

第 5 章

Python的图像处理

Python 有很强的图像处理能力，本章分别介绍应用 Pillow 模块和 Open CV 处理图像的方法。

5.1 图像像素的存储形式

为了方便介绍数据图像处理知识，首先要了解图像在计算机中的存储形式。

1. 灰度图

把黑白图像中的白色与黑色之间按对数关系分为若干等级，称为灰度。灰度图为单通道，一个像素块对应矩阵中的一个数字，数值为 0~255，其中 0 表示最暗（黑色），255 表示最亮（白色）。灰度图用矩阵表示如图 5.1 所示（为了方便起见，每列像素值都写一样了）。

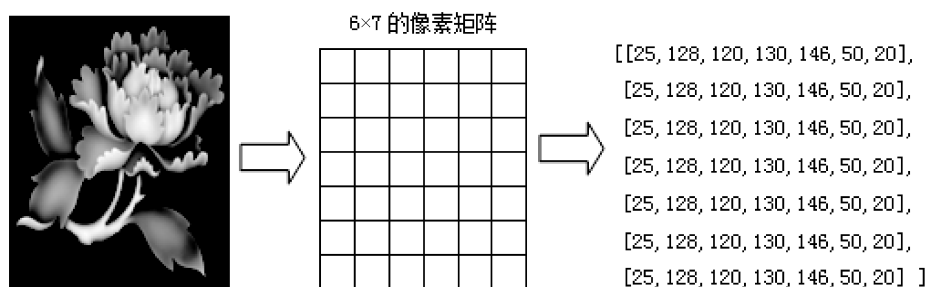


图 5.1 灰度图用矩阵表示

2. RGB 模式彩色图

RGB 模式彩色图用三维矩阵表示如图 5.2 所示。

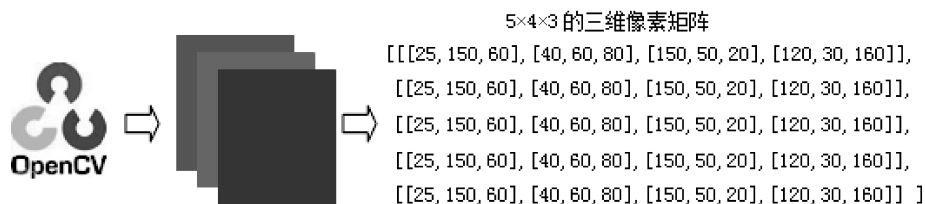


图 5.2 RGB 模式彩色图用三维矩阵表示



视频讲解

5.2 Pillow 模块处理图像

5.2.1 PIL 概述

PIL (Python Imaging Library, Python 图像处理库) 提供了通用的图像处理功能, 以及大量有用的基本图像操作, 如图像缩放、裁剪、旋转、颜色转换等。

1. 安装 Pillow 模块

PIL 仅支持到 Python 2.7 版本, Python 3.x 的 PIL 兼容版本称为 Pillow (Python 3.x 在使用引用模块语句时, 仍称为 PIL, 即 `import PIL`)。在命令行窗口中使用 `pip` 安装 Pillow 模块, 其命令为:

```
pip install pillow
```

安装过程如图 5.3 所示。

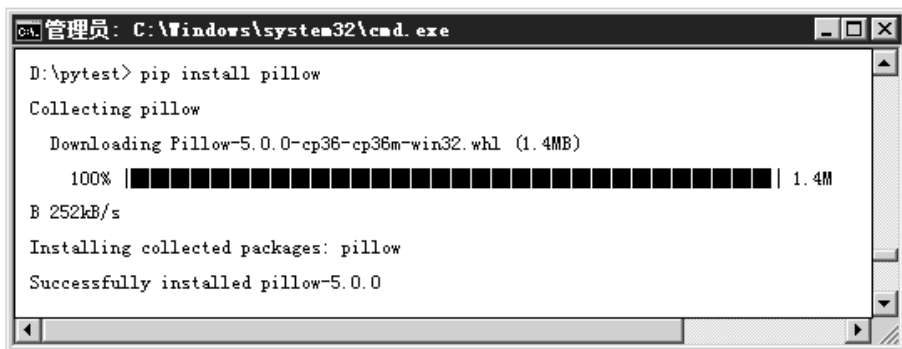


图 5.3 安装 Pillow 模块

2. Pillow 模块的方法

Pillow 模块提供了大量用于图像处理的方法, 通过创建的图像对象可以调用这些图像处理方法。Pillow 模块图像处理的常用方法如表 5.1 所示。

表 5.1 Pillow 模块图像处理的常用方法

方法	说明
<code>Image.open("图像文件名")</code>	打开图像文件, 返回图像对象
<code>show()</code>	显示图像
<code>Save("文件名")</code>	保存图像文件
<code>resize(宽高元组)</code>	图像缩放
<code>thumbnail()</code>	创建图像的缩略图
<code>rotate()</code>	旋转图像
<code>transpose(Image.FLIP_LEFT_RIGHT)</code>	图像水平翻转
<code>transpose(Image.FLIP_TOP_BOTTOM)</code>	图像垂直翻转
<code>crop(矩形区域元组)</code>	裁剪图像

续表

方法	说明
<code>paste(裁剪图像对象,矩形区域)</code>	粘贴图像
<code>ImageGrab.grab(矩形区域元组)</code>	屏幕截图, 若区域为空, 则表示全屏幕截图
<code>filter(ImageFilter.EDGE_ENHANCE)</code>	图像增强
<code>filter(ImageFilter.BLUR)</code>	图像模糊
<code>filter(ImageFilter.FIND_EDGES)</code>	图像边缘提取
<code>point(lambda i:i*r)</code>	图像点运算。 $r>1$, 图像变亮; $r<1$, 图像变暗
<code>format</code>	查看图像格式的属性值
<code>size</code>	查看图像大小的属性值, 格式为(宽度, 高度)
<code>getpixel(坐标元组)</code>	读取指定坐标点的像素的颜色值, 参数为(x,y)坐标元组, 返回值为红、绿、蓝三色分量的值
<code>putpixel((元组 1), (元组 2))</code>	元组 2 的值改变目标像素元组 1 的颜色值
<code>split()</code>	将彩色图像分离为红、绿、蓝三个分量通道。例如: <code>r, g, b = im.split()</code>
<code>Image.merge(im.mode, (r,g,b))</code>	将红、绿、蓝三个分量通道合并成一个彩色图像
<code>enhance(n)</code>	对比度增强为原来的 n 倍(n 为实数)。例如: <code>img = ImageEnhance.Contrast(img)</code> <code>img = img.enhance(1.5) # 对比度增强为原图的 1.5 倍</code>

5.2.2 PIL 的图像处理方法

利用 PIL 中的函数, 可以从大多数图像格式的文件中读取数据, 然后写入最常见的图像格式文件中。PIL 中常用的模块为 `Image` 和 `ImageTk`。`Image` 用于加载图像文件, 得到 PIL 图像对象, 而 `ImageTk` 模块负责对 PIL 对象进行各种图像处理。例如, 要读取一幅图像, 可以使用:

```
from PIL import Image
img = Image.open("img1.gif")
```

上述代码的返回值 `img` 是一个 PIL 图像对象。可以对这个 PIL 图像对象进行各种处理。下面介绍几个典型的图像处理的应用示例。

1. 图像的打开和显示

【例 5.1】 打开和显示图像示例。

程序代码如下:

```
import tkinter
from PIL import Image, ImageTk

win = tkinter.Tk()
win.title('图像显示')
win.geometry('300x300') # 定义窗体的大小 300 像素×300 像素

can = tkinter.Canvas(win, # 创建画布组件
```

```
bg='white',                                # 指定画布组件的背景色
width=300,                                  # 指定画布组件的宽度
height=300)                                # 指定画布组件的高度

image = Image.open("dukou.jpg")            # 打开图像文件
img = ImageTk.PhotoImage(image)            # 获取图像像素
can.create_image(160,120,image=img)        # 将图像添加到画布组件中
can.pack()                                  # 将画布组件添加到主窗口

win.mainloop()
```

程序运行结果如图 5.4 所示。



图 5.4 打开和显示图像

2. 建立图像的缩略图

使用 PIL 可以很方便地创建图像的缩略图。PIL 图像对象的 `thumbnail(size)` 方法将图像转换为由元组参数设定大小的缩略图。

【例 5.2】 建立图像缩略图示例。

程序代码如下：

```
import tkinter
from tkinter import Label
from PIL import Image, ImageTk
import os

win = tkinter.Tk()
win.title('建立图像缩略图')
```

```

win.geometry('200x200')                                # 定义窗体大小为 400 像素×200 像素

def imgshow():
    size = (64, 64)                                    # 设置缩略图尺寸的元组参数
    img = Image.open("dukou.jpg")
    img.thumbnail(size)
    img.save("dukou(1).jpg", "JPEG")                  # 保存缩略图为 dukou(1).jpg
    photo = ImageTk.PhotoImage(file='dukou(1).jpg')
    label = Label(win, image=photo)
    label.pack()
    label.image = photo

tkinter.Button(win, text='建立图像缩略图 ', command=imgshow).pack()
win.mainloop()

```

运行程序,单击“建立图像缩略图”按钮后,则将当前文件夹中名为 dukou.jpg 的图像文件生成 64×64 像素的缩略图,如图 5.5 所示。



图 5.5 生成图像缩略图

3. 增强图像处理

使用 PIL 可以很方便地对图像进行各种数字图像处理。例如,应用 filter()方法的 ImageFilter.EDGE_ENHANCE 属性可以将图像的对比度增强。

【例 5.3】 增加图像的对比度示例。

程序代码如下:

```

import tkinter
from tkinter import Label
from PIL import Image, ImageTk, ImageEnhance, ImageFilter

win = tkinter.Tk()
win.title('增强图像')
win.geometry('400x200')                                # 定义窗体大小为 400 像素×200 像素

photo = Image.open('dukou.jpg')
img1 = ImageTk.PhotoImage(photo)                      # 获取图像像素

```

```
label_1 = Label(win, image=img1) # 显示原图

def imgshow():
    img = photo.filter(ImageFilter.EDGE_ENHANCE)
    img2 = ImageTk.PhotoImage(img) # 获取图像像素
    label_2 = Label(win, image=img2).grid(row=1, column=1) # 显示增强后的图
    label_2.image = img2

button = tkinter.Button(win, text='增强图像处理', command=imgshow)

button.grid(row=0, column=0, columnspan=2)
label_1.grid(row=1, column=0)

win.mainloop()
```

程序运行结果如图 5.6 所示。



图 5.6 图像增强

5.3 Open CV 数字图像处理

Open CV 是 Open Source Computer Vision Library 的简称，是一个计算机视觉库。Open CV 可用于开发实时的图像处理、计算机视觉以及模式识别程序，目前使用十分广泛。



视频讲解

5.3.1 Open CV 模块的安装和导入

1. 安装 Open CV 模块

Python 的 Open CV 模块名为 opencv-python，在命令行窗口中使用 pip 安装 Open CV 模块，其命令为：

```
pip install opencv-python
```

安装过程如图 5.7 所示。

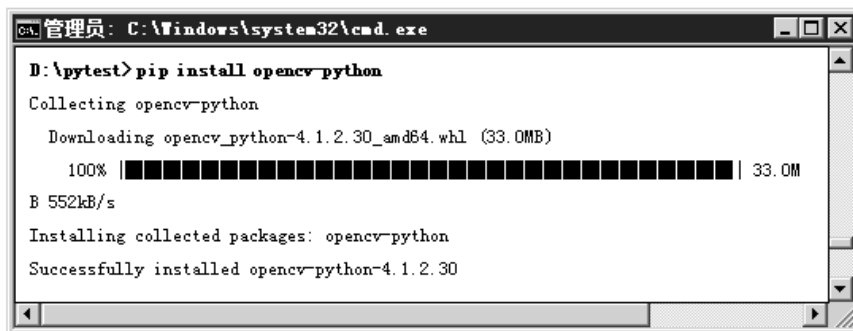


图 5.7 用 pip 安装 Open CV 模块的过程

2. Open CV 模块的导入

导入 Open CV 模块，其名称为 cv2，其导入语句如下：

```
import cv2
```

5.3.2 图像的读取、显示和保存

调用 Open CV 模块的相关函数，就可以很方便地对图像进行操作。下面介绍图像的加载、显示和保存函数及其使用方法。

1. 读取图像函数 imread()

Open CV 的 imread()函数可以读取图像文件，返回图像对象。其基本格式为

```
cv2.imread(img_path, flag)
```

该函数的参数含义如下：

- img_path: 图像的路径，即使路径错误也不会报错，但打印返回的图像对象为 None。
- flag:
 - cv2.IMREAD_COLOR，读取彩色图像，为默认参数，也可以传入 1。
 - cv2.IMREAD_GRAYSCALE，按灰度模式读取图像，也可以传入 0。
 - cv2.IMREAD_UNCHANGED，读取图像，包括其 alpha 通道，也可以传入 -1。

2. 显示图像函数 imshow()

Open CV 的 imshow()函数在自适应图像大小的窗口中显示图像。其基本格式为

```
cv2.imshow(window_name, img)
```

该函数的参数含义如下：

- window_name: 指定窗口的名字。
- img: 显示的图像对象。

该函数可以指定多个窗口名称，显示多个图像。

3. 保持窗体显示的函数

由于 `imread()` 函数显示图像的窗口会发生闪退，图像无法显示出来。因此，需要使用 `waitKey()` 使窗口保持显示状态。`waitKey()` 的格式为：

```
cv2.waitKey(milliseconds)
```

其中，参数 `milliseconds` 为设定时间数（毫秒），在该时间内等待键盘事件；当参数为 0 时，会无限等待。

通常与 `waitKey()` 配合使用的还有销毁窗口函数 `destroyAllWindows()`，其格式为：

```
cv2.destroyAllWindows(window_name)
```

其中，参数 `window_name` 为需要关闭的窗口。

4. 保存图像函数 `imwrite()`

保存图像函数的格式为：

```
cv2.imwrite((img_path_name, img)
```

其中，参数 `img_path_name` 为保存的文件名，`img` 为需要保存的图像对象。

【例 5.4】 使用 Open CV 模块打开和显示图像示例。

程序代码如下：

```
import cv2

img = cv2.imread('test.jpg', 0)    # 读取图像，参数 0 表示灰度
cv2.imshow('title', img)           # 显示图像，第 1 个参数为图像窗体的标题
cv2.imwrite('Grey_img.jpg', img)    # 保存灰度图像
cv2.waitKey(0)                     # 等待图像的关闭
cv2.destroyAllWindows()            # 关闭和销毁图像窗口
```

程序运行结果如图 5.8 所示。

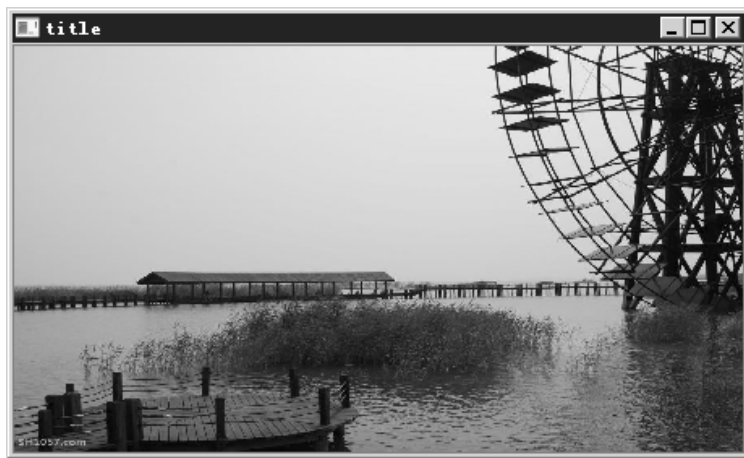


图 5.8 使用 Open CV 打开和显示图像

5.3.3 绘制基本几何图形



视频讲解

1. 在图像上绘制点和直线

1) 绘制点

在 Open CV 中, 当绘制一个半径很小的圆时, 就是一个点了。其函数为:

```
cv2.circle(img, center, radius, color[, thickness[, lineType[, shift]]])
```

函数的参数含义如下:

- **img**: 要画的圆所在的矩形或图像。
- **center**: 圆心坐标, 如 (100, 100)。
- **radius**: 半径, 如 10。
- **color**: 圆边框颜色, 如 (0, 0, 255) 表示红色, 为 BGR。
- **thickness**: 取正值时表示圆的边框宽度, 取负值时表示画一个填充圆形。
- **lineType**: 圆边框线型, 可为 0, 4, 8。
- **shift**: 圆心坐标和半径的小数点位数。

2) 绘制直线

绘制直线函数为:

```
cv2.line(img, pt1, pt2, color[, thickness[, lineType[, shift]]])
```

函数的参数含义如下:

- **img**: 要画的圆所在的矩形或图像。
- **pt1**: 直线的起点。
- **pt2**: 直线的终点。
- **color**: 线条的颜色, 如 (0, 0, 255) 表示红色, 为 BGR。
- **thickness**: 线条的宽度
- **lineType**: 取值 4 或 8 (-- 8: 8 连通线段, -- 4: 4 连通线段)。

【例 5.5】 在图像上绘制点和直线示例。

程序代码如下:

```
import cv2 as cv
point_size = 5
point_color = (0, 0, 255)          # BGR
thickness = 4                      # 可以为 0, 4, 8
img = cv.imread('xiamen.jpg', 1)
# 要画的点的坐标
points_list = [(80, 12), (320, 200)]
for point in points_list:
    cv.circle(img, point, point_size, point_color, thickness)
# 直线的起点和终点坐标
ptStart = (80, 14)
ptEnd = (318, 200)
lineType = 4
```

```
cv.line(img, ptStart, ptEnd, point_color, thickness, lineType)
cv.imshow('a_window', img)
cv.waitKey(0)
cv.destroyAllWindows()
```

程序运行结果如图 5.9 所示。



图 5.9 在图像上绘制点和直线

2. 在图像上绘制矩形和注释文字

1) 绘制矩形

绘制矩形函数为：

```
cv2.rectangle(img, pt1, pt2, color, thickness)
```

函数的参数意义如下：

- **img**: 指定的图片。
- **pt1**: 矩形左上角点的坐标。
- **pt2**: 矩形右下角点的坐标。
- **color**: 线条的颜色 (RGB) 或亮度 (灰度图像) (grayscale image)。
- **thickness**: 组成矩形的线条的粗细程度。取负值 (如 `CV_FILLED`) 时函数绘制填充了色彩的矩形。

2) 注释文字

显示文字函数为：

```
cv2.putText(img, str, origin, font, size, color, thickness)
```

其中，各参数依次是：图片、添加的文字、左上角坐标、字体、字体大小、颜色和字体粗细。

【例 5.6】 在图片上显示矩形框及文字。

程序代码如下：

```
import cv2 as cv

color1 = (0, 0, 255)
color2 = (255, 0, 0)
x = 140
y = 170
row = 150
col = 180
img = cv.imread('face.jpg', 1)
cv.rectangle(img, (x, y), (x+row, y+col), color1, 3)
cv.putText(img, 'face', (30, 150), cv.FONT_HERSHEY_COMPLEX, 3, color2,
11)

cv.imshow('a_window', img)
cv.waitKey(0)
cv.destroyAllWindows()
```

程序运行结果如图 5.10 所示。



图 5.10 在图像上绘制矩形框和文字

5.4 案例精选



5.4.1 用画布绘制图形

画布 Canvas 是图形用户界面 tkinter 的组件，它是一个矩形区域，用于绘制图形或作视频讲解 为容器放置其他组件。

Canvas 对象包含了大量的绘图方法，表 5.2 列出了常用的绘图方法。

表 5.2 Canvas 对象常用的绘图方法

方法	说明
<code>create_line(x1, y1, x2, y2)</code>	绘制一条从(x1,y1)到(x2,y2)的直线
<code>create_rectangle(x1, y1, x2, y2)</code>	绘制一个左上角为(x1,y1)、右下角为(x2,y2)的矩形
<code>create_polygon(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6)</code>	绘制一个顶点为(x1,y1)、(x2,y2)、(x3,y3)、(x4,y4)、(x5,y5)、(x6,y6)的多边形
<code>create_oval(x1, y1, x2, y2, fill='color')</code>	绘制一个左上角为(x1,y1)、右下角为(x2,y2)的外接矩形所包围的圆, fill 为填充颜色
<code>create_arc(x1, y1, x2, y2, start=s0, extent=s)</code>	绘制在左上角为(x1,y1), 右下角为(x2,y2)的外接矩形所包围的一段圆弧, 圆弧角度为 s, 从 s0 开始
<code>create_image(w, h, anchor=NE, image=filename)</code>	在 w 宽、 h 高的矩形区域内显示文件名为 filename 的图像
<code>move(obj, x, y)</code>	移动组件 obj。x 为水平方向变化量, y 为垂直方向变化量

【例 5.7】 绘制笑脸。

程序代码如下:

```
import tkinter
import tkinter.messagebox
win = tkinter.Tk()
win.title('画布示例')
win.geometry('250x250')

can = tkinter.Canvas(win, height=250, width=250)      # 定义画布
io1 = can.create_oval(35,30,210,210, fill='yellow')  # 画一个黄色的圆
io2 = can.create_oval(70,70,180,180, fill='black')
io3 = can.create_oval(65,70,185,170, outline='yellow', fill='yellow')
io4 = can.create_oval(80,100,110,130, fill='black')
io5 = can.create_oval(150,100,180,130, fill='black')

can.pack()
win.mainloop()
```

程序运行结果如图 5.11 所示。

【例 5.8】 用方向键移动矩形块。

tkinter 的画布 Canvas 类可以用于设计简单动画, 使用 `move(tags、dx、dy)` 方法实现移动图片或文字等组件。Canvas 的 `update()` 方法为刷新界面, 重新显示画布。

程序代码如下:

```
import time
from tkinter import *

x = 50
```

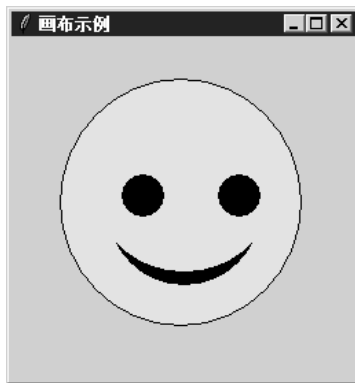


图 5.11 绘制笑脸

```

y = 50
# (1) 定义窗口
win = Tk()
win.title("移动小矩形块")
# (2) 定义画布
canvas = Canvas(win,width=400,height=400)
canvas.pack()          # 显示画布
# (3) 定义矩形块
rect = canvas.create_rectangle(x, y, x+30, y+30, fill='red')
print(rect)
# (4) 定义移动小矩形的函数
def moveRect(event):
    if event.keysym == 'Up':
        canvas.move(rect, 0, -3)
    elif event.keysym == 'Down':
        canvas.move(rect, 0, 3)
    elif event.keysym == 'Left':
        canvas.move(rect, -3, 0)
    elif event.keysym == 'Right':
        canvas.move(rect, 3, 0)
    win.update()        # 界面刷新
    time.sleep(0.05)    # 休眠

# (5) 绑定方向键
canvas.bind_all('<KeyPress-Up>', moveRect)
canvas.bind_all('<KeyPress-Down>', moveRect)
canvas.bind_all('<KeyPress-Left>', moveRect)
canvas.bind_all('<KeyPress-Right>', moveRect)

win.mainloop()

```

keysym == 键值 (方向键)
move(组件, x 坐标增量, y 坐标增量)

绑定键盘事件

程序运行结果如图 5.12 所示。

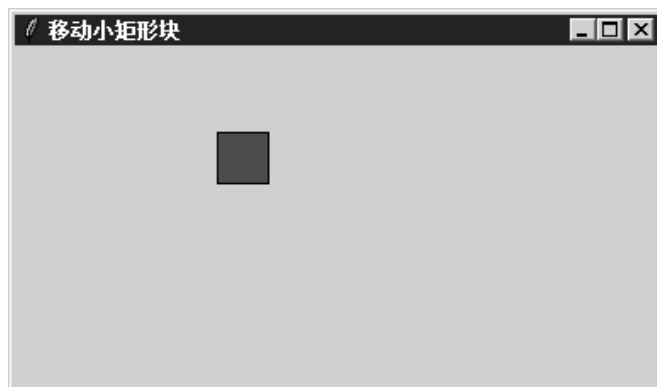


图 5.12 用方向键移动小矩形块

5.4.2 识别二维码及条形码

pyzbar 是 Python 的一个开源库，用于扫描、识别二维码和条形码信息。用 pip 安装 pyzbar 库，其命令为：



视频讲解

```
pip install pyzbar
```

【例 5.9】 设有条形码图片 bar_code.jpg 和二维码图片 two_bar_code.jpg，编写一个识别二维码及条形码的程序。

程序代码如下：

```
from pyzbar import pyzbar
import matplotlib.pyplot as plt
import cv2

# 条码定位及识别
def decode(image, barcodes):
    # 循环检测到的条形码
    for barcode in barcodes:
        # 提取条形码的边界框的位置
        # 画出图像中条形码的边界框
        (x, y, w, h) = barcode.rect
        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 5)
        # 条形码数据为字节对象，所以如果想在输出图像上
        # 画出来，就需要先将它转换为字符串
        barcodeData = barcode.data.decode("utf-8")
        barcodeType = barcode.type
        # 绘出图像上条形码的数据和条形码类型
        text = "{} ({}).format(barcodeData, barcodeType)
        cv2.putText(image, text, (x, y - 10),
        cv2.FONT_HERSHEY_SIMPLEX, .8, (255, 0, 0), 2)
        # 向终端打印条形码数据和条形码类型
        print("[INFO] Found {} barcode: {}".format(barcodeType,
        barcodeData))
        plt.figure(figsize=(10,10))
        plt.imshow(image)
        plt.show()

# (1) 读取条形码图片
image = cv2.imread('tiaoma.jpg')
# 找到图像中的条形码并进行解码
barcodes = pyzbar.decode(image)
# 识别条形码
decode(image, barcodes)
```

```
# (2) 读取二维码图片
image = cv2.imread('zsmma.jpg')
# 找到图像中的二维码并进行解码
barcodes = pyzbar.decode(image)
# 识别二维码
decode(image, barcodes)
```

程序运行结果:

[INFO] 识别 EAN13 barcode: 6902265501114

[INFO] 识别 QRCODE barcode: <http://qr6l.cn/odmOqs/qAI1Bft>

显示结果如图 5.13 所示。

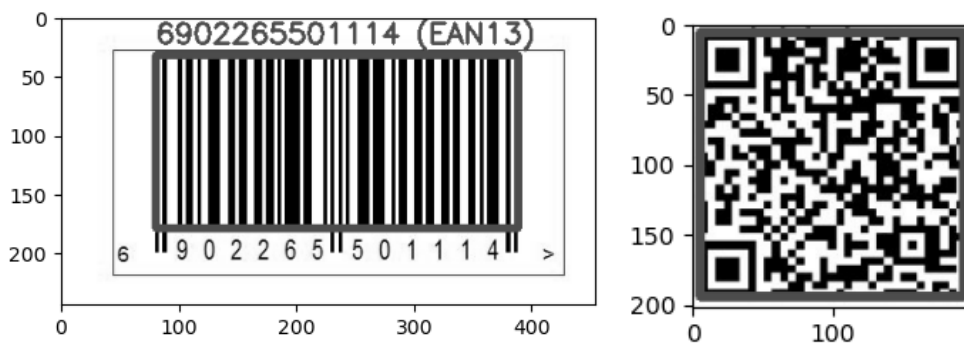


图 5.13 识别条形码及二维码

5.4.3 无人驾驶汽车车道线检测



视频讲解

车道线检测是无人驾驶汽车的一项重要技术。有效地获取车道线信息,对无人驾驶汽车的决策有至关重要的作用。

车道线检测主要步骤及主要算法说明如下。

1. 将视频流转换为一帧帧的图像

在本案例中,为了简化程序,将其省略为加载图像。

```
src = cv2.imread('gaosu_lane.jpg')
```

2. 对图像进行去噪处理

使用高斯滤波对图像进行去噪处理。

```
src1 = cv2.GaussianBlur(src, (5,5), 0, 0)
```

3. 把图像转换为灰度图

```
src2 = cv2.cvtColor(src1,cv2.COLOR_BGR2GRAY)
cv2.imshow('huidu',src2)
```

其效果如图 5.14 所示。



图 5.14 灰度图

4. 边缘处理，提取图像的轮廓

```
src3 = cv2.Canny(src2,lthrehlod,hthrehlod)
cv2.imshow('bianyuan',src3)
```

其效果如图 5.15 所示。



图 5.15 图像的轮廓

5. 保留感兴趣的区域并提取轮廓图中的直线

因为在实际应用中摄像头固定在车上，所拍摄的图像中特定的部分包含车道线，它一般都位于图片的中下部，所以只需要对这个区域进行处理即可；然后使用霍夫变换原理，提取轮廓图中的直线。

```
regin = np.array([[0,src.shape[0]],(460,325),
(520,325),(src.shape[1],src.shape[0])])
mask = np.zeros_like(src3)
mask_color = 255 # src3 图像的通道数是 1, 且是灰度图像, 所以颜色值为 0~255
cv2.fillPoly(mask,regin,mask_color)
```

```
src4 = cv2.bitwise_and(src3,mask)
cv2.imshow('bianyuan',src4)
```

其效果如图 5.16 所示。



图 5.16 在感兴趣区域提取轮廓图中的直线

6. 优化处理并画出车道线

在第 5 步中提取到的图像中包含了很多直线，其中有很多直线是不需要的，这就需要去掉冗余的直线。具体的办法如下：首先对直线点集根据斜率的正负分成左右两类直线集；然后再分别对左右直线集进行处理。计算直线集的平均斜率和每条直线的斜率与平均斜率的差值，求出其中差值最大的直线，判断该直线的斜率是否大于阈值，如果大于阈值，就将其去除，然后对剩下的直线继续进行相同的操作，直到满足条件为止。

经过优化处理，得到的直线集依然很多，但是范围已经很小了，进一步采用最小二乘拟合的方式，将这些都用直线拟合，得到最后的左右车道线。

```
good_leftlines = choose_lines(lefts, 0.1)      # 左边优化处理后的点
good_rightlines = choose_lines(rights, 0.1)    # 右边优化处理后的点

leftpoints = [(x1,y1) for left in good_leftlines for x1,y1,x2,y2 in left]
leftpoints = leftpoints+[(x2,y2) for left in good_leftlines for x1,y1,x2,y2
in left]
rightpoints = [(x1,y1) for right in good_rightlines for x1,y1,x2,y2 in right]
rightpoints = rightpoints+[(x2,y2) for right in good_rightlines for x1,y1,x2,y2
in right]

lefttop = clac_edgepoints(leftpoints,325,src.shape[0]) # 左右车道线的端点
righttop = clac_edgepoints(rightpoints,325,src.shape[0])

src6 = np.zeros_like(src5)

cv2.line(src6,lefttop[0],lefttop[1],linecolor,linewidth)
```

```
cv2.line(src6, righttop[0], righttop[1], linecolor, linewidth)
```

```
cv2.imshow('onlylane', src6)
```

其效果如图 5.17 所示。

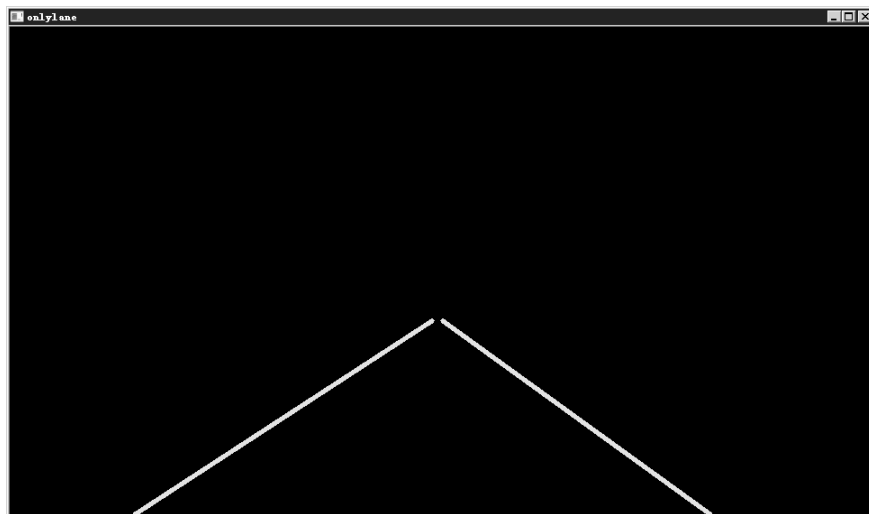


图 5.17 画出车道线

7. 图像合成

将所画的车道线与原图像进行叠加，得到合成的图像。

```
src7 = cv2.addWeighted(src1, 0.8, src6, 1, 0)
```

```
cv2.imshow('Finally Image', src7)
```

其效果如图 5.18 所示。



图 5.18 合成后的图像

8. 完整的无人驾驶汽车车道线检测程序代码

【例 5.10】 无人驾驶汽车车道线检测程序。

程序代码如下：

```
import cv2
import numpy as np

# 读取图片
src = cv2.imread('line.jpg')

# 高斯降噪
src1 = cv2.GaussianBlur(src, (5,5), 0, 0)
# cv2.imshow('gaosi', src1)

# 灰度处理
src2 = cv2.cvtColor(src1, cv2.COLOR_BGR2GRAY)
# cv2.imshow('huidu', src2)

# 边缘检测
lthrehlod = 50
hthrehlod =150
src3 = cv2.Canny(src2, lthrehlod, hthrehlod)
# cv2.imshow('bianyuan', src3)

# ROI 划定区间,并将非此区间变成黑色
regin = np.array([[ (0,src.shape[0]), (460,325),
(520,325), (src.shape[1], src.shape[0])]])
mask = np.zeros_like(src3)
mask_color = 255      # src3 图像的通道数是 1, 且是灰度图像, 所以颜色值为 0~255
cv2.fillPoly(mask,regin, mask_color)
src4 = cv2.bitwise_and(src3, mask)
# cv2.imshow('bianyuan', src4)

# 利用霍夫变换原理找出图中的像素点组成的直线, 然后画出来
rho = 1
theta = np.pi/180
threshold =15
minlength = 40
maxlengthgap = 20
lines = cv2.HoughLinesP(src4, rho, theta, threshold, np.array([]),
minlength, maxlengthgap)
# 画线
linecolor =[0, 255, 255]
linewidth = 4
src5 = cv2.cvtColor(src4, cv2.COLOR_GRAY2BGR)    # 转换为三通道的图像
```

```

lefts = []
rights = []
for line in lines:
    for x1,y1,x2,y2 in line:
        # cv2.line(src5, (x1,y1), (x2,y2), linecolor, linewidth)
        # 分左右车道
        k = (y2-y1)/(x2-x1)
        if k<0:
            lefts.append(line)
        else:
            rights.append(line)

# 优化处理
def choose_lines(lines, threshold):
    # 过滤斜率差别较大的点
    slope = [(y2-y1)/(x2-x1) for line in lines for x1,x2,y1,y2 in line]
    while len(lines) > 0:
        mean = np.mean(slope) # 平均斜率
        diff = [abs(s- mean) for s in slope]
        idx = np.argmax(diff)
        if diff[idx] > threshold:
            slope.pop(idx)
            lines.pop(idx)
        else:
            break

    return lines

def clac_edgepoints(points, ymin, ymax):
    # 寻找直线的端点
    x = [p[0] for p in points]
    y = [p[1] for p in points]

    k = np.polyfit(y, x, 1)
    func = np.polyld(k)
    xmin = int(func(ymin))
    xmax = int(func(ymax))

    return [(xmin, ymin), (xmax, ymax)]

good_leftlines = choose_lines(lefts, 0.1) # 处理后的点
good_rightlines = choose_lines(rights, 0.1)

leftpoints = [(x1,y1) for left in good_leftlines
for x1,y1,x2,y2 in left]
leftpoints = leftpoints+[(x2,y2) for left in good_leftlines
for x1,y1,x2,y2 in left]

```

```

rightpoints = [(x1,y1) for right in good_rightlines
for x1,y1,x2,y2 in right]
rightpoints = rightpoints+[(x2,y2) for right in good_rightlines
for x1,y1,x2,y2 in right]

leftttop = clac_edgepoints(leftpoints,325,src.shape[0]) # 左车道线的端点
rightttop = clac_edgepoints(rightpoints,325,src.shape[0]) # 右车道线的端点

src6 = np.zeros_like(src5)
cv2.line(src6, leftttop[0], leftttop[1], linecolor, linewidth)
cv2.line(src6, rightttop[0], rightttop[1], linecolor, linewidth)

#cv2.imshow('onlylane',src6)

# 图像叠加
src7 = cv2.addWeighted(src1, 0.8, src6, 1, 0)
cv2.imshow('Finally Image', src7)

cv2.waitKey(16000) # 等待图片的关闭
cv2.destroyAllWindows() # 关闭和销毁图片窗口

```

习 题 5

1. 绘制一个带阴影的小矩形块。
2. 设计一个图片浏览器，单击“上一张”按钮，则显示前一张图片；单击“下一张”按钮，则显示后一张图片。
3. 编写一个程序，显示图像的轮廓，如图 5.19 所示。



(a) 原图



(b) 轮廓图

图 5.19 显示图像的轮廓