第3章

视频

边 缘 检 测

图像中的特征，如边缘，是通过强度或颜色的局部变化来检测的[1]。它们在图像解释中起到重要作用。一幅图像主观的“清晰”与其中结构的不连续性和锐度相关。人类眼睛会对目标边缘给予较大的权重，因为正是通过边缘人才能分辨不同的特性以及区分实际的形状。所以，简单的痕迹就足够解释目标的类别。因此边缘是图像处理和机器视觉的重要主题。

3.1 边缘和轮廓

边缘在人类视觉中起到主导的作用，或许在其他生物视觉系统中也是如此。边缘不仅能被注意到，而且有可能仅借助很少的边缘线来重建目标（见图 3.1）。这里要讨论的主要议题之一是边缘如何起源于图像和如何定位它们以用于其后的处理步骤。

(a) (b)

图 3.1

（a）原始图像；（b）具有边缘和轮廓的图像

粗略地说，边缘可被看作图像中强度在各个方向显著变化的点[2]。特定像素处呈现的强度变化将是图像中该点的边缘值。变化的幅度常借助导数计算，这也是确定图像边缘的最重要方法。根据特定像素中呈现的强度变化，将可以确定图像中该点的边缘值。

3.2 用基于梯度的技术检测边缘

考虑一幅具有在中心的白色区域被黑色背景包围的图像，如图 3.2(a)所示，先看 1-D 情况。

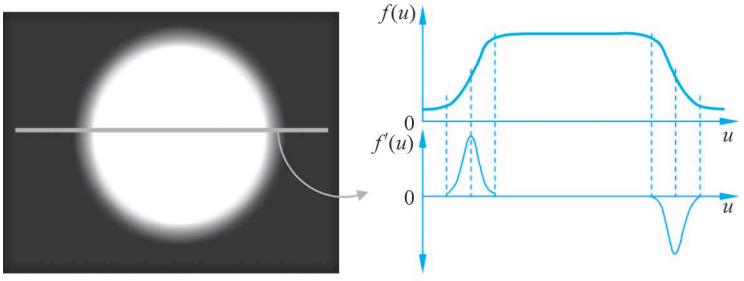

图 3.2 从图像水平剖面获得的 1-D 一阶导数
（a）原始图像；（b）水平剖面的导数

沿图像一条线的灰度剖面如图 3.2(b)所示。可以定义这个 1-D 信号为 $f(u)$。它的一阶导数定义如下：

$$f'(u)=\frac{\mathrm{d}f}{\mathrm{d}u}(u) \tag{3.1}$$

这样，在强度增加的地方有正的上升而在强度减少的地方有负的上升。不过，导数对离散函数 $f(u)$还没定义，需要有一种计算它的方法。

已知函数在点 x 的导数可解释成该点的切线斜率。不过对离散函数，在点 u（切线斜率计算点）的导数可根据 u 的邻域点之间的差除以两点之间的距离来计算[3]，如图 3.3 所示。所以，导数可以近似为

$$\frac{\mathrm{d}f}{\mathrm{d}u}(u)\approx\frac{f(u+1)-f(u-1)}{2}=0.5\times(f(u+1)-f(u-1)) \tag{3.2}$$

相同的过程也可用于沿图像列的垂直方向。

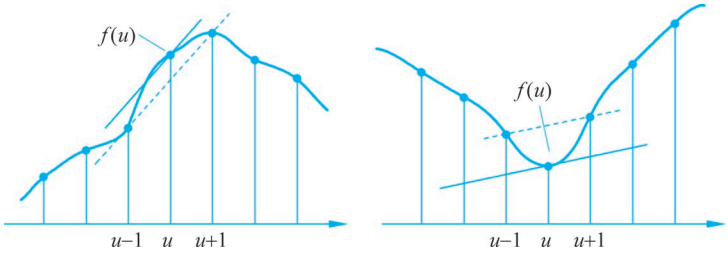

图 3.3 计算离散函数的一阶导数。用通过相邻点 $f(u-1)$和 $f(u+1)$的直线的斜率来间接计算 $f(u)$点的切线的斜率

3.2.1 偏导数和梯度

偏导数可以看作沿多维函数的一个坐标轴（相对于函数的变量之一）的导数，例如，

$$\frac{\partial I(x,y)}{\partial x}\quad 和\quad \frac{\partial I(x,y)}{\partial y} \tag{3.3}$$

图像函数 $I(u,v)$ 相对于 u 和 v 的偏导数可以用下列矢量定义：

$$\nabla I(x,y)=\begin{bmatrix}\dfrac{\partial I(x,y)}{\partial x}\\[2ex]\dfrac{\partial I(x,y)}{\partial y}\end{bmatrix} \tag{3.4}$$

这代表了函数 I 在点 (x,y) 的梯度矢量。梯度的值可计算如下：

$$|\nabla I|=\sqrt{\left(\frac{\partial I}{\partial x}\right)^2+\left(\frac{\partial I}{\partial y}\right)^2} \tag{3.5}$$

$|\nabla I|$ 的值在图像旋转时不变，所以它独立于图像中结构的朝向。这个性质对定位图像中的边缘点很重要，所以 $|\nabla I|$ 的值是大多数边缘检测算法中的一个实用值。

3.2.2　导出的滤波器

式(3.4)中的梯度分量是图像行和列的一阶导数。根据图 3.2 和图 3.3，水平方向的导数可按滤波器操作用下列系数矩阵计算：

$$\boldsymbol{H}_x^{\mathrm{D}}=[-0.5\quad \underline{0}\quad 0.5]=0.5\times[-1\quad \underline{0}\quad 1] \tag{3.6}$$

其中，系数 -0.5 影响像素 $I(x-1,y)$，0.5 影响像素 $I(x+1,y)$。中间像素 $I(x,y)$ 的值乘以 0 或忽略掉(这里，带有下画线的元素被看作参考点或其值已计算)。以相同的方式，滤波器在垂直方向的相同效果也可以计算出来，即系数矩阵为

$$\boldsymbol{H}_y^{\mathrm{D}}=\begin{bmatrix}-0.5\\ \underline{0}\\ 0.5\end{bmatrix}=0.5\times\begin{bmatrix}-1\\ \underline{0}\\ 1\end{bmatrix} \tag{3.7}$$

图 3.4 显示了使用定义在式(3.6)和式(3.7)中的滤波器所获得的结果。在图 3.4 中，对方向的依赖性很容易识别。水平梯度滤波器 $\boldsymbol{H}_x^{\mathrm{D}}$ 导致了水平方向大量的响应并突出了垂直方向的边缘(见图 3.4(b))。以相同的方式，滤波器 $\boldsymbol{H}_y^{\mathrm{D}}$ 导致了垂直方向大量的响应并突出了水平方向的边缘(见图 3.4(c))。在滤波器响应为 0 的图像区域(在图 3.4(b)和图 3.4(c)中)，其值用灰色像素来表示。

图 3.4　一阶偏导数

(a) 合成图像；(b) 水平方向的一阶偏导数 $\partial I/\partial u$；(c) 垂直方向的一阶偏导数 $\partial I/\partial v$；(d) 梯度 $|\nabla I|$ 的值。在图(b)和图(c)中，黑色属于负值，白色属于正值，灰色对应 0

3.3　边缘检测滤波器

计算对应每幅图像中各种像素局部梯度方式的根本区别在于各个边缘检测算子。主要的差别是梯度被计算为不同的方向分量。但同样的是，所有偏导数结果(按各个方向获得)

都被结合进一个最终结果。在梯度的计算中，梯度值和边缘的方向都值得关注。不过，由于两个元素（梯度值和方向）都隐含在梯度计算中，所以可以更方便地得到它们[4]。下面将介绍一些非常著名的边缘算子，包括实际使用的和历史上有名的。

3.3.1 蒲瑞维特算子和索贝尔算子

蒲瑞维特算子和索贝尔算子代表两个使用最多的边缘检测方法[5]，它们很相似，只有一些细节上的差别。

1. 滤波器

两个算子都使用 3×3 的系数矩阵作为滤波器。与式(3.6)和式(3.7)中给出的滤波器相比，这两个算子的尺寸和配置都使滤波器不易受到自身噪声的影响。蒲瑞维特算子使用如下定义的滤波器：

$$\boldsymbol{H}_x^{\mathrm{P}}=\begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{bmatrix},\quad \boldsymbol{H}_y^{\mathrm{P}}=\begin{bmatrix}-1 & -1 & -1\\ 0 & 0 & 0\\ 1 & 1 & 1\end{bmatrix} \tag{3.8}$$

这些滤波器很明显作用于所考虑像素的不同相邻像素。如果对它们的分解形式进行分析，有

$$\boldsymbol{H}_x^{\mathrm{P}}=\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}*\begin{bmatrix}-1 & 0 & 1\end{bmatrix},\quad \boldsymbol{H}_y^{\mathrm{P}}=\begin{bmatrix}-1\\ 0\\ 1\end{bmatrix}*\begin{bmatrix}1 & 1 & 1\end{bmatrix} \tag{3.9}$$

不管 $\boldsymbol{H}_x^{\mathrm{P}}$ 还是 $\boldsymbol{H}_y^{\mathrm{P}}$，带有 ±1 分量的矢量都给出构建式(3.8)所定义滤波器的 3 列或 3 行系数。不过，可以看出，矢量[−1　0　1]保持了 3.2.2 节定义的导数近似；而分量全为 1 的矢量[1　1　1]在两种情况下都隐含了一个数据平滑操作。这样一来，滤波器除了定位或增强属于边缘的像素，还执行了一个平滑操作。这使得滤波器对图像中的噪声更鲁棒。

索贝尔算子与蒲瑞维特算子基本相同，二者仅有的差别是在这个滤波器中，对中心行或列给予较大的权重。索贝尔算子的系数矩阵定义如下：

$$\boldsymbol{H}_x^{\mathrm{S}}=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix},\quad \boldsymbol{H}_y^{\mathrm{S}}=\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix} \tag{3.10}$$

蒲瑞维特算子和索贝尔算子都给出图像像素沿两个不同方向上的局部梯度估计结果，并保持了下列关系：

$$\nabla I(x,y)\approx\frac{1}{6}\begin{bmatrix}\boldsymbol{H}_x^{\mathrm{P}}\cdot\boldsymbol{I}\\ \boldsymbol{H}_y^{P}\cdot\boldsymbol{I}\end{bmatrix},\quad \nabla I(x,y)\approx\frac{1}{8}\begin{bmatrix}\boldsymbol{H}_x^{\mathrm{S}}\cdot\boldsymbol{I}\\ \boldsymbol{H}_y^{\mathrm{S}}\cdot\boldsymbol{I}\end{bmatrix} \tag{3.11}$$

2. 梯度大小和方向

不管是蒲瑞维特算子还是索贝尔算子，对每个不同方向的滤波结果都可如下刻画：

$$D_x(x,y)=\boldsymbol{H}_x*\boldsymbol{I},\quad D_y(x,y)=\boldsymbol{H}_y*\boldsymbol{I} \tag{3.12}$$

边缘 $E(x,y)$ 的幅度在两种情况下都定义为梯度的幅度：

$$E(x,y)=\sqrt{(D_x(x,y))^2+(D_y(x,y))^2} \tag{3.13}$$

在各个像素的梯度方向（角度）可计算如下（见图 3.5）：

$$\phi(x,y)=\arctan\left[\frac{D_y(x,y)}{D_x(x,y)}\right] \tag{3.14}$$

使用蒲瑞维特算子和索贝尔算子的原始版本计算梯度方向相对不太准确。所以，建议不使用式(3.10)定义的索贝尔算子而使用如下定义的能最小化角度误差的版本：

$$\boldsymbol{H}_x^{S'}=\frac{1}{32}\begin{bmatrix}-3 & 0 & 3\\ -10 & 0 & 10\\ -3 & 0 & 3\end{bmatrix},\quad \boldsymbol{H}_y^{S'}=\frac{1}{32}\begin{bmatrix}-3 & -10 & -3\\ 0 & 0 & 0\\ 3 & 10 & 3\end{bmatrix} \tag{3.15}$$

由于实现简单且效果好，索贝尔算子被大量数字图像处理的商业软件包采用(见图 3.6)。

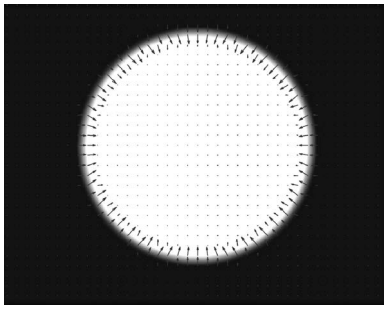

图 3.5　梯度幅度 $E(x,y)$和它的方向 $\phi(x,y)$的表达(完整的边缘检测过程总结在图 3.6 中；首先，将原始图像通过两个系数矩阵 $\boldsymbol{H}_x$ 和 $\boldsymbol{H}_y$ 进行滤波，接下来，将结果归于梯度幅度 $E(x,y)$和它的方向 $\phi(x,y)$)

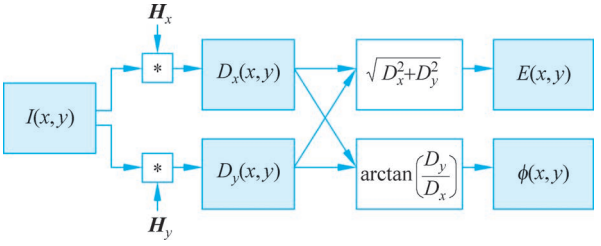

图 3.6　用来检测边缘的滤波器的典型操作：借助系数矩阵 $\boldsymbol{H}_x$ 和 $\boldsymbol{H}_y$，可得到梯度图像 $\boldsymbol{D}_x$ 和 $\boldsymbol{D}_y$，并计算出梯度幅度 $E(x,y)$和它的方向 $\phi(x,y)$

3.3.2　罗伯特算子

罗伯特算子是在图像中用于定位边缘的最老的滤波器之一[6]。该滤波器的一个特点是它非常小，只使用 2×2 的系数矩阵来确定它在两个不同对角线方向的梯度。这个算子定义如下：

$$\boldsymbol{H}_1^{R}=\begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix},\quad \boldsymbol{H}_2^{R}=\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix} \tag{3.16}$$

该滤波器对图像中具有对角线方向的边缘特别起作用(见图 3.7)。不过，这使得该滤波器在方向上不是很有选择性，特别在具有不同朝向的区域里计算时。梯度的幅度按照式(3.5)的定义，可考虑计算两个分量 $\boldsymbol{H}_1^{R}$ 和 $\boldsymbol{H}_2^{R}$。不过，由于滤波器对数据对角线的操作，也可考虑梯度的幅度由两个 45°的矢量($\boldsymbol{H}_1^{R}$ 和 $\boldsymbol{H}_2^{R}$)构成。图 3.8 展示了这个操作。

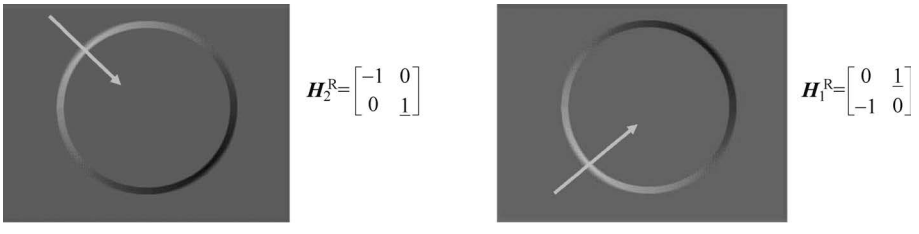

图 3.7　罗伯特算子的对角线分量

(a) 分量 H_2^R；(b) 分量 H_1^R(从图中可以容易地识别出这个算子的对角线特性以得到图像中每个像素的梯度幅度)

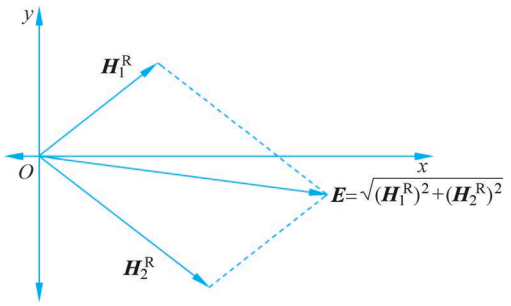

图 3.8　罗伯特算子的梯度幅度（梯度幅度 $E(x,y)$ 是两个正交滤波器 H_1^R 和 H_2^R 的和，可适用于各个对角线方向）

3.3.3　罗盘算子

在设计边缘检测滤波器中的一个问题是：对结构边缘的检测越敏感，对其方向的依赖性就越强。因此，为设计一个好的算子，需要在响应幅度和对梯度方向的敏感度之间进行平衡。

解决这个问题的一个方法是不仅仅使用将滤波器操作分解在两个方向上，如蒲瑞维特算子和索贝尔算子是在水平和垂直两个方向上，或如罗伯特算子是在对角线方向上，还可以在更多的方向上使用滤波器。一个典型的例子是基尔希算子，它包括 8 个不同的滤波器，互相之间相差 45°，如此就覆盖了所有方向以有效地进行边缘检测。这个算子的前 4 个系数矩阵如下定义：

$$\boldsymbol{H}_0^K=\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix},\quad \boldsymbol{H}_1^K=\begin{bmatrix}-2&-1&0\\-1&0&1\\0&1&2\end{bmatrix}$$
$$\boldsymbol{H}_2^K=\begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix},\quad \boldsymbol{H}_3^K=\begin{bmatrix}0&-1&-2\\1&0&-1\\2&1&0\end{bmatrix} \tag{3.17}$$

这个算子的后 4 个系数矩阵与前 4 个系数矩阵只差一个负号，例如，$\boldsymbol{H}_4^K=-\boldsymbol{H}_0^K$。所以对 8 个滤波器 $\boldsymbol{H}_0^K,\boldsymbol{H}_1^K,\cdots,\boldsymbol{H}_7^K$，根据卷积的线性性质，有

$$\boldsymbol{I}*\boldsymbol{H}_4^K=\boldsymbol{I}*(-\boldsymbol{H}_0^K)=-(\boldsymbol{I}*\boldsymbol{H}_0^K) \tag{3.18}$$

由基尔希滤波器 $D_0,D_1,\cdots,D_7$ 的操作所产生的图像是用下列方式生成的：

$$\begin{aligned}&D_0=\boldsymbol{I}*\boldsymbol{H}_0^K,\quad D_1=\boldsymbol{I}*\boldsymbol{H}_1^K,\quad D_2=\boldsymbol{I}*\boldsymbol{H}_2^K,\quad D_3=\boldsymbol{I}*\boldsymbol{H}_3^K\\&D_4=-D_0,\quad D_5=-D_1,\quad D_6=-D_2,\quad D_7=-D_3\end{aligned} \tag{3.19}$$

梯度的幅度应源自基尔希滤波器生成的所有图像的组合，是由 8 个滤波器得到的图像在像素 (x,y) 处的最大值。所以，在像素 (x,y) 处的梯度幅度值定义如下：

$$\begin{aligned}E^K(x,y)&=|\max[D_0(x,y),D_1(x,y),\cdots,D_7(x,y)]|\\&=\max[|D_0(x,y)|,|D_1(x,y)|,\cdots,|D_7(x,y)|]\end{aligned} \tag{3.20}$$

梯度的方向由对计算梯度幅度给出最大贡献的滤波器确定。所以，梯度方向由下式指定：

$$\phi^{K}(x,y)=\frac{\pi}{4}l,\quad l=\underset{0\leqslant i\leqslant 7}{\operatorname{argmax}}[D_i(x,y)] \tag{3.21}$$

3.3.4　用 MATLAB 检测边缘

借助前面讨论的理论基础，本小节介绍如何使用 MATLAB 来计算图像中的边缘。为生成可确定图像边缘的程序，使用任一个蒲瑞维特算子、索贝尔算子、罗伯特算子或基尔希算子，都需要将程序分成 3 部分。

在第 1 部分中，需要每个滤波器生成一幅图像。一般对应算子的定义（见式(3.4)），需要不同方向上的两幅图像。在这部分中，滤波器（即系数矩阵）需要在空间上与图像卷积。这个过程的结果是由各个滤波器定义的方向上的梯度幅度值。图 3.9 展示了这个过程（考虑使用索贝尔算子）。

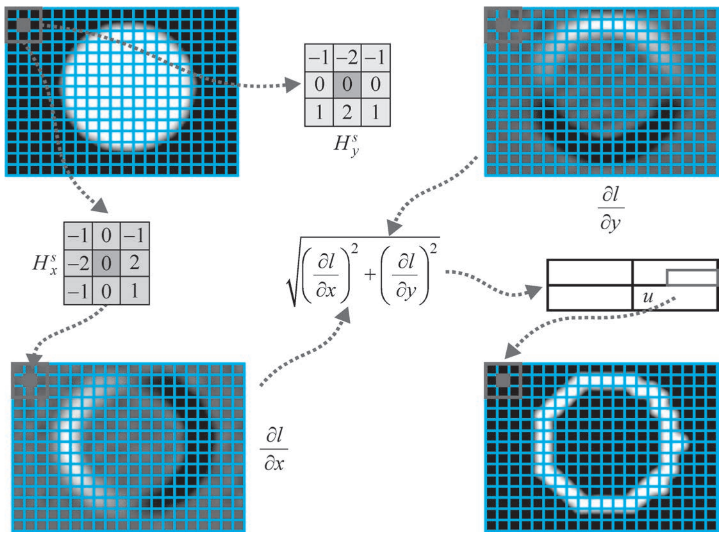

图 3.9　使用索贝尔算子确定梯度幅度的实现过程

在第 2 部分中，从（水平或垂直）滤波器和图像之间卷积得到的结果图像进一步计算梯度的幅度（见式(3.5)）。

在第 3 部分中，设置一个阈值 U，以根据所考虑像素的梯度幅度值确定其是否为边缘的一部分。这个值自然定义了目标的结构特性。通过使用阈值 U，梯度幅度图被二值化。该图的元素只有 1 和 0。一个像素将是边缘如果它的值是 1；否则，它将是 0。程序 3.1 给出了用 MATLAB 实现的使用索贝尔算子确定图像中边缘的代码。

程序 3.1　使用 MATLAB 确定图像的边界

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% Determining the Edges of an Image Using the Sobel Operator %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% The original image is brought into the MATLAB environment
I = imread('fotos/puente.jpg');
```

```
% Convert the original image to an intensity image
% to be able to operate on it
Im = rgb2gray (I);
% The values of the dimensions of the image are obtained [m,n] = size(Im);
% The image is converted to double to avoid problems
% in data type conversion
Im = double(Im);
% Matrices are created with zeros
Gx = zeros(size(Im));
Gy = zeros(size(Im));

% FIRST PART
% Sobel Filters (see Equation 6.10) are applied
% to the image Gx, in the x - direction and
% Gy in the y - direction
for r = 2:m - 1
    for c = 2:n - 1
        Gx(r,c) = - 1 * Im(r - 1,c - 1) - 2 * Im(r - 1,c) - Im(r - 1,c + 1)...
            + Im(r + 1,c - 1) + 2 * Im(r + 1,c) + Im(r + 1,c + 1);
        Gy(r,c) = - 1 * Im(r - 1,c - 1) + Im(r - 1,c + 1) - 2 * Im(r,c - 1)...
            + 2 * Im(r,c + 1) - Im(r + 1,c - 1) + Im(r + 1,c + 1);
    end
end

% SECOND PART
% Total Gradient Value is calculated
% (see Equation 3.5 or 3.13 )
Gt = sqrt (Gx. ^2 + Gy. ^2);
% The maximum value of the gradient is obtained
VmaxGt = max (max(Gt));
% Normalize the gradient to 255
GtN  = (Gt/VmaxGt)  * 255;

% The image is converted for display
GtN = uint8 (GtN) ;
% The minimum values for the Gradients x and y are obtained.
VminGx = min(min(Gx));
VminGy = min(min(Gy));

% Using the lows shifts to avoid negatives.
GradOffx = Gx - VminGx;
GradOffy = Gy - VminGy;

% The maximum values for Gx and Gy are obtained.
VmaxGx = max(max(GradOffx)) ;
VmaxGy = max(max(GradOffy));

% Gradients are normalized to 255
GxN = (GradOffx/VmaxGx) * 255;
```

```
GyN = (GradOffy/VmaxGy) * 255;

 % The image is converted for display
GxN = uint8 (GxN) ;
GyN = uint8 (GyN) ;

 % Display of the gradients in x and y
figure
imshow (GXN)
figure
imshow (GyN)

 % THIRD PART
 % The image is binarized considering a threshold of 100
B = GtN > 25;
 % Full gradient display
figure
imshow (GtN)
 % Edge Image Display
figure
imshow(B)
```

使用程序 3.1 中的代码，所得到的结果如图 3.10 所示，其中可以看到原始强度图像，用索贝尔算子得到的 x-轴方向和 y-轴方向的梯度，以及总梯度和边缘。

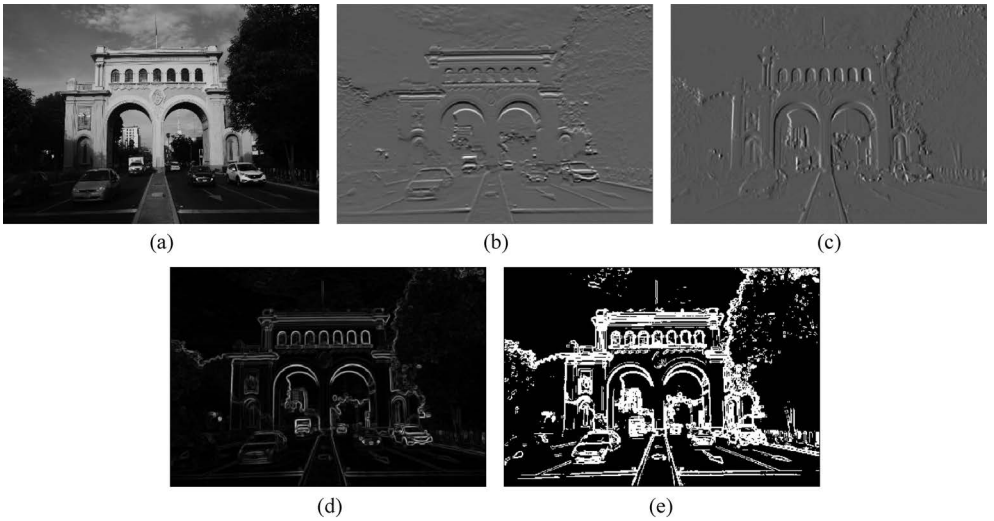

图 3.10　在强度图像上应用程序 3.1 的代码所得到的结果

(a) 原始灰度图像；(b) x 方向梯度；(c) y 方向梯度；(d) 总梯度；(e) 图像边缘

3.3.5　用于边缘检测的 MATLAB 函数

图像处理工具箱提供了函数 edge，它实现了前面各节讨论的不同算子(索贝尔算子、蒲瑞维特算子和罗伯特算子)[7]。对其中某些算子，还可以指定滤波器计算的方向。所以，可以指出梯度沿水平、垂直或两个方向执行的灵敏度。函数的通用结构可描述如下：

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

其中，f 是需要提取边缘的图像；'method'对应表 3.1 中列出的算子之一；parameters 代表根据所使用方法而需要指定的配置；输出 g 是一幅二值图像，其中属于 f 中要检测边缘的像素具有值 1（否则，它们具有值 0）；参数 t 是可选的，它提供了算法使用的阈值以确定什么样的梯度值可以被看作边缘。

表 3.1　图像处理工具箱中函数 edge 用到的边缘检测算子

算　　子	'method'	滤　波　器
蒲瑞维特	'prewitt'	$\boldsymbol{H}_x^{\mathrm{P}}=\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}$　$\boldsymbol{H}_y^{\mathrm{P}}=\begin{bmatrix}-1&-1&-1\\0&0&0\\1&1&1\end{bmatrix}$
索贝尔	'sobel'	$\boldsymbol{H}_x^{\mathrm{S}}=\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix}$　$\boldsymbol{H}_y^{\mathrm{S}}=\begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix}$
罗伯特	'roberts'	$\boldsymbol{H}_1^{\mathrm{R}}=\begin{bmatrix}0&1\\-1&0\end{bmatrix}$　$\boldsymbol{H}_2^{\mathrm{R}}=\begin{bmatrix}-1&0\\0&1\end{bmatrix}$

1. 索贝尔算子

索贝尔算子使用表 3.1 描述的滤波器近似偏导数$\partial I/\partial u$ 和$\partial I/\partial v$。根据式(3.5)结合这些偏导数，可以得到各个像素(u,v)的梯度值。所以，一个像素(u,v)被认为对应图像中的边缘，如果该值大于一个预先定义的阈值。

图像处理工具箱调用边缘检测函数（使用索贝尔方法）的一般命令是：

```
[g,t] = edge(f,'sobel',U,dir);
```

其中，f 是需要提取边缘的图像；U 是一个阈值，用作判断边缘的准则；dir 选择梯度的方向，可以是'horizontal'（对于 $\boldsymbol{H}_x^{\mathrm{S}}$）或'vertical'（对于 $\boldsymbol{H}_y^{\mathrm{S}}$）。选项'both'是默认值，代表两个滤波器都计算。作为结果，函数给出图像 g，它包含检测出的边缘。g 是一幅二值图像，其中边缘像素具有值 1，而非边缘像素具有值 0。如果指定了阈值 U，则 t = U。如果没有指定阈值 U，那么算法自动确定一个阈值进行边缘检测，并返回给 t。

2. 蒲瑞维特算子

蒲瑞维特算子使用表 3.1 指定的系数矩阵进行边缘检测。该函数的一般结构如下：

```
[g,t] = edge(f,'prewitt',U,dir);
```

该函数的参数与索贝尔的参数相同。蒲瑞维特算子实现起来比索贝尔算子稍简单一些（从计算角度看）。但是，所得结果的噪声多一点。这是因为索贝尔算子对数据进行了平滑，这可从计算梯度时沿像素行或列为 2 的系数看出。

3. 罗伯特算子

罗伯特算子使用表 3.1 指定的滤波器来近似在像素(x,y)的梯度幅度。该函数的一般结构如下：

```
[g,t] = edge(f,'roberts',U,dir);
```

该函数的参数与索贝尔的参数相同。罗伯特算子是用于计算图像梯度最老的方法之

一。尽管该方法实现最简单，但它的功能有限，因为它是不对称的。所以，它不能检测沿对角线 45°倍数的边缘。作为比较，图 3.11 给出应用前述各个不同的算子检测图像中边缘的比较。在图 3.11 中，所有函数在所有情况下使用相同的阈值。

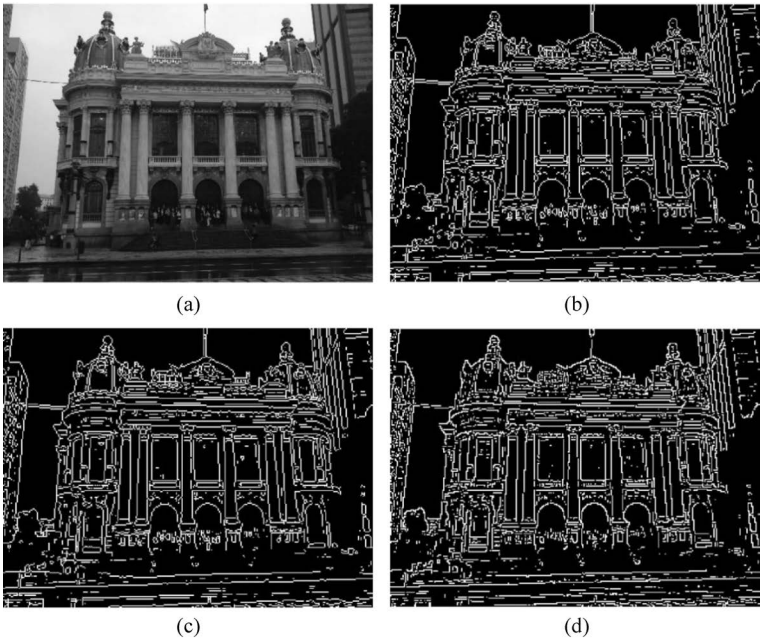

图 3.11　不同算子用于边缘检测的结果比较

（a）原始图像；（b）索贝尔算子检测出的边缘；（c）蒲瑞维特算子检测出的边缘；（d）罗伯特算子检测出的边缘（在所有情况下，都使用了 $U=0.05$）

3.4　基于二阶导数的算子

前面在 3.2 节介绍了一组基于一阶导数的算子，以近似图像的梯度。除了这些滤波器，还有其他类型的算子，它们基于图像函数的二阶导数[8]。在图像中，强度改变可以根据各个方向的导数值（最大梯度）或根据二阶导数（拉普拉斯算子）的过零点计算。将会看到，使用一阶导数来进行边缘检测的问题是，当代表高度方向性的方法或边缘没有很好定义时，定位很困难。

尽管可以利用梯度值完成对边缘的检测，有时候知道像素处在梯度的正过渡区或负过渡区也很重要。这等价于知道像素是在边缘暗的一边还是亮的一边。

3.4.1　使用二阶导数技术的边缘检测

这种技术基于确定什么称为过零点（零交叉）。过零点是从正到负或反过来的过渡，这可通过二阶导数来估计。

一个 1-D 函数的导数可定义如下：

$$\frac{\partial f}{\partial x}=f(x+1)-f(x) \tag{3.22}$$

类似地，从式(3.22)可以定义二阶导数如下：

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) - 2f(x) + f(x-1) \tag{3.23}$$

用二阶导数检测边缘基于任何方向上梯度值的过零点。为获得这种效果，可使用拉普拉斯算子，它对旋转不敏感且是各向同性的。

$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2} \tag{3.24}$$

这个滤波器使用二阶导数。在拉普拉斯算子的式(3.24)中考虑代入式(3.23)，则有

$$\frac{\partial^2 I(x,y)}{\partial x^2} = I(x+1,y) - 2I(x,y) + I(x-1,y) \tag{3.25}$$

$$\frac{\partial^2 I(x,y)}{\partial y^2} = I(x,y+1) - 2I(x,y) + I(x,y-1) \tag{3.26}$$

这样，如果将式(3.25)和式(3.26)代入式(3.24)，则有

$$\nabla^2 I(x,y) = I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y) \tag{3.27}$$

如果将上述方程写成滤波器的形式，则得到如图 3.12 所示的系数矩阵。

如图 3.12 所示的滤波器（根据式(3.27)描述的形式）没有考虑像素在对角线邻域的变化，但这可以结合进滤波器。为考虑这一点，可在滤波器中增加 4 个 1。因此，中心系数也需要增加到 8。此时拉普拉斯算子可定义为如图 3.13 所示。

0	1	0
1	−4	1
0	1	0

图 3.12 代表拉普拉斯算子在图像上计算的滤波器，由式(3.27)得到

1	1	1
1	−8	1
1	1	1

图 3.13 对中心像素扩展了对角线邻接像素的拉普拉斯滤波器

从对拉普拉斯算子的计算可见，在均匀灰度区域的响应为 0，而在灰度变化区域的响应是不同的。这是因为这类滤波器的效果类似高通滤波器。滤波器系数之和总为 0。

一个在灰度图像上应用拉普拉斯算子的例子见图 3.14。它代表了在每个方向上应用二阶导数的不同效果，最终这两个结果按拉普拉斯算子的定义加起来。

3.4.2 图像的锐化增强

如果将拉普拉斯算子应用于一幅图像，将得到它的边缘。不过，如果希望改进图像的锐度，就需要保留原始图像的低频信息，并通过拉普拉斯滤波器加强图像中的细节。为获得这样的效果，需要从原始图像中减去拉普拉斯滤波器值的一个缩放版本。因此，具有改进锐度的图像可定义如下：

$$I(x,y)_{\text{Enhanced}} = I(x,y) - w \cdot \nabla^2 I(x,y) \tag{3.28}$$

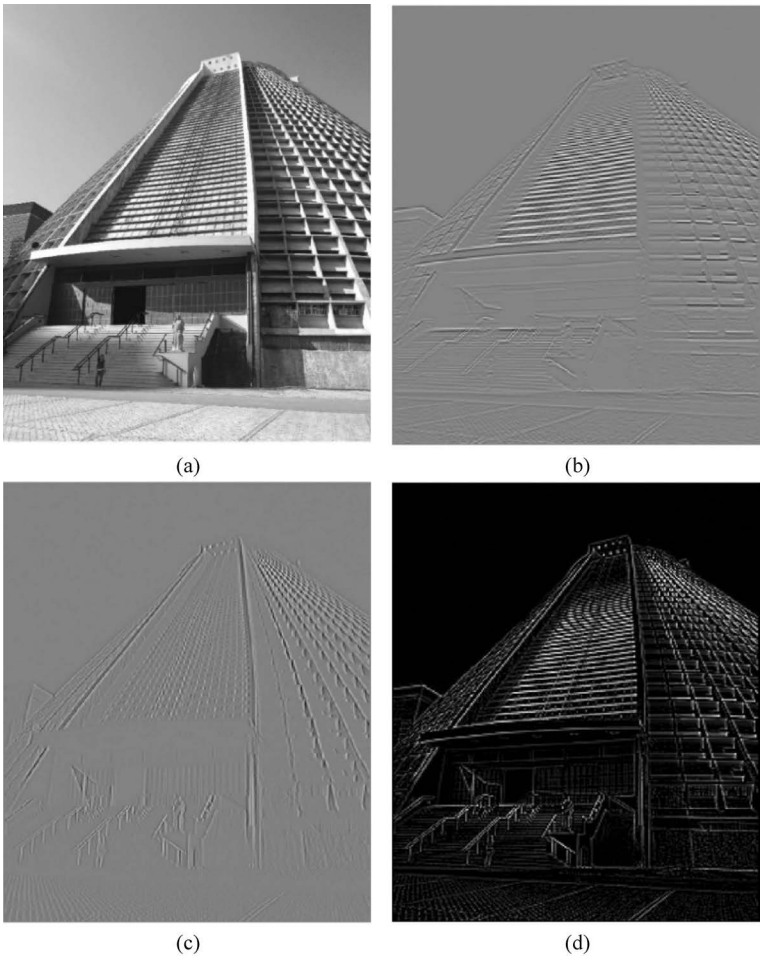

图 3.14　拉普拉斯滤波器的使用

(a) 原始图像；(b) 水平二阶导数$\partial^2 I(x,y)/\partial x^2$；
(c) 垂直二阶导数$\partial^2 I(x,y)/\partial y^2$；(d) 拉普拉斯算子$\nabla^2 I(x,y)$

图 3.15 展示了这个概念，图像借此通过使边缘更加明显而改进了其锐度。为方便解释，图中仅考虑了 1-D 的情况。

改进图像锐度的效果可通过一步操作实现。考虑 $w=1$，增强模型可定义如下：

$$I(x,y)_{\mathrm{En}} = I(x,y) - (1) \cdot \nabla^2 I(x,y) \tag{3.29}$$

如果用式(3.27)的表达式替换式(3.29)中的拉普拉斯算子，增强模型可定义如下：

$$I(x,y)_{\mathrm{En}} = 5I(x,y) - [I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1)] \tag{3.30}$$

如果式(3.30)表示了滤波器的形式，则其系数矩阵可定义如下：

$$\boldsymbol{I}(x,y)_{\mathrm{En}} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{3.31}$$

图 3.16 给出了对一幅图像使用这个滤波器得到的结果。

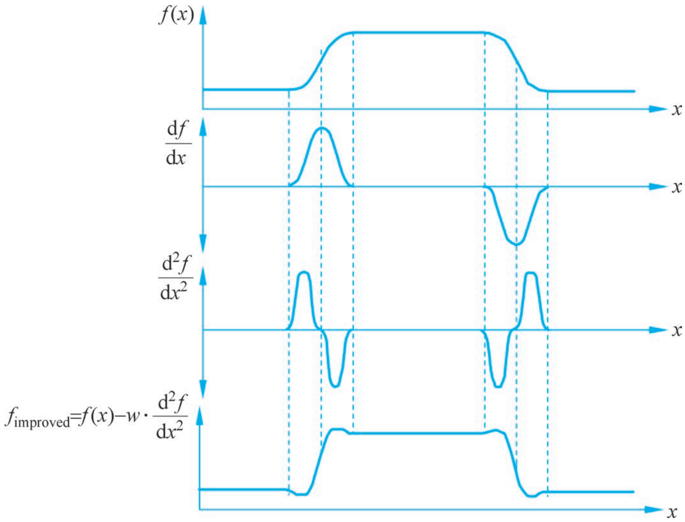

图 3.15　使用二阶导数进行锐化（通过从函数中减去其二阶导数的一个因子，可以最大化图像中的边界）

(a)　　(b)

图 3.16　使用拉普拉斯算子以改进图像的锐度

(a) 原始图像；(b) 使用 $I(x,y)_{\text{Enhanced}} - w \cdot \nabla^2 I(x,y)$ 得到的图像，其中 $w=1$

3.4.3　用 MATLAB 实现拉普拉斯滤波器和增强锐度

本小节将描述用 MATLAB 改进图像的锐度。在这个过程中，也需要计算拉普拉斯算子。改进图像锐度的 MATLAB 代码可被分成两个简单步骤。第 1 步，根据图 3.12 描述的滤波器形式计算拉普拉斯滤波器。第 2 步，考虑用原始图像中包含的低频信息减去拉普拉斯算子的值（见式（3.28））以生成图像锐度，这里拉普拉斯算子结合了允许增强图像细节的信息。程序 3.2 给出使用拉普拉斯算子改进图像锐度的 MATLAB 代码。

程序 3.2　使用 MATLAB 借助拉普拉斯算子改进图像的锐度

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%  Improving the sharpness of an image using the Laplacian operator  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%  The image is read in order to process it.
I = imread('img.jpg');
```

```
% A conversion is made in the color space to work with
% an intensity image
Im = rgb2gray(I);
% The factor with which the Laplacian operator
% affect the image (see Equation 3.32)
w = 1;
% The dimension values of the image are obtained:
[m,n] = size(Im);
% The image is converted to double to avoid problems in the conversion of the data type:
Im = double(Im);
% The matrix L is created with zeros:
L = zeros(size(Im));

% FIRST PART.
% The Laplacian filter is applied:
for x = 2:m - 1
      for y = 2:n - 1
         L (x,y) =  m(x + 1,y) + Im(x - 1,y) + Im(x,y + 1) + Im(x,y - 1)  - 4 * Im(x,y);
 end
end
% SECOND PART.
% The new pixels of the image whose sharpness
% is intended to improve (see Formula 3.32):
Init = Im - w * L;
% The minimum value for the Init image is obtained:
VminInit = min(min(Init));
% Using the minimum value shifts to avoid negatives:
GradOffL = Init - VminInit;
% The maximum value to normalize to 1 is obtained:
VmaxInit = max(max(GradOffL));
% The gradients are normalized to 255:
InitN = (GradOffL/VmaxInit) * 255;
% Convert the image for deployment:
GInitN = uint8(InitN);
% The image is displayed to analyze its results
figure
imshow(GInitN)
```

使用程序 3.2 的代码的结果如图 3.17 所示。从图 3.17 中可看到原始灰度图像与用 $w=1$ 增强图像对比度得到图像之间的差别。

(a)　(b)

图 3.17　使用程序 3.2 的 MATLAB 代码借助拉普拉斯算子改善图像锐度的结果

(a) 原始灰度图像；(b) 使用 $w=1$ 增强对比度得到的图像

3.4.4 坎尼滤波器

使用坎尼滤波器是一种广为人知的检测图像中边缘的方法[2]。这个方法基于应用一系列不同方向和分辨率的滤波器，它们最终结合并给出单个结果。这个方法有 3 个目标：

（1）最小化虚假边缘的数量；

（2）改善图像中边缘的位置；

（3）给出只有单个像素宽度的边缘。

坎尼滤波器本质上是一个基于梯度的滤波器，但它也使用了二阶导数（或拉普拉斯算子），作为边缘位置的准则。大多数情况下，这个算法使用其简单形式，即仅设置平滑参数 σ。图 3.18 给出应用该算法时采用不同参数值 σ 的例子。

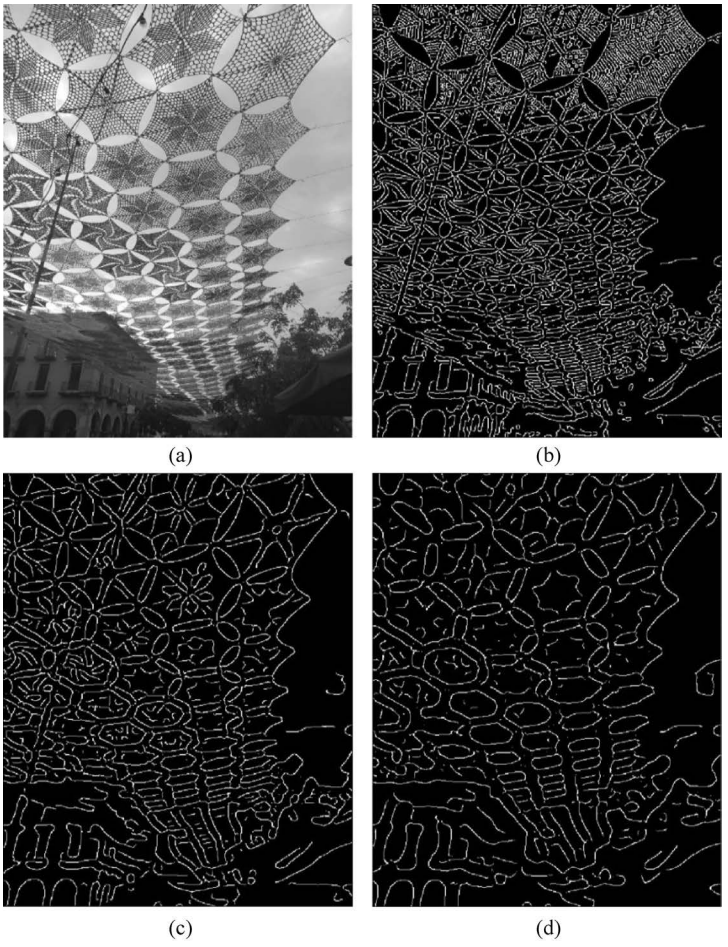

图 3.18 坎尼算法用于一幅图像

（a）原始图像；（b）$\sigma=2$ 时的图像边缘；（c）$\sigma=4$ 时的图像边缘；（d）$\sigma=6$ 时的图像边缘

3.4.5 实现坎尼滤波器的 MATLAB 工具

因为坎尼滤波器广泛应用于分割和目标分类的预处理阶段以及检测边缘的鲁棒性，大多数商业库和数字图像处理工具都包括坎尼滤波器。在 MATLAB 中坎尼算法可使用函数

edge 来计算。它的通用结构为

```
BW = edge(I,'canny',U,sigma);
```

其中，BW 是带有检测出边缘的图像(使用了坎尼算法)；I 是需要从中提取边缘的灰度图像；U 是用于判断系数为边缘的阈值；sigma 代表平滑参数(σ)，它有最小化虚假边缘数量的效果。

参考文献

[1] Gose E, Johnsonbaugh R, Jost S. *Pattern recognition and image analysis*. CRC Press, 2017.

[2] Canny J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986, 8(6), 679-698. https://ieeexplore.ieee.org/document/4767851.

[3] Bina S, Ghassemian H. A Comprehensive Review of Edge Detection Techniques for Images in Computer Vision. *Journal of Computer and Communications*, 2019, 7(1), 36-49. https://doi.org/10.14257/ijmue.2017.12.11.01.

[4] Hussain S, Hussain M. A Review of Edge Detection Techniques in Digital Image Processing. *International Journal of Scientific and Engineering Research*, 2014, 5(2), 222-231. https://www.researchgate.net/journal/International-Journal-of-Science-and-Research-IJSR-2319-7064.

[5] Singh S, Bhatnagar G. Comparative Analysis of Edge Detection Techniques for Digital Images. *Journal of Information Technology and Computer Science*, 2015, 7(1), 31-40. https://doi.org/10.1109/ICCCIS51004.2021.9397225.

[6] Abdullah-Al-Wadud M, Islam M A, Islam M M. A Review of Image Edge Detection Techniques and Algorithms. *Journal of Electromagnetic Analysis and Applications*, 2017, 9(10), 240-251. https://doi.org/10.1007/s11633-018-1117-z.

[7] Solomon C, Breckon T. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in MATLAB*. Wiley, 2010.

[8] Tariq M A, Raza A, Abbas M. A Comparative Study of Edge Detection Techniques in Digital Image Processing. *IEEE Access*, 2019, 7, 40793-40807. https://doi.org/10.46565/jreas.2021.v06i04.001.