3.1 Edge Detection

1. Display the image which is converted into gray-scale

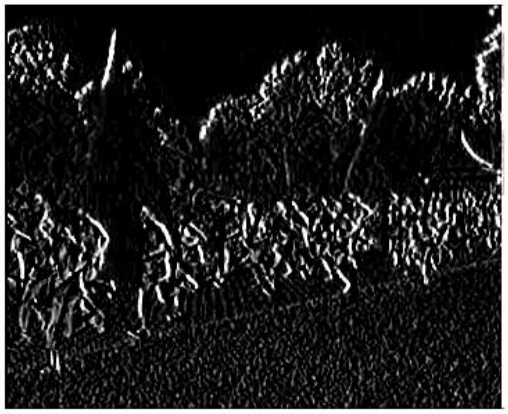
```
>>PC3=imread('macritchie.jpg');
>>P31 = rgb2gray(PC3);
>>imshow(P31);
```

2. Create 3x3 vertical and horizontal Sobel masks and filter the image.

```
>>verti=[-1,0,1;
        -2,0,2;
        -1,0,1];
>>hori=[-1,-2,-1;
       0,0,0;
       1,2,1];
>>v=conv2(double(P31),double(verti));
>>h=conv2(double(P31),double(hori));
```

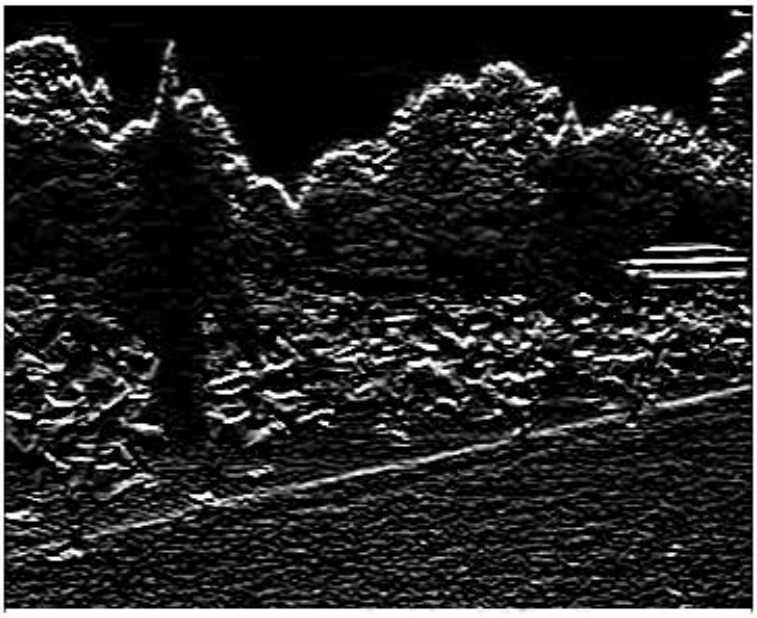
%Display image after applying vertical mask filter

```
>>imshow(uint8(v));
```



%Display image after applying horizontal mask filter

```
>>imshow(uint8(h));
```



We can see from the generated image, since diagonal edge can be decomposed with horizontal and vertical direction, if the edge is nor strictly horizontal or vertical, it could still be detected. However, for example the running path edge, with the horizontal filter, the edge is clearer and with the vertical filter, the edge can not be proper detected. And this is because the running path has larger horizontal vector components.

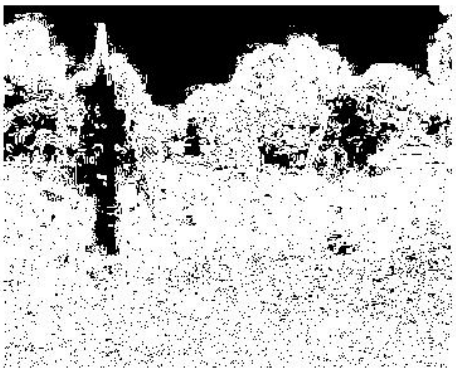
3. Carry out the square operation

```
>>Pa=imadd((v.^2),(h.^2));
>>Pa=sqrt(Pa);
```

The reason of using square operation is that there won't be any negative value left, and it is easier to calculate the sobel matrix. Since we don't have to care about the negative part.

4.Threshold the image at different values {20,60,100,150,300}

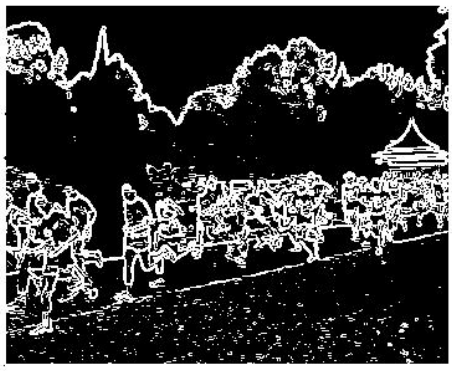
```
>>E20 = Pa>20;
>>imshow(E20);
```



```
>>E60 = Pa>60;
>>imshow(E60);
```

```
>>E100 = Pa>100;
>>imshow(E100);
```

```
>>E150 = Pa>150;
>>imshow(E150);
```



```
>>E300 = Pa>300;
>>imshow(E300);
```

If the threshold is chosen too low, we have too much details and can not really see the edge. If the threshold is too high, we omit too many information, and therefore can not see the details clearly. With the suitable threshold, we are allowed to get the result we want.

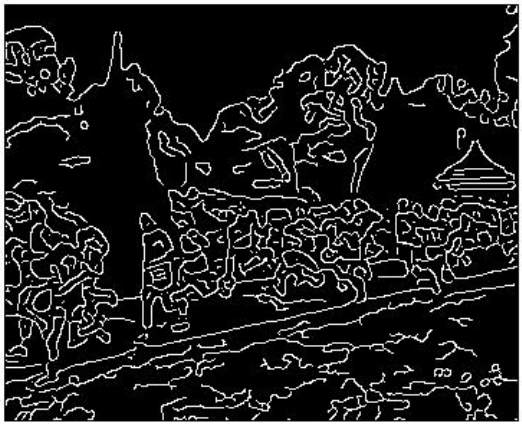
5. Compute edge image using Canny edge detection.

%Trying different sigma {1,2,3,4,5}

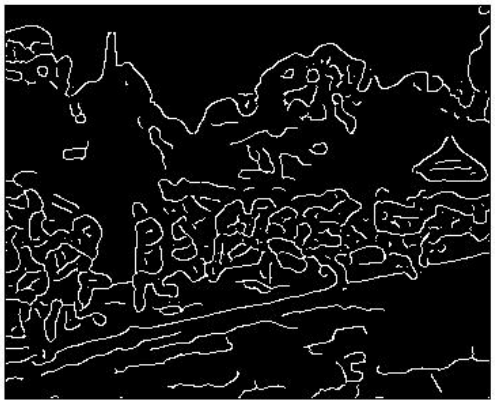
```
>>tl=0.04;th=0.1;
```

```
>>E1 = edge(P31,'canny',[tl th],1);
>>imshow(E1);
```

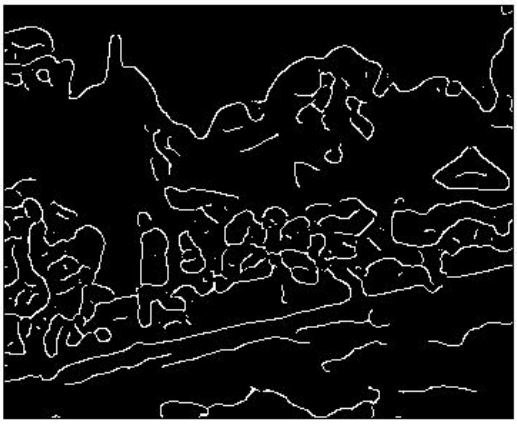
```
>>E2 = edge(P31,'canny',[tl th],2);
>>imshow(E2);
```



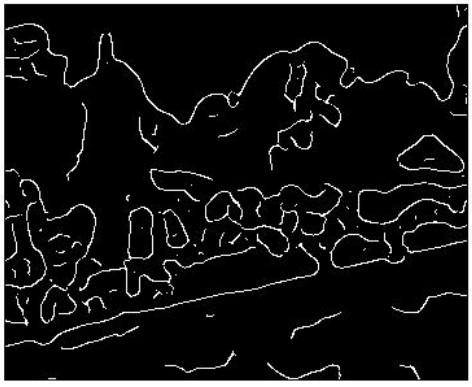
```
>>E3 = edge(P31,'canny',[tl th],3);
>>imshow(E3);
```



```
>>E4 = edge(P31,'canny',[tl th],4);
>>imshow(E4);
```



```
>>E5 = edge(P31,'canny',[tl th],5);
>>imshow(E5);
```



The sigma here is the standard deviation of the filter, and therefore the higher sigma has more influenced on adjacent pixels than the lower sigma.
From image, the lower the sigma is more edges and noises, for example the grass. The higher the sigma is more noises have been cancelled; however, we lost most of the edges.
The higher sigma is suitable for noise edgel removal, since it blurs the image and cancels the noise.
The lower sigma is suitable for location accuracy of edgels, since we get the more detailed image.
(Continued to next page)

%Trying different tl {0.01,0.02,0.04,0.07,0.09}

```
>>T1 = edge(P31,'canny',[0.01 th],1);
>>imshow(T1);
```

```
>>T2 = edge(P31,'canny',[0.02 th],1);
>>imshow(T2);
```

```
>>T4 = edge(P31,'canny',[0.04 th],1);
>>imshow(T4);
```

```
>> T7 = edge(P31,'canny',[0.07 th],1);
>>imshow(T7);
```

```
>>T9 = edge(P31,'canny',[0.09 th],1);
>>imshow(T9);
```

The lower the tl is more details and edges could be shown.From the canny algorithm, each pixel can be separated into edges, non-edges or in-between by the threshold value provided. And the pixel in-between could be edge pixel also, if it is connected to other edge pixels. Therefore the tl represents the minimum threshold, and the range is larger when tl is lower, so it could detects more edges.

3.2 Line Finding using Hough Transform

1. Reuse the edge image computed via the Canny algorithm with sigma=1.0

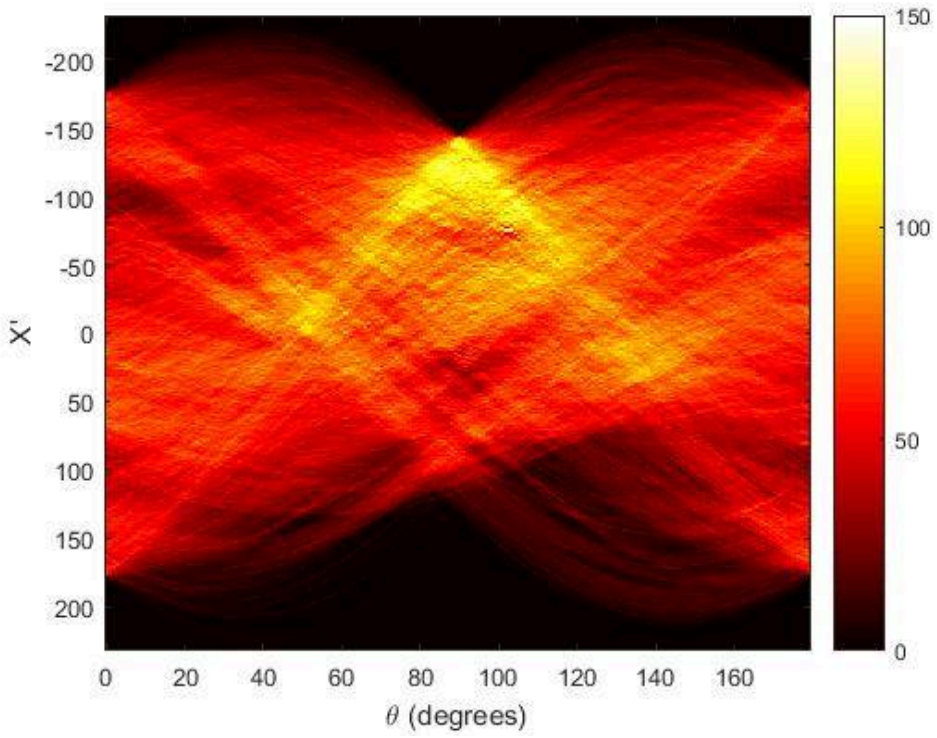
```
>>PC32=imread('macritchie.jpg');
>>PG32 = rgb2gray(PC32);
>>tl=0.04;th=0.1;
>>E=edge(PG32,'canny',[tl th],1);
>>theta=0:179;
```

2. The Radon transform and Hough transform are equivalent since this is a binary image. It wouldn't be the same, when the image is not binary since the intensity of the pixel may be bias, and could not fully represent the maximum intensity as the edge, where hough transform has the intersection as its straight edge.

```
>>[H, xp] = radon(E);
>>imagesc(theta,xp,H);
>>xlabel('\theta (degrees)');
>>ylabel('X''');
```

% To visualize the result, we apply the colormap

```
>>colormap(hot);
>>colorbar
```



2.Find the position of the maximum value, which the point is on the line that has the most edgels.

```
>>[max_value,idx]=max(H(:));
>>[x_max,y_max]=ind2sub(size(H),idx);
```

%Find line theta and radius with strongest edge support

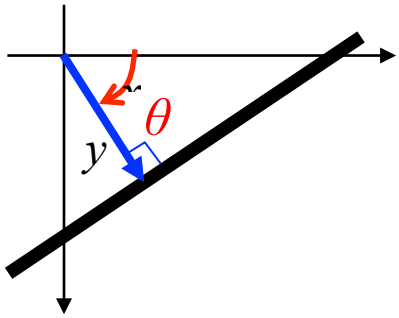
```
>>theta=theta(y_max);
>>radius=xp(x_max);
```

3. Convert the [theta, radius] line representation to the normal line equation form Ax + By = C in image coordinates, and y-axis is pointing downwards.

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

4. Compute C

C1 is the original intercept when the origin has not been shifted.



$$\rho = x\cos\theta + y\sin\theta$$
$$\Rightarrow y\sin\theta = \rho - x\cos\theta$$
$$\Rightarrow y = -\frac{\cos\theta}{\sin\theta}x + \frac{\rho}{\sin\theta}$$
$$\text{or } y = -\cot\theta\, x + \rho\csc\theta$$

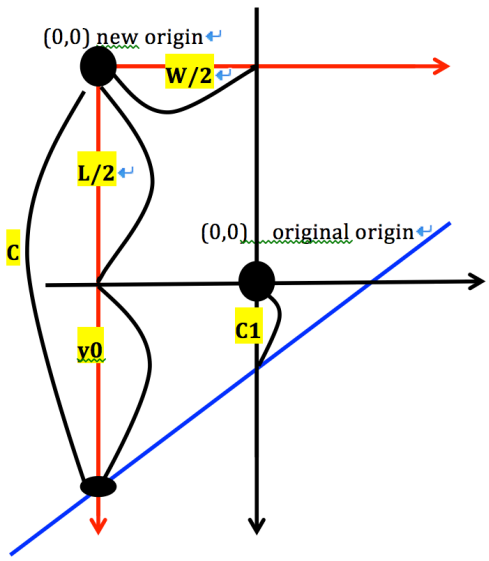
（Figures from tutorial material）

% Original intercept=radius*csc(theta*pi/180)= radius/sin(theta*pi/180),and since the y-axis has changed direction. C1=-radius/sin(theta*pi/180)

```
>>C1=-(radius/sin(theta*pi/180));
```

% C is the new intercept, and from the graph you can see that C=L/2+the y original value of the intersection of line and new y-axis

```
>>[L,W]=size(E)
>>x0=-W/2;
>>y0=(-A/B)*x0+C1;
>>C=y0+(L/2);
```



5. Compute yl and yr values by Ax+By=C for corresponding xl = 0 and xr = width of image - 1 (Width of image is 358)

```
>>xl=0;
>>yl=(-A/B)*xl+C;
>>xr=358-1;
>>yr=(-A/B)*xr+C;
```

% Display original image

```
>>imshow(PC32);
```

6. Superimpose the line in red

```
>>line([xl xr], [yl yr],'Color',[1,0,0]);
```

The line has matched up to the edge of running path. If there are errors, it could be possibly caused by theta, since from this case we given 0-179 degrees in integers, so if the theta is decimal digit, it may cause some errors.

3.3 3D Stereo

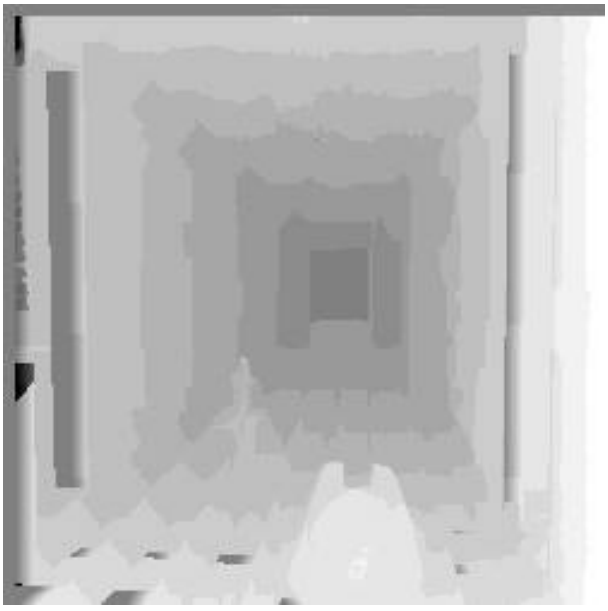
1. Load the image

```
>>Pl=imread('corridorl.jpg');
>>Pr=imread('corridorr.jpg');
>>Pd=imread('corridor_disp.jpg');
>>Plg = rgb2gray(Pl);
>>Prg = rgb2gray(Pr);
```

%Run the function

```
>>dismap = dis_map(Plg,Prg,11,11);
>>imshow(-dismap,[-15 15]);
```

(result)



(corridor_disp.jpg)

2. The result I get is similar to the expected image. The difference is the deepest part, which is also where the wall is. The quality of the disparities is not that good since the wall should be the deepest part which is the darkest color. When the part of the image is flat and doesn't have a lots of different components like the wall in this case, it's hard to find the match pixel from left and right pictures. It could lead to false result, since it match the wrong pixel and regards it as shallow rather than deep.

%dis_map function

```
function dismap = dis_map(Pl,Pr,h,w)
[x,y]= size(Pl);
xs=ceil((h+1)/2);
ys=ceil((w+1)/2);
dismap=ones(x,y);
```

% set the area outside the boundary of the image to 0, in order to avoid edge problems

```
Pln=zeros(x+xs,y+ys+150);
Prn=zeros(x+xs,y+ys+150);
Pln(1:x,15+(ys-1):y+14+(ys-1))=Pl;
Prn(1:x,15+(ys-1):y+14+(ys-1))=Pr;
```

% First move 15 pixels after in order to avoid any edge problem

```
for i=xs:x
    for j=ys+15+(ys-1):y+15+(ys-1)
    template=Pln(i-(xs-1):i+(xs-1),j-(ys-1):j+(ys-1));
          I=Prn(i-(xs-1):i+(xs-1),j-15-(ys-1):j+15+(ys-1));
        TD=double(template);
        TD=rot90(rot90(TD));
        Isquare=double(I).*(double(I));
```

%Create F as an all 1 matrix, and do the convolution with Isquare, in order to get the (h,w) size matrix. Use the equation from the sheet

```
        F=ones(h,w);
 Ss=conv2(Isquare,F,'same')-2*conv2(double(I),TD,'same');
```

%Generate the array with the SSD value

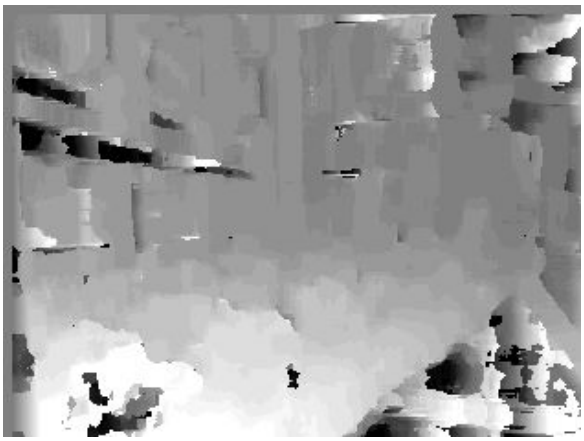
```
        [sh,sw]=size(Ss);
```

```
        % Get only the array which have not been influenced by the boundary
          and has been fully convolution.
        Ssd=Ss(xs,ys:sw-(ys-1));
        %We look for the disparity which is the minimum SSD value.
        [V,I]=min(Ssd);
        % We add the value to the disparity matrix and since we have moved
          the position, we have to move it back
        dismap(i,j-15-(ys-1))=I-15;
            end
        end
end
```

4. Rerun the algorithm

```
>>Kl=imread('triclopsi2l.jpg');
>>Kr=imread('triclopsi2R.jpg');
>>Kd=imread('triclopsid.jpg');
>>Klg = rgb2gray(Kl);
>>Krg = rgb2gray(Kr);
>>dismap = dis_map(Klg,Krg,11,11);
>>imshow(-dismap,[-15 15]);
```

(Result)



(`triclopsid.jpg`)

The result is very similar. However, the pavement part is not that accurate, and the reason is that firstly the pavement's components are very similar and it could easily detect to the same SSD, secondly, the angle of two photo for the pavement is slightly different and this could lead to low accuracy of estimated disparities.