1 方法简介

1.1 传统 Hough 变换

最简单的霍夫变换是在图像中识别直线。在平面直角坐标系 $(x-y)$ 中，一条直线可以用方程式

$$y = m_0 x + b_0 \tag{1.1}$$

表示,b_0 是直线的截距,m_0 是直线的斜率. 而 (m_0, b_0) 可以视为参数空间 (m, b) 中的一点. 当直线垂直于 x 轴时, 斜率为无限大, 若用电脑数值计算时会很不方便, 因此提出使用 $Hessenormalform$ 形式来表示直线的参数,r 是从原点到直线的距离,θ 是 $\vec{r}$ 和 x 轴的夹角.

$$r = x\cos(\theta) + y\sin(\theta) \tag{1.2}$$

考虑一个以参数 r 和 θ 定义的二维空间,x, y 平面的任意一条直线映射为 r, θ 空间的一个点, x, y 平面的一点 $p_1(x_1, y_1)$, 过该点的直线可有很多, 每一条都对应了 r, θ 空间中的一点, 这些点必然满足以 x_1, y_1 为常量的等式. 在参数空间中与 x, y 空间中所有这些直线对应的点的轨迹是一条正弦曲线, 而 x, y 平面上的任一点对应了 r, θ 空间的一条正弦曲线, 可以将 Hough 变换的算法设计为 Algorithm 1.

Algorithm 1 Conventional voting process of the Hough transform

Require: $I\,\{Binary\ image\}$
Require: $\delta\,\{Discretization\ step\ for\ the\ parameter\ sapce\}$
  $Votes \leftarrow 0\,\{Initialization\ of\ the\ voting\ matrix\}$
  for each feature pixel $I(x, y)$ **do**
    for $0° \leq \theta < 180°$,using a δ discretization step **do**
      $\rho \leftarrow x\cos(\theta) + y\sin(\theta)$
      $Votes(\rho, \theta) \leftarrow Votes(\rho, \theta) + 1$
    end for
  end for

1.2 基于核函数的 Hough 变换

根据前一节所述的算法, 传统 Hough 变换中效率低下和误检率高的最大来源是其穷尽式的投票过程. 这种从图像域投票到参数域的穷尽式搜索模式, 不仅计算量大, 占用内存多; 同时, 参数域中得票最多的主峰被得票数次多的次峰所包围, 容易对检测造成干扰, 导致对直线的误检或漏检. 其中主峰对应于更好地代表图中的线, 而次峰的产生主要由于图像空间的不确定性以及特征像素可能不是完全共线. 因此提出一个简单又自然的投票方式

1 对于一个边缘图像, 将其大致共线的特征像素聚类

2 对每一个聚类, 计算其最佳拟合直线和模型的不确定性

3 根据模型的不确定性得出一个高斯椭圆核函数, 并对每个聚类进行投票

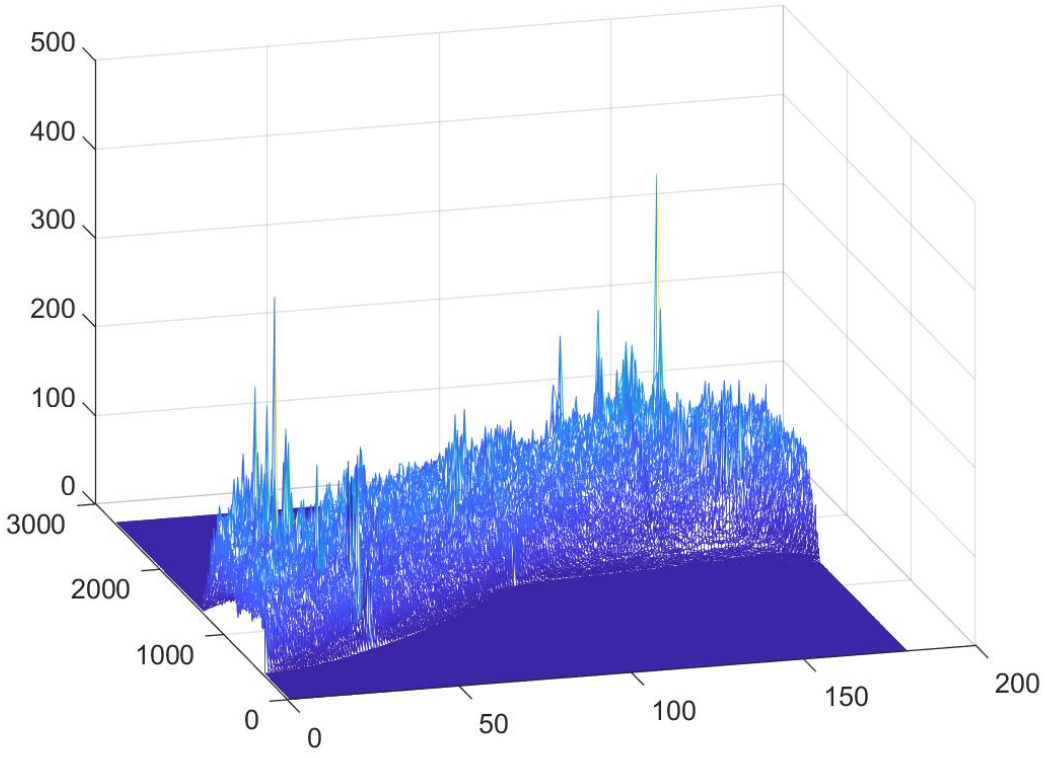

图 1.1　传统 Hough 变换的峰值分布

2 方法说明

2.1 相邻像素点的聚类

将图像中相连的特征像素进行聚类, 这样可以排除小短线和噪声影响, 解决了直线检测过程中过连接问题, 并且可以排除不含有直线特征的连通域, 提高算法的检测效率, 减少一定的计算量和占用内存. 设计算法通过 $O(n^2)$ 的时间复杂度对图像经行遍历搜索, 算法设计见 Algorithm 2,Algorithm 3.

Algorithm 2 Function $Next()$.

for each neighbor pixel of p_seed **do**
  $p \leftarrow$current neighbor pixel of p_seed
  if p is a feature pixel **then**
    Break the loop and return p $\{The\ next\ pixel\ was\ found\}$
  end if
end for
Return an invalid pixel position$\{The\ next\ pixel\ was\ not\ found\}$

2.2 直线的细分编组

图像聚类后, 相连的像素点可能为近似直线, 也可能是若干近似直线段组合成的折线. 为便于检测, 需对折线作进一步细分, 以使细分后的每一段更接近直线.

若 A 为聚类后的某线段,$(x_1, y_1), (x_2, y_2)$ 分别是该线段两端点的坐标, 而 B 为由这两端点确定的直线,(x_3, y_3) 是线段 A 偏离直线 B 的偏离距离为 d 的点的坐标,d 可有几何推导可得

$$d = \frac{(y_3 - y_1)(x_2 - x_1) - (x_3 - x_1)(y_2 - y_1)}{L} \tag{2.1}$$

式中 $L = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$.

用曲线两端点的直线距离与曲线相对直线的最大偏离距离之比代表此线段曲线接近直线的程度, 即

$$R = L/max_d \tag{2.2}$$

该比率 R 仅仅与近似直线本身的性质有关, 而与图像的尺寸无关, 不会随图像大小的变换而改变.

Algorithm 3 Linking of neighboring edge pixels into strings. The pseudo code for the function $Next()$ is presented in Algorithm 2

Require: $I\,\{Binary\ image\}$

Require: $p_{ref}\,\{A\ feture\ pixel\}$

Ensure: A chain of pixels

1: $S \leftarrow \varnothing$ $\{Create\ an\ empty\ string\ of\ pixels\ \}$
2: $\{Find\ and\ add\ feature\ pixels\ to\ the\ end\ of\ the\ string\}$
3: $p \leftarrow p_{ref}$
4: **repeat**
5: $\quad S \leftarrow S + p$
6: $\quad I(x_p, y_p) \leftarrow 0$
7: $\quad p \leftarrow Next(p, I)$
8: **until** p be an invalid pixel position
9: $\{Find\ and\ add\ pixels\ to\ the\ begin\ of\ the\ string\}$
10: $p \leftarrow Next(p_{ref}, I)$
11: **if** p is a valid pixel position **then**
12: $\quad$ **repeat**
13: $\quad\quad S \leftarrow p + S$
14: $\quad\quad I(x_p, y_p) \leftarrow 0$
15: $\quad\quad p \leftarrow Next(p, I)$
16: $\quad$ **until** p be an invalid pixel position
17: **end if**

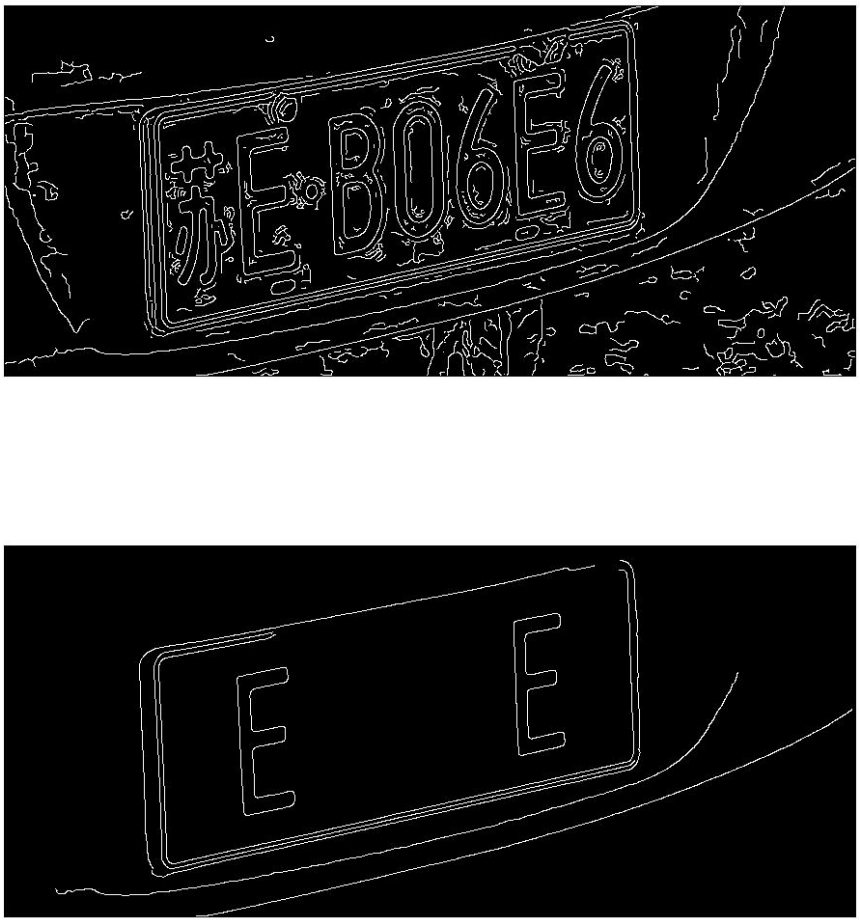

图 2.1　边缘图像和其聚类后的图像

从偏移距离最大的点对原线段进行二分割, 并计算分割后两子线段的比率 $R_i(i = 1, 2)$, 若他们均大于原线段的比率, 则用两子线段代替原线段, 并以相同代替段到最小细分值, 直到最小子线段达到阈值为止.

图像经过感知编组后, 每条线段更接近直线.

2.3　计算核函数

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{2.3}$$

算法由 Algorithm 4给出. 给定一个由几乎共线的特征像素组成的聚类 S, 其投票分布在其最佳拟合直线 l 周围. 选取 $\bar{p} = (x_{\bar{p}}, x_{\bar{p}})^T$,$\bar{p}$ 为聚类的质心像素. 类似于 PCA 主成分分析,$u = (x_u, y_u)^T$ 为最大特征值的特征向量,$v = (x_v, y_v)^T$ 为第二大特征值

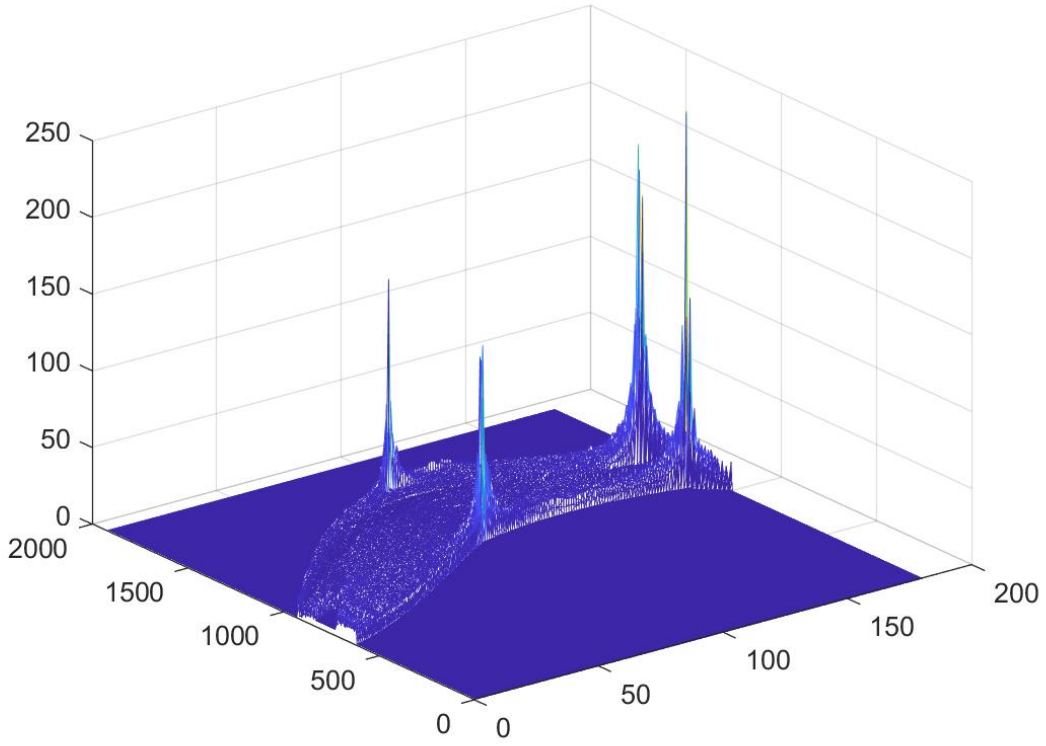

图 2.2　聚类后的峰

的特征向量. 则直线可以写成

$$Ax + By + C = x_v x + y_v y - (x_v x_{\bar{p}} + y_v y_{\bar{p}}) = 0 \tag{2.4}$$

联立2.3和2.4, 得到 ρ 和 θ 的表达式

$$\rho = -C = x_v x_{\bar{p}} + y_v y_{\bar{p}} \tag{2.5}$$

$$\theta = \cos^{-1}(A) = \cos^{-1}(x_v) \tag{2.6}$$

投票分布在最佳拟合直线 l 的不确定性可由 σ_ρ^2 和 σ_θ^2 来表示, 并且这两者用来定义核函数的形状和性质, 此外核函数的方向由协方差 $\sigma_{\rho\theta}$ 决定 为了避免直线垂直的情况而无法计算上述参数, 将直线映射到以 $\bar{p}$ 为原点,u' 为水平轴, 于是最佳拟合直线总是水平的.

$$p_i{}' = \begin{pmatrix} x_{p_i'} \\ y_{p_i'} \\ 1 \end{pmatrix} = RTp_i, l' = \begin{pmatrix} A' \\ B' \\ C' \end{pmatrix} = ((RT)^{-1})^T l = R(T^{-1})^T l \tag{2.7}$$

Algorithm 4 Computation of the Gaussian kernel parameters

for each group of pixels S_k **do**

$p \leftarrow$pixels in S_k $\{Pixels\ are\ 2D\ column\ vectors\}$

$n \leftarrow$ number of pixels in S_k

$\{Alternative\ reference\ system\ definition\}$

$\bar{p} \leftarrow$mean point of p $\{Also\ a\ column\ vector\}$

$V, \Lambda \leftarrow eigen((p-\bar{p})(p-\bar{p})^T)\ \{Eigen-decomposition\}$

$u \leftarrow$eigenvector in V for the biggest eigenvector in Λ

$v \leftarrow$eigenvector in V for the smaller eigenvector in Λ

$y_v \geq 0$ $\{condition\ verification\}$

if $y_v < 0$ **then**

$v \leftarrow -v$

end if

$\{Normal\ equation\ parameters\ computation\}$

$\rho_k \leftarrow \langle v, \bar{p} \rangle$

$\theta_k \leftarrow \cos^{-1}(x_v)$

$\sigma^2_{m'} \leftarrow 1/\sum \langle u, p_i - \bar{p} \rangle^2$

$\sigma^2_{b'} \leftarrow 1/n$

$M = \nabla f \begin{pmatrix} \sigma^2_{m'} & 0 \\ 0 & \sigma^2_{b'} \end{pmatrix} \nabla f^T$

$sigma^2_{\rho_k} \leftarrow 2^2 M_{(1,1)}$

$sigma^2_{\theta_k} \leftarrow 2^2 M_{(2,2)}$

$sigma_{\rho_k \theta_k} \leftarrow M_{(1,2)}$

end for

式中 $l=(A,B,C)^T$,R,T 为以下矩阵

$$R=\begin{pmatrix} x_u & y_u & 0 \\ x_v & y_v & 0 \\ 0 & 0 & 1 \end{pmatrix}, T=\begin{pmatrix} 1 & 0 & -x_{\bar{p}} \\ 0 & 1 & -y_{\bar{p}} \\ 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

对聚类中的特征像素进行线性拟合, 估计除 m', n' 和他们的协方差, 方差

$$\sigma_{m'}^2=\frac{S_\sigma}{\Delta}, \sigma_{b'}^2=\frac{S_{x^2}}{\Delta}, \sigma_{m'b'}=\frac{S_x}{\Delta} \tag{2.9}$$

其中

$$\begin{aligned} S_\sigma &= \sum_{i=1}^{n-1}\frac{1}{\sigma_i^2}, S_{x^2}=\sum_{i=0}^{n-1}\frac{(x_{p_i'})}{\sigma_i^2} \\ S_x &= \sum_{i=1}^{n-1}\frac{x_{p_i'}}{\sigma_i^2}, \Delta = S_\sigma S_{x^2}-S_x^2 \end{aligned} \tag{2.10}$$

其中 σ_i 是变换后 v' 坐标系的特征像素点的不确定性. 因为图像的像素的离散性质, 所以不确定性 ± 1 个像素, 于是假设 $\sigma_i^2=1$, 并且由于 $\bar{p'}=(0,0)^T$, 故 $S_x=0$, 于是式子可以简化成

$$\sigma_{m'}^2=\frac{1}{\sum_{i=0}^{n-1}(x_{p_i'})^2}, \sigma_{b'}^2=\frac{1}{n}, \sigma_{m'b'}=0 \tag{2.11}$$

再次由2.7的逆映射可得

$$l=\begin{pmatrix} A \\ B \\ C \end{pmatrix}=(R(T^{-1})^T)^{-1}l'=T^TR^T\begin{pmatrix} -m' \\ 1 \\ -b' \end{pmatrix}=\begin{pmatrix} x_v-x_um' \\ y_v-y_um' \\ (x_ux_{\bar{p}}+y_uy_{\bar{p}})m'-x_vx_{\bar{p}}-y_vy_{\bar{p}}-b' \end{pmatrix} \tag{2.12}$$

从2.12解得

$$\rho=-C=x_vx_{\bar{p}}+y_vy_{\bar{p}}+b'-(x_ux_{\bar{p}}+y_uy_{\bar{p}})m'\theta=\cos^{-1}(x_v-x_um') \tag{2.13}$$

于是 ρ 和 θ 可以从上述分析估计

$$\begin{pmatrix} \sigma_\rho^2 & \sigma_{\rho\theta} \\ \sigma_{\rho\theta} & \sigma_\theta^2 \end{pmatrix}=\nabla f\begin{pmatrix} \sigma_{m'}^2 & \sigma_{m'b'} \\ \sigma_{m'b'} & \sigma_{b'}^2 \end{pmatrix}\nabla f^T \tag{2.14}$$

$$\nabla f=\begin{pmatrix} \frac{\partial\rho}{\partial m'} & \frac{\partial\rho}{\partial b'} \\ \frac{\partial\theta}{\partial m'} & \frac{\partial\theta}{\partial b'} \end{pmatrix}=\begin{pmatrix} -x_ux_{\bar{p}}-y_uy_{\bar{p}} & 1 \\ x_u/\sqrt{1-x_v^2} & 0 \end{pmatrix} \tag{2.15}$$

2.4 使用高斯分布进行投票

一旦计算得到了 ρ 和 θ 的方差和协方差, 投票可以根据一个双变量的高斯分布进行

$$G_k(\rho_j, \theta_j) = \frac{1}{2\pi\sigma_\rho\sigma_\theta\sqrt{1-r^2}} e^{\frac{-z}{2(1-r^2)}} \tag{2.16}$$

其中

$$z = \frac{\rho_j - \rho}{\sigma_\rho^2} - \frac{2r(\rho_j - \rho)(\theta_j - \theta)}{\sigma_\rho\sigma_\theta} + \frac{(\theta_j - \theta)^2}{\sigma_\theta^2} r = \frac{\sigma_{\rho\theta}}{\sigma_\rho\sigma_\theta} \tag{2.17}$$

在过程中,2.16只需评估 (ρ, θ) 周围的一小部分邻域 ρ_j, θ_j. 超过邻域的特征像素, 计算结果很小以致可以忽略, 其中阈值 g_{min} 可计算为

$$g_{min} = min(G_k(\rho + \rho_\omega\sqrt{\lambda_\omega}, \theta + \theta_\omega\sqrt{\lambda_\omega})) \tag{2.18}$$

其中 $\omega = (\rho_\omega, \theta_\omega)^T$ 为 λ_ω 的特征向量,λ_ω 为2.14的较小特征值.

2.5 投票程序

一旦计算好每个聚类的高斯核参数后, 投票过程就按照 Algorithm 5 和 Algorithm 6 进行, 其中将核函数分为四个象限, 对四个方向进行投票.

另一个重要的优化在 Algorithm 5中是对核函数的剔除操作, 有助于避免过小的核参与投票导致计算量加大. 这样的核函数通常来自于一组间距较大的特征像素, 这些不确定性导致了核的长度过短. 于是, 剔除的方法是取最高核函数的高度, 用它将所有高斯核归一化到 $[0, 1]$ 范围内. 归一化后, 若其核高度小于 h_{min}, 则将其剔除, 在本文中选取 $h_{min} = 0.002$. 由于具有归一化操作, 故 h_{min} 是自适应的.

2.6 峰值检测

与一般排序不同的是, 先进行一次 $3 * 3$ 的高斯核卷积, 这样的卷积操作使得投票盒平滑, 有助于将相邻的峰合并为单个图像线.

最大的峰代表了图像中最重要的直线, 即车牌的旋转角度.

2.7 应用实例

通过检测到的最大峰值得到旋转角度, 再进行旋转. 可以看到明显基于核函数的 hough 变换所要计算的特征像素点远远小于传统 hough 变换, 因此效率得到了极大提升.

Algorithm 5 Proposed voting process.

Require: S $\{Kernels\ parameters\ computed\ by\ Algorithm\ 2\}$

Require: h_{min} $\{Minimum\ height\ for\ a\ kernel\ [0,1]\}$

Require: δ $\{Discretization\ step\ for\ the\ parameter\ sapce\}$

$\{Discard\ groups\ with\ very\ short\ kernels\}$

$h_max \leftarrow height$ of the tallest kernel in S

$S' \leftarrow$groups of pixels in S with $(height/h_{max}) \leq h_{min}$

$\{Findthe g_{min} threshold\}$

$g_{min} \leftarrow 0$

for each group of pixels $S_k{}'$ **do**

$M \leftarrow \begin{pmatrix} \sigma^2_{\rho_k} & \sigma_{\rho_k\theta_k} \\ \sigma_{\rho_k\theta_k} & \sigma^2_{\theta_k} \end{pmatrix}$

$V, \Lambda \leftarrow eigen(M)$

$\omega \leftarrow$eigenvector in V for the smaller eigenvalue in Λ

$\lambda_\omega \leftarrow$smaller eigenvalue in Λ

$g_{min} \leftarrow min(g_{min}, G_k(\rho_k + \rho_{omega}\sqrt{\lambda_\omega}, \theta_k + \theta_\omega\sqrt{\lambda_\omega}))$

end for

$\{Vote\ for\ each\ selected\ kernel\}$

$Votes \leftarrow 0$

$g_s \leftarrow max(1/g_{min}, 1)$

for each $S_k{}'$ group of pixels **do**

$Votes \leftarrow Vote(Votes, \rho_k, \theta_k, \delta, \delta, g_s, S_k{}')$

$Votes \leftarrow Vote(Votes, \rho_k, \theta_k - \delta, \delta, -\delta, g_s, S_k{}')$

$Votes \leftarrow Vote(Votes, \rho_k - \delta, \theta_k - \delta, -\delta, -\delta, g_s, S_k{}')$

$Votes \leftarrow Vote(Votes, \rho_k - \delta, \theta_k, -\delta, \delta, g_s, S_k{}')$

end for

Algorithm 6 Function $Vote()$.

Require: $Votes, \rho_{start}, \theta_{start}, \delta_\rho, \delta_\theta, g_s, {S_k}'$

$\rho_k, \theta_k, \sigma^2_{\rho_k}, \sigma^2_{\theta_k}, \sigma_{\rho_k\theta_k}$
$R \leftarrow \sqrt{\omega^2 + h^2}/2$
$\{Loop\ for\ the\ \theta\ coordinates\ of\ the\ parameter\ space\}$
$\theta_j \leftarrow \theta_{start}$
repeat
  $\{Test\ if\ the\ kernel\ exceeds\ the\ parameter\ space\ limits\}$
  if $\theta_j < 0^\circ$ or $\theta_j \leq 180^\circ$ **then**
    $\delta_\rho \leftarrow -\delta_\rho$
    $\rho_k \leftarrow -\rho_k$
    $\sigma_{\rho_k\theta_k} \leftarrow -\sigma_{\rho_k\theta_k}$
    $\rho_{start} \leftarrow -\rho_{start}$
    if $\theta_j < 0^\circ$ **then**
      $\theta_k \leftarrow 180^\circ - \delta_\theta + \theta_k$
      $\theta_j \leftarrow 180^\circ - \delta_\theta$
    else
      $\theta_k \leftarrow \theta_k - 180^\circ + \delta_\theta$
      $\theta_j \leftarrow 0^\circ$
    end if
  end if
  $\{Loop\ for\ the\ \rho\ coordinates\ of\ the\ parameter\ space\}$
  $\rho_j \leftarrow \rho_{start}$
  $g \leftarrow round(g_s G_k(\rho_j, \theta_j))$
  while $g > 0$ and $-R \leq \rho_j \leq R$ **do**
    $Votes(\rho_j, \theta_j) \leftarrow Votes(\rho_j, \theta_j) + g$
    $\rho_j \leftarrow \rho_j + \delta_\rho$
    $g \leftarrow round(g_s G_k(\rho_j, \theta_j))$
  end while
  $\theta_j \leftarrow \theta_j + \delta_\theta$
until no votes have been included by the internal loop

图 2.3　传统 Hough 变换

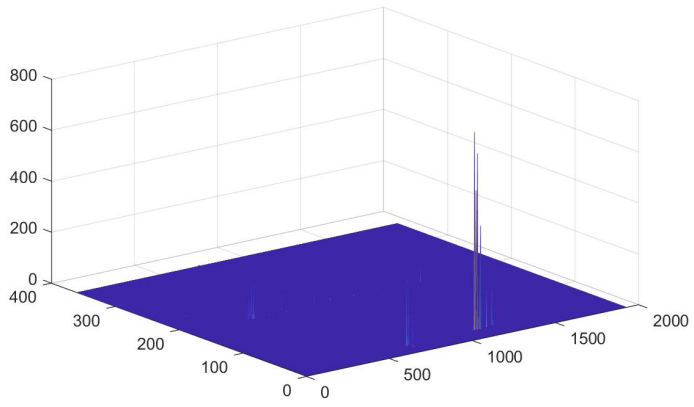

图 2.4　基于核函数的 Hough 变换得到的峰值图像

参考文献

[1] Fernandes, Leandro A. F. and Oliveira, Manuel M. Real-time line detection through an improved Hough transform voting scheme [J]. Pattern Recognition, 2008，41(1): 299–314. https://doi.org/10.1016/j.patcog.2007.04.003.

[2] 段汝娇, 赵伟, 黄松岭, 等. 一种基于改进 Hough 变换的直线快速检测算法 [J]. 仪器仪表学报, 2010(12): 2774-2780.

3 代码

3.1 传统 Hough 变换

3.1.1 main

```
%% Load image
clear;
img = imread("im02.jpg");
%% Find edges
img = img(:,:,1);
figure();
imshow(img);
title('Original Image');

img_edges = edge(img, 'Canny');
chains = {};
chains = find_chains(chains, img_edges, size(img_edges,2),
    size(img_edges,1), 1000);
clusters = {};
clusters = find_clusters(clusters,chains,10,100);

figure();
imshow(img_edges);
title('Edges found in image');

%% Perform Hough Transform for lines
tic
for i = 1:size(clusters,2)
cluster = clusters{i};
cluster = cluster(:,1:2);
cluster = fliplr(cluster);
[H, theta, rho] = hough_lines_acc(img_edges,cluster);
if i == 1
    temp = zeros(size(H,1),size(H,2));
end
temp = temp+H;
```

```
    end
    toc
    %% Plot/show accumulator array H
    H = temp;
    figure();
    imshow(imadjust(mat2gray(H)),'XData',theta,'YData',rho,...
          'InitialMagnification','fit');
    title('Hough transform');
    xlabel('\theta'), ylabel('\rho');
    axis on, axis normal, hold on;
    colormap(hot);

    %% Find Peaks
    gaussianFilter = fspecial('gaussian',3, 3);
    H_filted = imfilter(H, gaussianFilter,'symmetric');
    peaks = hough_peaks(H_filted, 1);

    %% Highlight peak locations on accumulator array
    imshow(imadjust(mat2gray(H)),'XData',theta,'YData',rho,
    'InitialMagnification','fit');
    title('Hough transform with peaks found');
    xlabel('\theta'), ylabel('\rho');
    axis on, axis normal, hold on;
    plot(theta(peaks(:,2)),rho(peaks(:,1)),'o','LineWidth',3,
    'color','red');
    %% Draw Lines
    hough_lines_draw(img, peaks,rho,theta);
    figure,imshow(imrotate(img,peaks(1,2)-180));
    x = 1:size(H,1);
    y = 1:size(H,2);
    [X,Y] = meshgrid(y,x);
    figure,mesh(X,Y,H);
```

```
function [H, theta, rho] = hough_lines_acc(BW, cluster,varargin)
% Compute Hough accumulator array for finding lines.
%
% BW: Binary (black and white) image containing edge pixels
% RhoResolution (optional): Difference between successive rho values, in
    pixels
% Theta (optional): Vector of theta values to use, in degrees
% Rows of H should correspond to values of rho, columns those of theta.
```

```
p = inputParser();
addParameter(p, 'RhoResolution', 1);
addParameter(p, 'Theta', linspace(-90, 89, 180));
parse(p, varargin{:});

rhoStep = p.Results.RhoResolution;
theta = p.Results.Theta;

dMax = sqrt((size(BW,1) - 1) ^ 2 + (size(BW,2) - 1) ^ 2);
numRho = 2 * (ceil(dMax / rhoStep)) + 1;
diagonal = rhoStep * ceil(dMax / rhoStep);% rho ranges from -diagonal to
    diagonal
numTheta = length(theta);
H = zeros(numRho, numTheta);
rho = -diagonal :rhoStep: diagonal;

%    for i = 1 : size(BW,1)
%        for j = 1 : size(BW,2)
%            if (BW(i, j))
%                for k = 1 : numTheta
%                    temp = j * cos(theta(k) * pi / 180) + i *
    sin(theta(k) * pi / 180);
%                    rowIndex = round((temp + diagonal) / rhoStep) + 1;
%                    H(rowIndex, k) = H(rowIndex, k) + 1;
%                end
%            end
%        end
%    end

for i = 1:size(cluster,1)
    for k = 1:numTheta
        temp =
            cluster(i,2)*cos(theta(k)*pi/180)+cluster(i,1)*sin(theta(k)*pi/180);
        rowIndex = round((temp + diagonal) / rhoStep) + 1;
        H(rowIndex, k) = H(rowIndex, k) + 1;
    end
end
end

function hough_lines_draw(img, peaks, rho, theta)
```

```
% Draw lines found in an image using Hough transform.
%
% img: Image on top of which to draw lines
% peaks: Qx2 matrix containing row, column indices of the Q peaks found
    in accumulator
% rho: Vector of rho values, in pixels
% theta: Vector of theta values, in degrees

figure();
imshow(img);
hold on;
for i = 1 : size(peaks,1)
   rho_i = rho(peaks(i,1));
   theta_i = theta(peaks(i,2)) * pi / 180;
   if theta_i == 0
       x1 = rho_i;
       x2 = rho_i;
       if rho_i > 0
           y1 = 1;
           y2 = size(img,1);
           plot([x1,x2],[y1,y2],'g','LineWidth',3);
       end
   else
       x1 = 1;
       x2 = size(img, 2);
       y1 = (rho_i - x1 * cos(theta_i)) / sin(theta_i);
       y2 = (rho_i - x2 * cos(theta_i)) / sin(theta_i);
       plot([x1,x2],[y1,y2],'g','LineWidth',3);
   end
end

function peaks = hough_peaks(H, varargin)
% Find peaks in a Hough accumulator array.
%
% Threshold (optional): Threshold at which values of H are considered to
    be peaks
% NHoodSize (optional): Size of the suppression neighborhood, [M N]

p = inputParser;
addOptional(p, 'numpeaks', 1, @isnumeric);
addParameter(p, 'Threshold', 0.5*max(H(:)));
```

```
    addParameter(p, 'NHoodSize', floor(size(H) / 100.0) * 2 + 1); % odd
        values >= size(H)/50
    parse(p, varargin{:});

    numpeaks = p.Results.numpeaks;
    threshold = p.Results.Threshold;
    nHoodSize = p.Results.NHoodSize;

    peaks = zeros(numpeaks, 2);
    num = 0;
    while(num < numpeaks)
        maxH = max(H(:));
        if (maxH >= threshold)
            num = num + 1;
            [r,c] = find(H == maxH);
            peaks(num,:) = [r(1),c(1)];
            rStart = max(1, r - (nHoodSize(1) - 1) / 2);
            rEnd = min(size(H,1), r + (nHoodSize(1) - 1) / 2);
            cStart = max(1, c - (nHoodSize(2) - 1) / 2);
            cEnd = min(size(H,2), c + (nHoodSize(2) - 1) / 2);
            H(rStart:rEnd,cStart:cEnd) = 0;
        else
            break;
        end
    end
    peaks = peaks(1:num, :);
end
end
```

3.1.2 find chains

```
function [chains, binary_image] = find_chains(chains, binary_image,
    image_width, image_height, min_size)
half_width = 0.5*image_width;
half_height = 0.5*image_height;
chains = {};
for y = 2:image_height-1
    for x = 2:image_width-1
        chain = [];
        [chain,binary_image] = linking_procedure(chain, binary_image,
```

```
            image_width, image_height, x, y, half_width, half_height);
        if size(chain,1) >= min_size
            chains = [chains,chain];
        end
    end
end
end

function [flag, x_seed, y_seed] = next(x_seed, y_seed, binary_image,
    image_width, image_height)
X_OFFSET = [0, 1, 0, -1, 1, -1, -1, 1];
Y_OFFSET = [1, 0, -1, 0, 1, 1, -1, -1];
for i = 1:8
    x = x_seed + X_OFFSET(i);
    if(1 <= x && x <= image_width)
        y = y_seed + Y_OFFSET(i);
        if(1 <= y && y <= image_height)
            if(binary_image(y,x))
                x_seed = x;
                y_seed = y;
                flag = true;
                return;
            end
        end
    end
end
flag = false;
end

function [chain, binary_image] = linking_procedure(chain, binary_image,
    image_width, ...
    image_height, x_ref, y_ref, half_width, half_height)
x = x_ref;
y = y_ref;
[flag,x,y] = next(x,y,binary_image,image_width,image_height);
while flag == true
    chain = [chain; x y x-half_width y-half_height];
    binary_image(y,x) = 0;
    [flag,x,y] = next(x,y,binary_image,image_width,image_height);
end
chain = flip(chain);
```

```
x = x_ref;
y = y_ref;
[flag,x,y] = next(x,y,binary_image,image_width,image_height);
while flag == true
    chain = [chain; x y x-half_width y-half_height];
    binary_image(y,x) = 0;
    [flag,x,y] = next(x,y,binary_image,image_width,image_height);
end
end
```

3.1.3 find clusters

```
function [clusters,chains] =
    find_clusters(clusters,chains,min_deviation,min_size)
clusters={};
for i = 1:size(chains,2)
    chain = chains{i};
    [ratio,clusters,chain]=subdivision_procedure(clusters, chain, 1,
    size(chain,1), min_deviation, min_size);
end
end

function
    [ratio,clusters,chain]=subdivision_procedure(clusters,chain,first_index,
last_index,min_deviation,min_size)
clusters_count = size(clusters,2);
% chain.emplace_back(x, y, x - half_width, y - half_height);
x = chain(first_index,3)-chain(last_index,3);
y = chain(first_index,4)-chain(last_index,4);
length = sqrt(x*x+y*y);
max_pixel_index = 0;
deviation = -1;
max_deviation = -1;
i = first_index;
count = size(chain,1);
while i~=0
    deviation = abs((chain(i,3)-chain(first_index,3))*(chain(first_index,4)-
    chain(last_index,4))+(chain(i,4)-chain(first_index,4))*
    (chain(first_index,3)-chain(last_index,3)));
    if(deviation>max_deviation)
```

```
        max_pixel_index = i;
        max_deviation = deviation;
    end
    i = mod(i+1,count);
end
max_deviation = max_deviation/length;
ratio = length/max(max_deviation,min_deviation);
if ((max_pixel_index-first_index+1)>=min_size &&
    last_index-max_pixel_index+1>=min_size)
    [ratio1,clusters,chain] = subdivision_procedure(clusters, chain,
        first_index, max_pixel_index, min_deviation, min_size);
    [ratio2,clusters,chain] = subdivision_procedure(clusters, chain,
        max_pixel_index, last_index, min_deviation, min_size);
    if(ratio1>ratio || ratio2>ratio)
        ratio = max(ratio1,ratio2);
        return;
    end
end
clusters = clusters(1:clusters_count);
clusters = [clusters,chain(first_index:last_index,:)];
end
```

3.2 基于核函数的 Hough 变换

源代码使用 cpp 和 OpenCv 编写, 最后通过 Mex 文件创建 Matlab 接口, 源代码使用 Cmake 编译, 并且已上传至学习通. 下面是在 matlab 中调用已经编译好的函数 kht 的例子

```
    close all
clear
clc
% Load input image.
im = imread('im02.jpg');
% gaussianFilter = fspecial('gaussian',20, 10);
% im = imfilter(im, gaussianFilter,'symmetric');
tic
[height,width,~] = size(im);

bw = uint8(edge(rgb2gray(im),'canny'));
figure,imshow(mat2gray(bw))
```

```
% Call the kernel-base Hough transform function.
[lines,H] = kht(bw);

% Show current image and its most relevant detected lines.
warning('off','Images:initSize:adjustingMag')
figure,imshow(im)
warning('on','Images:initSize:adjustingMag')

hold on

relevant_lines = 1;
% Convert from KHT to MATLAB's coordinate system conventions.
% The KHT implementation assumes row-major memory alignment for
% images. Also, it assumes that the origin of the image coordinate
% system is at the center of the image, with the x-axis growing to
% the right and the y-axis growing down.
if sind(lines(relevant_lines,2)) ~= 0
    y = [-height/2,height/2-1];
    x = (lines(relevant_lines,1)-y*cosd(lines(relevant_lines,2)))
    /sind(lines(relevant_lines,2));
else
    y = [lines(relevant_lines,1),lines(relevant_lines,1)];
    x = [-width/2,width/2-1];
end
x = x+width/2+1;
y = y+height/2+1;
patch(x,y,[1,1,0],'EdgeColor','y','LineWidth',0.5);
toc
hold off
figure,imshow(imrotate(im,-lines(relevant_lines,2)));
figure,imshow(imadjust(mat2gray(H)))
x = 1:size(H,1);
y = 1:size(H,2);
[X,Y] = meshgrid(y,x);
figure,mesh(X,Y,H);
```