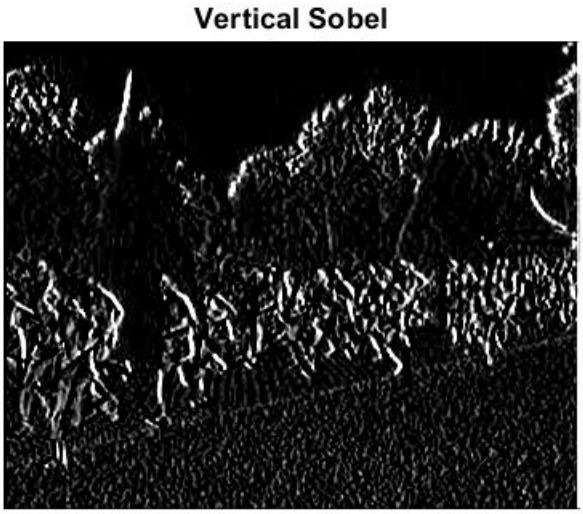
Computer Vision (CZ4003) - Lab 2

Part 1: EDGE DETECTION

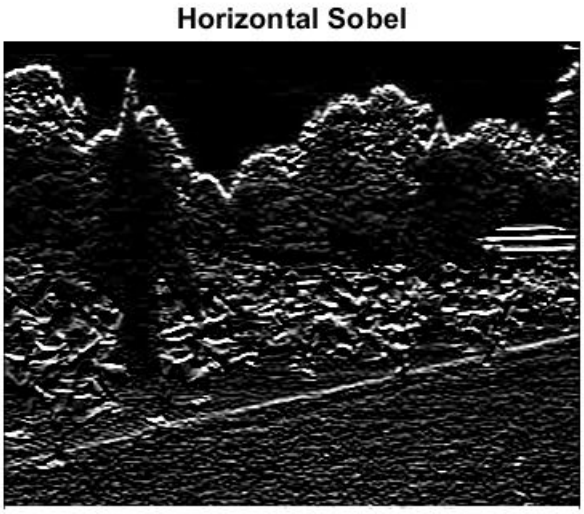
```
% part A: Read in image
P = imread('images/maccropped.jpg');
I = rgb2gray(P);
imshow(I);
```

```
% part B: Sobel filtering
sobel_v = [
    -1 0 1;
    -2 0 2;
    -1 0 1;
];
sobel_h = [
    -1 -2 -1;
    0 0 0;
    1 2 1;
];

res1 = conv2(I, sobel_v);
imshow(uint8(res1)), title("Vertical Sobel");
```



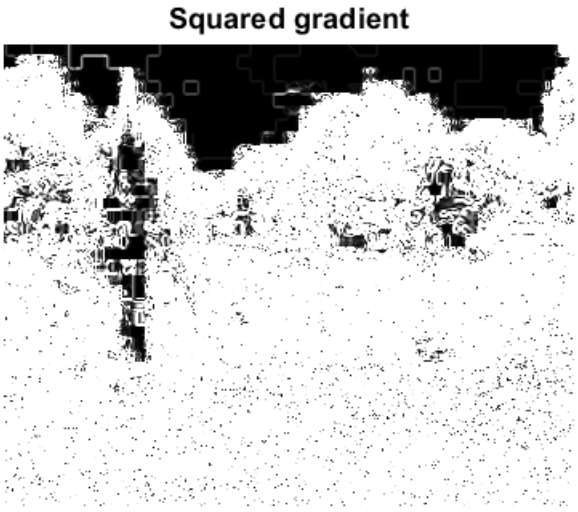

```
res2 = conv2(I, sobel_h);
imshow(uint8(res2)), title("Horizontal Sobel");
```



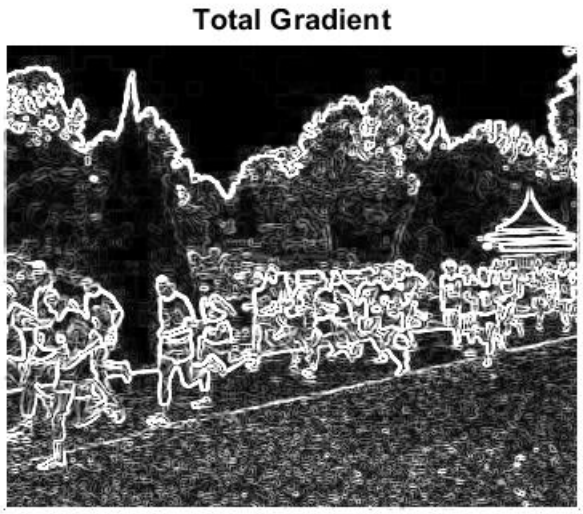
Since sobel kernel is designed to identify gradient along one specific direction (**x** and **y** in this case), region with diagnal gradient will appear with **low intensity, or invisible** in the resultant images.

For the intuition behind Sobel, it's a simple **gradient** operator $[-\frac{1}{2},\ 0,\ \frac{1}{2}]$ combined with a shortest non-trivial **Pascal/Gaussian smoothing** $[\frac{1}{4},\ \frac{1}{2},\ \frac{1}{4}]^T$, this however will not be affect if the global scale factor are normalized. Thus the product of 2 operations give a 3x3 filter. Notice that elements of Sobel filters are **dyadic numbers** so it's efficient when dealing with binary operations (less round-off error, robust to bit shifts, etc).

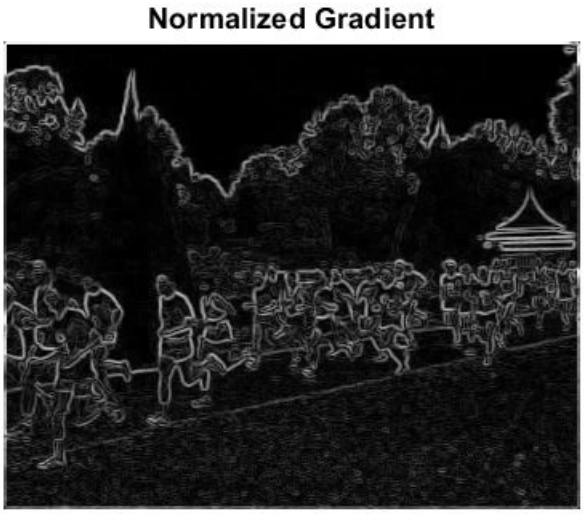
```
% part C: Get total Gradient
E = res1.^2 + res2.^2;
imshow(uint8(E)), title("Squared gradient");
```




```
E = sqrt(E);
imshow(uint8(E)), title("Total Gradient");
```



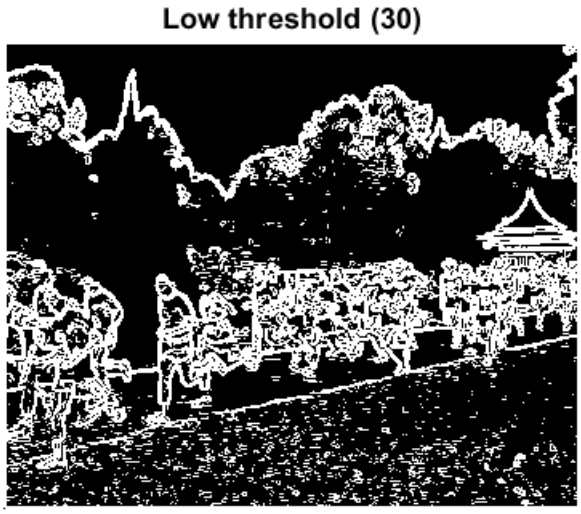

```
E = (E - min(E(:))) / (max(E(:)) - min(E(:))) * 255;
imshow(uint8(E)), title("Normalized Gradient");
```



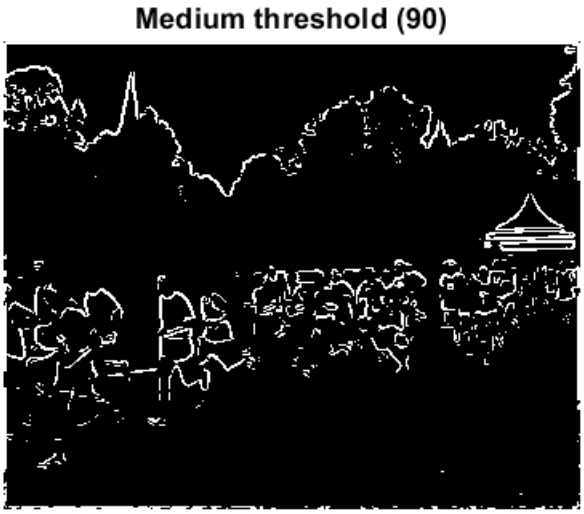
As stated above, Sobel kernel output magnitude of **images gradient projected onto a specific direction**. Thus, the sum of square is the **squared magnitude of gradient**. However, in the **original** Canny algorithm, square root of E and an orientation map $\Theta = atan(\frac{G_y}{G_x})$ is used. *Thus I will use the **magnitude of gradient (e.g: square root of sum square), then normalized the result by a MinMaxScaler (e.g: max value is 255)** for this experiment. The square root gives much better result and MinMaxScaler is to reduce noisy pattern due to automated clamping by MatLab.*

As expected, this can display diagonal edges, details and noise level is acceptable.

```
% part D: Simple thresholding
t = [30, 90, 150];
Et = E>t(1); imshow(Et), title("Low threshold (30)");
```




```
Et = E>t(2); imshow(Et), title("Medium threshold (90)");
```




```
Et = E>t(3); imshow(Et), title("High threshold (150)");
```

High threshold (150)

As aboved, images is normalized to be in range [0, 255]. Level of thresholding are: **30 = low**, **90 = medium**, **150 = high.** Low level is too noisy while high level filter out most of image details, thus medium thresholding is preferred. Nevertheless, this customized thresholding allow practitioner to extract and choose level detailedness based on preference or specific use case. (e.g: set high threshold if want to select sharp edge, low threshold if want to see every possible edges).

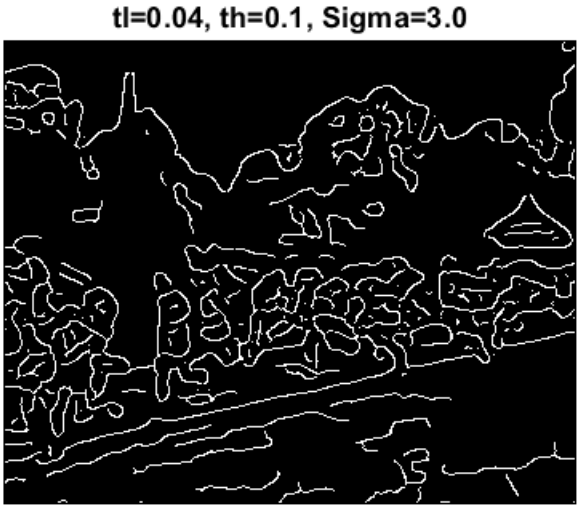
However, simple thresholding produce a binary image, thus resultant images' quality is very limited (in terms of brightness integrity). And to note, a systematic thresholding scheme (Otsu) might obtain a better (balance) result.

```
% Part E: MatLAB's Canny implementation.
tl = 0.04; th = 0.1; sigma = 1.0;
```

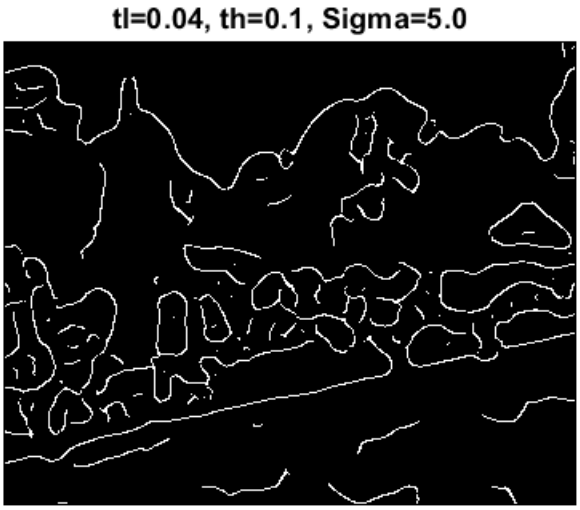
(i) Varying Sigma = [1.0, 3.0, 5.0]

```
sigmas = [1.0, 3.0, 5.0];
E = edge(I, 'canny', [tl th], sigmas(1));
imshow(E), title("tl=0.04, th=0.1, Sigma=1.0");
```


```
E = edge(I, 'canny', [tl th], sigmas(2));
imshow(E), title("tl=0.04, th=0.1, Sigma=3.0");
```




```
E = edge(I, 'canny', [tl th], sigmas(3));
imshow(E), title("tl=0.04, th=0.1, Sigma=5.0");
```



As can easily be seen from the experiment, **raising sigma filters out more details** in the resultant image.

This is due to the nature of **Gaussian filtering, smaller sigma** mean steeper filter: $\Psi(x, y) = \frac{1}{2\pi} * exp(-\frac{x^2 + y^2}{2\sigma^2})$,

steeper filter $\rightarrow$ **better localization,** but **weaker smoothing** effects. Thus, low sigma value ($\sigma = 1$) good at localize edgels, keeping much of the details while high sigma ($\sigma = 5$) is better at removing noisy edges.

(ii) Varying low threshold: th = [1.0, 3.0, 5.0]

```
tls = [0.07, 0.04, 0.01];
E = edge(I, 'canny' ,[tls(1) th], sigma);
imshow(E), title("tl=0.07, th=0.1, Sigma=5.0");
```

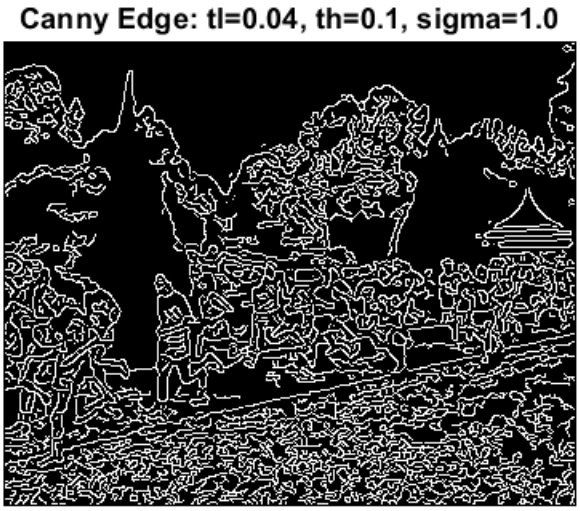
```
E = edge(I, 'canny' ,[tls(2) th], sigma);
imshow(E), title("tl=0.04, th=0.1, Sigma=5.0");
```


```
E = edge(I, 'canny' ,[tls(3) th], sigma);
imshow(E), title("tl=0.01, th=0.1, Sigma=5.0");
```

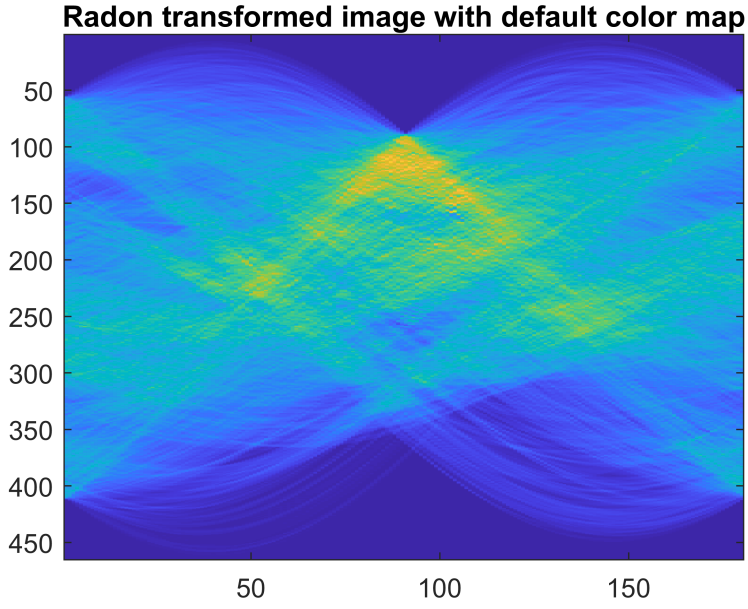
In Canny Edge detection algorithm, **hysteresis thresholding** is performed by selecting **tl** and **th,** then value above **th** is kept, below **tl** is discarded, between **tl** and **th** is decided to whether be kept or discarded by neighborhood voting (as in Ising model). So it is intuitively easy to see **lowering tl** will result in **more details** are kept, as we expected from the experiment above.

Part 2: LINE DETECTION

```
% Part A: Read in -> Canny EDGE.
P = imread('images/maccropped.jpg');
I = rgb2gray(P);
tl = 0.04; th = 0.1; sigma = 1.0;
E = edge(I, 'canny', [tl th], sigma);
imshow(E), title("Canny Edge: tl=0.04, th=0.1, sigma=1.0")
```

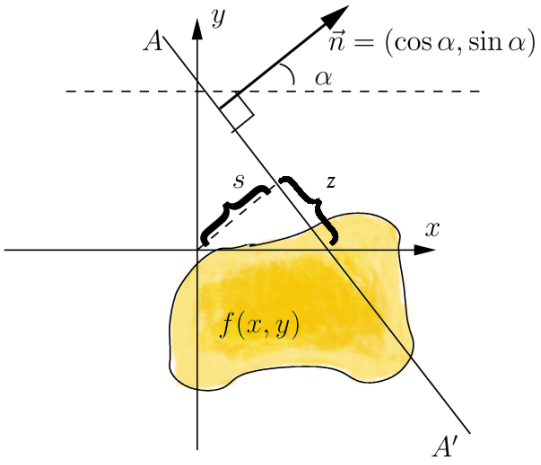



```
% Part B: Radon transform (Hough binary equivalent)
[R, xp] = radon(E, 0:179);
imagesc(uint8(R));
colormap('default'), title("Radon transformed image with default color map");
```



Here instead of Hough transform, we use Radon transform, which is a discrete equivalent relation because:

1. Recall **Hough transform** and its inverse move back and forth 2 spaces: $(\rho,\ \theta) \leftrightarrow (x,\ y)$ by the relation: $\rho = x * cos(\theta) + y * \sin(\theta)$. That is, a **line in** $(x,\ y)$ corresponding to a **point in** $(\rho,\ \theta)$ while a **wave in** $(\rho,\ \theta)$ is a **point in** $(x,\ y)$.
2. Now the **Radon transform** defined by:




```
% Part C: Pixel with maximum value.
[row, col] = find(ismember(R, max(R(:))))
```

```
row = 157
col = 104
```

```
% Part D:
theta = 103; % since theta is 0-indexed
radius = xp(157); % x is 1-indexed.
[A, B] = pol2cart(theta*pi/180, radius);
B = -B;

C = A*(A+179) + B*(B+145);
```

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

%f
imshow(I)
line([xl xr], [yl yr])
```

Part 3: STEREO VISION

```
% a

% b
l = imread('images/corridorl.jpg');
l = rgb2gray(l);
imshow(l);
```

```
r = imread('images/corridorr.jpg');
r = rgb2gray(r);
imshow(r);
```

```
% c
D = map(l, r, 11, 11);
```

```
Unrecognized function or variable 'map'.
```

```
imshow(D,[-15 15]);
res = imread('image/corridor_disp.jpg');
imshow(res);

%d
l = imread('image/triclopsi2l.jpg');
l = rgb2gray(l);
imshow(l);
r = imread('image/triclopsi2r.jpg');
r = rgb2gray(r);
imshow(r);

D = map(l, r, 11, 11);
imshow(D,[-15 15]);
res = imread('image/triclopsid.jpg');
imshow(res);
```