Point Operations and Spatial Filtering

Ranga Rodrigo

November 13, 2011
1. Point Operations
   - Histogram Processing

2. Spatial Filtering
   - Smoothing Spatial Filters
   - Sharpening Spatial Filters

3. Edge Detection
   - Line Detection Using the Hough Transform
Outline

1. Point Operations
   - Histogram Processing

2. Spatial Filtering
   - Smoothing Spatial Filters
   - Sharpening Spatial Filters

3. Edge Detection
   - Line Detection Using the Hough Transform
We can process images either in spatial domain or in transform domains. Spatial domain refers to image plane itself. In spatial domain processing, we directly manipulate pixels. Spatial domain processing is common. Frequency domain and wavelets are examples of transform domains.
Domains

Image enhancement

Spatial domain
Transform domain
Spatial Domain Processing

- Spatial domain processing
  - Range operations
  - Domain operations
  - Point operations
  - Neighborhood operations
Point Operations

- $g(x_0, y_0) = T[f(x_0, y_0)]$.
- These are known as gray level transformations.
- Examples:
  - Gamma correction.
  - Window-center correction.
  - Histogram equalization.
Neighborhood Operations

\[ g(x_0, y_0) = T_{(x, y) \in \mathcal{N}} [f(x, y)], \text{ where } \mathcal{N} \text{ is some neighborhood of } (x_0, y_0). \]

Examples:
- Filtering: mean, Gaussian, median etc.
- Image gradients etc.
Domain Operations

\[ g(x_0, y_0) = f(T_x(x_0, y_0), T_y(x_0, y_0)) \].

Examples:

- Warping.
- Flipping, rotating, etc.
- Image registration.
Point Operations: Recapitulation

- \( g(x_0, y_0) = T[f(x_0, y_0)] \).
- These are known as gray level transformations.
- The enhanced value of a pixel depends only on the original value of the pixel.
- If we denote the value of the pixel before and after the transformation as \( r \) and \( s \), respectively, the above expression reduces to
  \[
  s = T(r). \tag{1}
  \]
- For example, \( s = 255 - r \) gives the negative of the image.
Figure 1: Intensity transformation function for image negative.
Example

Write a program to generate the negative of an image.
im = imread('image.jpg');
imneg = 255 - im;
subplot(1,2,1)
imshow(im)
title('Original')
subplot(1,2,2)
imshow(imneg)
title('Negative')
Power-Law Transformations

\[ s = cr^\gamma, \]

(2)

where \( c \) and \( \gamma \) are positive constants.

Values of \( \gamma \) such that \( 0 < \gamma < 1 \) map a narrow range of dark input pixels into a wider range of output values, with opposite being true for higher values of input levels.

Values \( \gamma > 1 \) have the opposite behavior to above.

\[ c = \gamma = 1 \] gives the identity transformation.

Gamma correction is an application.
Figure 2: Intensity transformation function using power law.
Example

Write a program to carry out power-law transformation on an image.
```matlab
r = imread('image.jpg');
c = 1;
gamma = 0.9;
s1 = c * r.^gamma;
gamma = 1.2;
s2 = c * r.^gamma;
subplot(1,3,1)
imshow(r)
title('Original')
subplot(1,3,2)
imshow(s1)
title('Gamma = 0.9')
subplot(1,3,3)
imshow(s2)
title('Gamma = 1.2')
```
Piecewise-Linear Transformation Functions

These functions can be used to do gray level operations such as these:

- Contrast stretching.
- Window-center correction to enhance a portion of levels.
- Gray-level slicing.
Figure 3: Contrast stretching function.
Here is an example of a piecewise linear transformation.

```matlab
im = imread('Images/airplane.jpg');
r = rgb2gray(im);
line1 = 0:0.5:100;
line2 = 155/55*([201:1:255] - 200) + 100;
t = [line1, line2]
plot(t)
s = t(r + 1);
subplot(1,2,1)
imshow(r)
title('Original')
subplot(1,2,2)
imshow(mat2gray(s))
title('Output')
```
Example

Write a program to carry out contrast stretching.
Outline

1. Point Operations
   - Histogram Processing

2. Spatial Filtering
   - Smoothing Spatial Filters
   - Sharpening Spatial Filters

3. Edge Detection
   - Line Detection Using the Hough Transform
The histogram of a digital image with gray levels in the range $[0, L - 1]$ is a discrete function $h(r_k) = n_k$, where $r_k$ is the $k$th gray level and $n_k$ is the number of pixels having gray level $r_k$.

We can normalize the histogram by dividing by the total number of pixels $n$. Then we have an estimate of the probability of occurrence of level $r_k$, i.e., $p(r_k) = n_k / n$. 
The histogram that we described above has \( L \) bins. We can construct a coarser histogram by selecting a smaller number of bins than \( L \). Then several adjacent values of \( r \) will be counted for a bin.
Example

Write a program to compute the histogram with a given number of bins.
close all
im = imread('fruits.jpg');
img = rgb2gray(im);
imshow(img)
numbins = 10;
binbounds = linspace(0, 255, numbins + 1);
cumhist = zeros(numbins + 1, 1);
for i = 2: numbins + 1
    cumhist(i) = sum(sum(img <= binbounds(i)));
endfor
hist = cumhist(2:end) - cumhist(1:end-1);
bincenters = (binbounds(2:end) + binbounds(1:end-1))/2;
bar(bincenters', hist, 0.2)
Histogram Equalization

We saw that a given image may have a peaked histogram. Such a peaked or non-flat histogram shows that pixels do not equally occupy all the gray levels. This results in a low contrast image. We can remedy this by carrying out the operation called histogram equalization. Histogram equalization is a gray-level transformation that results in an image with a more or less flat histogram.
Consider, for now, that continuous intensity values of an image are to be processed. We assume that $r \in [0, L - 1]$. Let's consider the intensity transformation

$$s = T(r) \quad 0 \leq r \leq L - 1$$

(3)

that produces an output intensity level $s$ for every pixel in the input image having intensity $r$. We assume that

- $T(r)$ is monotonically increasing in the interval $0 \leq r \leq L - 1$, and
- $0 \leq T(r) \leq L - 1$ for $0 \leq r \leq L - 1$.

The intensity levels in the image may be viewed as random variables in the interval $[0, L - 1]$. Let $p_r(r)$ and $p_s(s)$ denote the probability density functions (PDFs) of $r$ and $s$. A fundamental result from basic probability theory is that if $p_r(r)$ and $T(r)$ are known, and $T(r)$ is continuous and differentiable over the range of values of interest, then the PDF of the transformed variable $s$ can be obtained using the simple formula

$$p_s(s) = p_r(r) \left| \frac{dr}{ds} \right|.$$

(4)
Now let’s consider the following transform function:

\[ s = T(r) = (L - 1) \int_0^r p_r(w) \, dw, \tag{5} \]

where \( w \) is the dummy variable of integration. The right-hand side of this equation is the cumulative distribution function (CDF) of random variable \( r \). This function satisfies the two conditions that we mentioned.

- Because PDFs are always positive and the integral of a function is the area under the curve, Equation 5 satisfies the first condition. This is because the area under the curve cannot decrease as \( r \) increases.

- When the upper limit in this equation is \( r = L - 1 \), the integral evaluates to 1, because the area under a PDF curve is always 1. So the maximum value of \( s \) is \( L - 1 \), and the second condition is also satisfied.
To find $p_s(s)$ corresponding to this transformation we use Equation 3.

$$\frac{ds}{dr} = \frac{dT(r)}{dr},$$

$$= (L - 1) \frac{d}{dr} \left[ \int_0^r p_r(w) dw \right],$$

$$= (L - 1) p_r(r). \quad (6)$$

Substituting this result in Equation 3, and keeping in mind that all probability values are positive, yields

$$p_s(s) = p_r(r) \left| \frac{dr}{ds} \right|,$$

$$= p_r(r) \left| \frac{1}{(L - 1) p_r(r)} \right|,$$

$$= \frac{1}{L - 1} \quad 0 \leq s \leq L - 1. \quad (7)$$

We recognize the form of $p_s(s)$ in the last line of this equation as a uniform probability density function.
For discrete values, we deal with probabilities (histogram values) and the summation instead of probability density functions and integrals. The probability of occurrence of intensity level \( r_k \) in a digital image is approximated by

\[
p_r(r_k) = \frac{n_k}{MN} \quad k = 0, 1, \ldots, L - 1, \tag{8}
\]

where \( MN \) is the total number of pixels in the image, \( n_k \) is the number of pixels that have intensity \( r_k \), and \( L \) is the number of possible intensity levels in the image (e.g., 256). Recall that a plot of \( p_r(r_k) \) versus \( r_k \) is commonly referred to as a histogram.

The discrete form of the Equation 4 is

\[
s_k = T(r_k) = (L - 1) \sum_{j=0}^{k} p_r(r_j),
\]

\[
= \frac{(L - 1)}{MN} \sum_{j=0}^{k} n_j \quad k = 0, 1, \ldots, L - 1. \tag{9}
\]
Thus the output image is obtained by mapping each pixel in the input image with intensity $r_k$ into a corresponding pixel level $s_k$ in the output image using Equation 9.
Example

Suppose that a 3-bit image \((L = 8)\) of size \(64 \times 64\) pixels \((MN = 4096)\) has the intensity distribution shown in Table 1, where the intensity levels are in the range \([0, L - 1] = [0, 7]\). carry out histogram equalization.
<table>
<thead>
<tr>
<th>$r_k$</th>
<th>$n_k$</th>
<th>$p_r(r_k) = n_k / MN$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_0 = 0$</td>
<td>790</td>
<td>0.19</td>
</tr>
<tr>
<td>$r_1 = 1$</td>
<td>1023</td>
<td>0.25</td>
</tr>
<tr>
<td>$r_2 = 2$</td>
<td>850</td>
<td>0.21</td>
</tr>
<tr>
<td>$r_3 = 3$</td>
<td>656</td>
<td>0.16</td>
</tr>
<tr>
<td>$r_4 = 4$</td>
<td>329</td>
<td>0.08</td>
</tr>
<tr>
<td>$r_5 = 5$</td>
<td>245</td>
<td>0.06</td>
</tr>
<tr>
<td>$r_6 = 6$</td>
<td>122</td>
<td>0.03</td>
</tr>
<tr>
<td>$r_7 = 7$</td>
<td>81</td>
<td>0.02</td>
</tr>
</tbody>
</table>

**Table 1:** Table for Example 5.
Example

Write a program to carry out histogram equalization.
img = imread('Images/Fig3.15(a)1top.jpg');
L = 256;
cumhist = zeros(L,1);
for k = 0:L-1
    s(k+1) = sum(sum(img <= k));
endfor
n = size(img,1)*size(img,2)
s = s/n;
imeq = mat2gray(s(img + 1));
subplot(2,2,1)
imshow(img)
subplot(2,2,2)
imshow(imeq)
subplot(2,2,3)
imhist(im2double(img))
subplot(2,2,4)
imhist(imeq)
Outline

1. Point Operations
   - Histogram Processing

2. Spatial Filtering
   - Smoothing Spatial Filters
   - Sharpening Spatial Filters

3. Edge Detection
   - Line Detection Using the Hough Transform
Spatial filtering is one of the main tools that we use in image processing.
There are many applications including image enhancement, e.g., smoothing.
We can accomplish effects such as smoothing by applying a spatial filter directly on the image.
Spatial filters are also called spatial masks, kernels, templates, and windows.
Applying a Spatial Filter

In the spatial filter operation we apply a predefined operation on the image pixels included in the neighborhood of the pixel in question. Filtering gives the value of the corresponding pixel at the coordinates of the center of the neighborhood, in the resultant image. If the operations performed on the imaged pixels is linear, then the filter is called a linear filter. Otherwise, the filter is non-linear. Figure 4 shows a sketch of the neighborhood of a particular pixel. Figure 5 shows the weights of a $5 \times 5$ kernel.
Figure 4: A $3 \times 3$ neighborhood for spatial filtering. The center of the neighborhood is the pixel $(i_0, j_0)$. 
<table>
<thead>
<tr>
<th>$w(-2,-2)$</th>
<th>$w(-2,-1)$</th>
<th>$w(-2,0)$</th>
<th>$w(-2,1)$</th>
<th>$w(-2,2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w(-1,-2)$</td>
<td>$w(-1,-1)$</td>
<td>$w(-1,0)$</td>
<td>$w(-1,1)$</td>
<td>$w(-1,2)$</td>
</tr>
<tr>
<td>$w(0,-2)$</td>
<td>$w(0,-1)$</td>
<td>$w(0,0)$</td>
<td>$w(0,1)$</td>
<td>$w(0,2)$</td>
</tr>
<tr>
<td>$w(1,-2)$</td>
<td>$w(1,-1)$</td>
<td>$w(1,0)$</td>
<td>$w(1,1)$</td>
<td>$w(1,2)$</td>
</tr>
<tr>
<td>$w(2,-2)$</td>
<td>$w(2,-1)$</td>
<td>$w(2,0)$</td>
<td>$w(2,1)$</td>
<td>$w(2,2)$</td>
</tr>
</tbody>
</table>

**Figure 5:** A $5 \times 5$ kernel showing weights.
Consider the $3 \times 3$ kernel shown in Figure 6.

<table>
<thead>
<tr>
<th></th>
<th>$w(-1, -1)$</th>
<th>$w(-1, 0)$</th>
<th>$w(-1, 1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w(0, -1)$</td>
<td>$w(0, 0)$</td>
<td>$w(0, 1)$</td>
<td></td>
</tr>
<tr>
<td>$w(1, -1)$</td>
<td>$w(1, 0)$</td>
<td>$w(1, 1)$</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 6**: A $3 \times 3$ kernel showing weights.
At any pixel \((i, j)\) in the image, the response \(g(i, j)\) of the filter is the sum of products of the filter coefficients and the image pixels encompassed by the filter:

\[
g(i, j) = w(-1, -1)f(i - 1, j - 1) + w(-1, 0)f(i - 1, j) + \cdots + w(1, 1)f(i + 1, j + 1).
\] (10)

Observe that the center coefficient of the filter, \(w(0, 0)\) aligns with the pixel at the location \((i, j)\). For a mask of size \(m \times n\), we assume that \(m = 2a + 1\) and \(n = 2b + 1\), where \(a\) and \(b\) are positive integers. This means that we always choose filters of odd dimensions for convenience. If general, linear spatial filtering for an image of size \(M \times N\), with the filter of size \(m \times n\) is given by the expression

\[
g(i, j) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t)f(i + s, j + t),
\] (11)

where \(i\) and \(j\) are varied so that each pixel in \(f\) is visited. Of course, when the kernel is not fully within the image, we have to use methods such as zero padding.
Example

Give a $3 \times 3$ kernel that can be used for averaging a $3 \times 3$ neighborhood.
Instead of using equal $\frac{1}{9}$ entries, we can have a $3 \times 3$ kernel of all ones and then divide the filter output by 9.
Example
Write a program to average filter an image using a $3 \times 3$ kernel.
w = 3;
h = 1/9*ones(w,w);
hw = floor(w/2);
imrgb = imread('Images/airplane.jpg');
im = im2double(rgb2gray(imrgb));
[row, col] = size(im)
result = zeros(row,col);
for i=hw+1:row−hw
    for j=hw+1:col−hw
        result(i,j) =
        sum(sum(h.*im(i−hw:i+hw, j − hw:j + hw)));
    end
end
figure;
imshow(result);
A faster and convenient implementation of the aforementioned loops is as follows:

```c
result = conv2 (im, h, "same");
```
Outline

1. Point Operations
   - Histogram Processing

2. Spatial Filtering
   - Smoothing Spatial Filters
   - Sharpening Spatial Filters

3. Edge Detection
   - Line Detection Using the Hough Transform
Smoothing filters are used for blurring and noise reduction. Blurring is used as a preprocessing operation to remove small noise-like objects before large object extraction and bridging small gaps in lines and curves. Noise reduction can be achieved by blurring with a linear filter and by nonlinear filtering.
Examples of Smoothing Filters

The averaging filter that we studied earlier is a smoothing filter. This reduces sharp transitions and noise. Figure 7 shows this kernel. Note that using 1 as filter coefficients instead of $1/9$ and later multiplying the sum by $1/9$ is computationally efficient.

\[
\begin{array}{ccc}
1 & 1 & 1 \\
\frac{1}{9} & \times & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

**Figure 7:** $3 \times 3$ averaging smoothing filter.
The weighted averaging kernel shown in Figure 8 gives more importance to pixels close to the center.

Figure 8: $3 \times 3$ averaging smoothing filter.
Example

Write a program to carry out weighted averaging using the kernel in Figure 8.
Order-Statistics (Non-Linear) Filters

- Order statistics filters determine the value of the resultant pixel by a value related to the ranking or order of the pixels within a kernel.

- The best known in this category is the **median filter**, which computes the value of the resultant pixel as the median of the neighborhood.

- Median filters are very effective in the presence of impulse noise called salt-and-pepper noise.

- In order to do median filtering, we sort the pixels in the neighborhood, find the median, and set the pixel of the resultant image with the median value.
Outline

1. Point Operations
   - Histogram Processing

2. Spatial Filtering
   - Smoothing Spatial Filters
   - Sharpening Spatial Filters

3. Edge Detection
   - Line Detection Using the Hough Transform
We were able to achieve image blurring by using pixel averaging, an operation similar to integration.

Sharpening, the opposite of blurring, can be achieved by spatial differentiation.

The strength of the response of the derivative operator is proportional to the degree of intensity discontinuity in the image at the point of interest.

Thus, image differentiation enhances the edges and other discontinuities, while deemphasizing areas of low intensity variations.
Using the Second Derivative for Sharpening—The Laplacian

- We can use a 2-D, second order derivative for image sharpening.
- We will use the discrete formulation of the second-order derivative and construct a filter mask.
- We are interested in isotropic filters, whose response is independent of the direction of discontinuities.
- Isotropic filters are invariant to the image rotation, or simply, rotation invariant.
A simple isotropic second-order derivative operator is the Laplacian. For a function (image) $f(x, y)$ of two variables it is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \quad (12)$$

The second-order derivative in $x$-direction is

$$\frac{\partial^2 f}{\partial x^2} = f(x + 1, y) + f(x - 1, y) - 2f(x, y). \quad (13)$$

In $y$-direction, we have
Laplacian

A simple isotropic second-order derivative operator is the Laplacian. For a function (image) $f(x, y)$ of two variables it is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \quad (12)$$

The second-order derivative in $x$-direction is

$$\frac{\partial^2 f}{\partial x^2} = f(x + 1, y) + f(x - 1, y) - 2f(x, y). \quad (13)$$

In $y$-direction, we have

$$\frac{\partial^2 f}{\partial y^2} = f(x, y + 1) + f(x, y - 1) - 2f(x, y). \quad (14)$$
Therefore, the discrete version of the Laplacian is

$$\nabla^2 f = f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1) - 4f(x, y).$$  \hspace{1cm} (15)$$

Figure 9 shows some implementations of the Laplacian.
Figure 9: Laplacian kernels.
Because the Laplacian is a derivative operator, its use highlights the intensity discontinuities in an image and deemphasizes regions with slowly varying intensity levels. We can produce a sharpened image by combining the Laplacian with the original image. If we use the first kernel in Figure 9, we can add the image and the Laplacian. If \( g(x, y) \) is the sharpened image

\[
g(x, y) = f(x, y) + \nabla^2 f(x, y).
\]  

(16)

Usually, we need to re-scale the graylevels to the \([0, L - 1]\) after this operation.
A process used in printing industry is unsharp masking. It sharpens the image by subtracting an unsharp (smoothed) version of the image from the original. Unsharp masking consists of the following steps:

1. Blur the original image.
2. Subtract the blurred image from the original. The resulting difference is called the mask.
3. Add the mask to the original.
First-order derivative based operations are implemented using the magnitude of the gradient. For a function $f(x, y)$, the gradient of $f$ at the coordinates $(x, y)$ is defined as the two-dimensional column vector

$$
\nabla f = \text{grad}(f) = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}.
$$

(17)

This vector has the important property that it points in the direction of the greatest rate of change of $f$ at location $(x, y)$. 
The magnitude (length) of vector $\nabla f$, denoted as $M(x, y)$, where,

$$M(x, y) = \text{mag}(\nabla f) = \sqrt{f_x^2 + f_y^2},$$

(18)

is the value at $(x, y)$ of the rate of change in the direction of the gradient vector. Note that $M(x, y)$ is an image of the same size as the original, created when $x$ and $y$ are allowed to vary over all the pixel locations in $f$. This is commonly referred to as the gradient image of simply the gradient.
Sobel operators are discrete approximations to the gradient. Figure 10 shows the Sobel operators.

(a) For $f_y$

(b) For $f_x$

Figure 10: Sobel operators.
Figure 11: Sobel filtering.
Outline

1. Point Operations
   - Histogram Processing

2. Spatial Filtering
   - Smoothing Spatial Filters
   - Sharpening Spatial Filters

3. Edge Detection
   - Line Detection Using the Hough Transform
Segmentation subdivides an image into its constituent regions or objects. The level to which the subdivision is carried depends on the problem being solved. Segmentation of nontrivial images is one of the most difficult tasks in image processing. Segmentation accuracy determines the eventual success or failure of computerized analysis procedures. For this reason, considerable care should be taken to improve the probability of rugged segmentation. In some situations, such as industrial inspection applications, at least some measure of control over the environment is possible at times. In others, as in remote sensing, user control over image acquisition is limited principally to the choice of image sensors.

Segmentation algorithms for monochrome images generally are based on one of two basic properties of image intensity values: discontinuity and similarity. In the first category, the approach is to partition an image based on abrupt changes in intensity, such as edges in an image. The principal approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria.
There are three basic types of intensity discontinuities in a digital image:

1. points,
2. lines, and
3. edges.

The most common way to look for discontinuities is to run a mask through the image.
For a $3 \times 3$ mask this procedure involves computing the sum of products of the coefficients with the intensity levels contained in the region encompassed by the mask. That is the response, $R$, of the mask at any point in the image is given by

$$R = \sum_{i=1}^{9} w_i z_i$$

where $z_i$ is the intensity of the pixel associated with the mask coefficient $w_i$.

Figure 12: A mask for point detection.
The detection of isolated points embedded in areas of constant or nearly constant intensity in an image is straightforward in principle. Using the mask shown in Figure 12 we say that an isolated point has been detected at the location on which the mask is centered if

$$R \geq T,$$

where $T$ is a nonnegative threshold. Point detection is implemented in MATLAB using functions `imfilter`, with a mask. The important requirements are that the strongest response of a mask must be when the mask is centered on an isolated point, and that the response be 0 in areas of constant intensity.
If $T$ is given, the following command implements the point-detection approach just discussed:

```matlab
    g = abs(imfilter(double(f), w)) >= T;
```

where $f$ is the input image, $w$ is an appropriate point-detection mask and $g$ is the resulting image.
The next level of complexity is line detection. Consider the mask in Figure 13.
Figure 13: Line detector masks.
If the first mask were moved around an image, it would respond more strongly to lines (one pixel thick) oriented horizontally. With a constant background, the maximum response would result when the line passed through the middle row of the mask. Similarly, the second mask in Figure 13 responds best to lines oriented at $+45^\circ$; the third mask to vertical lines; and the fourth mask to lines in the $-45^\circ$ direction. Note that the preferred direction of each mask is weighted with a larger coefficient (i.e., 2) than other possible directions. The coefficients of each mask sum to zero, indicating a zero response from the mask in areas of constant intensity.
Although point and line detection certainly are important in any discussion on image segmentation, edge detection is by far the most common approach for detecting meaningful in intensity values. Such discontinuities are detected by using first-and second-order derivatives. The first-order derivative of choice in image processing is the gradient. The gradient of a 2-D function, \( f(x, y) \), is defined as the vector

\[
\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}
\]
The magnitude of this vector is

$$|\nabla f| = [G_x^2 + G_y^2]^{1/2}.$$ 

To simplify computation, this quantity is approximated sometimes by omitting the square-root operation,

$$|\nabla f| \approx G_x^2 + G_y^2,$$

or by using absolute values,

$$|\nabla f| \approx |G_x| + |G_y|.$$
These approximations still behave as derivatives; that is, they are zero in areas of constant intensity and their values are proportional to the degree of intensity change in areas whose pixel values are variable. It is common practice to refer to a magnitude of the gradient or its approximations simply as “the gradient.” A fundamental property of the gradient vector is that it points in the direction of the maximum rate of change of $f$ at coordinates $(x, y)$. The angle which this maximum rate of change occurs is

$$\alpha(x, y) = \tan^{-1}\left(\frac{G_y}{G_x}\right).$$

One of the key issues is how to estimate the derivative $G_x$ and $G_y$ digitally. The various approaches used by function edge are discussed later in this section.
Second order derivatives in image processing are generally computed using the Laplacian. That is the Laplacian of a 2-D function \( f(x, y) \) is formed from second-order derivatives, as follows:

\[
\nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}.
\]

The Laplacian is seldom used by itself for edge detection because, as a second-order derivative, it is unacceptably sensitive to noise, its magnitude produces double edges, and it is unable to detect edge direction. However, the Laplacian can be powerful complement when used in combination with other edge-detection techniques. For example, although its double edges make it unsuitably for edge detection directly, this property can be used for edge location.
With the preceding discussion as background, the basic idea behind edge detection is to find places in an image where the intensity changes rapidly, using one of two general criteria:

1. find places where the first derivative of the intensity is greater in magnitude than a specified threshold, or

2. find places where the second derivative of the intensity has a zero crossing.
IPT’s function `edge` provides several derivative estimators based on the criteria just discussed. For some of these estimators, it is possible to specify whether the edge detector is sensitive to horizontal or vertical edges or to both. The general syntax for this function is

```
[g, t] = edge(f, 'method', parameters)
```

where `f` is the input image, `method` is one of the approaches listed in its help and `parameters` are additional parameters explained in the following discussion. In the output, `g` is a logical array with 1s at the locations where edge points where detected in `f` and 0s elsewhere. Parameter `t` is a optional; it gives the threshold used by `edge` to determine which gradient values are strong enough to be called edge points.
Sobel Edge Detector

The Sobel edge detector uses the masks in Figure 14 to approximate digitally the first derivatives $G_x$ and $G_y$. In other words, the gradient at the center point in a neighborhood is computed as follows by the Sobel detector:

$$g = |\nabla f| = [G_x^2 + G_y^2]^{1/2}.$$
(a) General mask

\[
\begin{array}{ccc}
  z_1 & z_2 & z_3 \\
  z_4 & z_5 & z_6 \\
  z_7 & z_8 & z_9 \\
\end{array}
\]

(b) Sobel, Vertical

\[
\begin{array}{ccc}
  -1 & -2 & -1 \\
  0 & 0 & 0 \\
  1 & 2 & -1 \\
\end{array}
\]

(c) Sobel, Horizontal

\[
\begin{array}{ccc}
  -1 & 0 & 1 \\
  -2 & 0 & 2 \\
  -1 & 0 & 1 \\
\end{array}
\]

(d) Prewitt Vertical

\[
\begin{array}{ccc}
  -1 & -1 & -1 \\
  0 & 0 & 0 \\
  1 & 1 & 1 \\
\end{array}
\]

(e) Prewitt Horizontal

\[
\begin{array}{ccc}
  -1 & 0 & 1 \\
  -1 & 0 & 1 \\
  -1 & 0 & 1 \\
\end{array}
\]
Sobel edge detection can be implemented by filtering an image, \( f \), (using \texttt{imfilter}) with the left mask in Figure 14 filtering \( f \) again with the other mask, squaring the pixels values of each filtered image, adding the two results, and computing their square root. Similar comments apply to the other and third entries in Figure 14. Function \texttt{edge} simply packages the preceding operations into the function call and adds other features, such as accepting a threshold value or determining a threshold automatically. In addition, \texttt{edge} contains edge detection techniques that are not implementable directly with \texttt{imfilter}.
The general calling syntax for the Sobel detector is

\[ [g, t] = \text{edge}(f, 'sobel', T, \text{dir}) \]

where \( f \) is the input image, \( T \) is specified threshold, and \( \text{dir} \) specifies the preferred direction of the edges detected: ‘horizontal’, ‘vertical’ or ‘both’ (the default). As noted earlier, \( g \) is a logical image containing 1s at locations where edges were detected and 0s elsewhere. Parameter \( t \) in the output is optional. It is the threshold value used by edge. If \( T \) is specified, then \( t = T \). Otherwise, if \( T \) is not specified (or is empty, [ ]), \text{edge} sets \( t \) equal to a threshold it determines automatically and then uses for edge detection. One of the principal reason for including \( t \) in the output argument is to get an initial value for the threshold. Function \text{edge} uses the Sobel detector as a default if the syntax \( g = \text{edge}(f) \), is used.
The Prewitt edge detector uses the masks in Figure 14. to approximate digitally the first derivatives. Its general calling syntax is

\[
[g, t] = \text{edge}(f, \text{'prewitt'}, T, \text{dir})
\]

The parameters of this function are identical to the Sobel parameters. The Perwitt detector is slightly simpler to implement computationally than the Sobel detector, but it tends to produce somewhat noisier results. (It can be shown that the coefficient with value 2 in the Sobel detector provides smoothing.)
Laplacian of a Gaussian (LoG) Detector

Consider the Gaussian function

\[ h(r) = -e^{-\frac{r^2}{2\sigma^2}} \]

where \( r^2 = x^2 + y^2 \) and \( \sigma \) is the standard deviation. This is a smoothing function which, if convolved with an image, will blur it. The degree of blurring is determined by the value of \( \sigma \). The Laplacian of this function (the second derivative with respect to \( r \)) is

\[ \nabla^2 h(r) = -\left[ \frac{r^2 - \sigma^2}{\sigma^4} \right] e^{-\frac{r^2}{2\sigma^2}} \]
This function is called the Laplacian of Gaussian (LoG). Because the second derivative is a linear operation, convolving (filtering) an image with $\nabla^2 h(r)$ is the same as convolving the image with the smoothing function first and then computing the Laplacian of the result. This is the key concept underlying the LoG detector. We convolve the image with $\nabla^2 h(r)$ knowing that it has two effects: It smoothes the image (thus reducing noise), and it computes the Laplacian, which yields a double-edge image. Locating edges then consists of finding the zero crossings between the double edges.
The general calling syntax for the LoG detector is

\[
[g, t] = \text{edge}(f, 'log', T, \sigma)
\]

where \(\sigma\) is the standard deviation and the other parameters are as explained previously. The default value of \(\sigma\) is 2. As before, \text{edge} ignores any edges that are not stronger than \(T\). If \(T\) is not provided, or it is empty, [ ], \text{edge} chooses the value automatically. Setting \(T\) to 0 produces edges that are closed contours, a familiar characteristic of the LoG method.
Zero-Crossings Detector

This detector is based on the same concept as the LoG method, but the convolution is carried out using a specified filter function, H. The calling syntax is

\[
[g, t] = \text{edge}(f, 'zerocross', T, H)
\]

The other parameters are as explained for the LoG detector.
Canny Edge Detector I

The Canny detector [1] is the most powerful edge detector provided by function edge. The method can be summarized as follows:

1. The image is smoothed using a Gaussian filter with a specified standard deviation, $\sigma$ to reduce noise.

2. The local gradient, $g(x, y) = \sqrt{G_x^2 + G_y^2}$, and edge direction, $\alpha(x, y) = \tan^{-1}(G_y/G_x)$, are computed at each point, using an operator such as Sobel or Prewitt. An edge point is defined to be a point whose strength is locally maximum in the direction of the gradient.

3. The edge point is determined in (2) give rise to ridges in the gradient magnitude image. The algorithm then tracks along the top of these ridges and sets to zero all pixels that are not actually on the ridge top so as to give a thin line in the output, a process known as nonmaximal suppression. The ridge pixels are then thresholded using two thresholds, $T_1$ and $T_2$, with $T_1 < T_2$. Ridge
pixels with values greater than $T_2$ are said to be "strong" edge pixels. Ridge pixels with values between $T_1$ and $T_2$ are said to be "weak" edge pixels.

Finally the algorithm performs edge linking by incorporating the weak pixels that are 8-connected to the strong pixels.
The syntax for the Canny edge detector is

\[ [g, t] = \text{edge}(f, 'canny', T, \text{sigma}) \]

where \( T \) is a vector, \( T = [T_1, T_2] \), containing the two thresholds explained in step 3 of the preceding procedure, and \( \text{sigma} \) is the standard deviation of the smoothing filter. If \( t \) is included in the output argument, it is a two-element vector containing the two threshold values used by the algorithm. The rest of the syntax is as explained for the other methods, including the automatic computation of thresholds if \( T \) is not supplied. The default value for \( \text{sigma} \) is 1.
We can extract and display the vertical edges in the image, \( f \),

\[
[gv, t] = \text{edge}(f, 'sobel', 'vertical');
\]

\[
\text{imshow}(gv)
\]

\[
t = 0.0516
\]
Outline

1. Point Operations
   - Histogram Processing

2. Spatial Filtering
   - Smoothing Spatial Filters
   - Sharpening Spatial Filters

3. Edge Detection
   - Line Detection Using the Hough Transform
Ideally, the methods discussed in the previous section should yield pixels lying only on edges. In practice, the resulting pixels seldom characterize an edge completely because of noise, breaks in the edge from nonuniform illumination, and other effects that introduce spurious intensity discontinuities. Thus edge-detection algorithms typically are followed by linking procedures to assemble edge pixels into meaningful edges. One approach that can be used to find and link line segments in an image is the Hough transform.
Given a set of points in an image (typically a binary image), suppose that we want to find subsets of these points that lie on straight lines. One possible solution is to first find all lines determined by every pair of points and then find all subsets of points that are close to particular lines. The problem with this procedure is that it involves finding $n(n - 1)/2 \sim n^2$ lines and then performing $n(n(n - 1))/2 \sim n^3$ comparisons of every point to all lines. This approach is computationally prohibitive in all but the most trivial applications.
With the Hough transform, on the other hand, we consider a point \((x_i, y_i)\) and all the lines that pass through it. Infinitely many lines pass through \((x_i, y_i)\) all of which satisfy the slope-intercept equation 

\[ y_i = ax_i + b \]

for some values of \(a\) and \(b\). Writing this equation as 

\[ b = -ax_i + y_i \]

and considering the \(ab\)-plane (also called parameter space) yields the equation of a single line for a fixed pair \((x_i, y_i)\). Furthermore, a second point \((x_j, y_j)\) also has a line in parameter space associated with it, and this line intersects the line associated with \((x_i, y_i)\) at \((a', b')\). where \(a'\) is the slope and \(b'\) the intercept of the line containing both \((x_i, y_i)\) and \((x_j, y_j)\) in the \(xy\)-plane. In fact, all points contained on this line have lines in parameter space that intersect at \((a', b')\).
In principle, the parameter-space lines corresponding to all image points \((x_i, y_i)\) could be plotted, and then the image lines could be identified where large numbers of parameter lines intersect. A practical difficulty with this approach, however, is that \(a\) (the slope of the line) approaches infinity as the line approaches the vertical direction. One way around this difficulty is to use the normal representation of a line:

\[
x \cos \theta + y \sin \theta = \rho.
\]
The computational attractiveness of the Hough transform arises from subdividing the $\rho \theta$ parameter space into so-called accumulator cells. Usually the maximum range of the values is $-90^\circ \leq \theta \leq 90^\circ$ and $-D \leq \rho \leq D$ where $D$ is the distance between corners in the image. The cell at the coordinates $(i, j)$ with accumulator value $A(i, j)$, corresponds to the square associated with parameter space coordinates $(\rho_i, \theta_j)$. Initially, these cells are set to zero. Then, for every non-background point $(x_k, y_k)$ in the image plane, we let $\theta$ equal each of the allowed subdivision values on the $\theta$ axis and solve for the corresponding $\rho$ using the equation $\rho_k = x_k \cos \theta + y_k \sin \theta$. The resulting $\rho$-values are then rounded off to the nearest allowed cell value along the $\rho$-axis. The corresponding accumulator cell is then incremented. At the end of this procedure, a value of $Q$ in $A(i, j)$, means that $Q$ points in the $xy$-plane lie on the line $x \cos \theta_j + y \sin \theta_j = \rho_i$. The number of subdivisions in the $\rho \theta$-plane determines the accuracy of the colinearity of these points.
Using `houghlines` to Find and Link Line Segments

```matlab
lines = houghlines(f, theta, rho, r, c)
figure, imshow(f), hold on
for k = 1:length(lines)
    xy = [lines(k).point1 ; lines(k).point2];
polt(xy(:,2), xy(:,1), 'LineWidth', 4, 'Color', [.6
end
```
J Canny.
A computational approach to edge detection.