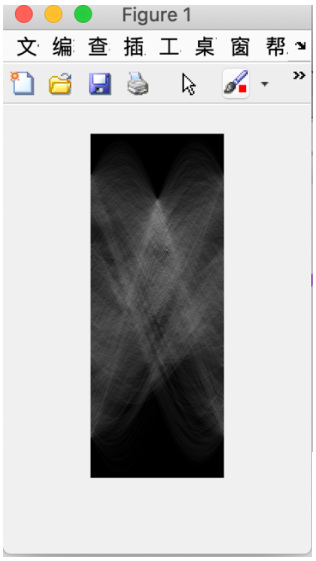



```
>> tl = 0.04;
th = 0.1;
sigma = 2.0;
canny_edge=edge(p_gary,'canny',[tl th],sigma);
imshow(canny_edge);
```

b) **As there is no function available to compute the Hough transform in MATLAB, we will use the Radon transform, which for binary images is equivalent to the Hough transform. Read the help manual on Radon transform, and explain why the transforms are equivalent in this case. When are they different?**

>> [H, xp] = radon(E);

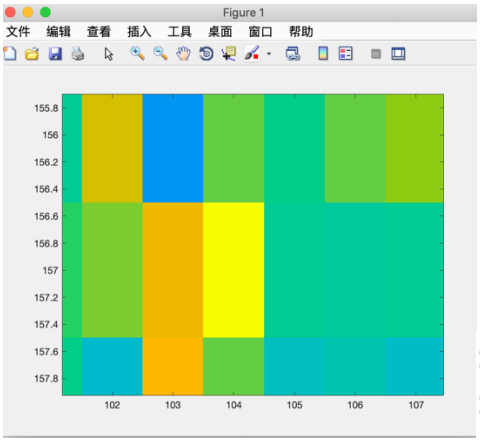
```
>> help radon
radon - Radon transform

    This MATLAB function returns the Radon transform R of the grayscale image I for
    angles in the range [0, 179] degrees.

    R = radon(I)
    R = radon(I,theta)
    [R,xp] = radon(___)
    [gpuarrayR,gpuarrayXp] = radon(gpuarrayI,theta)

    另请参阅 fan2para, fanbeam, ifanbeam, iradon, para2fan, phantom

    radon 的参考页
    名为 radon 的其他函数
```




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

Because of the binary format, the Hough Transform is discrete while the Randon is continuous. So apply a discrete input to the radon method, it will produce the discrete randon transform.

c) **Find the location of the maximum pixel intensity in the Hough image in the form of [theta, radius]. These are the parameters corresponding to the line in the image with the strongest edge support.**




```
>> imagesc(H);
>> colormap('default');
```

Theta = 104; and Radius = 157; Therefore, [theta, radius] = [104, 157].

d) **Derive the equations to convert the [theta, radius] line representation to the normal line equation form Ax + By = C in image coordinates. Show that A and B can be obtained via**

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

```
theta = 104;
 radius = xp(157);
 [A, B] = pol2cart(theta*pi/180, radius);
 B = -B;
 C = A*(A+179) + B*(B+145);
```

```
>> A

A =

   18.3861

>> B

B =

   73.7425

>> C

C =

   1.9760e+04
```

e) **Based on the equation of the line Ax+By = C that you obtained, compute yl and yr values for corresponding xl = 0 and xr = width of image - 1.**

```
xl = 0;
 yl = (C - A * xl) / B;
 xr = 357;
 yr = (C - A * xr) / B;
```

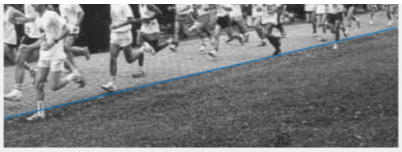
f) **Display the original 'macritchie.jpg' image. Superimpose your estimated line by >> line([xl xr], [yl yr]); Does the line match up with the edge of the running path? What are, if any, sources of errors? Can you suggest ways of improving the estimation?**

The line on the left picture does not completely match the line. Because the runway path is not a perfect straight line.

Also, the use of Canny edge and Randon transform will lose some details.

After I talk to my classmates I fund that their theta is 103, while my theta is 104 . so I tried 103 instead , and I fund that the result turns out greater than theta =104.

theta=104



theta=103

2.3 3D Stereo Vision

This is a fairly substantial section as you will need to write a MATLAB function script to compute disparity images (or maps) for pairs of rectified stereo images Pl and Pr. The disparity map is inversely proportional to the depth map which gives the distance of various points in the scene from the camera.

Estimating Disparity Maps

The overview of the algorithm is:

i. for each pixel in Pl,
ii. Extract a template comprising the 11x11 neighbourhood region around that pixel.

Using the template, carry out SSD matching in Pr, but only along the same scanline. The disparity is given by

$$d(x_l, y_l) = x_l - \hat{x}_r$$

where xl and yl are the relevant pixel coordinates in Pl, and x_hat is the SSD matched pixel's x-coordinate in Pr. You should also constrain your horizontal search to small values of disparity (<15).

Note that you may use conv2, ones and rot90 functions (may be more than once) to compute the SSD matching between the template and the input image. Refer to the equation in section 2.3 for help.

iii. Input the disparity into the disparity map with the same Pl pixel coordinates.

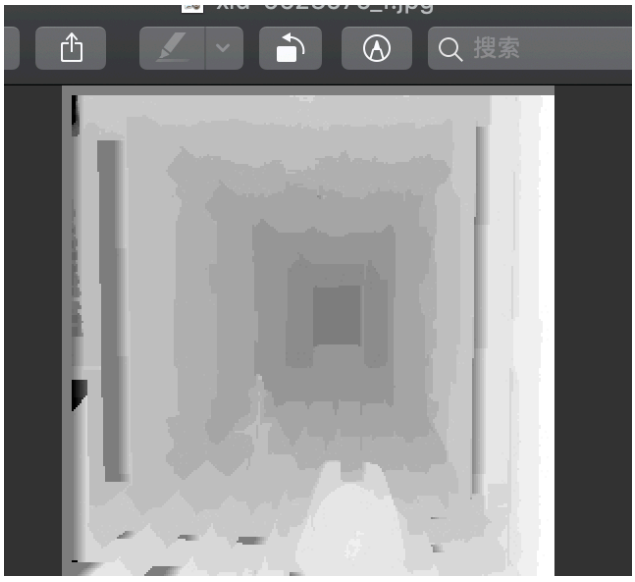
a) Write the disparity map algorithm as a MATLAB function script which takes two arguments of left and right images, and 2 arguments specifying the template dimensions. It should return the disparity map. Try and minimize the use of for loops, instead relying on the vector / matrix processing functions.

```
function desparity = map(image_left,image_right,template_x,template_y)

x_dim = floor(template_x/2);
y_dim = floor(template_y/2);
[x,y] = size(image_left);
desparity = ones(x-template_x+1,y-template_y+1);

for i=1+x_dim:-x_dim
    for j =1+y_dim:y-y_dim
        r = image_left(i-x_dim: i+x_dim, j-y_dim: j+y_dim);
        l = rot90(r, 2);
        c = j;
        min_ssd = 1000000;
        for k = max(1+y_dim , j-14) : j
            T = image_right(i-x_dim: i+x_dim, k-y_dim: k+y_dim);
            r = rot90(T, 2);
            conv_1 = conv2(T, r);
            conv_2 = conv2(T, l);
            ssd = conv_1(template_x,template_y) - 2 * conv_2(template_x, template_y);
            if ssd < min_ssd
                min_ssd = ssd;
                c = k;
            end
        end
        desparity(i - x_dim, j - y_dim) = j - c;
    end
end
```

b) Download the synthetic stereo pair images of 'corridorl.jpg' and 'corridorr.jpg', converting both to grayscale.

```
>> r=imread('right.jpg');
>> l=imread('left.jpg');
>> r=rgb2gray(r);
>> l=rgb2gray(l);
```

c) **Run your algorithm on the two images to obtain a disparity map D, and see the results via >> imshow(-D,[-15 15]); The results should show the nearer points as bright and the further points as dark. The expected quality of the image should be similar to `corridor_disp.jpg' which you can view for reference.**




```
D=map(l,r,11,11);
imshow(D,[-15 15]);
```

f) **Rerun your algorithm on the real images of 'triclops-i2l.jpg' and triclops-i2r.jpg'. Again you may refer to 'triclops-id.jpg' for expected quality. How does the image structure of the stereo images affect the accuracy of the estimated disparities?**

1. it may due to the quality of the picture we use in the part c is greater than above one we use in part f.
2. maybe because ,it has a large area that have the same color ,and it affect the performance of SSD matching.
3. In part c it has a lot of straight line and regular graphs(i.e. square triangle) while in the part f the picture is composed of many irregular graphs.