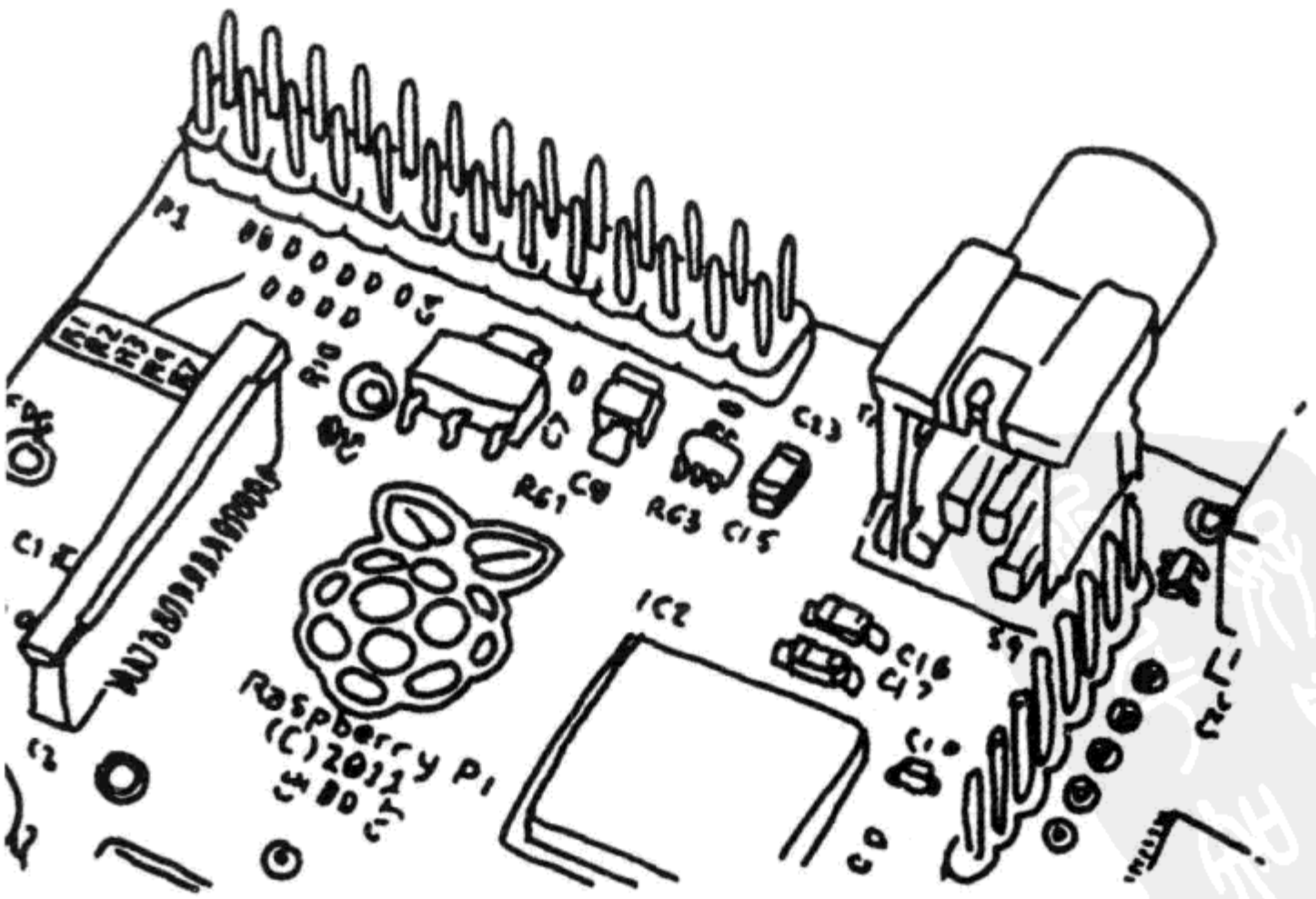
第 9 章

使用摄像头

Working with Webcams



使用像 Raspberry Pi 这样的平台开发各种项目有一个很大的优势——可以直接使用各种 USB 设备。你不但可以连接键盘、鼠标，还可以接连打印机、无线网卡、指纹识别器、读卡器、摄像头或移动硬盘等其他外设。在这一章中，我们会演示在 Raspberry Pi 上使用摄像头的一些方法。

虽然不像键盘和鼠标常见，摄像头也已经成为主流电脑的一个标准配置。这对大家来说都是一个好消息，因为这意味着我们可以用不到 25 美元的价格很容易买到一个知名品牌的摄像头，如果买非知名品牌的还可以更便宜。更重要的是，很多 USB 摄像头在 Linux 下都不需要额外的驱动程序就可以正常工作。

也许你还记得在第 1 章介绍主板接口时，我们提到过用于连接摄像头的 CSI 接口（图 9.1）。由于 Pi 所使用的 Broadcom 芯片组原本是为手机和平板电脑设计的，CSI 接口就是这些移动设备厂商

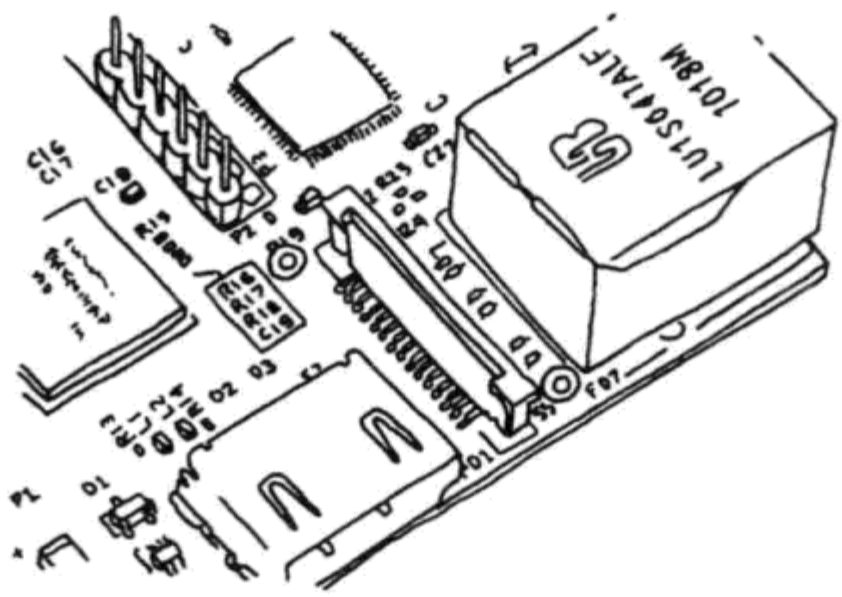


图9.1　Raspberry Pi的CSI接口

用来连接摄像头的接口。遗憾的是，作为普通消费者，CSI 接口的摄像头并不容易直接在商店中买到。截至本书出版时，Raspberry Pi 基金会仍在开发可以直接连接到 CSI 接口的摄像头[①]。在这种摄像头面市之前，我们仍然建议先使用 USB 摄像头（图 9.2）。

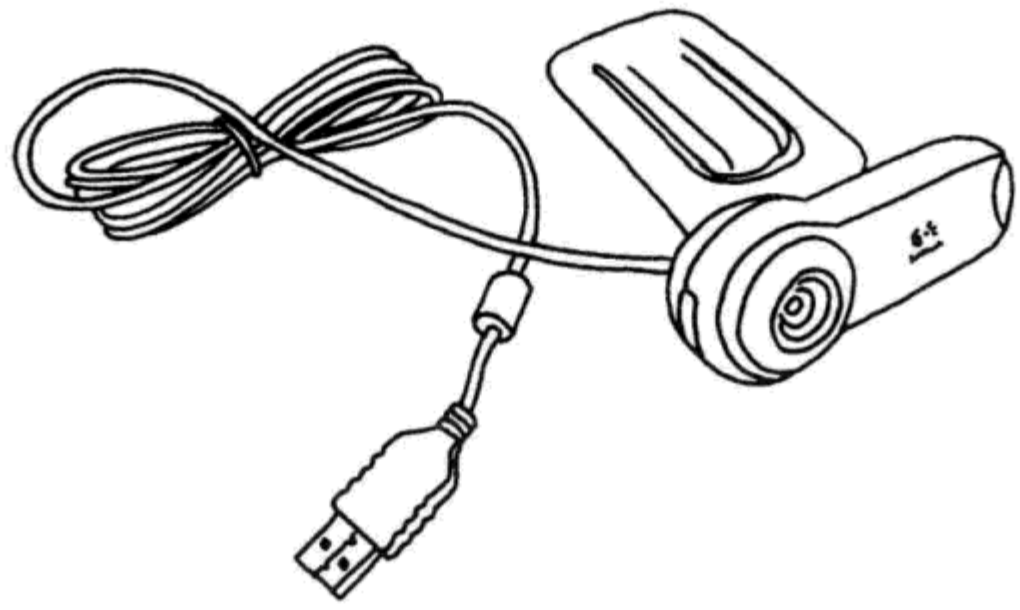


图9.2　USB摄像头

测试摄像头

市面上有很多型号的摄像头，没办法保证每一种都能直接在 Raspberry Pi 上正常工作。如果你准备购买某个型号的摄像头，最好先上网看看有没有他人成功使用这种摄像头的先例。你也可以参考 Raspberry Pi 已知可用的外设列表中摄像头相关的内容（*http://elinux.org/RPi_VerifiedPeripherals#USB_Webcams*）。

注意，你需要把摄像头和键盘、鼠标一起通过一个有源的 USB Hub 连接到 Raspberry Pi。必须使用有源的 USB Hub 是因为，Raspberry Pi 的 USB 接口只能提供有限的电流，这点电流很可能不

① 适用于 Raspberry Pi 的 CSI 接口摄像头已于 2013 年 5 月 14 日正式发布，请参考：*http://www.raspberrypi.org/archives/3890*。——译者注

足以让摄像头和键盘、鼠标一起正常工作。有源的 USB Hub 可以从独立电源获得电流给设备供电，就不再需要由 Raspberry Pi 自身来提供超出它供电能力的电流了。

如果你已经拥有了一个摄像头并准备在 Raspberry Pi 上开展测试，可以先在命令行上用 apt-get 命令来安装一个称为 luvcview 的摄像头图像预览程序：

```
pi@raspberrypi ~ $ sudo apt-get install luvcview
```

用 apt-get 完成程序的安装后，在图形化桌面环境的终端中输入 luvcview 运行这个程序。luvcview 会打开新的窗口显示它从 */dev* 目录下找到的第一个视频设备（通常是 */dev/video0*）中采集到的图像。画面的尺寸会在终端上显示出来。如果显示的视频图像上有一些波纹，你可以试着缩小画面的尺寸。例如，如果默认的视频大小是 640 × 480，你可以关闭并重新运行 luvcview，在命令行参数中把画面尺寸设置成原来的一半：

```
pi@raspberrypi ~ $ luvcview -s 320x240
```

如果无法看到视频画面，你需要在这时候就把问题原因找到。排查问题的方法之一是，先把摄像头断开再插上，并同时用 dmesg 命令观察系统所显示的诊断信息，这些信息也许可以为你提供一些线索。

安装并测试 SimpleCV

我们将在 Python 中使用 SimpleCV 库（图 9.3）来操作摄像头，SimpleCV 是与电脑视觉相关的开源库。通过使用 SimpleCV，我们可以很方便地从摄像头采集图像并显示在屏幕上或保存为文

件，但 SimpleCV 最强大的功能在于它所包含的计算机视觉相关的算法，可以用来实现一些让人惊叹的效果。除了基本的图像变换，SimpleCV 还可以用来跟踪、检测和识别图像或视频中的特定对象。后面我们会完成一个使用 SimpleCV 实现的人脸检测程序，见“人脸检测”一节。

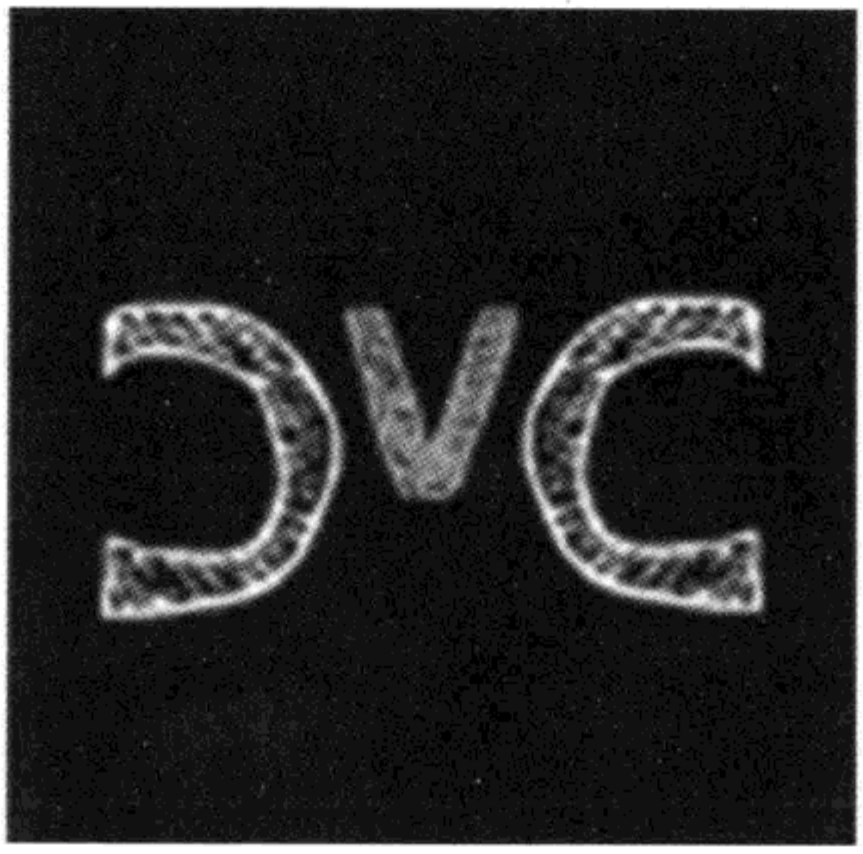


图9.3　SimpleCV的标识

在安装 SimpleCV 前，先要安装一些它所依赖的库，可以用 apt-get 来完成安装：

```
    pi@raspberrypi ~ $ sudo apt-get install python-opencv
python-scipy python-numpy python-pip
```

要安装的程序比较多，所以安装过程中可能需要耐心等待一下。下一步，真正开始安装 SimpleCV 库：

```
    pi@raspberrypi ~ $ sudo pip install https://github.com/
ingenuitas/SimpleCV/zipball/master
```

当安装完成后，可以用 Python 的交互式命令行通过导入这个库来验证安装是否成功：

```
    pi@raspberrypi ~ $ python
    Python 2.7.3rc2 (default, May 6 2012, 20:02:25)
    [GCC 4.6.3] on linux2
    Type "help", "copyright", "credits" or "license" for more
information.
    >>> import SimpleCV
    >>>
```

如果库的导入一切顺利，你就已经完成了准备工作，可以开始在 Raspberry Pi 上开展计算机视觉实验。

显示图片

本章中的很多实验都需要在图形化桌面环境下完成，因为这样你才能在屏幕上显示图片。你可以用 IDLE，也可以直接用 Leafpad 把代码保存为 *.py* 文件，然后再从终端窗口中运行。

下面我们一起来了解一些 SimpleCV 操作图像的基础知识，从中可以学到如何从摄像头上捕获图像。只有当你学会了如何从摄像头上捕获图像，后面才能尝试进行人脸检测。

1. 在你的主目录中创建一个新的目录，名为 *simplecv-test*。

2. 打开 Midori 浏览器，在网上找一张你喜爱的图片。在我们的例子中，使用 Wikipedia 上的树莓照片，把它保存为 *raspberrypies.jpg*。

3. 在图片上点击鼠标右键，选择 Save Image。

4. 把图片保存到先前创建的 *simplecv-test* 目录中。

5. 在文件管理器（File Manager）中［在附件（Accessories）菜单中］打开 *simplecv-test* 目录，点击右键，选择 Create New →

Blank File。

6. 把新建的文件命名为 *image-display.py*。

7. 双击新创建的 *image-display.py* 文件，用 Leafpad 打开它。

8. 输入代码 9.1 中的代码。

9. 保存 *image-display.py* 文件并从终端运行它。如果一切顺利，你会看到如图 9.4 所示的窗口，里面显示了你选择的图片。你可以把这个窗口关闭或者在终端中按 Control-C 键中断程序运行。

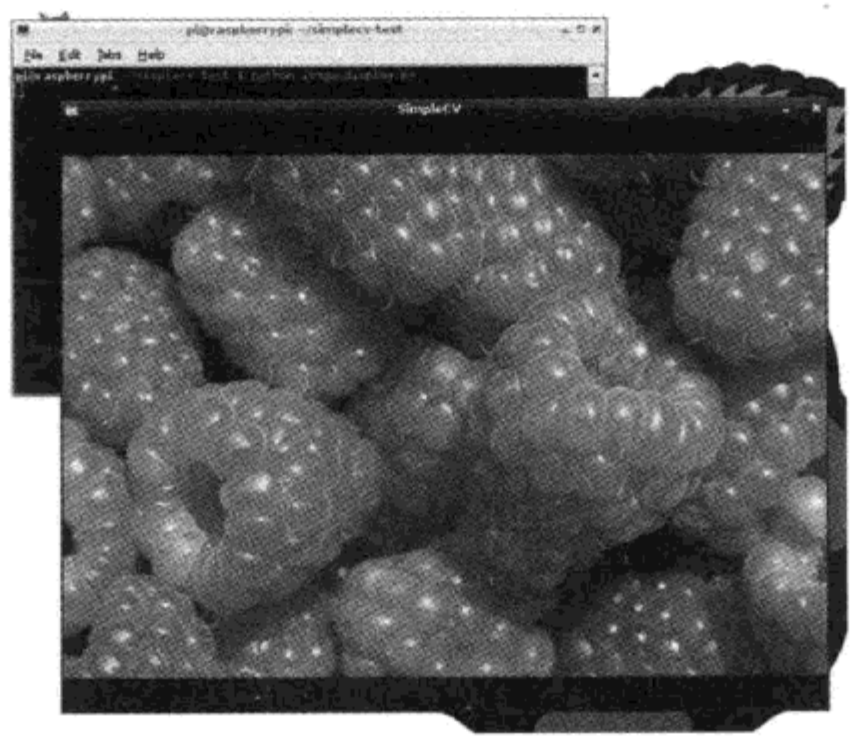


图9.4　在窗口中显示的树莓照片

代码 9.1　*image-display.py* 的源代码

```
from SimpleCV import Image, Display❶
from time import sleep

myDisplay = Display()❷
raspberryImage = Image("raspberries.jpg")❸
raspberryImage.save(myDisplay)❹
while not myDisplay.isDone():❺
    sleep(0.1)
```

❶ 导入 SimpleCV 中的图像与显示相关的函数。

❷ 创建一个新的窗口对象。

❸ 把 *raspberries.jpg* 文件加载到内存成为 `image` 对象。

❹ 在窗口中显示这个图像。

❺ 让程序在显示完图像后不立即退出，等待图像窗口关闭。

修改图片

现在你已经学会了如何把图片加载到内存并在屏幕上显示它，下一步是学习如何在显示图片前对它进行一些修改。这里所说的修改，是指修改内存中的图片对象，而并不是真正地修改磁盘上的图片文件。

1. 把 *image-display.py* 文件另存为 *superimpose.py*。
2. 按照代码 9.2 修改代码，增加处理逻辑。
3. 保存文件，并从命令行上运行它。
4. 你会看到与之前相同的图片，但图片上被加上了图形和文字。

代码 9.2 *superimpose.py* 的源代码

```
from SimpleCV import Image, Display, DrawingLayer, Color❶
from time import sleep

myDisplay = Display()

raspberryImage = Image("raspberries.jpg")

myDrawingLayer = DrawingLayer((raspberryImage.width,
raspberryImage.height))❷

myDrawingLayer.rectangle((50,20), (250, 60), filled=True)❸
myDrawingLayer.setFontSize(45)
myDrawingLayer.text("Raspberries!", (50, 20), color=Color.
WHITE)❹

raspberryImage.addDrawingLayer(myDrawingLayer)❺
```

```
raspberryImage.applyLayers()❻

raspberryImage.save(myDisplay)

while not myDisplay.isDone():
    sleep(0.1)
```

❶ 与上一个示例一样，导入 SimpleCV 库的图像和显示相关的函数，再加上绘画层和颜色相关的函数。

❷ 创建一个与原图片一样大的绘画层。

❸ 在绘画层上画一个矩形并填充上颜色，起止坐标是 (50,20) ~ (250,60)。

❹ 在绘画层上用白色绘制“Raspberries!”的文字，起始坐标是 (50,20)。

❺ 把 myDrawingLayer 这个绘画层添加到原始的 raspberryImage 图片对象上。

❻ 把绘画层与原始图片合并（图 9.5）。

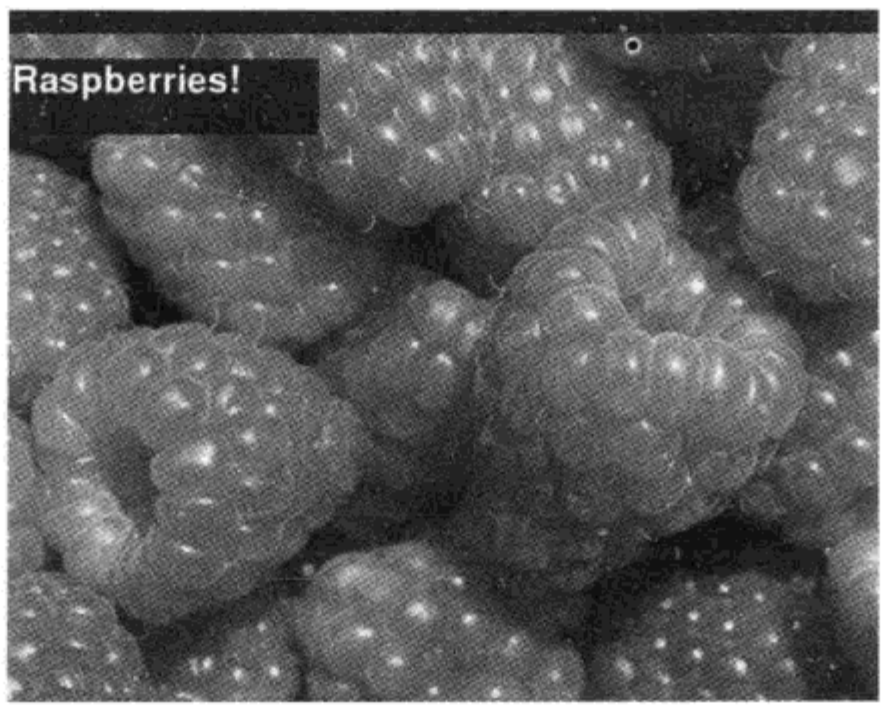


图9.5　修改过的树莓照片

除了把修改过的图片显示在屏幕上，你也可以很方便地把它保

存到文件中，如代码 9.3 所示。

代码 9.3 *superimpose–save.py* 的源代码

```
from SimpleCV import Image, DrawingLayer, Color
from time import sleep

raspberryImage = Image("raspberries.jpg")

myDrawingLayer = DrawingLayer((raspberryImage.width,
raspberryImage.height))

myDrawingLayer.rectangle((50,20), (250, 60), filled=True)
myDrawingLayer.setFontSize(45)
myDrawingLayer.text("Raspberries!", (50, 20), color=Color.
WHITE)

raspberryImage.addDrawingLayer(myDrawingLayer)
raspberryImage.applyLayers()
raspberryImage.save("raspberries-titled.jpg")❶
```

❶ 把内存中修改过的图片保存为一个新的文件，命名为 *raspberries-titled.jpg*。

上面的代码中没有打开窗口把图片显示在屏幕上，所以你可以脱离图形桌面环境直接在纯命令行模式下运行这个程序。你还可以通过修改这个程序，实现用一个命令给一批图片加上水印的功能。

你对图片的修改也不仅仅局限于添加文字或绘制矩形。下面是 SimpleCV 中其他一些可以使用的绘图函数。可以在 *http://simplecv.org/docs/SimpleCV.html#SimpleCV.DrawingLayer.DrawingLayer* 上查阅它们完整的文档。

- circle（圆）
- ellipse（椭圆）
- line（线条）

- polyon（多边形）
- bezier curve（贝塞尔曲线）

操作摄像头

用 SimpleCV 获取摄像头的视频流与打开一个图片文件加载到内存中的方式非常相似，可以很轻松地写出属于你自己的摄像头图像预览程序。

1. 新建一个文件，名为 *basic-camera.py*，输入代码 9.4 中的代码。

2. 接上你的 USB 摄像头，运行上一步中创建的脚本。你会看到如图 9.6 所示的窗口，里面显示了你的摄像头上捕捉到的画面。

3. 在终端中按 Control-C 键可以关闭该视频窗口。

代码 9.4　*basic-camera.py* 的源代码

```
from SimpleCV import Camera,Display
from time import sleep

myCamera = Camera(prop_set={"width": 320, "height": 240})❶

myDisplay = Display(resolution=(320, 240))❷

while not myDisplay.isDone():❸
    myCamera.getImage().save(myDisplay)❹
    sleep(.1)❺
```

❶ 创建一个摄像头对象，并把图像大小设置为 320 × 240 以提高程序性能。

❷ 窗口大小也设置成 320 × 240。

❸ 循环执行缩进的代码块直到窗口被关闭。

❹ 从摄像头获取一帧图像并在窗口中显示。

❺ 在显示每一帧图像之间暂停 0.1s。

图9.6　把摄像头图像显示在屏幕上

你可以把上面两个例子的代码结合起来，让 Python 脚本可以从摄像头捕获一帧图像并保存为 *.jpg* 文件：

```
from SimpleCV import Camera
from time import sleep

myCamera = Camera(prop_set={'width':320, 'height': 240})

frame = myCamera.getImage()
frame.save("camera-output.jpg")
```

人脸检测

SimpleCV 有一个很强大的函数——`findHaarFeatures`，这是一个在图像中搜索匹配某一种特定模式（或称 cascade）的算法，在 SimpleCV 中自带了几种模式，包括脸、鼻子、眼睛、嘴和身体，如果必要，你也可以下载或生成你自己模式文件。`findHaarFeatures` 可以分析图像并从中匹配出对应的模式，然后返回匹配到的部分在图像中的位置。这就意味着，你可以从像文件或摄像头捕获的图像中匹配汽车、动物或人。下面以人脸识别为例，

实验一下 findHaarFeatures 的功能。

1. 在 *simplecv-test* 目录中创建一个新文件，命名为 *face-detector.py*。

2. 输入代码 9.5 中的代码。

3. 连接你的摄像头并对准人的脸部，从命令行运行新建的脚本。

4. 在终端窗口中，你可以看到一系列 findHaarFeatures 所检测到的人脸坐标。尝试调整一下摄像头的角度，可以看到这些数据会发生变化。你还可以尝试着将人脸的照片放到摄像头前，看看会发生什么。

代码 9.5　*face-detector.py* 的源代码

```
from SimpleCV import Camera, Display
from time import sleep

myCamera = Camera(prop_set={"width":320, "height": 240})

myDisplay = Display(resolution=(320, 240))

while not myDisplay.isDone():
    frame = myCamera.getImage()
    faces = frame.findHaarFeatures('face')❶
    if faces:❷
        for face in faces:❸
            print "Face at: " + str(face.coordinates())
    else:
        print "No faces detected."
    frame.save(myDisplay)
    sleep(.1)
```

❶ 在图像中检测人脸并把检测到的人脸图像保存到 faces 对象中。

❷ 如果 findHaarFeatures 检测到至少一张人脸，则执行下面的代码块中的逻辑。

❸ 对于检测结果 faces 中的 face 对象（对应一张检测出来的人脸），运行下面的代码（print 语句）。

如果你的尊容出现在了屏幕上，但你仍然看到“No faces detected”的提示，可以尝试用下面的步骤排查问题。

- 光线是不是够亮？如果你所处的环境比较黑暗，会影响算法的检测能力。可以尝试增加一点照明。
- 程序中使用的 Haar cascade 适用于检测一个正面的人脸。如果你把头抬得太高或摄像头摆放得不够水平，都会影响到算法检测人脸的准确性。

项目：Raspberry Pi 照相馆

使用 Python，你可以把各种库都整合在一起，设计出复杂的项目。把第 8 章中所提到的 GPIO 库与 SimpleCV 结合起来，你可以实现一个 Raspberry Pi 照相馆，这个项目一定能使你在朋友圈中出尽风头（图 9.7）。通过使用 SimpleCV 中的 findHaarFeatures 函数，你还可以给你的照相馆中拍摄的人像添加各种元素，如添加帽子、眼镜、胡子或胡须。这个示例是基于 Kurt Demaagd、Anthony Oliver、Nathan Oostendorp 和 Katherine Scott 所著的 *Practical Computer Vision with SimpleCV*（*http://shop.oreilly.com/product/0636920024057.do*）一书中的例子所设计的。

这里是把你的 Raspberry Pi 变为照相馆所需要的材料：

- USB 摄像头；
- 显示器；

- 按钮，任何你所喜欢的样式都可以；
- 长度合适的连接线；
- 一个 10kΩ 的电阻。

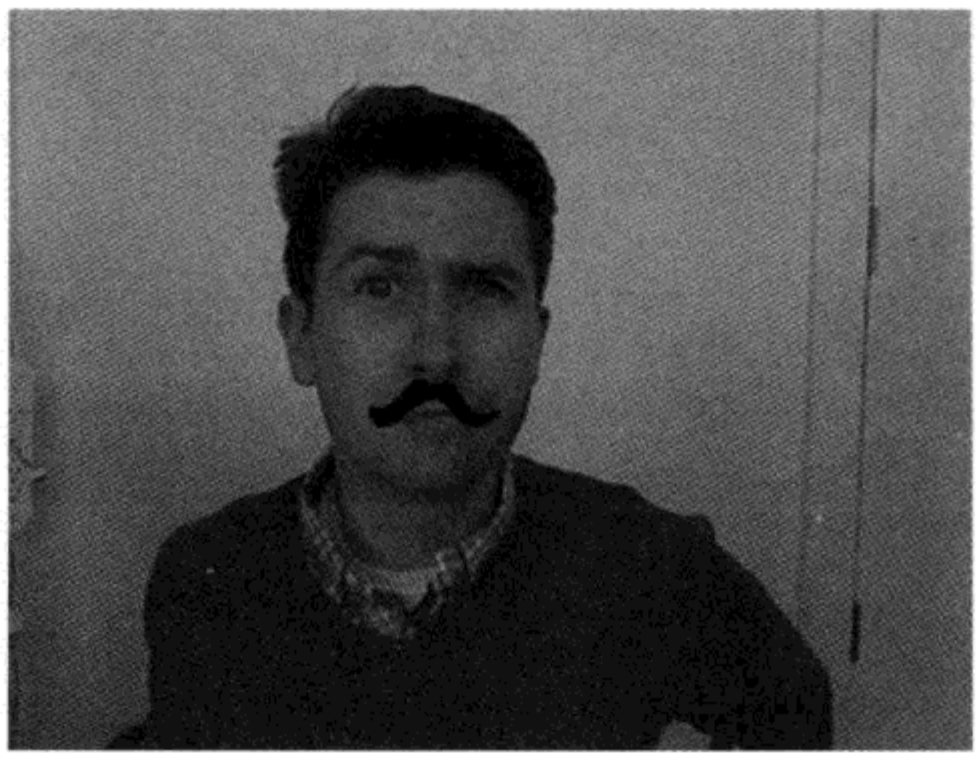


图9.7　Raspberry Pi 照相馆的输出

在开始之前，先确认你的 Raspberry Pi 上已经正确安装了 Python 的 RPi.GPIO 和 SimpleCV 库。请参考“在 Python 中安装并测试 GPIO”和“安装并测试 SimpleCV”的内容。

1. 与第 8 章中所做的一样，把按钮接在 GPIO 24 接口上。按钮的一个引脚应该接 3.3V，另一个接 GPIO 24。不要忘记在地线和接 GPIO 的按钮引脚间加上一个 10kΩ 的下拉电阻。

2. 找一张背景为白色胡子的图片，把它命名为 *mustache.png* 并保存到 Raspberry Pi 上的一个新目录中。你可以从本书的支持网站上下载各章节的代码（参考“如何联系我们”），其中 *ch09* 子目录中有我们制作好的胡子图片。

3. 在这个目录中，创建一个新文件，名叫 *photobooth.py*，输入代码 9.6 中的代码并保存。

代码 9.6　*photobooth.py* 的源代码

```
from time import sleep, time
from SimpleCV import Camera, Image, Display
import RPi.GPIO as GPIO

myCamera = Camera(prop_set={"width":320, "height": 240})
myDisplay = Display(resolution=(320, 240))
stache = Image("mustache.png")
stacheMask = \
        stache.createBinaryMask(color1=(0,0,0),
color2=(254,254,254))❶
stacheMask = stacheMask.invert()❷

GPIO.setmode(GPIO.BCM)
GPIO.setup(24, GPIO.IN)

def mustachify(frame):❸
    faces = frame.findHaarFeatures("face")
    if faces:
        for face in faces:
            print "Face at: " + str(face.coordinates())
            myFace = face.crop()❹
            noses = myFace.findHaarFeatures("nose")
            if noses:
                nose = noses.sortArea()[-1]❺
                print "Nose at: " + str(nose.coordinates())
                    xmust = face.points[0][0] + nose.x -
(stache.width/2)❻
                    ymust = face.points[0][1] + nose.y +
(stache.height/3)❼
            else:
                return frame❽
            frame = frame.blit(stache, pos=(xmust, ymust),
mask=stacheMask)❾
        return frame❿
    else:
        return frame⓫
```

```
while not myDisplay.isDone():
    inputValue = GPIO.input(24)
    frame = myCamera.getImage()
    if inputValue == True:
        frame = mustachify(frame)❿
        frame.save("mustache-" + str(time()) + ".jpg")⓭
        frame = frame.flipHorizontal()⓮
        frame.show()
        sleep(3)⓯
    else:
        frame = frame.flipHorizontal()⓰
        frame.save(myDisplay)
    sleep(.05)
```

❶ 创建胡子的蒙版，把黑色设置为透明（color1 和 color2 这两个参数用于设置 RGB 值，范围是 0 ~ 255）。

❷ 把蒙版取反，再绘制这个图片时就只有黑色的像素会被显示出来了，其他颜色都变成透明色。

❸ 定义一个函数，接受一帧图像的输入，如果在这帧图像中可以检测到脸和鼻子，就给这个图像上添加一个胡子。

❹ 创建只包含脸的图片对象，这样检测鼻子时可以处理得更快一些。

❺ 如果在脸上找到了多个具有鼻子特征的对象，选择其中最大的一个作为鼻子。

❻ 设置胡子的 x 坐标。

❼ 设置胡子的 y 坐标。

❽ 如果没有找到鼻子，就返回原始的图像。

❾ 使用 blit 函数（块图像传输，Block Image Transfer）把胡子画到原始的图像上。

❿ 返回添加了胡子的图像。

⓫ 如果没有找到人脸，直接返回原始图像。

⓬ 把图像传入 mustachify 函数。

⓭ 把图像保存为 JPG 文件，以当前的时间来给文件命名。

⓮ 在显示图像之前，先进行一次水平翻转，这样显示出来的图像是摄像头所拍摄到的画面的镜像画面。

⓯ 在屏幕上把保存下来的图片保持显示 3s。

⓰ 如果按钮没有被按下，直接把实时图像翻转后显示出来。

现在可以试一下你的作品了，把摄像头连接好，在终端上进入你的胡子图片和 *photobooth.py* 所在的目录，运行脚本：

```
pi@raspberrypi ~ $ sudo python photobooth.py
```

屏幕上会显示摄像头所捕捉到的图像。当按钮被按下时，程序会检测人脸、添加胡子并保存照片（所有的照片都会保存在与脚本相同的路径下）。

进一步学习

Practical Computer Vision with SimpleCV

（*http://shop.oreilly.com/product/0636920024057.do*）

由 Kurt Demaagd、Anthony Oliver、Nathan Oostendorp 和 Katherine Scott 合著的《SimpleCV 实践指南》。它包含了大量示例代码和项目，适合进一步学习用 Python 进行图像和计算机视觉开发。