**2022 – 2023 第一学期 实验报告**

课程编号： 2801000060， 课程名称：机器人学导论， 主讲教师： 邱国平

|  |  |  |
| --- | --- | --- |
| 学号：2022110131 | 姓名：廖祖颐 | 专业年级：电子信息工程大二 |

**Coursework 2 （20%）**

教师评语/成绩

手机拍摄的原始测试图像如图 1.1 所示。



图1.1 原始图像

原始图片尺寸为 300×365, 32bpp，大小: 138.93 k

完成这项任务大概分为以下几个步骤

**A. 初始图像处理**

1. **导入图像：**
   * 加载待处理的RGB彩色图像。
2. **计算灰度值：**
   * 通过加权RGB通道值获取灰度图像。权重分别为R通道的0.11、G通道的0.59和B通道的0.3。

**B. 图像二值化**

1. **自动阈值确定（大津法）：**
   * 使用大津法自动计算最佳阈值，用于将灰度图像转换为二值图像。
2. **标记像素：**
   * 将灰度图像中的像素根据阈值标记为0（黑色）或255（白色）。

**C. 图像分割**

1. **连通组件标记：**
   * 应用连通组件标记算法，识别并标记二值图像中的独立对象。
2. **用不同颜色标记不同对象：**
   * 对识别的每个独立对象使用不同的颜色进行标记，以便于区分。

**D. 绘制对象轮廓**

1. **确定边界点：**
   * 通过检查每个像素周围的像素，确定该像素是否为对象的边界点。
2. **用不同颜色标记不同对象的边界：**
   * 将识别的边界点用与对象相对应的颜色标记，形成清晰的边界轮廓。

**E. 计算最小二阶矩轴**

1. **计算面积的总二值：**
   * 计算每个对象区域内二值的总和，作为面积的度量。
2. **计算质心位置：**
   * 根据对象区域的二值分布，计算其质心（中心点）的位置。
3. **计算tanθ：**
   * 根据对象的空间分布，计算其最小二阶矩轴的方向（tanθ）。

**F. 绘制每个对象的最佳逼近椭圆**

1. **确定X轴上的搜索区域：**
   * 确定每个对象在X轴方向的搜索区域，用于找到椭圆的边界点。
2. **找到椭圆的端点‘E’和‘F’：**
   * 在确定的搜索区域内找到椭圆长轴的两个端点，标记为‘E’和‘F’。
3. **计算长短轴长度：**
   * 根据找到的端点和质心位置，计算椭圆的长轴和短轴长度，并绘制出椭圆。

# A. 初始图像处理

**a.** 导入图像

我们使用**cv2**库中的**imread**函数来导入图像，并获取一些重要的图像参数：

1. **pic\_size**：图像的尺寸，等于行数乘以列数，代表图像的总像素数。
2. **pic\_color**：一个大小为行数×列数×3的数组，由**imread**函数返回，存储图像的RGB值。

**b.** 通过加权RGB通道值计算灰度值

基于公式(1.1)，我们可以计算每个像素的灰度值，并将其存储在大小为行数×列数的数组**pic\_gray**中。得到**pic\_gray**后，我们创建一个图形，标题为“灰度图像”，并使用**matplotlib**的**imshow**函数显示灰度图像。

I(x,y) = 0.3 \* I\_R(x,y) +0.59 \* I\_G(x,y)+ 0.11 \* I\_B(x,y) (1.1)

第一部分的结果显示在图1.2中。

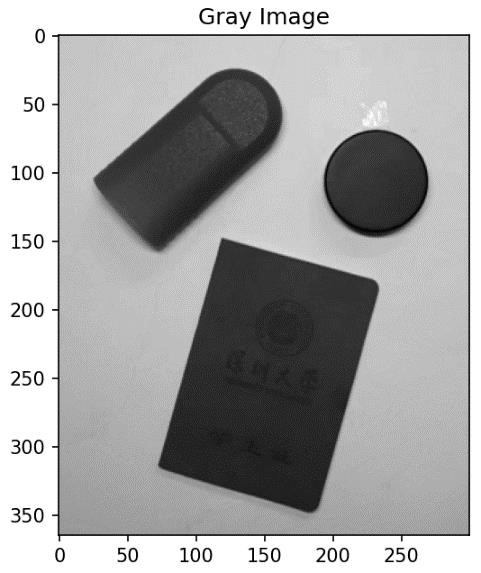


图1.2 灰度图像

# B. 图像二值化

**a.** 使用大津法自动获取最优阈值

大津法也被称为最大类间差异法，我们按照以下六个步骤执行此部分：

计算并输出灰度值分布的直方图：

直方图的水平表示为0至255的灰度值，垂直表示为图像中该灰度值的像素总数。

灰度值分布的直方图如图2.1所示：

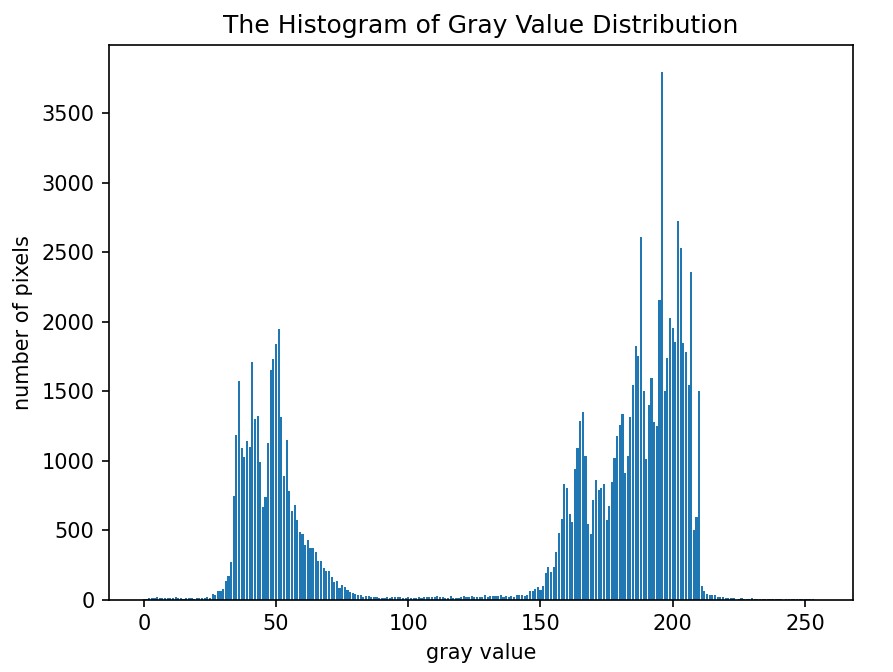


图2.1 灰度值分布直方图

1. 不同阈值下的像素分类： 遍历从0到255的阈值。在每种情况下，灰度值小于或等于阈值的像素为背景，灰度值大于阈值的像素为前景。计算背景像素和前景像素的数量。
2. 在不同阈值下，计算背景像素数量占总像素数的比例，用参数‘w0’表示，以及背景像素的平均灰度值，用参数‘u0’表示。
3. 在不同阈值下，计算前景像素数量占总像素数（pic\_size）的比例，用参数‘w1’表示，以及前景像素的平均灰度值，用参数‘u1’表示。
4. 计算图像中所有像素的平均灰度值，用参数‘u’表示。
5. 使用公式1.2计算类间方差，用参数‘v’表示，并找到最大的类间方差，用参数‘v\_max’表示，以及最适合的阈值。

v = w0 \* (u0 - u)2 + w1 \* (u1 - u)2  (1.2)

注意：考虑到当阈值较小时，背景像素的数量可能为零，使得u0的分母为零，无法计算。为了防止这种情况，我们必须添加一个条件：“如果w0\_temp == 0：u0 = 0”。对于u1也是如此。

**b**. 用0和255标记像素

对于二值图像，0代表背景，255代表前景。

对于我们的原始图像，背景是白色的，其灰度值为255。所以当一个像素的灰度值大于阈值时，我们给它赋值为0，否则赋值为255。像素的二值存储在数组‘pic\_array’中。

图像二值化的结果如图2.2所示：

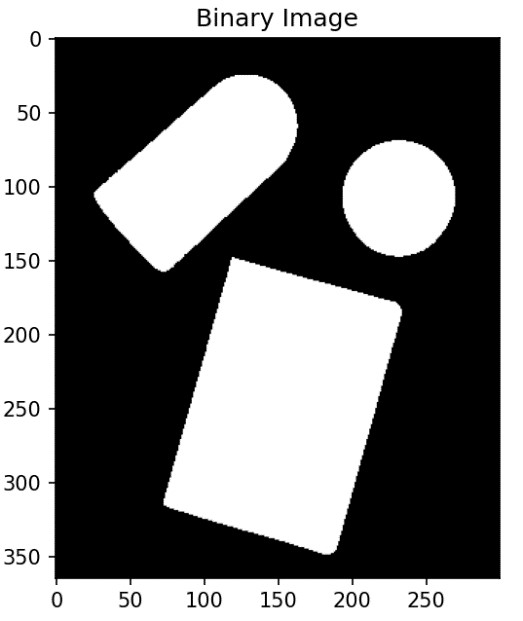


图2.2 二值图像

# C. 图像分割

**a.** 连通组件标记

为了区分不同的对象，我们可以通过两次标记像素的方法来实现：

(1) 第一次遍历

在程序中，我们有一个标签字典**label\_table**和一个数组**pic\_label**来存储像素的标签。

我们使用0表示背景像素，‘X’表示未判断的像素，‘A’和‘B’表示不同的对象标签，‘L’表示正在被标记的像素。

对于要标记的像素，有四种情况：

**Ⅰ.** 左边和上面的像素是背景

0

0

L

L = label + 1 我们将变量 **label** 的值加1。在程序中，**label** 是一个已经设置了初始值的参数。

在标签字典 **label\_table** 中，我们创建一个新的键 **L**，并将其值设为它本身。

**Ⅱ.** 左边像素为对象，上面像素为背景

0

A

L

L = A

**Ⅲ.** 上面的像素为对象，左边的像素为背景

B

0

L

L = B

**Ⅳ.** 左边和上面的像素是对象

B

A

L

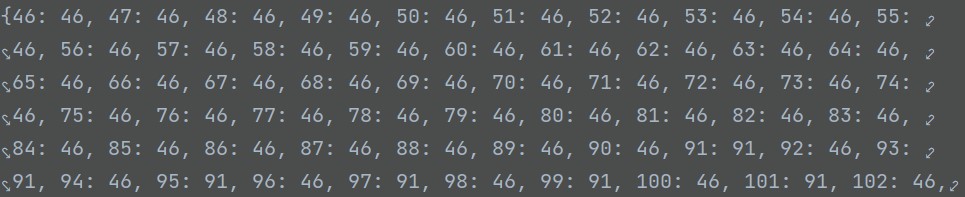
L = min(A,B)

在标签字典中，A 和 B 的对应值是它们中较小的一个。（例如，如下面的代码所示）：pic\_label[m, n] = min(pic\_label[m-1, n], pic\_label[m, n-1])

1. label\_table[pic\_label[m, n-1]] = label\_table[pic\_label[m-

1, n]] = min(label\_table[pic\_label[m-1, n]], label\_table[pic\_label[m, n-1]])

标签字典**label\_table**的部分结果如下所示（**label**的起始值为45）：

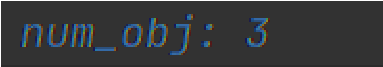


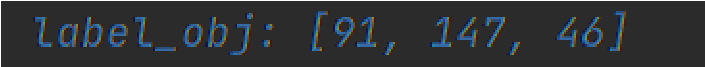
(2） 第二次遍历

Ⅰ. 给每个像素的标签赋予标签字典中该像素的值

Ⅱ. 在执行Ⅰ的同时，将标签值添加到列表**label\_obj**中，该列表用于记录对象的数量。通过使用**len**函数，我们可以知道图像中有多少个对象

在第二次遍历之后，我们得到了对象的数量及其标签，如下所示：





**b.** 用不同颜色标记不同的对象

一个大小为行数×列数×3的数组**pic\_color**用于存储不同对象的颜色。为了给每个对象一个独特的颜色，R/G/B通道的值是其标签值的（k+1）倍（如下面的代码所示）：

1.

pic\_color

=

np.zeros((row,

column,

3))

2.

**for**

i

**in**

range(row):

3.

**for**

j

**in**

range(column):

4.

**for**

k

**in**

range(3):

5.

**if**

pic\_label[i,

j]

==

255:

6.

pic\_color[i,

j,

k]

=

255

7.

**else**

:

8.

pic\_color[i,

j,

k]

=

pic\_label[i,

j]\*(k+1)

图像分割的结果展示在图3.1中

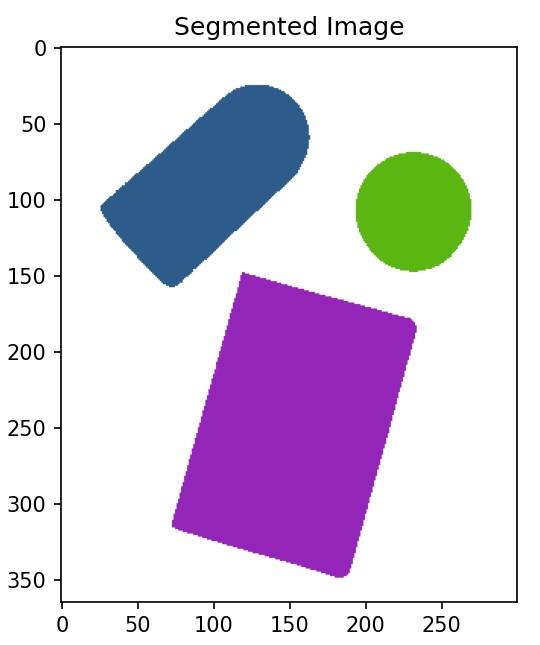


图3.1 分割后的图像

# 绘制对象轮廓

在这部分，我们有两个数组：**obj\_box**用于存储每个对象的上、下、左、右边界点，以及**pic\_bound**用于存储边界图像中像素的颜色。还有一个列表**b\_box**（边界框）用于存储每个对象的边界点。

1. 判断一个像素是否是边界点

如下图所示的像素窗口，只有当一个像素周围的8个相邻像素的标签值都等于该像素本身的标签值时，该像素才不被判断为边界点。如果一个像素被判断为边界点，它的坐标将被添加到**b\_box**中。

|  |  |  |
| --- | --- | --- |
| L | L | L |
| L | L | L |
| L | L | L |

1. 对边界框进行排序并记录每个对象的四个边界点

按行数的升序排列，我们可以得到上边界点。按行数的降序排列，我们可以得到下边界点。同样的方法，我们可以得到左右边界点。

（例如，如下面的代码所示）

* + 1. **def** take(elem):
    2. **return** elem[1]

3. **for** i **in** range(num\_obj):

* + 1. b\_box[i].sort() # 根据y坐标排序找到上边界点
    2. obj\_box[i, 0] = b\_box[i][0][0]
    3. b\_box[i].sort(reverse=True) # 根据y坐标倒序排序找到下边界点
    4. obj\_box[i, 1] = b\_box[i][0][0]
    5. b\_box[i].sort(key=take) # 根据x坐标排序找到左边界点
    6. obj\_box[i, 2] = b\_box[i][0][1]
    7. b\_box[i].sort(key=take, reverse=True) # 根据x坐标倒序排序找到右边界点
    8. obj\_box[i, 3] = b\_box[i][0][1]
    9. pic\_bound[obj\_box[i, 0]:obj\_box[i, 1], obj\_box[i, 2], 0] = 0 # 左边的边界
    10. pic\_bound[obj\_box[i, 0]:obj\_box[i, 1], obj\_box[i, 3], 0] = 0 # 右边的边界
    11. pic\_bound[obj\_box[i, 1], obj\_box[i, 2]:obj\_box[i, 3], 0] = 0 # 上方的边界
    12. pic\_bound[obj\_box[i, 0], obj\_box[i, 2]:obj\_box[i, 3], 0] = 0 # 下方的边界

obj\_box 的输出结果如下所示：

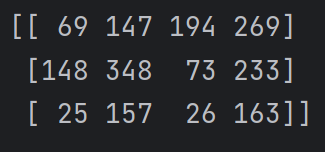
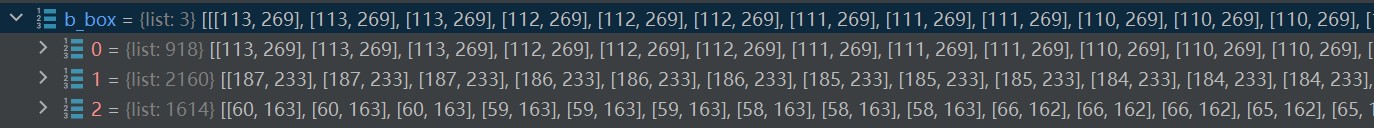


图4.1 边界点坐标

边界框的部分输出如下所示： 

边界图像的结果如图4.2所示

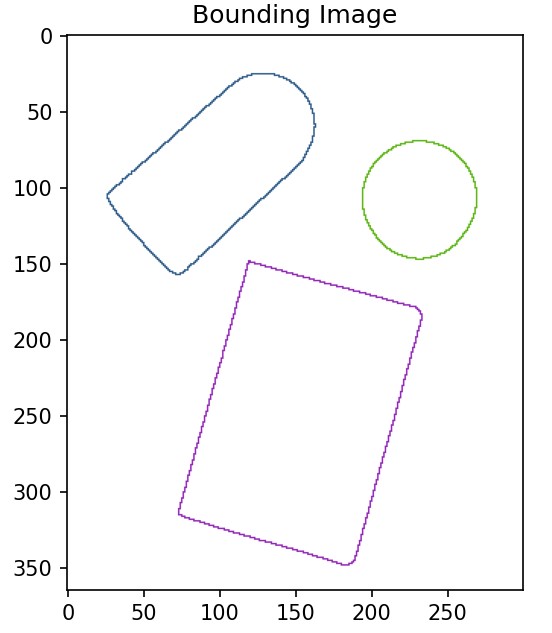
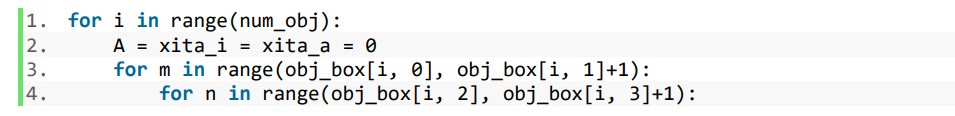


图4.2 边界图像

# E. 计算最小二阶矩轴

最小二阶矩轴的方程式为：*y*=-(*x*−*x-*)tan*θ*+*y*-

我们使用在部分D中找到的四个边界点为每个对象构建一个矩形搜索范围。（例如，如下面的代码所示）

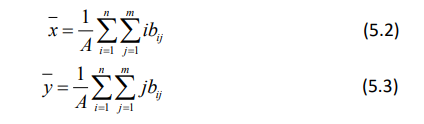


为了绘制最小二阶矩轴，我们按照以下三个步骤计算未知参数（在程序中，对应的参数分别是**Xa**、**Ya**和**tan\_xita\_i**）：

1. 计算A：区域的总二值，使用公式（5.1）

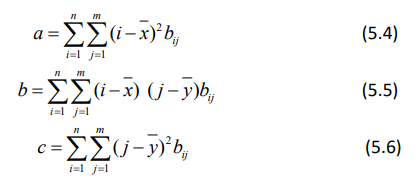


1. 计算（*x*ˉ,*y*ˉ）：使用公式（5.2-3）计算面积中心的位置

****

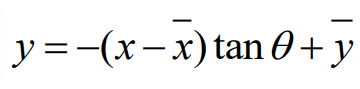
**c.** 计算

因为 ，我们需要使用公式（5.4-6）来计算a，b, c

**

通过推导，我得到：

如果(a-c)>0,那么 θ = arctan(b/(a-c)) ；如果 (a-c)<0,那么 θ = arctan(b/(a-c))+Π/2.

同时，因为最大二阶矩轴的方程是 ，我们也可以绘制最大二阶矩轴的直线。

# F. 绘制每个对象的最佳逼近椭圆

本实验中最佳近似椭圆的定义：椭圆的中心与对象的中心相同，长轴的长度等于沿最小二阶矩轴的对象长度，短轴的长度等于沿最大二阶矩轴的对象长度。

**a.** 确保在X轴上的搜索区域

如下图6.1和6.2所示，我绘制了一个任意对象及其最小二阶矩轴。有六种情况线与矩形相交，为方便起见，我只绘制了其中的两种。

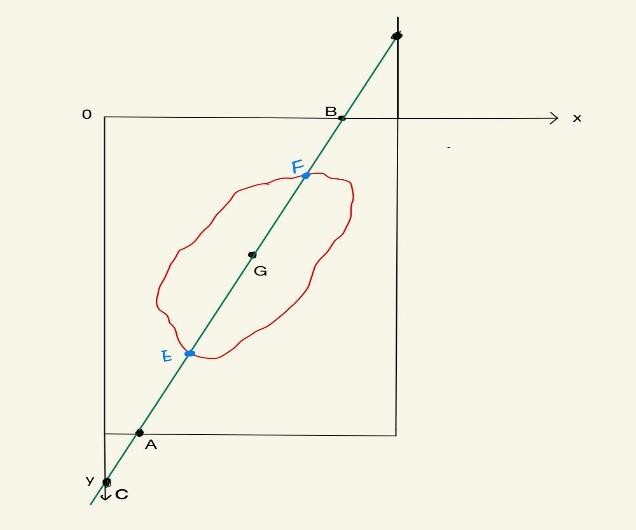
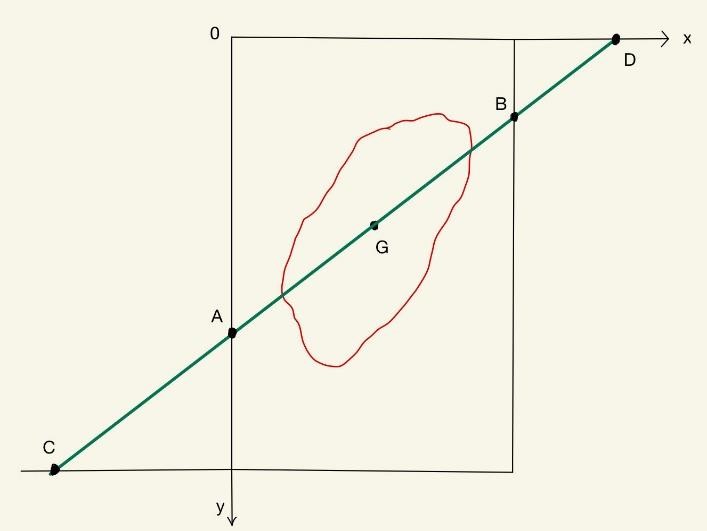


图6.1 图6.2

通过分析，我们可以发现线与矩形之间只有两个交点。如果交点在左/右边界上，那么搜索范围的开始/结束点就是左/右边界的水平坐标；如果交点在上/下边界上，那么搜索范围的起点是两个交点中水平坐标较小的那个，终点是水平坐标较大的那个。（如果一个点在上/下边界，另一个点在左/右边界，可以用相同的方法分析。）

经过进一步整理，搜索范围可以通过下面显示的代码确保：

1. xis = int(max(xis1, obj\_box[i, 2]))
2. xie = math.ceil(min(xis2, obj\_box[i, 3])) # 四舍五入

**b.** 在图中找到点'E'和'F'

从X轴的搜索范围中，我们可以计算它们的纵坐标。如果pic\_label[y,x]等于label\_obj[i]，则将此坐标x添加到列表'xi'中。最后，'xi'的最大值可能是点'F'的x坐标，'xi'的最小值可能是点'E'的x坐标。但在实验中，我发现当|tanθ|很大时，图6.3所示的情况可能会发生：

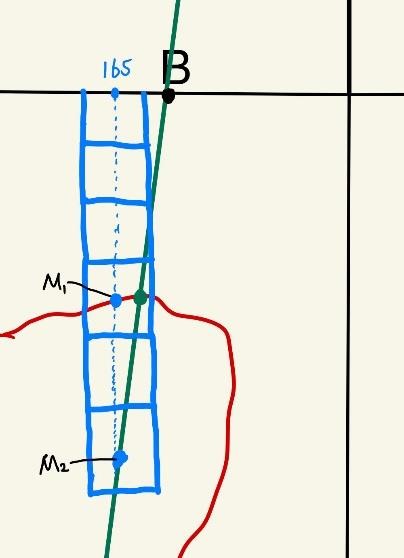


图 6.3

M1是我们想要的像素点，但由于图像坐标是离散的，而线的坐标是连续的，上述算法将M2视为点'F'。

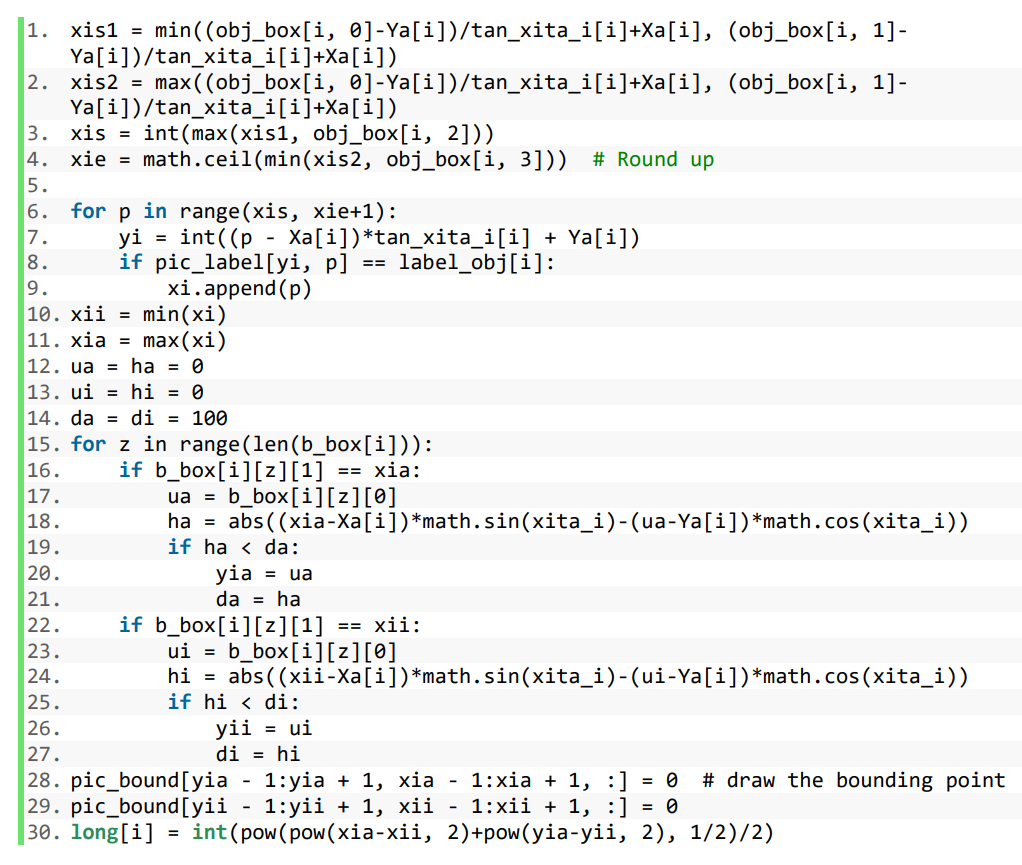
当(x1, y1)在目标区域内，而(x1+1, y2)不在区域内时，x1必须等于xF。因此，修改后的计划是在边界框中查找yF，而不是计算它。在边界框中，横坐标为xF的点中，距离线最近的点就是我们想要的点

**c.** 计算长轴和短轴的长度

在找到'E'点和'F'点之后，椭圆的长轴长度等于 。而我们可以沿着物体的第二时刻轴计算短轴的长度。

最后，我们可以使用库'cv2'中的'ellipse'函数，其旋转角度θ等于tan(θi)。。

该部分的代码如下所示：



我们可以同时绘制这两条线。具有最小二阶矩轴的边界图像如图6.4所示。

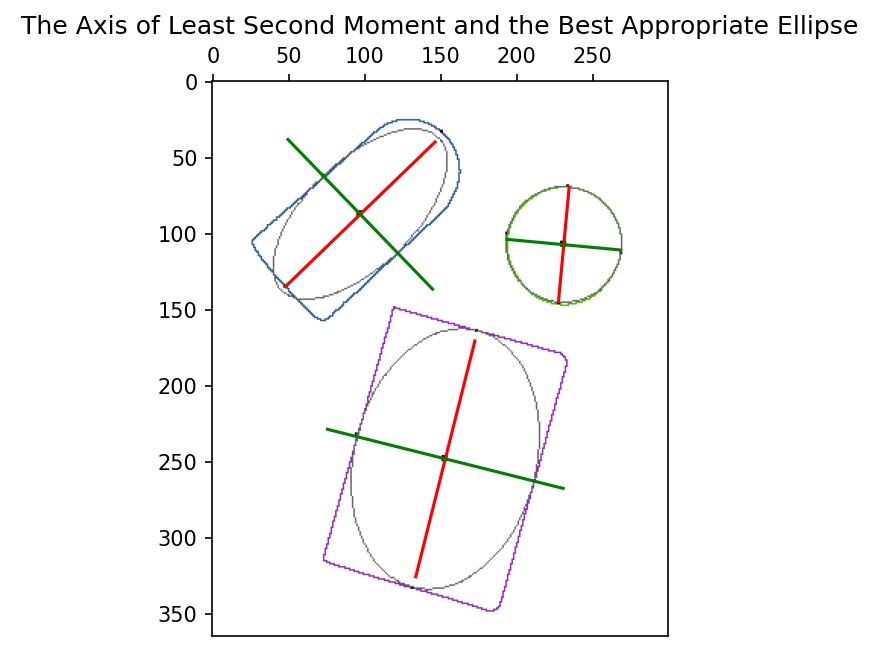


图6.4 最小二阶矩轴和最佳适应的椭圆。

红线表示最小二阶矩轴，绿线表示最大二阶矩轴。