摘 要

数字图像中，物体的几何形状是图像信息的重要组成部分。直线、圆和椭圆是组成常见物体的基本图形。本文研究的对其进行检测识别在图像分析中具有重要的意义。

本文主要研究了数字图像的边缘处理技术，着重介绍了 Hough 变换原理和以其为基础的直线、圆、椭圆检测算法。实现了直线检测算法和以其为基础的多边形检测，研究了一种基于弦中点 Hough 变换拟合的算法。对各个算法进行 MATLAB 编程实现并模拟实验，对算法进行分析研究。

关键词：边缘检测，直线检测，圆检测，椭圆检测，Hough 变换

Abstract

The shape of objects is an important part of information in digital image. Line, circle and ellipse are basal shapes of normal objects. Detection of them researched in this dissertation is important in image analysis.

In this thesis, edge detections of digital image are discussed, and the theory of Hough transform is illustrated in detail. Some detecting algorithms of lines, circles and ellipses are compared and analyzed. Realized a detection of lines and a recognition algorithm of conventional polygons which are based on detected lines. A chord-middle-point Hough transform fitting method of ellipses are researched. These algorithms are realized and analyzed with MATALB codes and tested results.

Keywords: Edge Detection, Lines Detection, Circles Detection, Ellipses Detection, Hough Transform

目 录

第一章 引言

1.1 研究背景及意义

随着计算机科学和电子技术的发展，数字图像处理作为多学科的融合结晶，诞生于 20 世纪 60 年代，并逐步发展起来。进入 21 世纪，计算机的普及和处理器性能的大幅提高使得数字图像处理成为图像处理的常见方法之一。目前数字图像处理广泛应用于人脸识别、人工智能、卫星成像等领域。

图像分析是数字图像处理的重要部分。图像分析主要从数字图像中提取有意义的信息。一般利用数学模型并结合图像处理的技术来分析图像的底层特征和上层结构，从而提取具有一定智能性的信息。在图像分析中物体的形状是比较高级的信息。图像的主要特征有灰度、纹理和形状，可以表示图像的亮度、颜色变化和几何属性。形状特征具有对空间、旋转、伸缩以及运动的不变形，满足计算机视觉技术对物体识的基本要求。几何形状是含信息量较高的图像特征。

直线、圆和椭圆作为最基本的几何形状，是自然界和人造物体的基础构成。在数字图像中对其进行有效和准确检测识别具有重要意义。对直线、圆和椭圆进行变换组合进而来识别完整的几何形状。在图像分割[1],工程图纸矢量化[2], 非接触式检测[3],计算机辅助设计[4]，物体定位等方面有着广泛地应用。

1.2 研究现状及发展

在计算机中目标的形状是通过目标的轮廓或者目标轮廓所包围的区域来获取的。基于不同的理解，已经提出了许多形状识别方法，如：Hough 变换、链码技术、不变矩、傅立叶描述自回归模型等，这些方法在某些特定环境下能成功识别对象，但都存在一些局限性。

由 Paul Hough 在 1962 年提出的 Hough 变换[5]，是目前一种应用非常广泛的图形识别检测技术。Hough 变换实现了一种从图像空间到参数空间的映射关系。用 Hough 变换方法检测直线对噪声具有较强的抗干扰能力，不受直线间断、局部遮挡等缺陷的影响；但是，传统的 Hough 变换也存在缺陷：如计算速度慢、消耗内存大。对圆直接进行 Hough 变换将会产生一个三维的参数空间，消耗的计算时间和

二维相比增大了数百倍乃至数千倍。对椭圆则需要 5 个参数，这对资源的消耗将是不可估计的。因而目前已经有许多学者提出基于 Hough 变换的改进型算法。对于直线检测，Shapiro 在文章[6]中说明了标准变换的投票过程对其检测精度的影响,并给出了一个新的变换算法,精度超越了标准变换,达到了连续 Hough 变换的水平;Lo 和 Tsai[7]结合了 Hough 变换和 Radon 变换提出了一种新的直线检测方法。对于圆检测,多数方法采用分步 Hough 变换,先找出圆心,然后再对半径进行投票。Chiu[8]为标准变换给出了一个新的投票方法,让每一个像素只对应一个候选圆参数,大大改善了计算复杂度,有效地减少了存储空间。Ramirez[9]等人使用遗传算法，能够在图形中检测出残缺的圆,并且检测结果的精确度能达到亚像素级别。

链码通过一个指定长度与方向的直线段的连接顺序来表示一条边界。典型情况下，这一表示是建立在线段的 4 连接或者 8 连接之上。每条线段的方向通过一个编号加以编码。最早由 Freeman[10]在 1961 年提出来的链码来描述线条的模式，至今仍然是一种被广泛使用的最主要的编码方式。在图像处理和识别中广泛被采用。链码也存在一些局限性，如确定链码的起始点坐标一个复杂的过程。对于同一个边界采用不同的边界点作为链码起始点，将得到不同的链码。另外，噪声干扰会导致边界变化，会导致链码发生与目标整体形状无关的较大变化。为了克服链码描述带噪声点直线的困难，采用平均链码描述直线，效果较好。并且不断有学者提出新的链码编码方式，如 Bribiesca[11]在 1992 年就提出了对 Freeman 链码的修改链码，并且在 1999 年有提出了一个新的链码编码方式来表示区域形状，称其为“顶点链码”。该链码基于他与 Guzman[12]在 1980 年提出的“形状数”的概念。

1962 年，由 M.K.Hu[13]提出的不变矩，具有平移、旋转和尺度变化的不变性，同时又可有效地完成从采样空间到模式空间的数据压缩，但计算复杂； Zahn[14]首先使用傅立叶描述子来描述和识别物体的形状特征，具有简单、高效的特点。这些方法也具有一定的适用性和局限性。

经过多年的研究和发展，各种检测算法已经日趋完善和成熟，并建立起一套完整的理论体系和实现方法，但其中仍然存在一些问题。如上面提到的 Hough 变换在计算量大和消耗内存，链码边界起点的确定计算复杂。因此有必要对已有算法进行进一步研究改进或者提出全新的算法。

1.3 本文结构

第一章，介绍几何形状检测算法的研究背景及意义，各类检测算法的国内外发

展及研究现状。

第二章，介绍数字图像处理的基础知识，各个边缘检测算子和彩色图像的边缘检测技术，并通过实验选取合适的边缘处理方法。

第三章，介绍 Hough 变换的基本原理，对比分析目前主流的直、圆和椭圆的检测算法。

第四章，对直线、多边形、椭圆检测算法进行编程实现和实验对比分析。进行图片分割检测和合并实现和分析。

第五章，对全文进行总结，并对后续相关工作进行探讨

第二章 数字图像的边缘检测技术

2.1 数字图像处理基础

2.1.1 图像数字化

图像的数字化是对一幅图像依照一定顺序，比如行优先顺序进行扫描，生成一个与图像一一对应的二维整数矩阵，扫描顺序决定了矩阵中每一个像素的位置，采样又决定了每一个像素的灰度值，再经过量化得到每一个像素灰度值的整数表示。这样，一幅图像数字化后所得到的最终结果就是一个二维的整数矩阵，即数字图像。彩色图像是由单个二维图像组合的。例如，在 RGB 彩色系统中，一副彩色图像是由三个独立的分量图像（红、绿、蓝）组成的。取样和量化的结果是一个实数矩阵。假设一副图像 $f\left(x,y\right)$ 取样后，得到一副有着 M 行 N 列的图像。则称这幅图像的大小为 $M\times N$ 。坐标 $\left(x,y\right)$ 的值是离散量，同时必须是整数。图像的坐标定义为 $\left(x,y\right)=\left(0,0\right)$ 处。但需要注意的是在后续的 MATLAB 编程实现中采取的坐标原点则是 $\left(x,y\right)=\left(1,1\right)$ 处。

基于上面的讨论，对数字化图像的表示：

$$f\left(x,y\right)=\begin{bmatrix} f\left(0,0\right) & f\left(0,1\right) & \text{L} & f\left(0,N-1\right) \\ f\left(1,0\right) & f\left(1,1\right) & \text{L} & f\left(1,N-1\right) \\ \text{M} & \text{M} & \text{O} & \text{M} \\ f\left(M-1,0\right) & f\left(M-1,1\right) & \text{L} & f\left(M-1,N-1\right) \end{bmatrix} \tag{2-1}$$

等式右边是由定义给出的一副数字图像。该数组的每一个元素都称为像元或像素。

2.1.2 图像类型

MATLAB 图像处理工具箱支持以下 4 种图像类型：亮度图像(Intensity image)、二值图像(Binary image)、索引图像(Indexed image)和 RGB 图像(RGB image)。大多数单色图像的处理运算是通过二值图像和亮度图像来进行的。

亮度图像：一副亮度是一个数据矩阵，其归一化的取值表示亮度。

二值图像：一副二值图像是一个取值只有 0 和 1 的逻辑数组。

RGB 图像：一副 RGB 图像就是彩色像素的一个$M \times N \times 3$数组，其中每一个彩色像素点都是在特定空间位置的彩色图像相对应的红、绿、蓝三个分量。RGB 也可以看成是有一个三幅灰度图像形成的“堆”，当将其送到彩色监视器的红、绿、蓝输入端时，便在屏幕上产生了一副彩色图像。

索引图像：直接把像素值直接作为 RGB 调色板下标的图像。也可把像素值直接映射为调色板数值。索引图像有两个分量，即整数的数据矩阵 X 和彩色映射矩阵 map。矩阵 map 是一个大小为$m \times 3$且范围在$[0,1]$之间的浮点数构成的数组。map 的长度 m 与它所定义的颜色数目相等。

2.2 边缘检测

2.2.1 边缘检测要求

为了检测数字图像中的直线、圆和椭圆等几何形状,我们首先要从数字图像中提取出有用且可靠的对象的边缘信息。通常分为两个步骤,首先使用边缘检测算子获得边缘信息；然后加人后期处理,移除边缘图片上多余的点。为有效地检测出数字图像中这些几何形状,我们选用的边缘检测算子得到的边缘信息要有足够的数量与较好的质量。Canny[15]指出一个好的边缘检测算子需要满足以下三条准则：

- 好的检测：算法能要尽可能多地标识出图像中的实际边缘。
- 好的定位：标识出的边缘要尽可能与图像中的实际边缘接近。
- 最小响应：图像中的所有边缘只标识一次,并且可能存在的图像噪声不应该被标识为边缘。

但运用边缘检测算子来完成基本几何形状检测的时候,此标准就变得有些不适合。比如,图像中出现的一些纹理上的细节边缘部分,不但对我们的检测没有,而且会参数更多无意义甚至干扰运算。因此需要的边缘检测算子要满足以下四点要求：

- 适合检测物体轮廓
- 唯一的边缘响应
- 尽可能少的控制参数
- 高的运算效率

第一条确保检测出来的边缘是对物体实际边界的响应,而非图像纹理边界中的细节性的边界（不包括不同纹理之间的边界）,这有利于减少算法的计算量。第二条保

证获得的边缘是唯一的,这有利于我们减少后续检测的计算量和保持好的准确率。第三条是要让边缘检测算子能适应不同质量图片的边缘检测要求,这也是后面直线、圆、椭圆检测算法能否在实际环境下可靠地运行的最基本要求。第四条则是出于对整体的考虑,后续的检测算法需要较大计算量,在满足一定质量的前提下,尽可能的使用复杂度小的检测算子显然是人们追求的目标。

2.2.2 边缘检测算子

一幅数字图像的一阶导数是基于各种二维梯度的近似值。图像 $f(x,y)$ $f(x,y)$ 在位置 (x,y) 的梯度定义为下列向量：

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix} \tag{2-2}$$

该向量的幅值为

$$|\nabla f| = \left|G_x^2 + G_y^2\right|^{1/2} = \left|\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right|^{1/2} \tag{2-3}$$

为简化计算，该数值有时也省略掉平方根的计算来近似，即

$$|\nabla f| \approx G_x^2 + G_y^2 \tag{2-4}$$

或者取绝对值来近似，即

$$|\nabla f| \approx |G_x| + |G_y| \tag{2-5}$$

这些近似值仍然具有导数性质，在不变亮度区中的值为零，而他们的值与像素值可变区域的亮度变化成比例。在实际中，通常将梯度的幅值或它的近似值称为“梯度”。

梯度向量的基本性质是它指向 f 在 (x,y) 处的最大变化率方向，最大变化率出现时的角度为

$$\alpha(x,y) = \arctan\left(\frac{G_y}{G_x}\right) \tag{2-6}$$

在图像处理中，二阶导数通常由拉普拉斯算子来计算。二维函数 $f(x,y)$ 的拉普拉斯由二阶导数形成，形式如下：

$$\nabla^2 f(x,y)=\frac{\partial^2 f(x,y)}{\partial x^2}+\frac{\partial^2 f(x,y)}{\partial y^2} \tag{2-7}$$

普拉斯算子自身很少直接用来做边缘检测，因为二阶导数对噪声的敏感性，它的幅度会产生双边缘，且它不能检测出边缘的方向。然而，当与其他边缘检测技术组合使用时，拉普拉斯算子是一种有效的补充方法。

边缘检测的基本意图是使用任意一个如下两个基本准则在图像中找到亮度快速变化的地方：

- 找到亮度的一阶导数在幅度上比指定的阈值大的地方。
- 找到亮度的二阶导数有零交叉的地方。

2.2.2.1 Sobel 检测算子

图像的领域表示如图 2-1。Sobel 边缘检测算子[16]使用图 2-2 中的掩模来数字化地近似一阶导数 G_x 和 G_y。一个邻域的中心点处的梯度可由 Sobel 检测器按如下方式计算：

$$\begin{aligned} g=\left|G_x^2+G_y^2\right|^{1/2}=\Big\{&\left[\left(z_7+2z_8+z_9\right)-\left(z_1+2z_2+z_3\right)\right]^2 \\ &+\left[\left(z_3+2z_6+z_9\right)-\left(z_1+2z_4+z_3\right)\right]^2\Big\} \end{aligned} \tag{2-8}$$

因此，我们指定一个阈值 T，若在位置 (x,y) 处 $g\geq T$，则在该位置的一个像素是一个边缘像素。

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

图 2-1 图像领域

-1	-2	-1
0	0	0
1	2	1

(a)

-1	0	1
-2	0	2
-1	0	1

(b)

图 2-2 Sobel 掩模

(a)垂直方向；(b)水平方向

2.2.2.2 Prewitt 检测算子

Prewitt 边缘检测算子使用图 2-3 中的掩模来数字化地近似一阶导数G_x和G_y。一个邻域的中心点处的梯度可由 Prewitt 检测器按如下方式计算：

$$g=\left|G_x^2+G_y^2\right|^{1/2}=\left\{\left[\left(z_7+z_8+z_9\right)-\left(z_1+z_2+z_3\right)\right]^2+\left[\left(z_3+z_6+z_9\right)-\left(z_1+z_4+z_3\right)\right]^2\right\} \tag{2-9}$$

-1	0	1
-1	0	1
-1	0	1

(a)

-1	-1	1
0	0	0
1	1	1

(b)

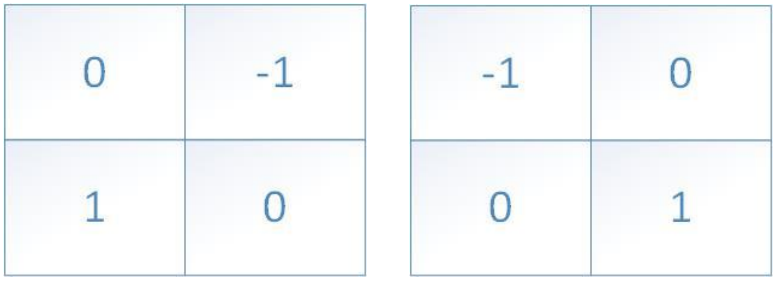
图 2-3 Prewitt 掩模

(a)垂直方向；(b)水平方向

2.2.2.3 Robert 检测算子

Robert 边缘检测算子使用图 2-3 中的掩模来数字化地近似一阶导数G_x和G_y。一个邻域的中心点处的梯度可由 Robert 检测器按如下方式计算：

$$g=\left|G_x^2+G_y^2\right|^{1/2}=\left[\left(z_9-z_5\right)^2-\left(z_8-z_6\right)^2\right]^{1/2} \tag{2-10}$$



(a) (b)

图 2-4 Robert 掩模

(a)垂直方向；(b)水平方向

2.2.2.4 LoG 检测器

考虑高斯函数

$$h(r)=-e^{-\frac{r^2}{2\sigma^2}} \tag{2-11}$$

其中，$r^2=x^2+y^2$，σ 是标准偏差。这是一个平滑函数，若和一幅图像卷积，则会使图像变模糊。模糊的程度由 σ 的值决定。该函数的拉普拉斯算子（关于 r 的二阶导数）为

$$\nabla^2 h(r)=-\left[\frac{r^2-\sigma^2}{\sigma^4}\right]e^{-\frac{r^2}{2\sigma^2}} \tag{2-12}$$

该函数称为 Laplacian of a Gaussian(LoG)。因为求二阶导数是线性运算，所以用 $\nabla^2 h(r)$ 对图像进行卷积（滤波）与先用平滑函数对图像卷积再计算结果的拉普拉斯算子是一样的。这是 LoG 检测器的最关键概念。用 $\nabla^2 h(r)$ 对图像卷积会产生两个效果:使图像变平滑（从而减少噪声）；计算拉普拉斯算子，以便产生双边缘图像。然后，通过定位边缘就是找到两个边缘之间的零交叉。

2.2.2.5 Canny 检测算子

Canny 检测算子是最有效的边缘检测器。其结合了上面介绍的几个算子的思想。其实现主要步骤为：

1、图像使用带有指定标准偏差 σ 的高斯滤波器来平滑，从而可以减少噪声。

2、在每一点计算局部梯度 $g(x,y)=\left|G_x^2+G_y^2\right|^{1/2}$ 和边缘方向 $\alpha(x,y)=\arctan\left(G_y/G_x\right)$。Sobel 掩模、Prewitt 掩模和 Robert 掩模都可用来计算 G_x

和G_y。边缘点定义为梯度方向上其强度局部最大的点。

3、第 2 条中确定的边缘点会导致梯度幅度图像中出现脊。然后，算法追踪所有脊的顶部，并将所有不在脊的顶部的像素设为零，以便在输出中给出一条细线，这就是非最大值抑制处理。对脊像素使用两个阈值 Tl 和 T2 来处理，其中$T1 < T2$。值大于 T2 的脊像素称为强边缘像素，Tl 和 T2 之间的脊像素称为弱边缘像素。

4、最后一步，算法通过将 8 连接的弱像素集成到强像素，执行边缘链接。

Sobel 算子、Prewitt 算子和 Robert 算子检测结果丢失了一些边缘细节，但能很好保留较明显的边缘，LoG 和 Canny 算子保留了大部分边缘细节，不同的是 LoG 算子由于不能判断边缘的方向信息而出现了很多不连续的边缘，而 Canny 算子所提取的边缘轮廓比较完整，连续性也较好。

2.2.3 彩色图像的边缘检测

目前通过数码设备拍摄的图片一般均为彩色 RBG 图像。对于一副 RGB 图像，首先要将其转换为通过边缘检测的灰度图像，然后进行其他处理用于后续检测算法。

由于 RGB 图像具有三通道，为将其转换为灰度边缘图像，有两种方案：

•直接将三通道的图像按照一定的比例融合为一副灰度图像，然后再运用上文提到的边缘检测算子进行边缘检测。

•分别对三通道进行边缘检测，然后再综合已有边缘信息，得到一副具有边缘信息的灰度图。

第一个方案中，直接将三通道图像矩阵信息变成了一个，实际上将数据量大大的减少了，再进行边缘检测得到的信息量将严重不足。反之，第二个方案，就充分利用了三通道的信息量，能得到较好的结果。下面将较为详细的介绍第二种方案。

然而，在前面 2.2.2 节推导的梯度等公式，并不能直接扩展到高维空间。令 c 代表 RGB 彩色空间的任意向量：

$$c = \begin{bmatrix} c_R \\ c_G \\ c_B \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2-13}$$

该公式表明 c 的分量是一副彩色图像在一个点上的 RGB 分量。彩色分量是坐标(x, y)的函数，表示为

$$c(x,y)=\begin{bmatrix}c_R(x,y)\\c_G(x,y)\\c_B(x,y)\end{bmatrix}=\begin{bmatrix}R(x,y)\\G(x,y)\\B(x,y)\end{bmatrix} \tag{2-14}$$

对于一个大小为 $M\times N$ 的图像来说，有 MN 个这样的向量 $c(x,y)$，其中，$x=0,1,2,\mathrm{L}\ ,M-1$ 和 $y=0,1,2,\mathrm{L}\ ,N-1$。下面，为了得到的向量 c 的梯度，需要把梯度的概念扩展到向量函数。

令 r, g 和 b 是 RGB 彩色空间沿 R, G 和 B 的单位向量。并定义向量

$$u=\frac{\partial R}{\partial x}r+\frac{\partial G}{\partial x}g+\frac{\partial B}{\partial x}b \tag{2-15}$$

和

$$v=\frac{\partial R}{\partial y}r+\frac{\partial G}{\partial y}g+\frac{\partial B}{\partial y}b \tag{2-16}$$

令 g_{xx}，g_{yy} 和 g_{xy} 是这些向量的点积，如下所示：

$$g_{xx}=u^Tu=\left|\frac{\partial R}{\partial x}\right|^2+\left|\frac{\partial G}{\partial x}\right|^2+\left|\frac{\partial B}{\partial x}\right|^2 \tag{2-17}$$

$$g_{yy}=v^Tv=\left|\frac{\partial R}{\partial y}\right|^2+\left|\frac{\partial G}{\partial y}\right|^2+\left|\frac{\partial B}{\partial y}\right|^2 \tag{2-18}$$

$$g_{xy}=u^Tv=\frac{\partial R}{\partial x}\frac{\partial R}{\partial y}+\frac{\partial G}{\partial x}\frac{\partial G}{\partial y}+\frac{\partial B}{\partial x}\frac{\partial B}{\partial y} \tag{2-19}$$

需要注意的是，以上 R，G 和 B 及 g 项均是 x 和 y 的函数。函数 $c(x,y)$ 的最大变化率的方向由角度：

$$\theta(x,y)=\frac{1}{2}\arctan\left(\frac{2g_{xy}}{g_{xx}-g_{yy}}\right) \tag{2-20}$$

给出，并且变化率的值[17]在由 $\theta(x,y)$ 的元素给出的方向上由下式给出：

$$F_\theta(x,y)=\left\{\frac{1}{2}\left[(g_{xx}+g_{yy})+(g_{xx}-g_{yy})\cos 2\theta+2g_{xy}\sin 2\theta\right]\right\}^{1/2} \tag{2-21}$$

$\theta(x,y)$和$F_\theta(x,y)$是与输入图像大小相同的图像。$\theta(x,y)$的元素是计算梯度后每点的角度，$F_\theta(x,y)$是梯度图像。

由于正切函数是周期为π的周期函数，若θ_0是上面方程arctan的一个解，则$\theta_0 \pm \pi/2$也是该方程的一个解。 此外，由于$F_\theta(x,y)=F_{\theta+\pi}(x,y)$, 所以 F 只要在半开区间$[0,\pi)$上计算$\theta$的值。arctan方程提供两个相隔 90° 的值意味着该方程与两个正交方向上的每一个点(x, y)相关。沿着这两个方向之一 F 最大，而沿着另 一个方向 F 最小，最终结果是由选择每个点上的最大值产生的。要完成上面的计算，需要计算偏导数，可以运用 Sobel 算子来完成。

下面，运用彩色图 2-5 来进行边缘检测实验。图 2-5 的大小为 2048×1152，是运用手机拍摄的。

图 2-5 纸板

图 2-6 纸板的灰度图像

图 2-6 是对图 2-5 运用 MATLAB 函数 rgb2gray 得到的结果，图 2-7 是对 2-6 调用 MATLAB 函数 edge 运用 Canny 算子得到的结果。图 2-8 是运用冈萨雷斯[17]编写的 colorgrad 函数得到的结果。为了便于观察对两者都运用了 MATLAB 函数 imcomplement 对图片进行取补。

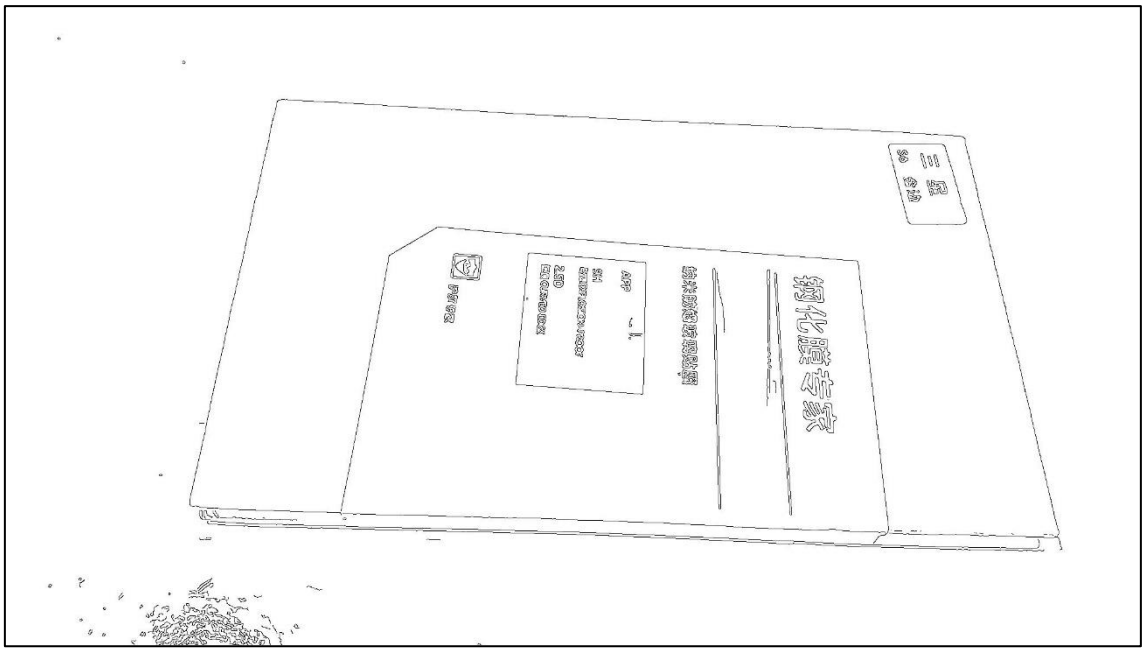


图 2-7 灰度图像的 Canny 算子检测的边缘图像

观察图 2-6 可以发现纸板的灰度图像丢失了很多彩色图像 2-5 的细节。在运用其为基础检测出来的边缘图像与图 2-8 相比，虽然边缘轮廓较粗，但是却没有 2-8 光滑，轮廓周围产生了较多干扰点，这对后续的识别是极其不利的；同时注意原图左下角较强的“光团”，2-8 采用的方法对其处理明显优于 2-7；从整体清晰度来看，2-8 要比 2-7 好很多。综上所述，对彩色图像的边缘检测采用上面详细讨论的方法是合理有效的。在后续的几何形状检测过程中都将采用这种方法来获取边缘图像。

图 2-8 纸板三通道融合检测的边缘图像

2.3 本章小结

本章，先介绍了一些数字图像处理的基本概念，如：图像的数字化即矩阵表示；数字图片的常见类型，灰度图像、二值图像、RGB 图像和索引图像，要完成完整的直线、圆、椭圆和其他基本几何形状的检测，需要多次进行图像类型的转换。为了获得边缘图像，我们需要明确边缘检测的基本要求和方法。介绍了几个常见的边缘检测算子，各个算子虽然各有特点，但总的来说结合了几个算子的优点的 Canny 算子较为优秀。虽然没有直接采取介绍的几种算子来进行边缘检测，而是选取了对于 RGB 图像更合适的方案。但是介绍的算子及梯度的思路是实现后者的基础。最后通过实验证明：讨论的彩色图像的边缘检测方法是合理有效的，能满足边缘检测的基本要求，能运用于后续的几何形状检测算法。

第三章 几何形状检测算法

3.1 直线检测

3.1.1 Hough 变换

Hough 变换是在图像全局范围内组织数据的方法。其基本策略是：首先根据曲线的解析表达式，将图像空间中的边缘特征信息转换到定义的参数空间；然后搜索参数空间中的峰值点，它在参数空间中的坐标代表了所检测曲线的参数；最后用曲线的参数在图像空间中重建曲线的完整描述。通过在参数空间所累积的结果作为原始空间是否存在该目标的证据。

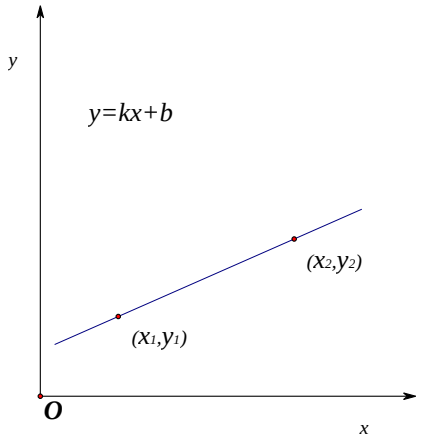


图 3-1 图像空间的直线$y = kx + b$

下面我们通过图像中的直线来解释上述过程。如图 3-1，直线 $y = kx + b$ 由两点 $A = (x_1, y_1)$ 和 $B = (x_2, y_2)$ 构成。通过 A 的所有直线可表示为 $y_1 = kx_1 + b$，其中 k 和 b 是一系列值。这就意味着这个方程可以解释为参数空间 k 和 b 的方程,因此,在参数空间，如图 3-2，通过 A 点的所有直线可表示为 $b = -kx_1 + y_1$。同理可得，通过 B 点的直线可以表示为 $b = -kx_2 + y_2$。可以看到，在参数空间和中两条直线产生了一个交点。这个唯一公共点表示的就是在原图像空间中连接两点的直线。由此可见,图像中的每条直线都可以由参数空间和中一个点来表示。直线的任何一部分都能映射为参数空间中这个点。如果我们对参数空间中的点进行累积,通过寻找峰值就

可以找到图像空间中的直线。

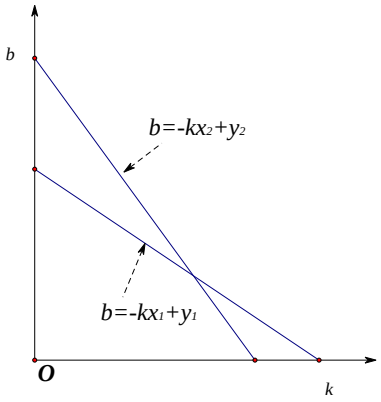


图 3-2 参数空间(k, b)

该方法有一个突出优点：检测分割结果对数据的不完全或噪声不是非常敏感。但是缺点是其较大的计算量。

3.1.2 标准 Hough 变换检测直线(SHT)

如果按照上面介绍的方法来进行图像空间到参数空间的积累，会出现一些问题。例如斜率不存在的直线或接近正无穷时，映射到参数空间都比较麻烦。因此有必要改变直线方程的形式。如图 3-3，采用极坐标的直线方程：

$$\rho = x\cos\theta + y\sin\theta \tag{3-1}$$

其中 ρ 原点到直线的距离，θ 是过原点与直线的垂线与 x 轴的夹角。

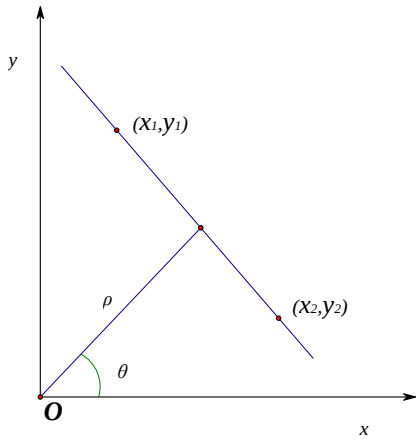


图 3-3 在 xy 平面上直线(ρ, θ)参数化

$\theta \in [-90°, 90°]$。对于水平线来说,θ=90°,ρ 等于正的截距，或者$\theta = -90°$,ρ 等于负的截距。图 3-4 中的两条曲线分别表示过点(x_1,y_1)和(x_2,y_2)的直线簇。交点(ρ_1,θ_1)对应于过点点(x_1,y_1)和(x_2,y_2)的直线。

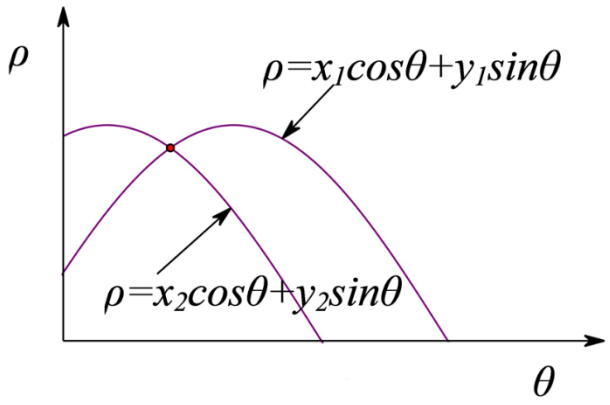


图 3-4 (ρ, θ)空间上的正弦曲线

在计算过程中，需要对参数空间进行离散化，一般来说 $-90° \le \theta \le 90°$，$-D \le \rho \le D$，其中 D 为图像的对角线长度。对于图像空间中每一个点(x,y)，θ按顺序选取累加器空间中允许细分的值，通过公式 3-1 求出相应的 ρ 值（四舍五入到最接近允许的细分值），增加相应累加器单元中，参数空间选出累加器峰值的位置即为可能直线的参数。

3.1.3 概率 Hough 变换检测直线(PHT)

概率 Hough 变换[18]的主要思路：是如果选择的数据点准确,那么定位一条直线就只需要两个点。这表明标准 Hough 变换过程中,有大量的数据是浪费的。选择图像中的一部分足够精确的点进行变换,可以做到不影响最终检测的结果,而且能大大减少检测时间和计算量。

要实现这一算法,问题的关键就在于如何选择这些样本点上。一种方法是通过定义图像的一致概率密度来选择,另外一种更加简便的方法是从另一个角度考虑这个问题。由于图像数据中的冗余量,一部分点的 Hough 变换结果可以代表这幅图像做标准 Hough 变换之后的结果,那么可以认为累加器中的值,只要达到一定阈值之后就认为检测到了一条直线。这样的话,我们就无需考虑到底要选取多少样本点来计算。

检测算法分为 4 个步骤：

(1) 随机选取数据点集合中的数据(x_k, y_k),若此点是没有被检测过的,则对这个点进行 Hough 变换。

(2) 若存在一累加器单元$A(\rho_k, \theta_k) > T$,T 为设定的阈值,则认为检测到了一条直线，输出对应参数,清空累加器单元$A(\rho_k, \theta_k)$

(3) 按照上一步得到的直线参数,在原图中做直线连接,将属于该直线上所有的数据点都标记为已检测。

(4) 检查数据点集合中的未处理过的点的总数,若小于一定值,结束计算；否则返回步骤(1)。

3.1.4 随机 Hough 变换检测直线(RHT)

不同于传统的 Hough 变换，随机 Hough 变换[19]采用多对一的映射机制，避免了一到多产生的庞大计算量，而且随机算法采用动态链表存储结构，只对多对一映射所得到的参数分配单元进行累积，大大降低了对内存资源的需求，同时使得该算法具有参数空间大、精度高的优点。但是在处理更复杂的图像时，会因为随机采样而引入大量无效采样和累积计算量，使得算法的性能大大降低。

RHT 算法的核心在于：图像空间随机选取的两个或者多个像素点来确定参数空间的一个点；在检测多条直线过程中，通过随机采样两点进行投票累积，如果超过一定阈值，检测到一条直线后，就将这条直线上的点从原图像点集中去除，避免了传统 Hough 变换一到多映射的庞大计算量。算法可以简述为：

(1) 生成图像的边界点集 S, 并初始化参数空间 P(用于存储直线的两个参数和相应累加值）。

(2) 如果边界点集 S 中点数少于某一阈值，结束计算，否则，从边界点集 S 中随机选取点(x_1,y_1)和(x_2,y_2)，两点的距离必须大于某个确定的最小距离条件 d，否则重新选取。

(3) 计算两点所确定的直线参数$p(\rho,\theta)$,在 P 中寻找$p_c(\rho_c,\theta_c)$，使得$\|p - p_c\| < \delta$，δ是允许的误差。增加参数空间p_c的累加器，进入第 4 步。如果不存在p_c，则将$p(\rho,\theta)$添加进 P 返回第 2 步。

(4) 若p_c的累加器大于阈值T_c。则认为检测到直线。输出对应的参数。将对应直线上的点从 S 中清除，并重置 P; 否则，返回第 2 步。

3.2 圆检测

3.2.1 Hough 变换检测圆

最早由 Duda 和 Hart 提出用 Hough 变换的方法来检测圆。公式 3-2 描述了，一个中心为(x_c, y_c)，半径为 r 的圆。

$$(x-x_c)^2+(y-y_c)^2=r^2 \tag{3-2}$$

基本的圆变换公式是在三维参数空间坐标(a, b, r)进行累加并通过统计来完成检测任务。图像空间中利用式 3-2 每变换一个点，对应参数空间就出现一个圆锥面，所有边缘点变换后，一簇圆锥面相交于同一点，然后累计相交于同一点的次数，超过一个设定阈值，就得到检测到的圆参数。

该方法可靠性高，在噪声、变形、部分区域残缺的状态下也能取得比较理想的结果，但计算量大，内存需求大。一般圆检测的算法基本都采用降低检测维数分步检测和减少累加器存储空间等措施。

3.2.2 梯度 Hough 变换检测圆

梯度 Hough 变换[20]，就是通过分步检测来减少计算量和内存需求。算法分为两步：

(1) 二维 Hough 变换检测圆的中心

(2) 一维 Hough 变换,求解对应的半径长度

$$\begin{aligned} x &= x_c + r\cos\theta \\ y &= y_c + r\sin\theta \end{aligned} \tag{3-3}$$

式 3-3 是圆的极坐标表达式。观察该式，如果已知圆心到圆上点的矢量方向那么方程就只有 r 是未知量。化简可得：

$$y_c = x_c\tan\theta - x\tan\theta + y \tag{3-4}$$

原来需要三个参数的圆方程被的维度被降低了,化简为一条直线方程。可以按照前面 Hough 变换直线检测的方法,建立累加器$A(x_c, y_c)$,按照公式 3-4 来进行类 Hough 变换,定位圆的中心坐标。然后初始化累加器$A(r)$，按照公式 3-2 进行累

加。最后找到的峰值就是对应圆的半径。

该方法主要利用了边缘的方向信息，一般边缘检测算子都通过求解出边缘方向来降低维数,继而减少累加器需要的空间。该算法内存空间占用少,但对小圆检测结果不理想,容易出现漏检。

3.2.3 随机圆检测(RCD)

随机圆检测是以随机 Hough 变换的思想为基础发展起来的。

随机圆检测的基本思想为：随机选取四个边缘点，采取其中任意三点拟合一个圆，判断另外一个点是否在该圆上。如果在圆上，再对候选圆进行证据累积来判断是否为真圆。如下图 3-5 与 3-6 分别为采样三点位于不同圆上和同一个上的示意图。

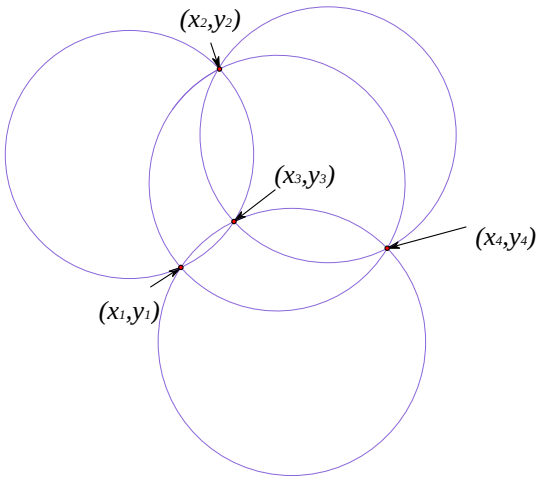


图 3-5 四点不在同一圆

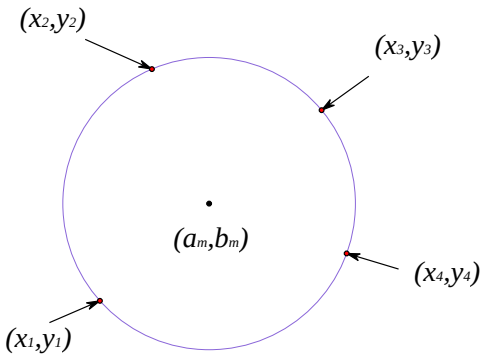


图 3-6 四点在同一圆

若 S 为图像边缘点集，随机采样四点，其中前三点$(x_m, y_m), m=1,2,3$所确定圆的圆心为(a_m, b_m) 半径为r_m，则它们满足

$$(x_m - a_m)^2 + (y_m - b_m)^2 = r_m^2 \tag{3-5}$$

上式可改写为

$$2x_m a_m + 2y_m b_m + d_m = x_m^2 + y_m^2 \tag{3-6}$$

其中，$d_m = r_m^2 - a_m^2 - b_m^2$。非共线的三个边缘点能确定一个圆。为了避免三点中任意两点太靠近而检测不到真实圆，选取点时，任意两点的距离需要大于一定的阈值。将三个点的坐标代入式 3-6 后可以得到三个方程，联立方程可求得圆参数联立三个圆方程求解圆参数的计算量显然是比较大的，可以用圆心在弦的中垂线上的性质来求得圆参数。先求出圆心(a_m, b_m)然后再求解半径r_m。

第四个点(x_4, y_4)到圆的距离为

$$d_4 = \sqrt{(x_4 - a_m)^2 + (y_4 - b_m)^2} - r_m \tag{3-7}$$

理想情况下，如果采样的四点共圆，第四个点到轮廓的距离为零，但是考虑到数字图像量化误差以及计算误差，因此给定一个阈值T_d，如果$d_4 < T_d$则认为采样的四点共圆。

在确定一个候选圆后，对图像边缘点集合 S 中的每个点 p,计算点 p 到候选圆的距离。如果距离小于阈值，则将 p 从点集合 S 中移除，加入一个候选圆点积 C 中，遍历完 S 后，如果 C 的点数大于阈值 Nc， Nc 与候选圆的半径，残缺率成比例。如果 C 的点数小于 Nc，则认为检测为圆的条件不足。将 C 中的点返回到点集 S 中。

相比于 Hough 变换和梯度 Hough 变换检测圆，RCD 所需内存非常少，具有运算速度快和较好抗噪声的优点。但对于 RCD 方法中随机采样四点的方法而言，只有采样到的点在实际的同一圆上时才有效，那么就会产生大量的无效采样。

3.3 椭圆检测

3.3.1 椭圆的表示及参数变换

式 3-8 表示了椭圆的一般方程也称为非标准方程。该方程为二次曲线型。如果

直接进行 Hough 变换，参数空间将会是五维的，累加器的构造，存储和峰值查找将会是极其困难的。因此 Hough 变换椭圆检测的算法基本上采用分步求解的方法，而且大多都运用了椭圆的某些几何性质来求解。

$$ax^2+bxy+cy^2+dx+ey+f=0 \tag{3-8}$$

其中，$b^2-4ac<0$。

上面的二次曲线参数在描述椭圆时，并不够直观。因此需要对其进行参数转换。设椭圆长轴与 X 轴正向的夹角为θ，也称为旋转角度，椭圆的半长轴为 A，半短轴为 B，(x_c,y_c)为椭圆的中心。如图 3-7，直观的表示了各个变量的含义。

$$\theta=\frac{1}{2}\arctan\frac{b}{b-c} \tag{3-9}$$

$$\begin{cases} x_c=\dfrac{be-2cd}{4ac-b^2} \\ y_c=\dfrac{bd-2ae}{4ac-b^2} \end{cases} \tag{3-10}$$

$$\begin{cases} A^2=\dfrac{2\left(ax_c^{\,2}+cy_c^{\,2}+bx_cy_c-f\right)}{a+c-\sqrt{(a-c)^2+b^2}} \\ B^2=\dfrac{2\left(ax_c^{\,2}+cy_c^{\,2}+bx_cy_c-f\right)}{a+c+\sqrt{(a-c)^2+b^2}} \end{cases} \tag{3-11}$$

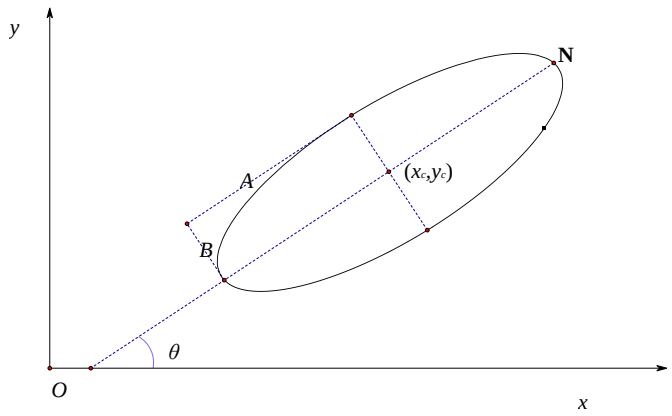
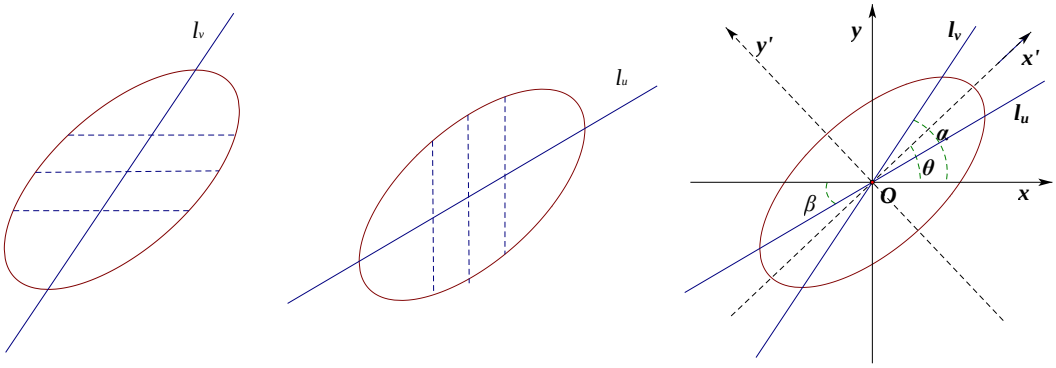


图 3-7 椭圆参数说明

3.3.2 利用椭圆几何对称性的检测算法(CSA)

最早由 Chen 和 Ho 提出[21]了利用几何对称性来检测椭圆的算法。如图 3-8(a)所示，假设一个椭圆 E,对其按照从左向右,自上而下进行扫描。假设水平扫描线场与椭圆相交于两点，那么这两点的中点都将位于一条直线l_v上。如图 3-8(b)所示，同理，假设一个椭圆 E,对其按照自上而下，从左向右进行扫描。中点又将参数另外一条直线l_u。两条直线l_v和l_u为椭圆的对称轴，则两者的交点为椭圆的中心，如图 3-8(c)所示。



(a) (b) (c)

图 3-8 扫描椭圆

(a) 水平扫描；(b)垂直扫描；

(c)两对称轴交于椭圆心

后来 Sheu 等人[22]对这个算法进行了发展,推导出了两条对称轴和椭圆偏转角度以及椭圆长短轴之比的关系公式。当α，$\beta>0$时，满足$\alpha>\theta>\beta$的解为椭圆旋转角,其它解则是短轴的偏转方向；当α，$\beta<0$时,满足$\alpha<\theta<\beta$的解为椭圆的方向。在椭圆长短轴,不在坐标轴上时,通过公式 3-12 和 3-13,可求解出。

$$\theta=\arctan\left(\frac{\tan\alpha\tan\beta}{2\tan\beta}\pm\frac{\sqrt{1-2\tan\alpha\tan\beta+\tan^2\alpha\tan^2\beta+4\tan^2\beta}}{2\tan\beta}\right) \tag{3-11}$$

$$\tan(\alpha-\theta)=(B/A)^2\tan(90°-\theta) \tag{3-12}$$

$$\tan(\theta-\beta)=(B/A)^2\cot(90°-\theta) \tag{3-13}$$

详细的推倒过程见 Sheu 的文献。

3.3.3 弦中点 Hough 变换检测椭圆(CMHT)

屈稳太[23]提出了弦中点变换椭圆检测算法。该方法在椭圆内切椭圆概念的基础上,利用椭圆上全部点的内切椭圆必经过椭圆中心的性质,分两步求解出椭圆的参数:先求出椭圆的中心,再求解其他三个参数。

内切椭圆：在椭圆上任取一点,将该点与椭圆上其他点相连构成椭圆的一组弦,这组弦的中点构成一个新的椭圆,称该椭圆为原椭圆在该点的内切椭圆。由于任意通过椭圆中心的弦都被平分中心平分，所以椭圆上的所有内切椭圆的交点就是椭圆的中心。

算法的原理：在二值图像中，椭圆上点的内切椭圆交于椭圆中心，非椭圆上的点的中点曲线，与椭圆中心相交的概率很小。因而在椭圆中心处,必有很多的内切椭圆通过,同时在其它位置上,中点曲线通过的次数就少得多。这样我们就能利用 Hough 变换的基本思想,得到椭圆的中心。然后求出其他三个参数。

3.3.4 三点检测椭圆(RHT3)

如图 3-9，椭圆上任意两个点 P_1 和 P_2，是过这两点椭圆的两条切线，它们相交于 T，就称 T 为椭圆的极，弦 P_1P_2 称为椭圆的极弦。设 P_1P_2 的中点为 M，MT 的中点为 G，那么就有以下两结论：

- 线段 MT 与椭圆边界的交点 P_3 必在线段 MG 上；
- P_3 处的切线和弦 P_1P_2 平行。

设直线 P_1T、P_2T 和 P_1P_2 的方分别为

$$\begin{cases} l_1(x,y)\equiv u_1x+v_1y+w_1=0 \\ l_2(x,y)\equiv u_2x+v_2y+w_2=0 \\ l_3(x,y)\equiv u_3x+v_3y+w_3=0 \end{cases} \tag{3-14}$$

其中 u、v 和 w 是直线的参数。则过,几两点的二次曲线簇为

$$\lambda l_3^2(x,y)+l_1(x,y)l_2(x,y)=0 \tag{3-15}$$

其中λ为任意常数。为确定二次曲线，我们还需要另外一个点，这里就选择 P_3，将 P_3 带入 3-14 就确定λ，从而椭圆的二次型表达式就完全确定了。

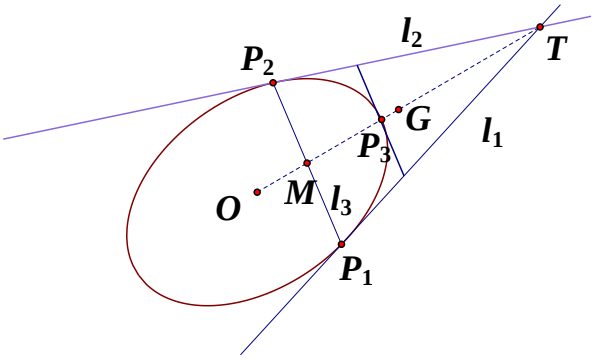


图 3-9 椭圆的极和极线

该算法很好利用了边缘点的边缘方向信息，当确定了 P_1、P_2 和 P_3，如果 P_3 的切线方向与弦 P_1P_2 的方向相差超过一定的误差范围，可以认为三者不在同一个椭圆上，就不需要后续计算了。可以避免大量的无效计算，加快了计算速度。

3.3.5 最小二乘法拟合椭圆

由 Fitzgibbon 提出的使用最小二乘法,从给定的数据中拟合出椭圆二次曲线方程的系数,二次曲线的多项式表示

$$F\left(x,y\right)=ax^2+bxy+cy^2+dx+ey+f \tag{3-16}$$

其中$b^2-4ac<0$。多项式$F\left(x,y\right)$称为点(x,y)到椭圆的距离。当$F\left(x,y\right)$=0时表示椭圆方程。引入向量

$$\begin{aligned}\alpha&=\left[a,b,c,d,e,f\right]^T\\ X&=\left[x^2,xy,y^2,x,y,1\right]\end{aligned} \tag{3-17}$$

式 3-16 改为

$$F_\alpha\left(X\right)=X\cdot\alpha \tag{3-18}$$

通过求解点到二次曲线的代数距离平方和的最小值,我们就可以将这些点 N 个数据点$\left(x_1,y_1\right),\left(x_2,y_2\right),\mathrm{L}\ ,\left(x_N,y_N\right)$，拟合到一条二次曲线上去。

$$\min\sum_{k=1}^{N}F\left(x_k,y_k\right)^2=\min\sum_{k=1}^{N}\left(F_\alpha\left(x_k\right)\right)^2=\min\sum_{k=1}^{N}\left(x_k\cdot\alpha\right)^2 \tag{3-19}$$

上式可以直接用最小二乘法求解。但是这个算法本身在求解过程中还有不稳定之处。下面介绍 Halir[24]提出的改进的基于最小二乘法拟合的椭圆的算法。

设有 N 个数据点$\left(x_1,y_1\right),\left(x_2,y_2\right),\mathrm{L}\ ,\left(x_N,y_N\right)$。设计矩阵 D 为一个$N\times6$的矩阵，$D=[D_1,D_2]$

$$D_1=\begin{bmatrix}x_1^2 & x_1y_1 & y_1^2\\ \mathrm{M} & \mathrm{M} & \mathrm{M}\\ x_k^2 & x_ky_k & y_k^2\\ \mathrm{M} & \mathrm{M} & \mathrm{M}\\ x_N^2 & x_Ny_N & y_N^2\end{bmatrix} \tag{3-20}$$

$$D_2=\begin{bmatrix}x_1 & y_1 & 1\\ \mathrm{M} & \mathrm{M} & \mathrm{M}\\ x_k & y_k & 1\\ \mathrm{M} & \mathrm{M} & \mathrm{M}\\ x_N & y_N & 1\end{bmatrix} \tag{3-21}$$

散射矩阵 S

$$S=D^TD=\begin{bmatrix}S_1 & S_2\\ S_2^T & S_3\end{bmatrix} \tag{3-22}$$

因为式 3-8 中的各个系数是成比例的。所以为了限定求解出来的二次曲线是椭圆，我们令$b^2-4ac=1$。限制矩阵 C 为6×6的矩阵

$$C=\begin{bmatrix}C_1 & 0\\ 0 & 0\end{bmatrix},\ C_1=\begin{bmatrix}0 & 0 & 2\\ 0 & -1 & 0\\ 2 & 0 & 0\end{bmatrix} \tag{3-23}$$

令上式 3-8 的二次曲线的系数为列向量α

$$\alpha=\begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix},\alpha_1=\begin{bmatrix}a\\ b\\ c\end{bmatrix},\alpha_2=\begin{bmatrix}d\\ e\\ f\end{bmatrix} \tag{3-24}$$

通过结合式 3-19 化简，可以得到 M 为缩小的散射矩阵，大小为 3×3，

$$M=C_1^{-1}\left(S_1-S_2S_3^{-1}S_2^T\right) \tag{3-25}$$

求解矩阵 M 的特征值和特征向量就能完成椭圆的拟合。得到的特征向量即为椭圆参数 α。

目前的椭圆检测算法大致可以分为投票类聚和最优化两大类方法。Hough 变换和 RANSAC 都是采用映射的方法,将样本点投影到参数空间,用累加器或者类聚的方法来检测椭圆。这类算法有很好的健壮性,能一次检测多个椭圆,但是需要复杂的运算和大量的存储空间。另一类方法包括最小二乘拟合算法,遗传算法以及其他最优化椭圆拟合方法。这类方法的主要特点在于准确性高,不过通常需要预先进行分割或分组处理,无法直接用于多个椭圆的检测,对噪声的敏感程度高于前一类方法。

3.4 本章小结

本章首先引入了 Hough 变换，研究了通过 Hough 来检测直线的算法及其改进算法，然后讨论了利用 Hough 变换思想为基础的几种圆的检测算法，接着对以结合椭圆的几何对称性和 Hough 变换思想的几种椭圆检测算法做出了研究对比。最后对直接通过最小二乘法拟合椭圆的算法做出了详细的介绍，后续章节中将会对上述某些算法进行实现然后实验对比，并提出改进方法。

第四章 检测算法编程实现及分析

4.1 直线检测

直线是众多几何形状（如三角形，矩形等）的基本构成。对直线检测算法进行实现是接下来 4.2 节多边形检测的基础。下面详细介绍标准 Hough 变换检测直线的实现过程。本章提到的编程实现均是 MATLAB 中进行。

4.1.1 算法流程及实现

直线检测算法的流程如下：

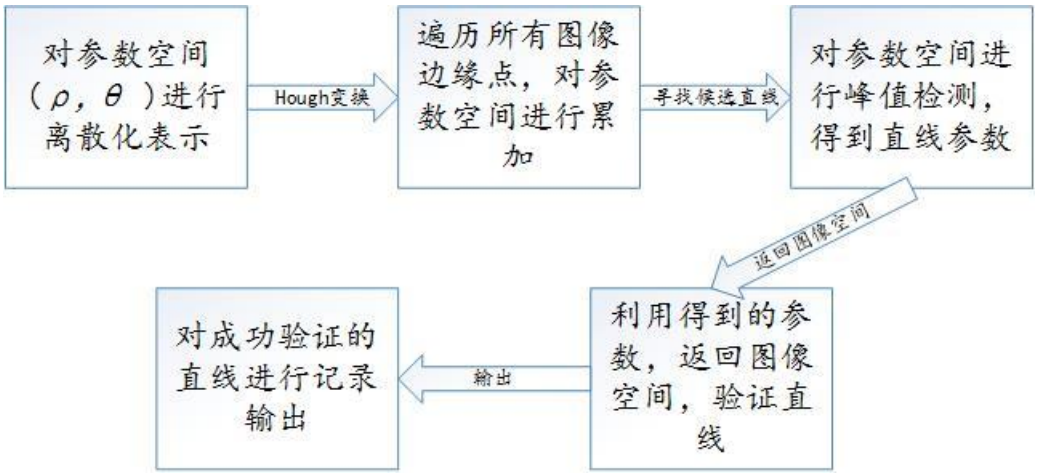


图 4-1 标准 Hough 变换流程图

（1） 对参数空间的(ρ,θ)进行离散化处理，其中$-D \le \rho \le D$，D 为图片的对角线长度，$-90° \le \theta < 90°$。设ρ的间距为$d\rho$（默认值为 1），θ的间隔为$d\theta$（默认值为 0.5）。

（2） 对于每一个边缘点(x,y)按照公式 3-1 计算所有θ对应的ρ，对参数空间的(ρ,θ)进行累加，即为编写 MATLAB 函数 hough_t.m。

（3） 在参数空间中寻找 n 个峰值，记录峰值的位置存入 S，将峰值位置的邻域的值置为零，n 为需要求解的直线的数量所有峰值都需要大于某个阈值，这里称为峰值最小值。注意这里将多条共线的线段看做一条直线。默认邻域的大小为图像大小的 1/50 向上取整并取最接近的奇数。默认

峰值最小值为最大值的一半,即为编写函数 hough_peaks.m。

（4） 对得到参数位置集合 S 中的每一对参数(ρ,θ)，在图像空间中验证是否存在真实的直线。对(ρ,θ)对应直线将会有多条线段，将间隔较小（默认最大间隔为 20）的线段连接起来。如果一条线段的长度达到了直线的最小值（默认值为 40），就判定为一条直线，存储到结果 Lines 中。Lines 是一个结构体，其包含了直线的两个端点坐标，直线长度，与 x 轴正向的夹角，以及参数空间对应的ρ和θ。即编辑函数 hough_lines.m

注意上面提到的各个默认值都可以根据需要进行修改。

4.1.2 实验结果及分析

实验一：对第二章（图 2-5）用于边缘检测对比实验的图进行直线检测

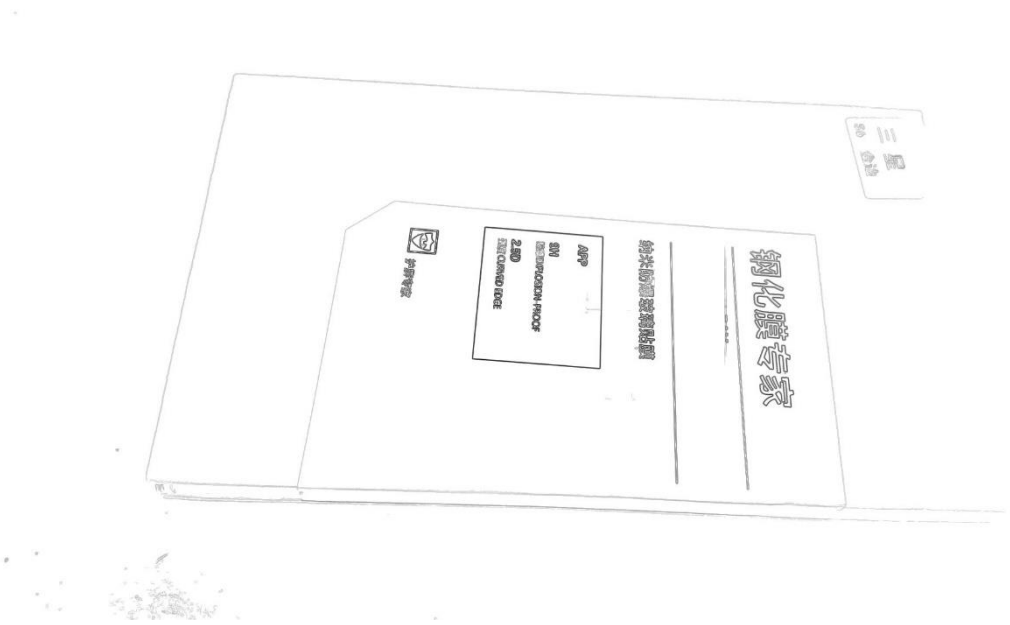


图 4-2 对纸板边缘图像的开运算重建后的图像

图 4-2 是对图 2-7 进行了开运算重建后即调用 MATLAB 函数 imopen 然后取补的图像，对比图 2-8，上图中的多数孤立的干扰点被排除。有利于接下来的直线检测。这也符合第二章开头提到的对边缘图像进行后期处理的原则。

图 4-3 显示了 Hough 变换的参数空间，是调用自己编写函数 hough_peaks 得到的结果，其中黑色线条表示了，参数空间累加器不为零的地方。其中小方块标记的位置为检测到的峰值的地方。一共检测到了 10 个峰值。具体检测结果见表 4-1。上面提到的能设置为默认值的参数都采用了默认值进行计算。

对于参数空间的可视化还可采用后面图 4-7 的三维视图。这两种方法各有优势，本方法可以直接观察参数空间峰值的具体坐标。而后面的方法则是更加立体直观，可以看到参数空间点数及其变化趋势。

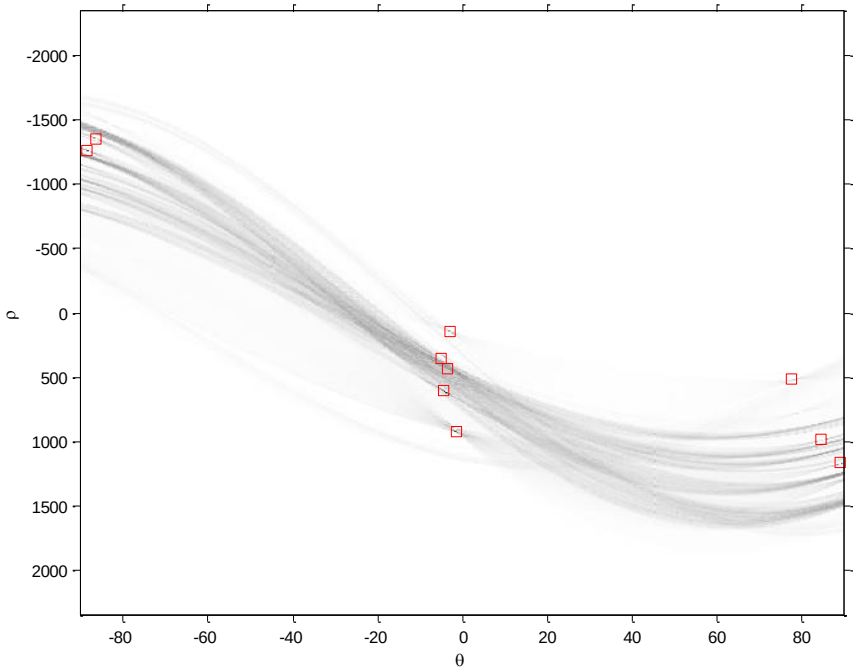


图 4-3 Hough 变换空间及其峰值

对于上图中小方块所对应的具体坐标，请对照表 4-1。

表 4-1 参数空间检测到的峰值的位置

n	1	2	3	4	5	6	7	8	9	10
θ	-88.5	-86.5	-5	-4.5	-3.5	-3	-1.5	77.5	84.5	89
ρ	-1264.5	-1359.6	358.2	606.3	436.2	142.1	920.4	517.2	982.4	1164.5

表 4-2 检测到直线的位置

n	1	2	3	4	5	6	7	8	9	10
θ	-88.5	-86.5	-5	-4.5	-3	-3	-1.5	77.5	84.5	89
ρ	-1264.5	-1359.6	358.2	606.3	142.1	142.1	920.4	517.2	982.4	1164.5

表 4-2 是检测到的直线参数，是调用编写函数 hough_lines 的结果，一共检测到了 10 条，但是和表 4-1 对比，发现$(\rho,\theta)=(436.2,-3.5)$的直线并没有检测出来。而$(\rho,\theta)=(142.1,-3)$检测出了两条，图 4-4 的粗实线是根据检测出来的直线参数绘

制在原图上的结果，可以看到，最上面的轮廓在原图中是一条直线，这里却检测出了两条，是由于两者的间隔超过了最大间隔阈值，所以判断为两条直线。而对于没有检测出的$(\rho,\theta)=(436.2,-3.5)$是由于每一小段的线段的距离没有达到设定的最小直线长度。由于图中还存在很多干扰文字，导致了还有些直线没有检测出来。注意为了便于观察，已经将原图像进行了取补作为底图。后面的很多图形为了显示的需要也进行了这样的处理。

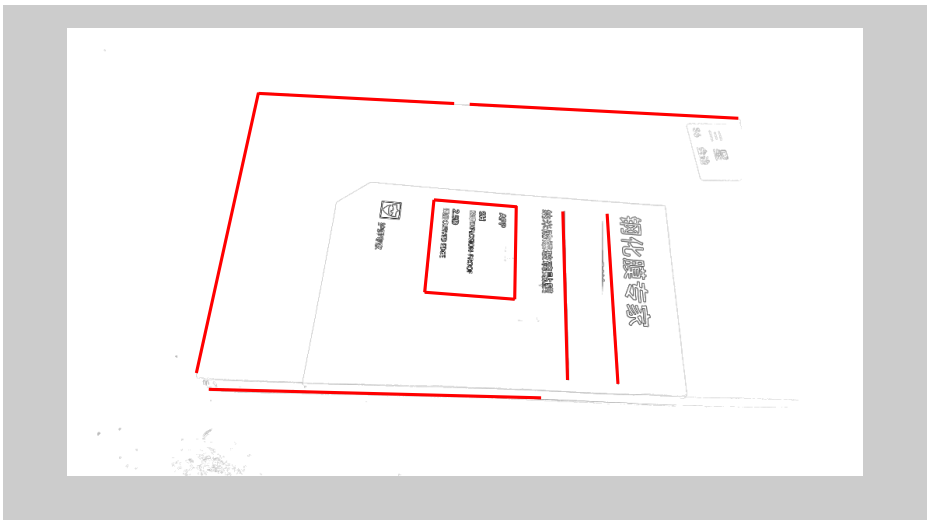


图 4-4 检测出来的直线

实验二：对简单的手绘图像进行直线检测

如图 4-5 是在白纸上用蓝色笔画的一些多边形，其中左上角的直角三角形和正方形相切。图 4-9 同样是手绘的，与 4-5 基本相似，但是左上角的直角三角形和正方形是分离的。两张图片的大小均为 1152×2048。

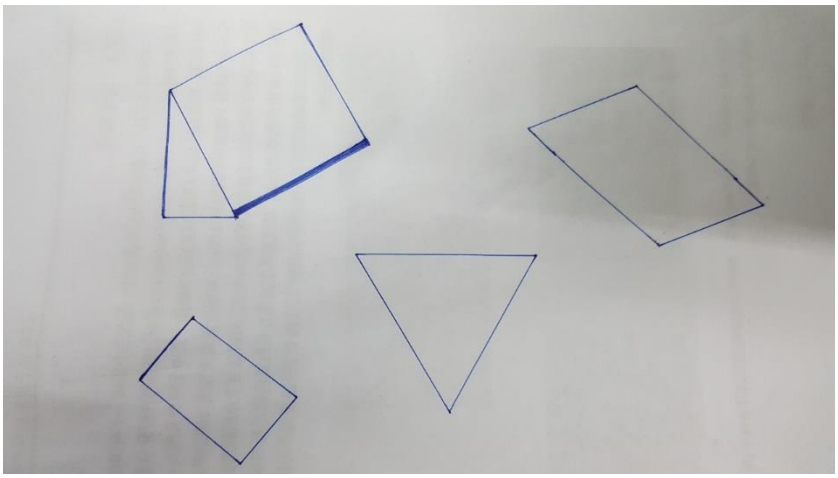


图 4-5 手绘连接的多边形

首先对上图进行边缘检测和直线检测。

参数设置：检测直线数量 17

Hough 变换的峰值检测的邻域设为[120,13]

峰值最小值设为最大值的 1/6

其余为默认值。

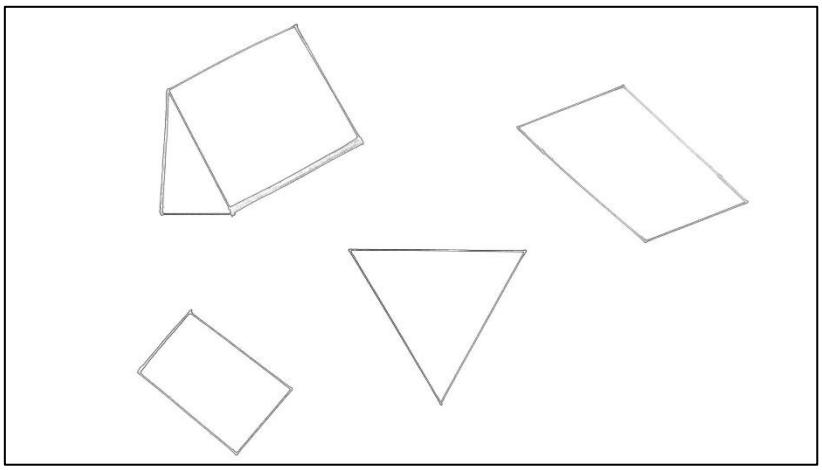


图 4-6 连接多边形边缘处理图像

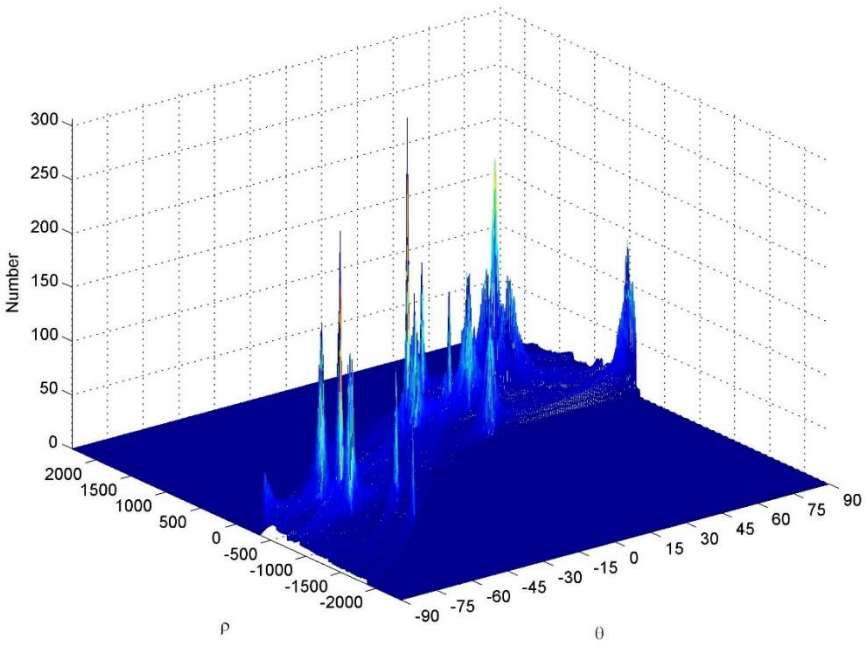


图 4-7 连接多边形的参数空间

图 4-6 是图 4-5 进行边缘检测后并后续处理后的图像，运用了 colorgrad 函数

和 imopen 函数，然后取补的结果。图 4-7 是对边缘图像进行 Hough 变换后得到的参数空间的三维图像，运用了 hough_peaks。(ρ,θ)对应了参数空间的位置。纵坐标 Number 表示了经过某个位置的点的个数。可以看到上图中存在多个峰值。而这些位置参数正是检测出来的可能直线参数。这些参数的具体值见表 4-3。一共检测了 17 个峰值。这正好完全对应了原图中的 17 条直线。不存在漏检、误检的情况。具体检测的结果如图 4-8。

表 4-3 连接多边形检测出的峰值参数

n	1	2	3	4	5	6	7	8	9
θ	-62.5	-60.5	-59.0	-43.0	-42.0	-39.5	-37.0	-0.5	0.0
ρ	-267.1	-616.3	-427.2	-915.4	-638.3	495.2	332.1	602.3	520.2
n	10	11	12	13	14	15	16	17	
θ	21.0	21.5	27.0	28.0	49.0	49.5	61.0	86.5	
ρ	1131.0	754.3	371.2	728.3	853.4	1177.0	1444.0	421.2	

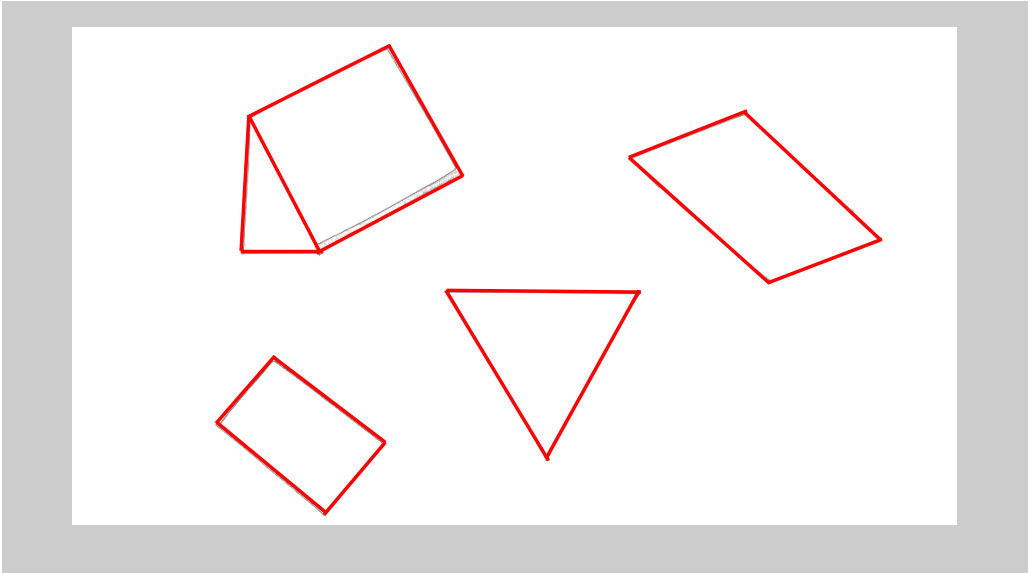


图 4-8 连接多边形的直线检测结果

下面对图 4-9 进行边缘处理和直线检测

参数设置：检测直线数量 18

Hough 变换的峰值检测的邻域设为[120,13]

峰值最小值设为最大值的 1/6

其余为默认值。

在参数设置上没有大的变化，只是将需要检测的直线的数量进行了增加。

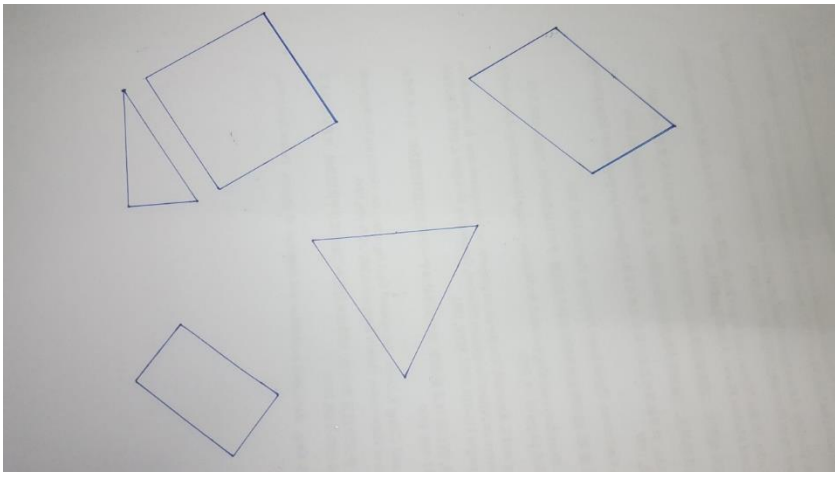


图 4-9 手绘分离的多边形

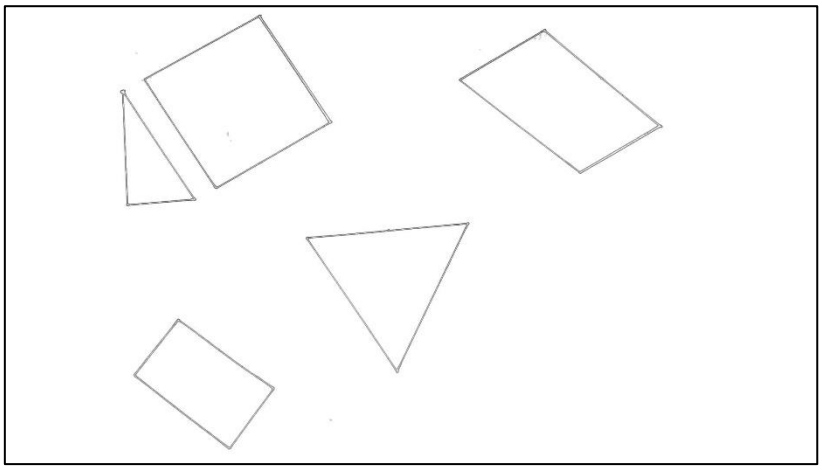


图 4-10 分离多边形的边缘处理图像

图 4-10 是对图 4-9 手绘分离的多边形的边缘处理的结果，用于后续的直线检测。图 4-11 是 Hough 变换空间的可视化图像，与图 4-7 很相似。也存在多个峰值。但是可以看到图 4-11 的最大值要略小于图 4-7，这是由于两图的差异造成的。线条的粗细和长短在很大程度上决定了在参数空间的峰值的大小。表 4-4 是检测出峰值的相关参数。一共检测出了 18 个峰值，这也完全对应了待检测的 18 条直线，最后的检测结果见图 4-12，可以发现所有的直线都很好的被检测出了。

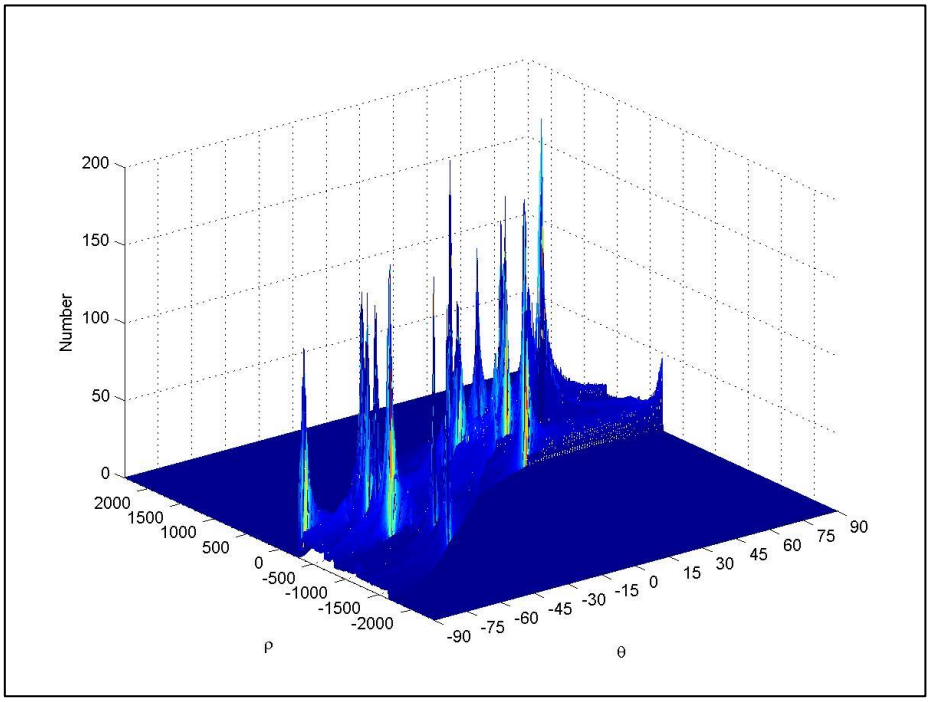


图 4-11 分离多边形的参数空间

对上图参数空间进行峰值检测（调用 hough_lines）出来的结果和表 4-4 相同。图 4-12 是最终的检测结果

表 4-4 分离多边形检测出的峰值参数

n	1	2	3	4	5	6	7	8	9
θ	-87.5	-56.5	-56.5	-56	-56	-39.5	-37.5	-37.5	-36.5
ρ	-285.1	-190.1	-129.1	-516.2	-307.1	-817.3	-550.2	537.2	372.2
n	10	11	12	13	14	15	16	17	18
θ	4.5	5	29	30	30	30.5	51.5	53.5	64.5
ρ	522.2	644.3	331.1	660.3	1088	741.3	833.4	1115	1287

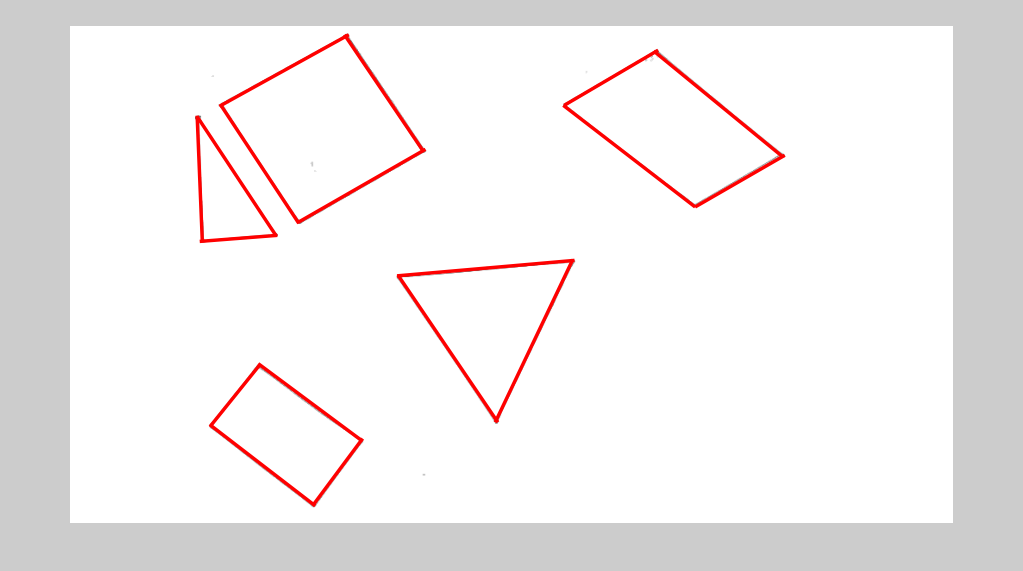


图 4-12 分离多边形的直线检测结果

对运行时间进行比较。

表 4-5 是对上面两张图片，连接多边形和分离多边形，在相同条件下进行 10 次实验记录的运算时间。记录的时间不包括 Hough 变换前的图像边缘处理和其他绘图时间。时间单位为秒。

表 4-5 两次实验的运行时间

次数	1	2	3	4	5	6	7	8	9	10	平均时间
连接	2.349	2.318	2.335	2.317	2.330	2.337	2.334	2.327	2.324	2.344	2.332
分离	2.017	2.004	2.070	1.905	2.058	1.931	1.933	1.954	1.937	1.916	1.973

观察上表可以看到对于同一幅图像，在其他条件不变的情况下，每次的运行时间都很稳定，只有微小的波动。纵向对比，可以看到对于分离多边形的平均计算时间要比连接多边形少 0.359 秒，这是由于分离多边形中边缘点的数量为少于连接的边缘点。分离多边形的边缘点为 29872 个，连接多边形的边缘点数量为 37436 个，两者相差了 20.2%，而在运行时间上两者相差为 15.4%，差异并不大。由于边缘点的个数决定了 Hough 变换的计算量，而 Hough 变换又占用了整个过程的大部分计算量，所以可以得出结论：运算时间和边缘点数量是息息相关的，边缘点越多，计算越耗时。

4.2 多边形检测

多边形可以是直线的组合图形，通过 Hough 变换算法检测出了直线，并记录了直线的参数，包括直线两个端点坐标、长度、与 x 轴正向的夹角。运用这些直线参数，并结合多边形的几何性质可以完成多边形的判断算法。

拟完成对如下图中的多边形检测：其中 $N \geq 5$

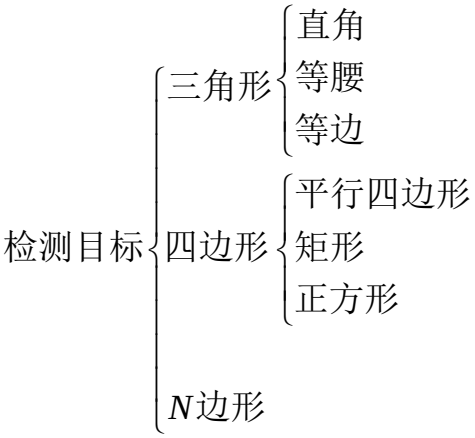


图 4-13 多边形检测目标

4.2.1 算法原理

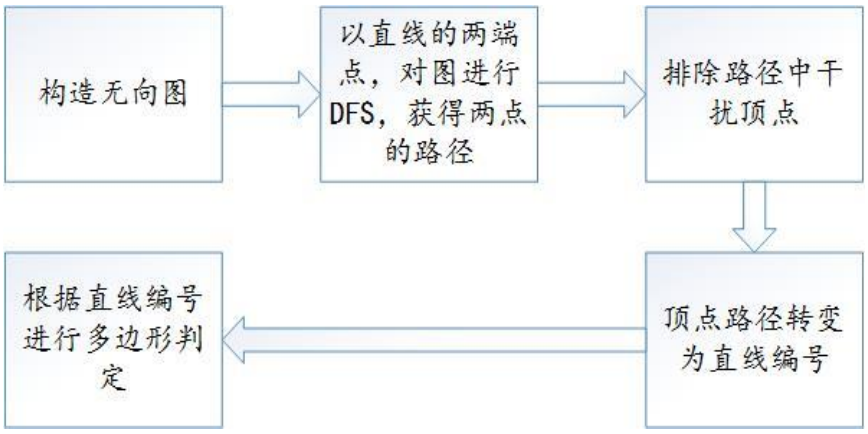


图 4-14 多边形检测算法流程图

图 4-14 是多边形检测算法的流程图，下面详细介绍其实现步骤：

（1） 将一副图像中的检测出所有直线的端点作为顶点(Vertex)，连接的直线和任一顶点某半径范围的其他顶点作为边(Edge)，构成一个无向图（Undirected

Graphs）。参阅附录中本人编写的polygon.m 中的子函数adjacent.

（2）对以某条直线为基准，检测所有经过该直线的多边形。具体实现为：以直线的一个端点为起点，另外一个端点为终点，对无向图进行深度优先搜索(Depth-first-search)，直到搜索到终点。得到起点到终点的路径。参阅 polygon.m 中的子函数 dfs。

（3）对路径进行排序，剔除干扰顶点。从一条直线端点 A 跳到另外一条直线的端点 B 时，可能会先跳到一个第三直线的端点 C，然后再到达 B。这样中间的 B 点就定义为干扰点，需要将其排除。

（4）将处理后的顶点路径，变换成对应直线的编号，并记录。然后根据得到的直线编号(line_ind)来获取直线相关参数。

（5）得到直线的数量和参数，用来进行多边形检测。

下面将列出获取直线编号和参数后进行多边形检测的伪代码，来介绍多边形判定的原理。

设 n 为第 5 步获得的直线数量，设 linen 为已经检测的多边形线段编号。如果 $n<3$ 结束检测，不在下述代码介绍范围内。对于 $n\geq 5$ 时，多边形规则性小，且不常见，故直接判定为 n 边形。如果不能判定为比较特殊的三角形和四边形，就判定为普通的三角形和四边形。

```
1 initialize linen to empty
2    if (!isempty(line_ind))
3       if(!Find line_ind in linen)
4             add line_ind to linen
5           if( n==3)
6               dis3=lengths of those lines
7                  if (dis3 fits Pythagorean)
8                              print Right triangle
9                       elseif (all of dis3 equal)
10                               print Equilateral triangle
11                      elseif (only 2 of dis3 equal)
12                             print Equilateral triangle
13                      else    print a triangle
14
```

```
15          if(n==4)
16              deg4=degree of 4 lines
17              dis4= lengths of 4 lines
18              degm=abs(deg4(2:4)-deg4(1))
19              degm=sort(degm);
20              if (degm(1)==0 && degm(2)==degm(3))
21                  if degm(2)==90
22                      if(all of dis4 equal)
23                      print Square
24                      else print Rectangle
25                  else print Parallelogram
26              else print Quadrilateral
27
28          else print n-sided polygon
```

从上面伪代码中，可以看出算法的主要利用了多边形的几何特性。由于多边形不存在像圆和椭圆这种一个完整的表达式，所以没有 Hough 变换，最小二乘拟合之类的检测算法。这也是没有必要的，因为进行直线检测完成后再进一步利用其几何特性就能够完成多边形的检测。关于具体的编程代码，请参阅附录中的polygon.m文件。

4.2.2 实验结果及分析

端点的间隔采用默认值 20，如果两端点距离小于 20 就认为这两点在无向图中是连通的，即在原图像中位于同一位置。

采用上面图 4-5 和 4-9 作为多边形的检测实验材料。对其进行的直线检测参数及结果见表 4-3 和 4-4。将直线检测得到的结构体参数 lines 用于多边形检测的参数输入。

如表 4-6 是对上面两图进行多边形检测实验结果：

表 4-6 对两图进行多边形检测结果

图 4-5	等边三角形	正方形	直角三角形	矩形	平行四边形	五边形
图 4-9	等边三角形	正方形	直角三角形	矩形	平行四边形	

其中所有检测出来的多边形均为一个。观察原图，对于分离的多边形图像的检

测结果是很准确的；对于连接的多边形图像，算法也很好的检测出了每个分离的几何图形，只是最后还将直角三角形和正方形连接在一起的部分再次进行了检测，判定为一个五边形。这并不能看做是算法的错误，这个可以满足不同检测目的的需要，如果不想进行类似的判决，只需要对算法进行细小的修改即可完成。另外算法的整体运行时间稳定且快速，平均时间为 0.05 秒左右。

4.3 椭圆检测及成像

4.3.1 随机椭圆检测算法(RED)

随机椭圆检测算法先选取样本点进行椭圆拟合，然后通过判断另外的点是否在拟合的椭圆上，来进行椭圆检测。下面介绍结合陈海峰[23]和李良福[24]的随机椭圆检测算法。

算法主要分为下面 6 步：

（1） 将所有边缘点加人集合V 中，n_p为边缘点的数量。初始化失败计数器$f=0$。令T_f为最大失败次数，Te为最小边缘点数量，如果n_p小于Te时结束计算；T_a距离参数，选取的任意两个点距离必须小于T_a，同时大于$T_a/\sqrt{3}$。T_d第四点到可能椭圆边界上的最大距离阈值。T_r为椭圆残缺率。Tsm为自适应迭代最大次数。

（2） 当$f=T_f$或$n_p<Te$时计算计算；否则，从V 中随机选取四个点，如果前三个点满足先提到的距离限制，对着三个点的邻域的边缘点进行 Halir 最小二乘拟合椭圆法。判断第四个点到椭圆的距离，如果小于T_d，将上面四个点从V 中移除并加空集合 S 中，进入下一步；否则递增 f，从新开始第二步。

（3） 是E_k是上一步拟合的椭圆，遍历所有边缘点，所有到E_k距离小于T_d的边缘点从V 中移除，加入 S 中。

（4） 如果 S 中边缘点的数量$n_s \geq kT_r \times C_k$，其中 k 为倍率，按需求设置，C_k为椭圆周长，认为E_k是真实存在的椭圆，初始化迭代次数Ts=0，进入下一步；否则递增 f，将 S 的点返回V 中，清空 S，回到第二步。

（5） 如果$Ts<Tsm$，运用得到的边缘点 S 再次进行椭圆拟合，此时边缘点数为n_s，再次遍历所有边缘点，将满足距离要求的点从V 中移除，加入 S 中，得到新的边缘点数量为n_{s1}；否则认为没有成功检测椭圆，递增

f，将 S 的点返回V 中，清空 S，回到第二步。

（6） 如果$\left|n_s - n_{s1}\right| / n_s \le tol$，$tol$ 为点数变化率，认为成功检测到椭圆，重置$f = 0$，记录椭圆参数；否则回到第五步进行自适应拟合。

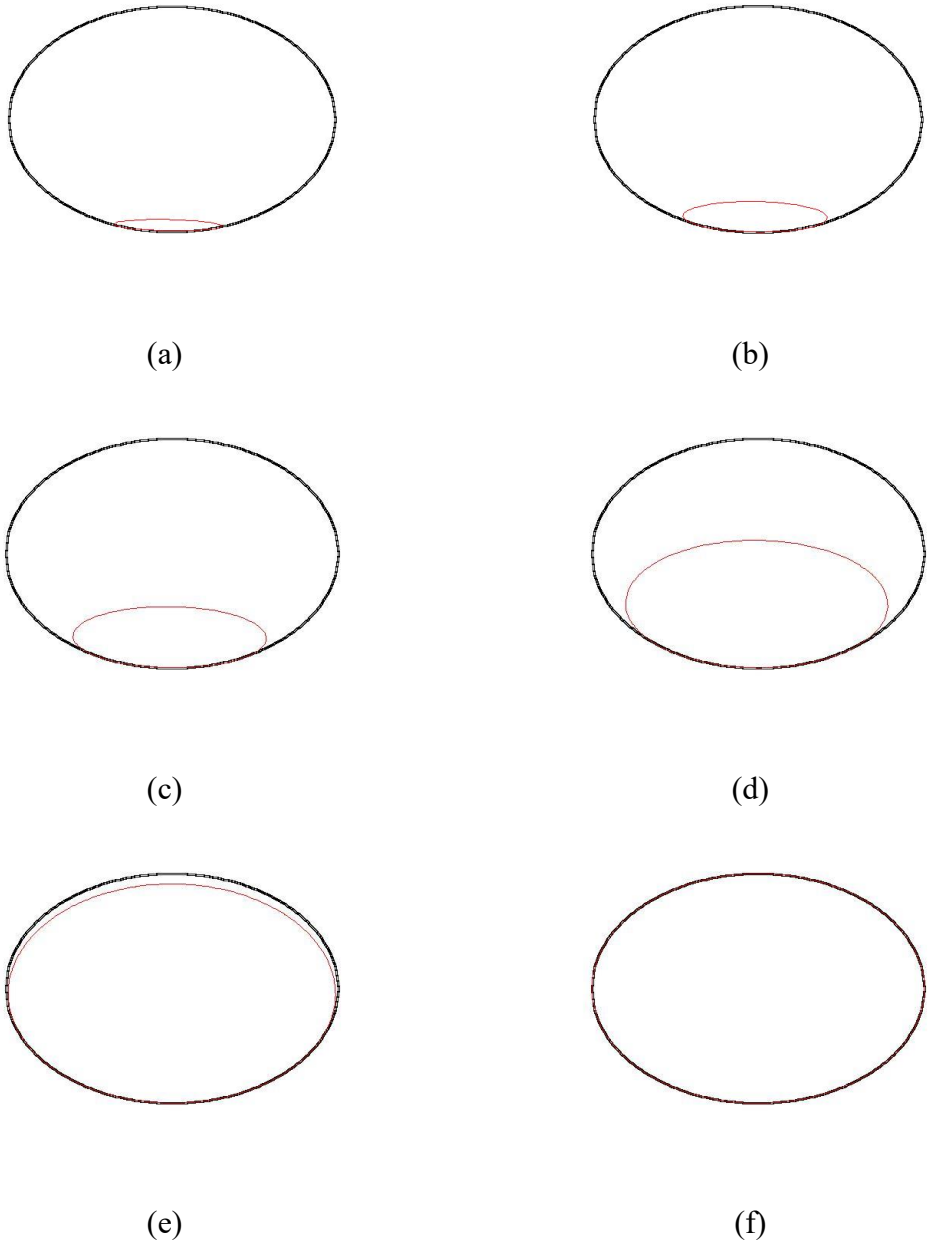
该算法类似于第三章中介绍的随机圆的检测算法，都是直接通过采样点，然后进行拟合，而后验证。而本算法加入了，迭代过程，对于椭圆而言，仅仅采样三个点，拟合的椭圆是极其不准确的，必须通过迭代来对椭圆进行修正。具体效果，将通过接下来的实验证明。

对一个椭圆进行随机椭圆检测,调用自己编写函数 RED1，来观察迭代过程。

表 4-7 随机椭圆检测迭代过程参数

迭代次数	x_c	y_c	A	B	θ
1	343.9005	285.3143	72.9646	7.5720	1.5900
2	332.2840	286.5637	95.4825	20.2585	1.5846
3	312.5015	287.3446	126.1556	39.9393	1.5831
4	269.3550	288.5817	170.9516	83.1735	1.5801
5	209.9742	290.1182	213.4285	142.9506	1.5737
6	203.4112	290.5019	216.7872	149.6876	1.5708
7	203.4109	290.5012	216.7876	149.6870	1.5708

表 4-7 是迭代过程中的参数，其中各符号含义，$\left(x_c, y_c\right)$表示拟合椭圆的中心，A 和 B 分别表示椭圆的长短半轴，θ表示长轴与 x 轴的夹角，注意这里是弧度制，另外图像的空间的坐标原点在左上角，x 轴竖直向下，y 轴水平向右。这些参数的设置与 3.3.1 节介绍的参数变换是相一致的。



(a) (b)

(c) (d)

(e) (f)

图 4-15 随机椭圆检测迭代过程

一共进行了 7 次迭代，图 4-15 的(a)~(f)显示了前 6 次，最后一次的迭代图像和上图的(f)几乎一样，肉眼已经不能分辨了，故没有列出。其中的细红线就是每次拟合出来的椭圆，外侧的粗黑线为待检测椭圆。观察上图和表 4-7 可以发现：随着每一次迭代，拟合出来的椭圆逐渐变大，椭圆的中心在移动，长短半轴都增加；最后几次迭代是已经很接近检测椭圆了，参数变化也很小了，第六次和第七次迭代参数还是发生了细小的变化。可见提出的这种迭代随机椭圆检测算法是正确合理的。

本方法对于一个椭圆和分离的两个椭圆能够得到较好的结果，但是运行时间不稳定，且速度较慢，同时本算法参数设置繁琐，算法结果对参数的敏感性较高，如果参数设置不合理，将很难检测出椭圆，特别是参数T_a的值至关重要。

4.3.2 弦中点随机椭圆检测算法

对于上面讨论的随机椭圆检测算法，选取的样本点，完全是随机的，只要满足一定的距离限制即可。这就导致了选取的大量的无效采样和不必要计算，下面我们考虑，结合椭圆的几何特性来采取样本点进行椭圆检测，对于椭圆的拟合仍然采用 Halir 的改进最小二乘方法，其在 MATLAB 中的具体实现参与附录中的 Halir.m。

对于随机椭圆检测算法而言，选取的样本点在同一椭圆在来进行拟合是至关重要的，对于上一小节中限制样本点距离在某个范围内也是出于这样的考虑。回顾 3.3.3 节的同一椭圆上任意点的内切椭圆都将相交于一点——椭圆的中心。理论上，只要选择的三个点在同一个椭圆上，那么这三个内切椭圆将会产生一个交点，这个点就是椭圆的中心。反过来，利用这个性质就可以判断样本点是否在同一个椭圆上了，从而可以避免大量的无效采样和计算。其具体编程代码参阅附件 CMPN.m。

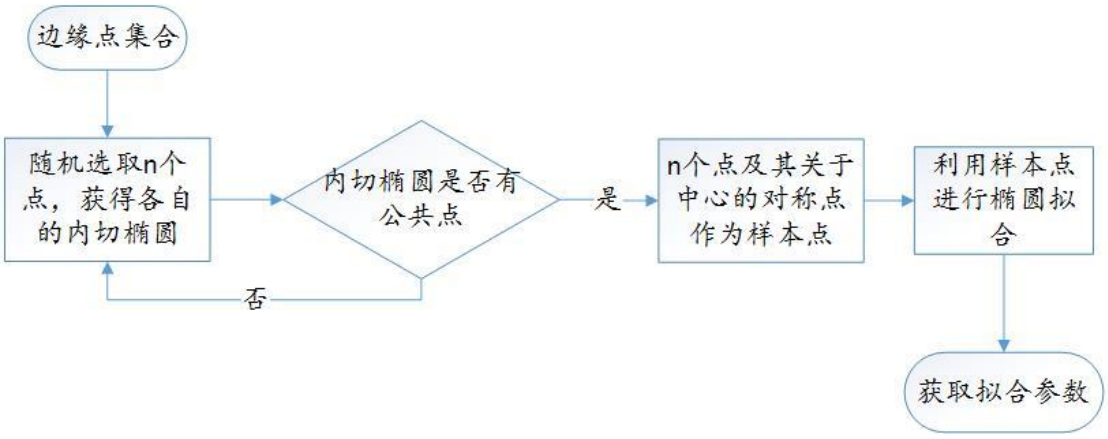
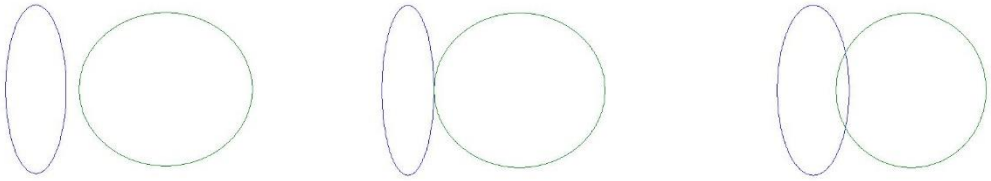


图 4-16 弦中点随机椭圆检测算法流程图

该算法的流程图如 4-16 所示，可以看到算法在逻辑的复杂度上，相较于随机椭圆检测算法已经大大的下降了。上图$n(n\geq 3)$需要根据实际情况来选取，n越大检测出来的结果越准确，但是需要付出更多的计算，具体变化将通过接下来的实验证实。

实验及分析：

下面对两个椭圆的情形进行实验，两个椭圆存在三种位置关系，分离、外切和相交如图 4-17，其中每幅图片的大小为 560×420。



(a) (b) (c)

图 4-17 两椭圆的位置关系

（a）分离；（b）外切；（c）相交

表 4-8 两椭圆图像检测结果及运行时间

位置	点数	失败	差	成功		
				最短时间	最长时间	平均时间
分离	5	2	1	0.25	3.97	1.62
	6	2	0	1.45	11.45	4.89
	7	1	0	1.55	30.96	13.47
外切	5	4	5	1.02	4.02	2.09
	6	4	2	0.80	11.55	3.97
	7	1	3	1.42	16.53	8.51
相交	5	3	2	0.85	1.78	1.11
	6	2	1	0.96	5.86	2.44
	7	0	2	1.12	21.37	4.93

表 4-8 是调用编写的函数 CMPN.m, 分别对图 4-17 的图片进行 20 次检测的结果，点数表示每次检测选取的样本点的个数；失败没能成功检测的次数；差表示检测结果不是很理想，但是椭圆的大概形状拟合正确，小部分边缘有偏差；成功指成功检测并且结果很好，并记录了运行时间，单位为秒。

观察上表，对于同一张图，采样的点数越多，拟合的成功率越高，但是计算时间却急剧增加；对于两分离的椭圆检测的成功率最高，而外切情形是成功率最低，这是由于相切时，很容易将中心判断到切点的附近，从而导致将错误的样本点，带入椭圆的拟合计算中。

对于三个椭圆的图像计算量大幅增加，且成功率急剧下降。

本算法，在一到两个椭圆的情形下，能较好的完成检测，虽然运行时间不太稳定，但是整体的运行时间并不大，是可以接受的。多椭圆时需要对图像进行分割处理，或采用其他算法。关于分割处理，将在 4.4 节详细介绍。

4.3.3 弦中点 Hough 变换拟合检测算法

对于弦中点随机椭圆检测算法，在复杂图像和多样本点时，内切椭圆不易相交于真实椭圆的中心，从而导致对椭圆的漏检和误检。由此可见选择正确的样本点，对基于最小二乘法的曲线拟合的重要性。而样本的正确性我们可以通过内切椭圆的相交中心来判断。利用 Hough 变换的思想，我们可以准确的定位椭圆的中心，然后利用椭圆的中心来选择样本点。

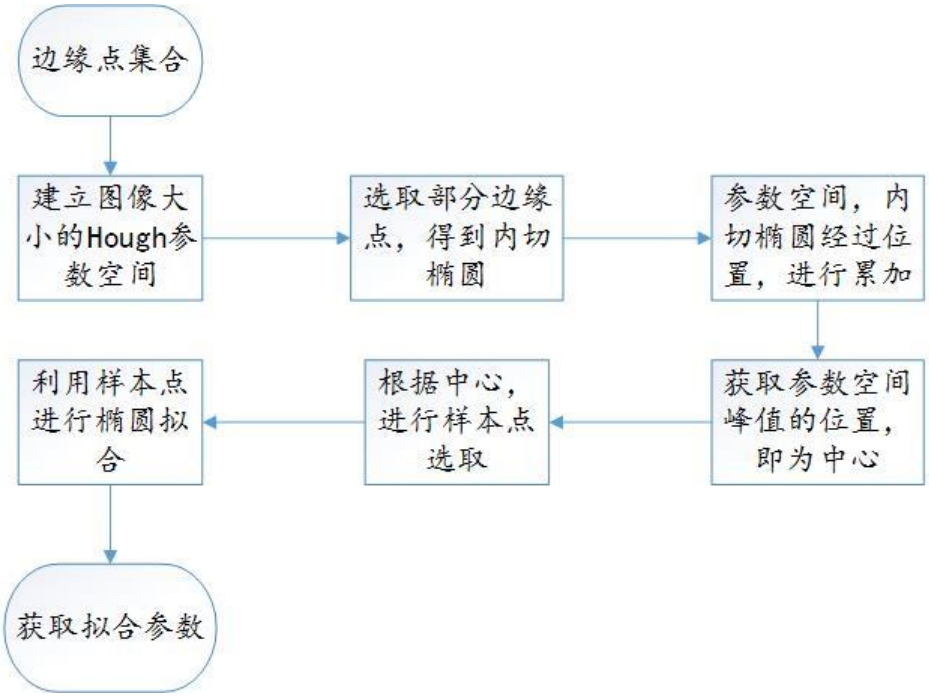


图 4-18 弦中点 Hough 变换拟合算法流程图

选取部分边缘点时，点数越多，累加的参数空间中心越准确，在本程序中随机选择一半的边缘点。获取峰值位置时，可以一次将所有中心全部得到。利用中心位置进行样本点选取时，一个样本点只能对应一个中心，如果对应多个中心，则视为干扰点，将其排除，可以大大的降低误检的概率。其编程实现参阅附录 CMHTN.m。

对上面图 4-17 调用 CMHTN 函数进行实验

表 4-9 两椭圆图像 20 次检测用时

位置	最短时间	最长时间	平均时间
分离	1.251	1.545	1.317085
外切	1.2611	1.4296	1.291245
相交	1.3362	1.6223	1.387795

表 4-9 是得到的运行时间的统计结果，时间单位为秒，一共进行了 20 次实验，不论是那种位置关系的椭圆，都能很好的检测，没有出现一次错误。且算法的运行时间短和稳定，都在平均时间的小范围内波动。运行时间明显快于上面的弦中点随机椭圆检测算法。

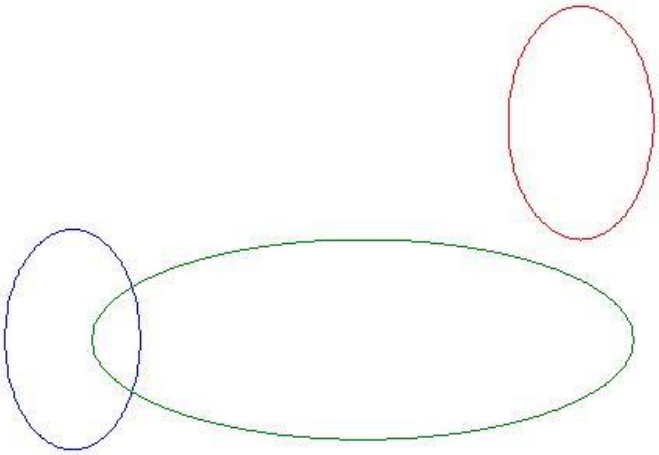


图 4-19 三椭圆检测

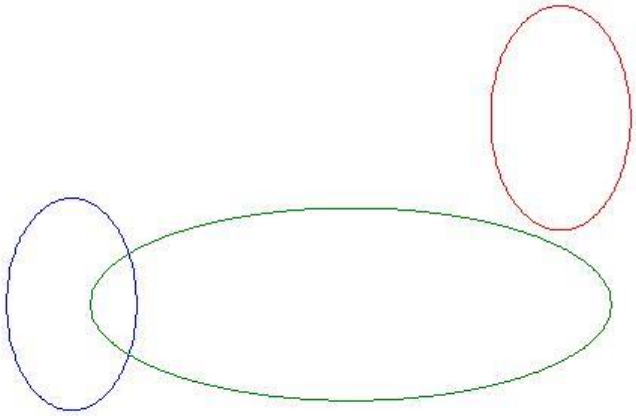


图 4-20 非理想三椭圆检测样本

对图 4-19 的三椭圆调用 CMHTN 函数进行 20 次检测，其中 19 次成功，一次失败。对于成功的 19 次，进行运行时间统计，最短时间为 1.5519s；最长时间为 1.823s；平均时间为 1.6066s，运行时间短且稳定。这正是 Hough 变换类算法的突出特点。

图 4-20 是对图 4-19 的右上角的椭圆进行了一点向下平移了一点得到的三椭圆图像。对其调用 CMHTN 函数，进行椭圆检测的结果见图 4-20。可以看到，成功检测出了两椭圆，同时出现了一个误检的椭圆。观察原图，其中左下角和右上角的椭圆，形成了很好的对称性，导致了参数空间在非真实椭圆中心处大量累加了，从而导致误检。

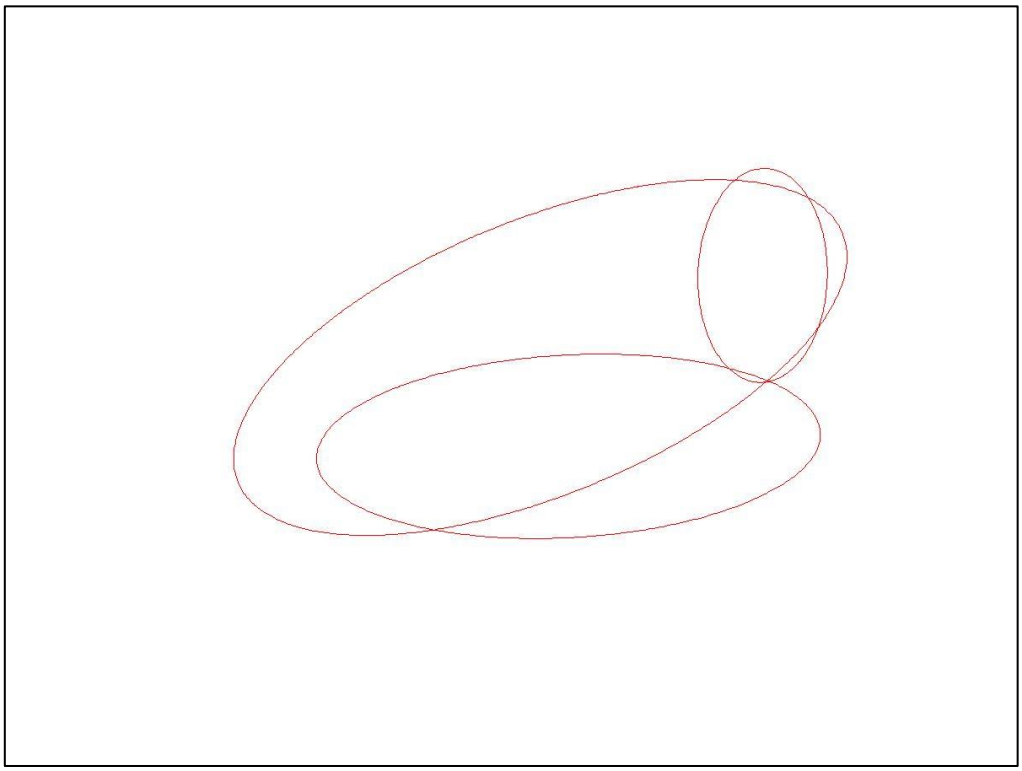


图 4-21 非理想三椭圆检测结果

对于出现的这种问题，可以通过图像分割然后在进行检测，或者采取拟合椭圆验证方法，拟合出椭圆后计算该椭圆范围邻域的边缘点数量来验证该椭圆是否真实存在。

这种问题只会出现在图像中存在较好的对称性时，几率较低，所以对弦中点 Hough 变换拟合算法的适用性的影响不大。对比上面三种方法，本算法不但对多椭圆存在较好的检测，同时计算快且稳定。

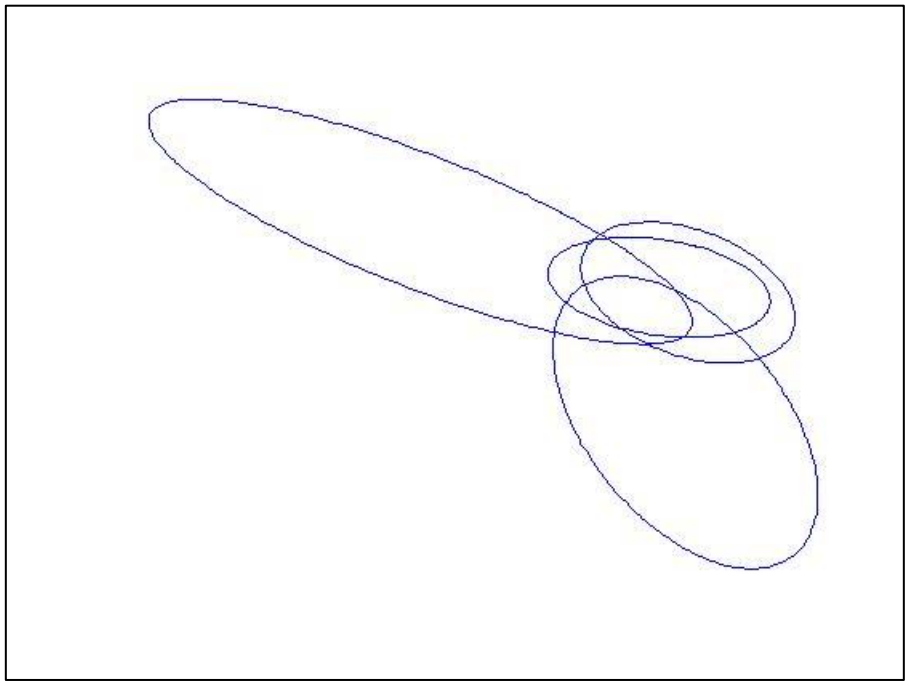


图 4-22 复杂四椭圆

图 4-22 是一副较为复杂的四椭圆图像，对其采用本方法进行 20 次检测，其中 4 次结果较差；成功检测的 16 次，最短运行时间为 1.9195s，最长为 2.2484s，平均时间为 1.9741s。可见弦中点 Hough 变换拟合算法，能对多椭圆较好的检测。

4.4 图像分割和椭圆检测

对于需要检测的几何形状，无论是直线、圆和椭圆，都是一个连接的图形。为了尽可能的得到一个完整的图形，这里采用 8 连接。利用 MATLAB 的 IPT 函数 bwlabel，可以对二值图像进行连接区域标记。对于同一个连接的图形会标记为同一值，然后将每一个值对应的图像$\left[x_{\min}:x_{\max},y_{\min}:y_{\max}\right]$保存在各自的二值矩阵中，其具体实现过程参阅附录中的 seg.m 函数文件。然后对各个分割图像进行检测。这里只考虑对椭圆的检测。

设$a\sim f$是分割图像检测出来的椭圆参数，而我们需要的是原始图像中的椭圆参数，就需要进行参数变换：坐标平移。再对椭圆进行平移的过程中，椭圆的偏转角，长轴、短轴的长度都没有变化，只是椭圆的中心发生了变化。假设分割得到的椭圆中心为$\left(x_c,y_c\right)$，则原始图像中心应为$\left(x_{cn},y_{cn}\right)=\left(x_c-1+x_{\min},y_c-1+y_{\min}\right)$。设合并图像中的椭圆表达式为：

$$a_n x^2 + b_n xy + c_n y^2 + d_n x + e_n y + f_n = 0 \tag{4-1}$$

由于椭圆旋转角度不变，故设

$$(a_n, b_n, c_n) = k(a, b, c) \tag{4-2}$$

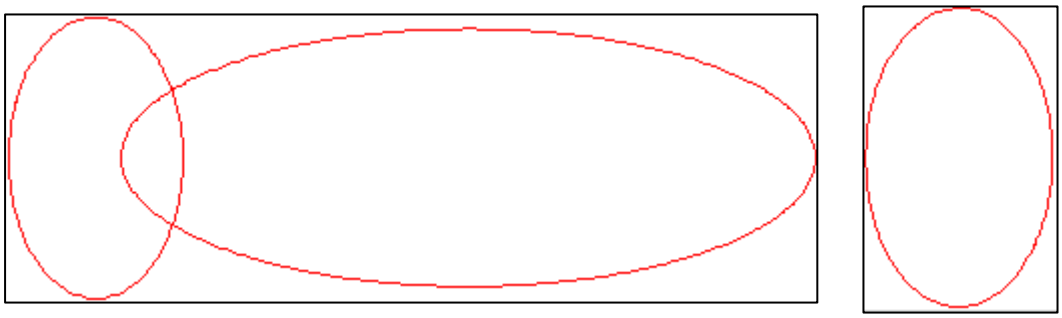
又

$$\begin{cases} x_{cn} = \dfrac{b_n e_n - 2c_n d_n}{4a_n c_n - b_n^{\,2}} \\ y_{cn} = \dfrac{b_n d_n - 2a_n e_n}{4a_n c_n - b_n^{\,2}} \end{cases} \tag{4-3}$$

$$A^2 = \frac{2\left(a_n x_{cn}^{\,2} + c_n y_{cn}^{\,2} + b x_{cn} y_{cn} - f_n\right)}{a_n + c_n - \sqrt{(a_n - c_n)^2 + b_n^{\,2}}} \tag{4-4}$$

这里不妨设 $f_n = 1$,利用上面这些公式就可以求见出 $a_n \sim f_n$，得到原图坐标系内的椭圆相关参数。其具体运算求解过程参阅附录中代码 coefficients_trans.m。

利用分割合并方法，对上图 4-20 进行检测：



(a) (b)

图 4-23 分割图片的检测结果

(a) 左下角；(b)右上角

对图 4-20 直接采用弦中点 Hough 变换拟合，成功检测出三个椭圆的概率很低，但是采用分割检测技术将该图分割成两部分（调用 seg 函数），然后分别进行椭圆检测（调用 CMHTN 函数），却能很好的完成。将上述检测得到椭圆参数进行坐标系变换（调用 coefficients_trans 函数），合并到原图像的检测结果见图 4-24。可以看到，合并后的拟合椭圆和图 4-20 几乎一样，同时对于本图像的分割计算到后来

合并只用时 0.45s 左右，速度是非常快的。由此可见，采取分割检测的弦中点检测算法，能够克服对称性较高的图片的缺点。下面，来通过对比研究分割检测对运行时间的影响。

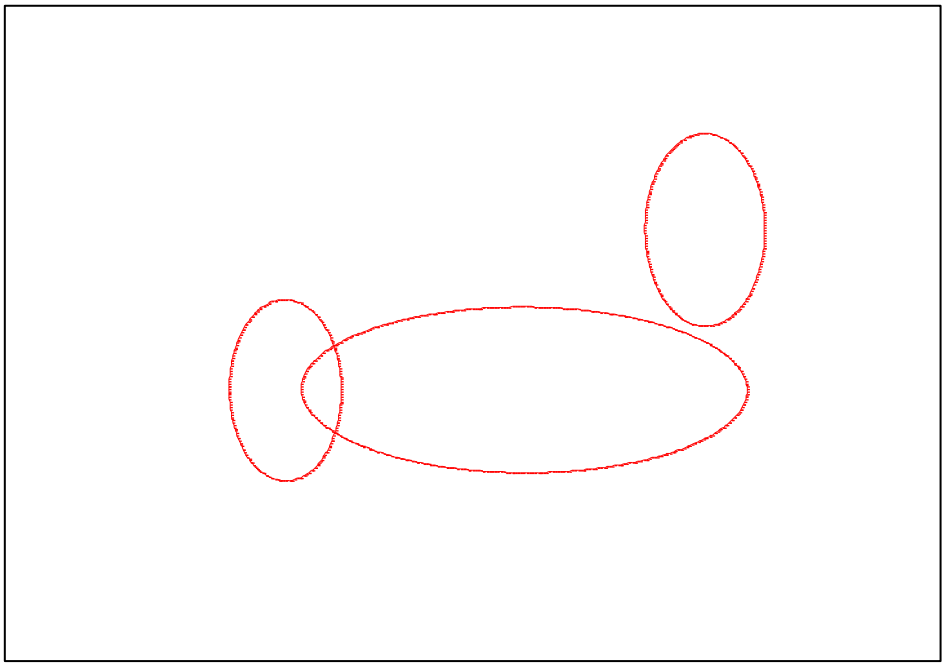


图 4-24 分割合并的椭圆检测结果

对图 4-19 的三个椭圆图片，进行分割合并检测和直接检测实验，两种方法都能很好的完成检测。分别进行 20 次实验均成功检测并统计计算时间。

表 4-10 两种方法的运行时间

方法	最短时间	最长时间	平均时间
分割合并	0.5348	0.5950	0.5532
直接计算	0.6117	0.6734	0.6349

由于本次实验时，没有绘制拟合后的图像，同时对程序进行了一些优化处理，所以运行时间较先前有较大的提升。但我们关心的是相同情况下，两种算法的对比，所以这些处理是可以接受的。观察表 4-10，可以发现分割合并算法，计算较快，且其最长时间也比直接运算的最短时间少，平均时间约减少了 12.89%。由此可见，分割合并算法，不但能解决对称性问题，同时也有助于加快运算速度。

4.5 本章小结

本章对直线、多边形、椭圆的检测算法进行了编程实现和研究。直线采用标准Hough 变换，利用检测出来的直线，结合多边形的几何特性，从而对多边形进行检测。对椭圆进行了三种算法的对比，随机椭圆检测算法的适用性和效率较低，而提出的弦中点 Hough 变换拟合算法，不但适用性大，稳定性高，同时计算速度快；但是对于某些对称性较高的图片，检测成功率不高，可以结合分割合并的方法克服该问题。

第五章 结束语

本文主要研究了数字图像中的直线、多边形、圆和椭圆的检测算法。介绍了这些检测算法的研究现状、发展过程和应用前景；对数字图像的基本概念和边缘检测算子进行了阐述，同时对彩色图像的边缘检测进行了实验对比，选取出了合适的方法。对目前一些主流的直线，圆和椭圆的检测算法进行了对比分析。实现了标准 Hough 变换检测直线和随机椭圆检测算法，同时依据直线检测结果结合多边形的几何特性，提出了一种多边形的检测算法，能较好的完成常见多边形的判断；结合了椭圆的弦中点内切椭圆几何性质和最小二乘拟合椭圆，实现了弦中点随机椭圆检测算法，并发现其缺点，将其改进提出了弦中点 Hough 变换拟合椭圆检测算法。该算法不但适用性高，计算快且稳定，通过实验证实了：结合图像的分割合并，能进一步提高该算法的性能。

当然，本文的算法，还有一些地方需要改进和验证，例如弦中点 Hough 变换拟合算法，对于同心椭圆的检测是不能很好完成的，接下来可以考虑对其进行修改和完善。当然对于实现的代码还可以进一步优化处理，减少某些不必要的计算，提高性能。

参考文献

[1] 陈凯，刘青.一种随机化的椭圆拟合方法[J]. 计算机科学也工程，2005，27(6):48-49

[2] 陆宗骐，张秋萍.工程图纸矢量化中的线条轮廓跟踪法[J]. 中国图象图形学报，1997,12(2):878-882

[3] 陶小平，冯华君，徐之海等.基于显微成像的光纤连接器端面检测系统设计[J].光学仪器 2006,28(5):3-7

[4] 黎自强，腾弘飞.广义 hough 变换:多个圆的决速随机检测[J]. 计算机辅助设计与图形学学报，2006,18(1):27-33

[5] P. V. C Hough. A Method and means for recognizing complex patterns[P]. Us patent:3069654, December 18, 1962

[6] V. Shapiro. Accuracy of the straight line hough transform: The non-voting approach[J]. Computer Vision and Image Understanding, 2006, 103(1):1-21

[7] R. C. Lo and W H Tsai. Gray-scale bough transform for thick line detection in gray-scale images[J]. Pattern Recognition, 1995, 28(5):647-661

[8] S. H. Chiu and J. J. Liaw. An effective voting method for circle detection[J]. PRL: Pattern Recognition Letters, 2005, 26(2):121-133

[9] V. A. Ramirez, C. H. Capulin, A. P. Garcia, et al. Circle detection on images using genetic algorithms[J]. Pattern Recognition Letters, 2006, 27(6):652-57

[10] H. Freeman. Computer Processing of Line-drawing Image [J]. Computer Surveys, 1974, 6(1):47-97

[11] E. Bribiesca. A Geometric Structure for Two-dimensional Shapes and Three-dimensional Surfaces [J]. Pattern Recognition, 1992, 24(4):483-496

[12] E. Bribiesca. A Guzman. How to Describe Pure Form and How to Measure Differences in Shapes Using Shape Numbers [J]. Pattern Recognition, 1981, 12(1):101-112

[13] M K Hu. Visual Pattern Recognition by Moment Invariants[J]. IEEE Trans on Information Theory, 1962, 57(8):179-187

[14] C. T. Zahn, R. Z. Roskies. Fourier descriptors for plane closed curves[J]. IEEE Trans on Computers, 1972, C-21(3):269-281

[15] F Canny. A computational approach to edge detection[J]. IEEE-PAMI, 1986, 8(6):679-698

[16] R. C. Gonzalez, R. E. Woods. Digital Image Processing Second Edition(M). Prentices Hall, 2003, 572-584

[17] 冈萨雷斯.数字图像处理（MATLAB 版）[M].（阮秋琦等）. 北京：电子工业出版社，2005,172-173

[18] J. R. Bergen and H. Shvaytser (Schweitzer). A probabilistic algorithm for computing hough transforms[J]. ALGORITHMS: Journal of Algorithms, 1991, 12(4): 639-656

[19] L. Xu, E. Oja, P. Kultanen. A new curve detection method: Randomized hough transform (RHT)[J]. Pattern Recognition Letters, 1990, 11(5): 331-338

[20] H. K. Yuen, J. Princen, J. Illingworth, et al. Comparative study of hough transform methods for circle finding[J]. Image Vision Compute, 1990, 8(1):71-77

[21] C. T. Ho and L. H. Chen. A fast ellipse/circle detector using geometric symmetry[J]. Pattern Recognition, 1995, 28(1):117-124

[22] H. T. Sheu, H. Y Chen, W. C. Hu. Consistent symmetrical axis method for robust detection of ellipses[J]. IEE Proceedings-Vision Image and Signal Processing, 1997, 144(6):332-338

[23] 屈稳太.基于弦中点 hough 变换的椭圆检测方法[J].浙江大学学报(工学版)，2005, 39(8):1132-1135

[24] 陈海峰.数字图像中基本几何形状检测算法的研究与应用[D]. 杭州:浙江大学，2007

[25] 李良福，冯祖仁，贺凯良. 一种基于随机 hough 变换的椭圆检测算法研究[J]. 模式识别与人工智能，2005,18(4):459-464

致谢

衷心感谢我的指导老师董宇亮副教授。在整个毕业设计中，对我细心指导和严格要求。同时还从赖生建老师那里得到了很多建设性的意见。在完成毕业设计这段时间里，他们不断的鼓励和引导我，同时对于我的错误也婉言指正。对于我拙劣的论文，多次提出修改意见。两位不但知识渊博，实事求是的学术精神更是给我留下了深刻的印象。由衷的感谢两位老师及其他给以我帮助的人。

附录

部分编程代码 ：包含多边形检测 polygon.m; 弦中点随机椭圆检测算法 CPMN.m; 弦中点 Hough 变换拟合检测 CMHTN.m; Halir 最小二乘拟合椭圆 Halir.m; 图像分割函数 seg.m; 分割椭圆参数合并参数函数 coefficient_trans 函数及其他相关函数。注：由于篇幅有限，某些代码没有加入本附录，如随机椭圆检测 RED1.m。代码全部有本人独立编写。由于附录较长，外文资料在后面较远位置。

多边形检测 polygon.m

```
function    polygon(lines,dis)
% POLYGON Dectects polygons of an image.
%     POLYGON(LINES,DIS) finds polygons of deteced lines. LINES is a result
%     of HOUGH_LINES. DIS is the maximum distance of two adjacent points
%     which are on seperate lines. In this case, those two lines are
%     considered to connected.
%     Polygons:
%     Triangle: Equilateral triangle, Isoceles triangle, Right triangle.
%     Quadrilateral: Parallelogram, Rectangle, Square.
%     Pentagon...n-side polygon.

if nargin<2
    dis=20;    %Default max distance of two vertex.
end

n=size(lines,2);
end_p=zeros(n,4);
deg_dis=zeros(n,2);

if n<3
    error('wrong imput')
```

```
else

    %Initialize
    for k=1:n
        end_p(k,:)=[lines(k).point1,lines(k).point2];   %Endpoints
        deg_dis(k,:)=[lines(k).theta,lines(k).length];   % Degrees and length of lines
end

    adj=adjacent(end_p,dis); % Creat an adjacent matrix
    linen=[];

    for k=1:n % Every line
        adjk=adj;
        adjk{2*k-1}(1)=[];
        adjk{2*k}(1)=[];
        path=findPath(2*k-1,2*k,adjk); % Find the path form an endpoint to another.
        path=sort(path);

        i=1;
        while i<length(path)     %Delete some bad vertex.
            if mod(path(i),2)==0 %Even
                if i==1   %The first point
                    path(i)=[];
                else
                    if path(i)-1==path(i-1) %Successive index is good
                        i=i+2;
                    else   %Un-successive.
                        path(i)=[];
                    end
                end

            else    %Odd
```

```
        if path(i)+1==path(i+1)
            i=i+2;
        else
            path(i)=[];
        end
    end
end

m=length(path);
% The indexes of lines which are on the same polygon.
line=round(path(1:2:m)/2);
line_num=m/2; %The number of sides of polygon.
line_ind=0;
  % Index of lines in decimal system. eg.matrix [2 3 5 6]-> number '6532'
for j=1:line_num   %Code the line indexes.
    line_ind=line_ind+line(j)*10^(j-1);
end

tol=2e-2;                %Find if this polygon has been detected
if (~isempty(line))&(isempty(linen)|(isempty(find(linen==line_ind))))
        linen=[linen;line_ind];   %Not been detected
    if line_num==3   %Triangle
        dis3=deg_dis(line,2);
        dis3=sort(dis3);       %Pythagorean
     if (sqrt(abs(dis3(1)^2+dis3(2)^2))-dis3(3))/dis3(3)<(4*tol)
            fprintf('This is a Right triangle.\n')
        elseif (abs(dis3(1)-dis3(2))/dis3(2)<tol)|(abs(dis3(3)-
                dis3(2))/dis3(2)<tol) % Lsoceles triangle
            if abs(mean(dis3)-dis3(2))/dis3(2)<tol   % Equilateral triangl
                fprintf('This is an Equilateral triangle.\n')
            else
                fprintf('This is a Isoceles triangle.\n')
```

```
                end
            else
                fprintf('This is a Triangle\n');
            end

        elseif line_num==4 %Quadrilateral
            deg4=(deg_dis(line,1)); %Degrees of four lines
            dis4=deg_dis(line,2);     %Lengths of four lines

            degm=abs(deg4(2:4)-deg4(1)); %Use the first as a standard.
            degm=sort(degm);
            if degm(1)<=5&&(degm(3)-degm(2))<=8   %Parallelogram

                if abs(degm(2)-90)<=5&&abs(degm(3)-90)<=5   %rectangle
                    if abs((dis4-mean(dis4)))/mean(dis4)<4*tol %Square
                        fprintf('This is a Square.\n');
                    else
                        fprintf('This is a Rectangle.\n');
                    end
                else
                    fprintf('This is a Parallelogram.\n');
                end
            else
                fprintf('This is a general Quadrilateral.\n');
            end

        else   % Others   num>=5
            fprintf('This is a %d-sided polygon.\n',line_num);
        end
    end
end
end
```

```
end
function adj_mat=adjacent(endpoints,dis)
%ADJACENT Computes and connects the adjacent vertex.
%      ADJ_MAT=ADJACENT(ENDPOINTS,DIS) creats an adjacent matrix,
ADJ_MAT,
%      which is a graph. DIS is the maximum threshold distance of two points.

 n=length(endpoints);
disq=dis^2;
adj_mat=cell(2*n,1); %2*n vertexes of n lines. vertex k and k+1 are endpoints of line
k/2;

%Initialize ADJ_MAT. Connect endpoints of a line.
for k=1:2:2*n
     adj_mat{k}=k+1;
     adj_mat{k+1}=k;
end

%Compute adjacent matrix with    a distance.
for k=1:2*n
     ke=uint8(k/2);    %The index of vertex k in ENDPOINTS.

     if mod(k,2)==1
          Pk=endpoints(ke,1:2);% The current point.
     else
          Pk=endpoints(ke,3:4);% The current point.
     end

     %Connect two near vertexes.
     for j=ke+1:n    %Traverse
          ak=endpoints(j,:);
          %Two endpoints of one line.
```

```
        dis1=((ak(1)-Pk(1))^2+(ak(2)-Pk(2))^2);
        dis2=((ak(3)-Pk(1))^2+(ak(4)-Pk(2))^2);

        if dis1<disq   %Connect each other.
            adj_mat{k}=[adj_mat{k},2*j-1];
            adj_mat{2*j-1}=[adj_mat{2*j-1},k];
        end

        if dis2<disq
            adj_mat{k}=[adj_mat{k}, 2*j];
            adj_mat{2*j}=[adj_mat{2*j},k];
        end
    end
end
end

function path1=findPath(fs,ls,adj_mat)
%FINDPATH Finds a path from form vertex FS to vertex LS.
%     PATH=FINDPATH(FS,LS,ADJ_MAT), the driver function of DSF. ADJ_MAT
is
%     the adjacent matrix of vertexes.   IF PATH1 is empty,   there is no such a
circulate.

n=size(adj_mat,1);
path=1:n;
visited=false(n,1);

path=dfs(fs,ls,visited,adj_mat,path);   %Call DFS to get path form FS to LS.
if path(ls)==ls   %Not find such a path
    path1=[];
else
    k=ls;
```

```
    res=[];
    while k~=fs
        res=[res,k];
        k=path(k);
    end
    res=[res,fs];
    path1=res(end:-1:1); %Reverse PATH
end
end

function [path,visited]=dfs(fs,ls,visited,adj_mat,path)
%DFS: Depth-first Search method to traverse form vertex FS to vertex LS.
%     PATH=DFS(FS,LS,VISITED,ADJ_MAT,PATH) computes and return a route:
%     PATH contains previous vertex index. VISITED is a vector which stores
%     the access information of vertexes. DFS is a recusive function.
%     information of whether vertexes is vistited.

visited(fs)=true;
na=length(adj_mat{fs});

if find(adj_mat{fs}==ls) % Find the last vertext
    path(ls)=fs;
else
    for k=1:na
        if ~visited(adj_mat{fs}(k))
            path(adj_mat{fs}(k))=fs;   %Set PATH
            [path,visited]=dfs(adj_mat{fs}(k),ls,visited,adj_mat,path); %DFS
        end
    end
end
end
```

弦中点随机椭圆检测算法

```
function [samples,center,cnt]=CMPN(BW,num,ns)
%CMPN Uses Chord Middle Points method to find NS points randomly which are
%  the same ellipse.
%    [SAMPLES,CENTER,CNT]=CMPN(BW,T) computes the center of an ellipse
%    in binary image BW and returns samples which are the same ellipse.
%    SIZE(SAMPLES) is the number of ellipses. SAMPLES(K) are three points.
%    CENTER store all centers of ellipses found by this algorithm.
%    CNT is a counter of found ellipes.
%    NUM is the max number of ellipses which need to find.
%    NS is the number of samples.

 [M,N]=size(BW);
idx=find(BW);   % All edge points. A column vector
[X,Y]=idx2xy(idx,M); %Numbers of row and column.

n=length(idx);
done=true; %Loop control
cnt=1;   %Counter of picking points
fn=0; %Fail counter

samples_c=cell(num,1); %Samples of true ellipses.
center=[];   %Centers of ellipses.
while (done&&(cnt<n)&&fn<num)
  % Get ns points with RAND
idxn=round(1+(n-1)*rand(ns,1));
[Sx,Sy]=idx2xy(idx(idxn),M); % Get row and column indexes.

% All middle points
MS=zeros(n,2,ns);

for k=1:ns   % Compute middle points.
```

```
    MS(:,:,k)=[X+Sx(k),Y+Sy(k)]/2;
end
 A=MS(:,:,1); %The crossover points of inscribed ellipses
for k=1:ns-1   %Find the intersecting pointof all inscribed ellipses.
    A=intersect(A,MS(:,:,k+1),'rows');
end


    if ~isempty(A)   %A is the center of a ellipse.
        center_new=mean(A,1);
        if isempty(center)||(~isfind(center_new(1),center_new(2),center))
            %Whether center_new has been detcected
          fn=fn+1;     %This is a new center.
         samples_c{fn}=[Sx,Sy];   % Add points to SAMPLES.
         center=[center;center_new]; %Add this new center.
        end
    else
        cnt=cnt+1;   %Increase the counter
    end
end
samples=zeros(ns,2,fn); %Place samples.

for k=1:fn   %Cell to matix
     samples(:,:,k)=samples_c{k};
end

for k=1:fn   % Double samples
    sak=samples(:,:,k);
     a=round([2*center(k,1)-sak(:,1),2*center(k,2)-sak(:,2)]);
      %Another endpoints. Center is the middle points.
     done1=true;
     while(done1)     %Waive endpoints which are out of the boundary
         idx_out=find(a(:,1)<0|a(:,1)>M);
```

```
            if ~isempty(idx_out)
                a(idx_out,:)=[];
            end
             idy_out=find(a(:,2)<0|a(:,2)>N);
             if ~isempty(idy_out)
                a(idy_out,:)=[];
             end
             done1=length(idx_out)|length(idy_out);
        end
        samples_new(:,:,k)=[samples(:,:,k);a];
end

if fn>=1    % At least one ellipse
samples=samples_new;
else samples=[];
end
end
% Convert an index of matrix to coordinate x,y. M is the row of this matrix
function [X,Y]=idx2xy(idx,M)
  Y=ceil(idx/M);
X=idx-M*(Y-1);
end

function r=isfind(Xc,Yc,XY)
%ISFIND Finds (Xc,Yc) in matrix XY.
% IF (XC,YC) is found, R is true;
%Consider a 3*3 neighbourhood

D=(XY(:,1)-Xc).^2+(XY(:,2)-Yc).^2;
if find(D<100)
      r=true;
else r=false;
```

```
end
end
```

弦中点Hough变换拟合检测算法

```
function [samples,center,H]=CMHTN(BW,num,ns)
%CMHTN Combines Chord Middle Method and Huogh Transform to get samples.
%      [SAMPLES,CENTER,H]=CMHT(BW,NUM,NS) computes the ceter of an
%        ellipse in binary image BW
%      and returns samples which are the same ellipse. SIZE(SAMPLES) is the
%      number of ellipses. SAMPLES(K) are three points.
%      CENTER store all centers of ellipses found by this algorithm.
%      H is the Hough transformed matrix.
%      NUM is the max number of ellipses which need to find.
%      NS is the number of samples.


[M,N]=size(BW); .
H=zeros(M,N,'uint16'); %Hough transformed matrix, as a counter of centers of ellipses
idx=find(BW);   % All edge points. A column vector
[X,Y]=idx2xy(idx,M); %Numbers of row and column.
n=length(idx); %The number of edge point.
nc=uint16(n/2); %The number of points which are used to count center
idxn=uint32(1+(n-1)*rand(nc,1)); % NC samples randomly
[Sx,Sy]=idx2xy(idx(idxn),M);

%Similar to Hough transform
for k=1:nc
    MXY=round([X+Sx(k),Y+Sy(k)]/2);
    idxm=sub2ind(size(H),MXY(:,1),MXY(:,2));
    H(idxm)=H(idxm)+1;
end
%Find peaks of Hough transformed space:H. The locations of peaks are centers of
ellipses.
nhood=size(BW)/25;
nhood=max(2*round(nhood/2)+1,1);
```

```
[r,c]=hough_peaks(H,num,max(H(:)/6),nhood); %Find peaks of H
center=[r',c'];
Nd=length(r);   %The number of centers.
SE=zeros(nc,2,Nd); %SE is the another endpoint. The middle points of SE and(Sx,Sy)
is CENTER.
for   k=1:Nd
    SE(:,:,k)=[2*r(k)-Sx,2*c(k)-Sy]; %Compute SE
end.
samples=zeros(2*ns,2,Nd); % Container of samples
cnsk=zeros(Nd,1); %Counter for samples of each ellipse.
for k=1:nc
    d=zeros(num,1);
    for j=1:Nd
        d(j)=isEllipse(SE(k,1,j),SE(k,2,j),7,M,N); %Whether the point SE(k,:,j) is on
the ellipse with the center(j)
    end
    d=uint8(d);
    id_first=find(d,1,'first');   %The first ellipse
    id_last=find(d,1,'last');   % The last ellipse
    if ~isempty(id_first)&id_first==id_last
    %(SE(x,y,id_fist)) is a unique sample of an ellipse.
        if cnsk(id_first)<2*ns;   %The numbner of samples of an ellipse.
            %Push back two sampes
                samples(cnsk(id_first)+1:cnsk(id_first)+2,:,id_first)=[Sx(k),Sy(k);SE(k,1
                ,id_first),SE(k,2,id_first)];
                %Increase counter of samples(:,:,id_first).
            cnsk(id_first)=cnsk(id_first)+2;
        end
    end
    if min(cnsk(:))>=2*ns
        break;   %Samlpes of all ellipses are enough
    end
```

```
end
for k=1:size(samples,3)
    if(find(samples(:,:,k)==0))   %If samples are not enough, give up them
        samples(:,:,k)=[];
        break;
    end
end
% A nested function To see if the point(x,y) is on an ellipse. [M,N]=SIZE(BW);
    function d=isEllipse(x,y,n,M,N)
        if x<1||x>M||y<1||y>N   %Not in the image.
            d=false;
        else
            n1=floor(n/2); %n*n neighbour-hood of (x,y)
            if x-n1<1
                X1=1:n;
            elseif x+n1>M
                X1=M-n1+1:M;
            else
                X1=x-n1:x+n1;
            end
            if y-n1<1
                Y1=1:n1;
            elseif y+n1>N
                Y1=N-n1+1:N;
            else
                Y1=y-n1:y+n1;
            end
            BC=BW(X1,Y1); %Neighbour-hood
            % Once there is 1 in BC, we consider it's on a ellipse.
            if max(BC(:))
                d=true;
            else
```

```
                d=false;
            end
        end
    end
end
```

Halir最小二乘拟合椭圆

```
function coef=Halir(samples,BW,nh)
%HALIR Fits ellipses with a least square method modified by Halir.
%   COEF=HALIR(SAMPLES,BW,NH) fits ellipses based SMAPLES which is
%   a result of Function:CMHTN or CMPN. BW the binary image. NH the
%   neighbour-hood of samples. COEF are coefficients of fitted ellispes.
%   SIZE(COEF)=[NE,6]. NE is the number of ellispes.
%   Every ellipse have 6 coefficients. See line 50.
if nargin<3
    nh=27; % Set the default neighbour-hood
end
 [M,N,ne]=size(samples); % NE is the number of ellipses
%Constrain matrix, C1=[0,0,2; 0,-1,0; 2,0,0]; Cinv=inv(C1)
Cinv=[0, 0, 0.5; 0, -1, 0; 0.5, 0, 0];
coef=zeros(ne,6);
hold on
for k=1:ne
    XY=[];
    for i=1:M
      san=neighbour(samples(i,1,k),samples(i,2,k),nh,BW);
    XY=[XY;san];
    end
    N=length(XY);
    D1=[XY(:,1).^2, XY(:,1).*XY(:,2),XY(:,2).^2];    D2=[XY,ones(N,1)];
    %Design matrix D=[D1, D2]; %Scatter matrix S=D' *D=[S1, S2; S2', S3]
    S1=D1'*D1;    S2=D1'*D2;    S3=D2'*D2;
    Mat=Cinv*(S1-S2/S3*S2');
```

```
    [V,D]=eig(Mat,'nobalance');     %Eigenvector and    Eigenvalue
    %Find the smallest positive egienvalue
    D_diag=diag(D);    D_diag(D_diag<=0)=inf;    [lamda, idx]=min(D_diag);
    alpha1=V(:,idx); alpha2=-inv(S3)*S2'*alpha1;
    %The general equation of ellipse: Ax2+Bxy+Cy2+Dx+Ey+F=0
F=@(p,x)p(1)*x(:,1).^2+p(2)*x(:,1).*x(:,2)+p(3)*x(:,2).^2+p(4)*x(:,1)+p(5)*x(:,2)+p(6
);
    pr1=[alpha1',alpha2']/alpha2(3); %Set F to 1
    coef(k,:)=pr1;
xmin=0;    xmax=550;    ymin=0;    ymax=500;
hold on;
h=ezplot(@(x,y)F(pr1,[y,x]),[-1+xmin,1+xmax,-1+ymin,1+ymax]); set(h,'Color','r');
end
end
.
function XY=neighbour(x,y,n,f) % Return points whose value is 1 in n*n neighbour-
hoods of f(x,y)
[M,N]=size(f);
n1=floor(n/2); % n*n neighbour-hood
if x-n1<1
    x1=1:n;
elseif x+n1>M
    x1=(M-n+1):M;
else
    x1=x-n1:x+n1;
end
if y-n1<1
    y1=1:n;
elseif y+n1>N
    y1=(N-n+1):N;
else
    y1=y-n1:y+n1;
```

```
end
fc=f(x1,y1);    [XY(:,1),XY(:,2)]=find(fc~=0);    % m*1
xc=find(x1==x);      yc=find(y1==y); %Find (x,y) in the new matrix
XY(:,1)=XY(:,1)+x-xc;    XY(:,2)=XY(:,2)+y-yc %Back to image coordinate.;
end
```

图像分割函数seg.m

```
function    [S,Origin]=seg(bw)
%Function SEG Segments a binary image, based on IPT Function:BWLABEL.
%       [S,ORIGIN]=SRG(BW) divides BW, a binary image, into K parts which are
placed in
%       S. LENGTH(S)=K; S is a cell. ORIGIN contains the positions of all
%       segmented images' origin in BW.

[BL,k]=bwlabel(bw);    %Label connected region

% There are k parts
S=cell(k,1);
Origin=ones(k,2);
for j=1:k

      [x,y]=find(BL==j); %One label
      %A rectangle
      xmin=min(x);
      xmax=max(x);
      ymin=min(y);
      ymax=max(y);

      S{j}=zeros(xmax-xmin+1,ymax-ymin+1); %Size
      idx=sub2ind(size(S{j}),x-xmin+1,y-ymin+1); %Indexes of row and column.
      S{j}(idx)=1;      %Only set this object to 1.

      Origin(j,:)=[xmin,ymin];
      S{j}=logical(S{j});    %Double to binary image.
end

end
```

分割检测的椭圆系数转换为合并图像参数coefficients_trans.m

```
function coef_t= coefficients_trans(coef,Origin)
%COEFFICIENTS_TRANS Transforms coefficients of segmented images to a merged
image.
%       COEF_T=COEFFICIENTS_TRANS(COEF,ORIGIN) changes coordinate.
%       COEF is the coefficients of segmented ellipses. COEF is a return of HALIA,
%       however, COEF's origins are all (1,1). When merging    divided images,
%       COEF should be transformed. COEF_T is the transformed coefficients.
%       ORIGIN is origins of segmented image in the merged one.

n=size(coef,1);
coef_t=zeros(n,6);

for k=1:n
      A=coef(k,1);       B=coef(k,2);       C=coef(k,3);
      D=coef(k,4);       E=coef(k,5);       F=coef(k,6);
      xc=(B*E-2*C*D)/(4*A*C-B^2);
      yc=(B*D-2*A*E)/(4*A*C-B^2);
      %New center of ellipse
      xcn=Origin(1,1)+xc-1;       ycn=Origin(1,2)+yc-1;
      temp=2*(A*xc^2+C*yc^2+B*xc*yc-F);
      temp2=A*xcn^2+C*ycn^2+B*xcn*ycn;
      t=-1/(temp/2-temp2+eps);    %Assign new f to 1;
      % The ratio of a b c don't be changed.
 n       a=t*A;    %new coefficients.
      b=t*B;       c=t*C;       d=-t*(B*ycn+2*A*xcn);
      e=-t*(B*xcn+2*C*ycn);       f=1;
      coef_t(k,:)=[a,b,c,d,e,f];
 end
end
```

来源：L. Maisonobe. Quick computation of the distance between a point and an ellipse[EB/OL]. https://www.spaceroots.org/documents/distance/distance-to-ellipse.pdf, Feb, 2006.

外文资料原文

Quick computation of the distance between a point and an ellipse

L. Maisonobe

September 2003, revised May 2005, minor revision February 2006

1 Introduction

We consider the following 2D problem: given a test point A on a plane and an ellipse E, find the point of the ellipse which is the closest to the test point.

This problem occurs in several contexts. The first one is to geopositioning. Planet bodies are often roughly modeled as biaxial ellipsoids, i.e. ellipsoids having a rotational symmetry axis (the polar axis). For such shapes, the meridian planes are ellipses. Given a point position in 3D Cartesian coordinates, we want to compute the longitude, geodesic latitude and altitude. The last two data are obtained by solving this kind of problem. Another context was encountered while computing approximations of graphical representations of 3D circles on a 2D display. In order to build an error model of a Bezier-based approximation of the ellipse, the distance of thousands of approximated curves had to be computed. This later problem is described in another technical note which is available on the same place as this one.

2 Problem description

We will use the traditional notations used in geodesy to describe the problem. We assume the

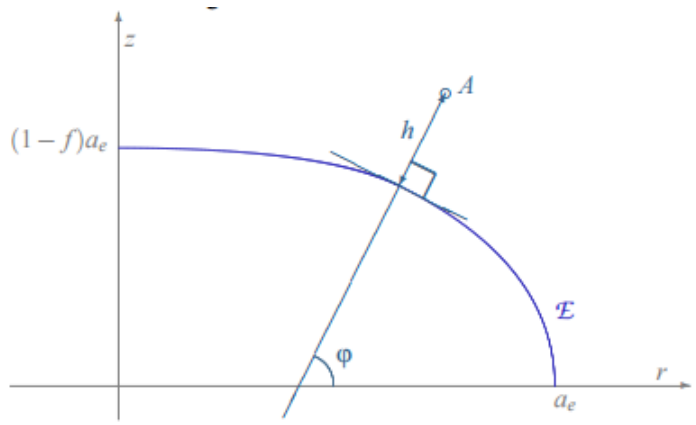
case of longitude λ has already been solved using $\lambda = a\tan 2(y, x)$ and that the remaining problem has been restricted to the 2D meridian plane using the coordinates $r = \sqrt{x^2 + y^2}$ and *z*.

We consider a test point *A* and an ellipse E. After a suitable coordinates change, we can consider the coordinates of the test point in the canonical reference frame of the ellipse. This frame is centered on the ellipse center, has its abscissae axis along the major axis and has its ordinates axis along the minor axis. In this reference frame, the coordinates of *A* are (*r*, *z*). Using suitable axes orientations, we can arrange to have *r* >=0. This assumption *is* used in the following equations.

The ellipse is defined by its semi-major axis is a_e, and either its flattening $f\ (0 \le f < 1)$ or its semi-minor axis $a_p = a_e(1-f)$. The signed distance between the point and the ellipse *h* and the inclination φof the projection of the point on the ellipse are related to the Cartesian coordinates:

$$\begin{cases} r = \left(\dfrac{a_e}{\sqrt{1 - f(2-f)\sin^2\varphi}} + h \right) \cos\varphi \\ z = \left(\dfrac{(1-f)^2 a_e}{\sqrt{1 - f(2-f)\sin^2\varphi}} + h \right) \sin\varphi \end{cases} \tag{1}$$

Figure 1: coordinates definition



In the geopositioning domain, a_e is the equatorial radius of the body, a_p is the polar radius, *h* is the altitude above the ellipsoid (negative when the point is below the surface of the ellipsoid) and φ is the geodesic latitude.

The equation (1) is easy to apply when h and φ are known and *r* and *z* are desired, but it is impossible to reverse in the general case. It can be reversed in the specific case where *h* is known to be null:

$$h = 0 \rightarrow \varphi = a\tan 2\left(z, r\left(1 - f^2\right)\right) \tag{2}$$

3 Preliminary results

3.1 Special case handling: center or the ellipse

A special case we will discard in the algorithm is the ellipse center. This point is easily detected in a preliminary check using a test like $\sqrt{r^2 + z^2} < \varepsilon_0$. The ellipse points closest to the center are both endpoints of the minor axis, we arbitrarily select the point having $\varphi > 0 : \varphi = +\pi / 2, h = -\left(1 - f\right)a_e$.

3.2 Inside/outside check

Given a test point A, it is very simple to check if it lies inside the ellipse or outside of it. This check is done by computing the sign of expression:

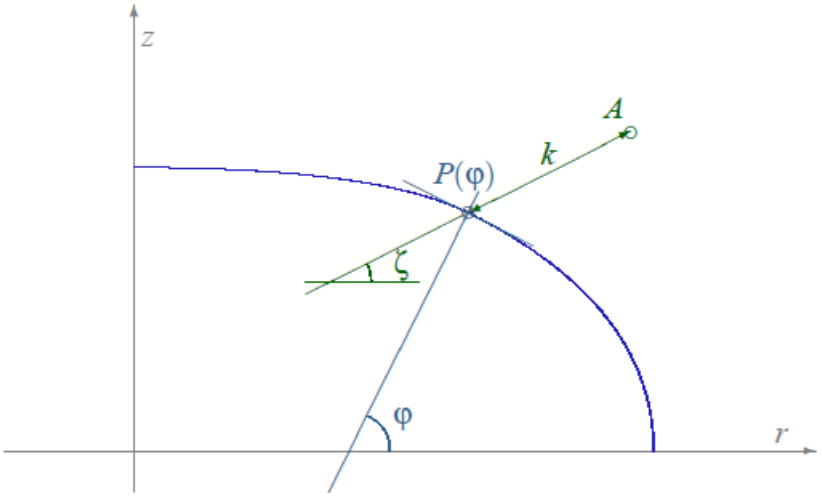
$$\left(\frac{r}{a_e}\right)^2 + \left(\frac{z}{\left(1 - f\right)a_e}\right)^2 - 1 \tag{3}$$

the point is outside if the sign is positive, inside otherwise.

3.3 Intersection between a line and an ellipse

Lets consider a line having slope ζ and containing the test point *A*.

Figure 2: line/ellipse intersection



For well chosen ζ values, this line intersects the ellipse. Let (r', z') be the Cartesian coordinates of the intersection point $p(\varphi)$:

$$\begin{aligned} r' &= r - k\cos\zeta \\ z' &= z - k\sin\zeta \end{aligned} \tag{4}$$

Where k is the signed distance between the test point A and the intersection point $p(\varphi)$. k is positive if the test point is outside of the ellipse and negative otherwise. Note that the sign of k implies the choice of the line orientation, so depending on the test point location inside or outside of the ellipse, we have to choose either ζ or π−ζ; the following equations take care of this choice. Since the intersection point $p(\varphi)$ belongs to the ellipse, its coordinates verify equation:

$$\left(\frac{r'}{a_e}\right)^2 + \left(\frac{z'}{(1-f)a_e}\right)^2 = 1$$

introducing the test point coordinates and the signed distance k

$$\left(\frac{r-k\cos\zeta}{a_e}\right)^2+\left(\frac{z-k\sin\zeta}{(1-f)a_e}\right)^2=1$$

$$\Leftrightarrow (1-f)^2\left[(r-k\cos\zeta)^2-a_e^2\right]+(z-k\sin\zeta)^2=0 \tag{5}$$

$$\Leftrightarrow ak^2-2bk+c=0 \quad where \begin{cases} a=(1-f)^2\cos^2\zeta+\sin^2\zeta \\ b=(1-f)^2 r\cos\zeta+z\sin\zeta \\ c=(1-f)^2\left(r^2-a_e^2\right)+z^2 \end{cases}$$

4 Iterative method

We intend to compute h and φ from r and z using a very quick iterative method based on a suite of lines containing the test point A and intersecting the ellipse on varying points $P(\varphi_n)$. The lines will converge to the projection line.

4.1 Initialization

When the A point is not at the center of the ellipse, we initialize the algorithm using a first line containing both the test point A and the center of the ellipse. The cosine and sine of this line slope ζ_0 are defined by equation (6). Note that we do not compute ζ_0 itself, to avoid numerical problems when $r \ll z$ (i.e. when we are close to the minor axis), we also compute $t_0 = \tan\zeta_0/2 = \sin\zeta_0/(1-\cos\zeta_0)$ directly from $\cos\zeta_0$ and $\sin\zeta_0$ (using a stable formula when $\cos\zeta_0 >= 0$) as we will need it soon.

$$\begin{cases} \cos\zeta_0 = \dfrac{r}{\sqrt{r^2+z^2}} \\ \sin\zeta_0 = \dfrac{z}{\sqrt{r^2+z^2}} \\ t_0 = \dfrac{z}{r+\sqrt{r^2+z^2}} \end{cases} \tag{6}$$

Regardless of its slope, this line is known to have exactly two intersections with the ellipse. We choose the closest one to the A point, which corresponds to the smallest root of binomial equation (5) in absolute value.

Since $1-f>0$, $\cos\zeta_0>0$ and $z\sin\zeta_0\geq 0$, we can deduce $a>0$ and $b>0$. This implies that when the binomial has real roots, their sum is strictly positive, so the smallest root in absolute value is also the smallest root in algebric value.

The classical expression for the smallest root is:

$$k_0=\frac{b-\sqrt{b^2-ac}}{a}$$

However, this expression is not numerically accurate when $b^2 \gg ac$ because in involves computing something similar to $b-(b-\varepsilon)$, leading to a numerical cancellation when ε is too small. This case occurs when the test point A is very close to the ellipse $\mathcal{E}$. The limit case is for a test points lying exactly on the ellipse, for which $k=0$ is a root so $c=0$.

The cancellation can be avoided even at the limit case by using the dual expression of the root:

$$k_0=\frac{c}{b+\sqrt{b^2-ac}} \tag{7}$$

Since we know $b>0$, we know this expression is more stable than the other one in all cases.

外文资料截图

外文资料译文

快速计算点到椭圆的距离

L. Maisonobe

2003 年 9 月，修改版 05 年 5 月，最终版 06 年 2 月

1 引言

考虑如下二维问题：给定平面上的一个测试点 A 和椭圆 E，在椭圆上寻找一点使其到 A 点的距离最短。

这个问题在到多种情况下都会发生。第一是地理定位，行星体通常可以近似为两轴的椭球，也就是说椭球具有一个转动对称轴（极轴）；对于这种形状，经线平面是一个椭圆。给定一个三维笛卡尔坐标系的点坐标，我们想要计算经度、纬度和海拔高度。后面两个数据可以通过求解上面的问题得到。当计算三维球面在二维平面上的近似图像表征时，会遇到另外一种情况。为了建立一个以贝塞尔曲线为基础的近似椭圆的误差模型，成千上万的点到近似曲线的距离需要计算。后面这个情形将在在另外一篇文献中讨论。

2 问题描述

我们将会运用测地学中使用的标记法来描述这个问题。假设在已经利用 $\lambda = a\tan 2(y,x)$ 求解出经度 λ 的情形下，那么余下的问题将被限制在二维经线平面上，该平面的坐标为 $r = \sqrt{x^2 + y^2}$ 和 z.

假定测试点为 A，椭圆为 E。经过合适的坐标系变换后，测试点变换为了椭圆的标准坐标系下的坐标。该坐标系的原点是椭圆的中心，横轴沿着椭圆的长轴，纵轴沿着椭圆的短轴。A 点的坐标为 (r,z)。利用合适的坐标轴旋转，可是使得 $r \geq 0$ 。

在接下来的公式中，都采用了 $r \geq 0$ 这个假设。

椭圆采用如下定义方法：长半轴为 a_e，扁平度为 f 或者短半轴为 $a_p = a_e(1-f)$。

点到椭圆的距离 h 和椭圆上投影点的倾斜度 φ，在笛卡尔坐标系下式是相关的：

$$\begin{cases} r = \left(\dfrac{a_e}{\sqrt{1-f(2-f)\sin^2\varphi}} + h \right) \cos\varphi \\ z = \left(\dfrac{(1-f)^2 a_e}{\sqrt{1-f(2-f)\sin^2\varphi}} + h \right) \sin\varphi \end{cases} \tag{1}$$

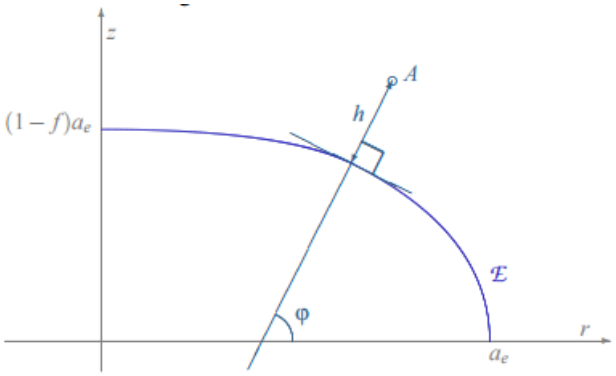


图 1 坐标定义

在地理定位领域，a_e 是球体的赤道半径；a_p 是到两极距离；h 是椭球面以上的海拔高度（当点在椭球面以下时，高度是负的）；φ 是地质学的中纬度。

当 h 和 φ 已知时，r 和 z 是很容易应用公式 1 求解的，但是在普遍情形下是行不通的。不过这个方法可以保留，用于当已知 h 为零时的特殊情况。当 $h = 0$ 时，

$$\varphi = a\tan 2\left(z, r(1-f)\right) \tag{2}$$

3 基础理论

3.1 处理特殊情形：中心或者椭圆上

对于椭圆中心这个特殊情形，我们的算法中将不予考虑。如果运用一个像这

样的测试$\sqrt{r^2+z^2}<\varepsilon_0$，这个点时是很容易被检测出来的。短轴的两端点是椭圆上到中心距离最小的点，我们不妨仅选取$\varphi>0:\varphi=\pi/2,h=-(1-f)a_e$情况下的点。

3.2 内外部检查

对于一个测试点A，是很容易判断该点是在椭圆的内部或者外部。可以通过判断下面表达式的正负来确定：

$$\left(\frac{r}{a_e}\right)^2+\left(\frac{z}{(1-f)a_e}\right)^2-1 \tag{3}$$

如果上式的符号是正的，那么该点在外部，反之则在内部。

3.3 直线椭圆的交点

考虑一条倾斜度为ζ的直线，其经过A点。

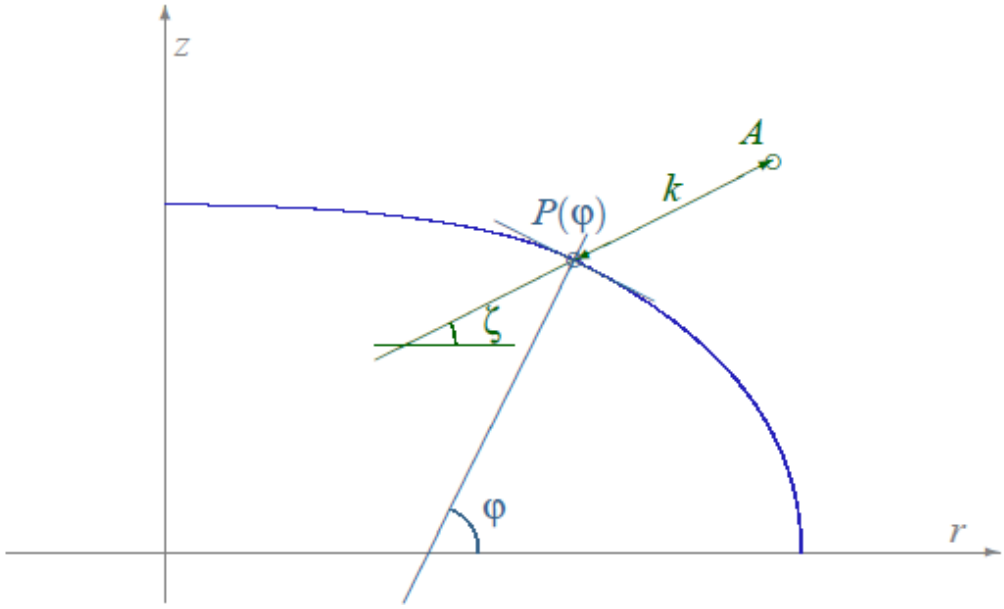


图 2 直线椭圆相交

选取合适的ζ可以使得，直线和椭圆相交，令交点为$P(\varphi)=(r',z')$，其在笛卡尔坐标系的具体值为：

$$\begin{aligned} r' &= r - k\cos\zeta \\ z' &= z - k\sin\zeta \end{aligned} \tag{4}$$

其中 k 是图中标记的 A 点与椭圆交点的距离。如果点在椭圆的外面，k 是正值，反之为负。注意，k 的符号已经包含了直线的方向选择，所以取决于测试点的相对于椭圆的内部或者外部，我们必须选择 ζ 或者 $\pi-\zeta$。接下来的公式将讨论关乎这个选择。因为交点 $P(\varphi)$ 位于椭圆上，所以满足

$$\left(\frac{r'}{a_e}\right)^2 + \left(\frac{z'}{(1-f)a_e}\right)^2 = 1$$

将测试点A的坐标和距离k表示的 r' 和 z' 代入

$$\begin{aligned} &\left(\frac{r-k\cos\zeta}{a_e}\right)^2 + \left(\frac{z-k\sin\zeta}{(1-f)a_e}\right)^2 = 1 \\ &\Leftrightarrow (1-f)^2\left[(r-k\cos\zeta)^2 - a_e^2\right] + (z-k\sin\zeta)^2 = 0 \\ &\Leftrightarrow ak^2 - 2bk + c = 0 \quad where \begin{cases} a = (1-f)^2\cos^2\zeta + \sin^2\zeta \\ b = (1-f)^2 r\cos\zeta + z\sin\zeta \\ c = (1-f)^2\left(r^2 - a_e^2\right) + z^2 \end{cases} \end{aligned} \tag{5}$$