第8章

机器视觉

8.0 引言

机器视觉（Computer vision，CV）可以让树莓派睁眼看世界。从实用的角度看，这就意味着你的树莓派能够分析图像，寻找感兴趣的物品，甚至识别面部和文本。

如果你连接一台照相机来提供图像的话，这一切都能成为可能。

8.1 安装 SimpleCV

面临问题

你想在树莓派上面安装机器视觉软件 SimpleCV。

解决方案

为了安装 SimpleCV，首先要使用下列命令来安装必须的软件包。

```
$ sudo apt-get update
$ sudo apt-get install ipython python-opencv python-scipy
$ sudo apt-get install python-numpy python-setuptools python-pip
$ sudo pip install svgwrite
```

然后，利用下列命令来安装 SimpleCV 本身。

```
$ sudo pip install https://github.com/sightmachine/SimpleCV/zipball/master
```

安装完成之后，你可以通过下列命令来检查是否一切正常。

```
$ simplecv
+-------------------------------------------------------------+
 SimpleCV 1.3.0 [interactive shell] - http://simplecv.org
```

```
+---------------------------------------------------------+
Commands:
    "exit()" or press "Ctrl+ D" to exit the shell
    "clear()" to clear the shell screen
    "tutorial()" to begin the SimpleCV interactive tutorial
    "example()" gives a list of examples you can run
    "forums()" will launch a web browser for the help forums
    "walkthrough()" will launch a web browser with a walkthrough
```

这将打开 SimpleCV 控制台。它实际上是一个 Python 控制台，只是为 SimpleCV 提供了额外的功能而已。

进一步探讨

SimpleCV 是机器视觉软件 OpenCV 的 Python 封装。SimpleCV，顾名思义，就是 OpenCV 的简化应用。如果你想释放 OpenCV 的全部威力，请浏览 http://opencv.org/。

机器视觉需要消耗大量的处理器和内存资源，因此，尽管 SimpleCV 和 OpenCV 可以在老版的树莓派上面运行，但是对于树莓派 3 或 2 之前的版本来说，其运行速度将会慢得让人无法忍受。

参考资料

关于 OpenCV 的详细介绍，请参考 http://opencv.org/。

关于 SimpleCV 项目的主页，请访问 http://simplecv.org。

在本章中，直到 8.4 节才首次应用 SimpleCV，你可以阅读该节以获取 SimpleCV 入门的详细知识。

8.2 为机器视觉配置 USB 摄像头

面临问题

你想设置一个 USB 摄像头以供机器视觉之用。

解决方案

你可以使用一个与树莓派兼容的 USB 摄像头（参见 http://elinux.org/RPi_USB_Webcams）。请选择一款优质的摄像头，如果你的项目要求摄像头靠近物体的话，请选择可以手动调焦的摄像头。为了能够真正近距离拍摄物体，廉价的 USB 内窥镜将会派上大用场。

有时候你可能希望为自己的 CV 项目建立一个照明良好的区域，当然是否需要要根据自己的 CV 项目的具体情况而定。图 8-1 展示了一个由半透明的塑料储物箱做成的简单灯箱，并且是同时从侧面和顶部均匀给光的。摄像头连接固定在灯箱顶部，这种布局将

会在 8.4 节中用到。

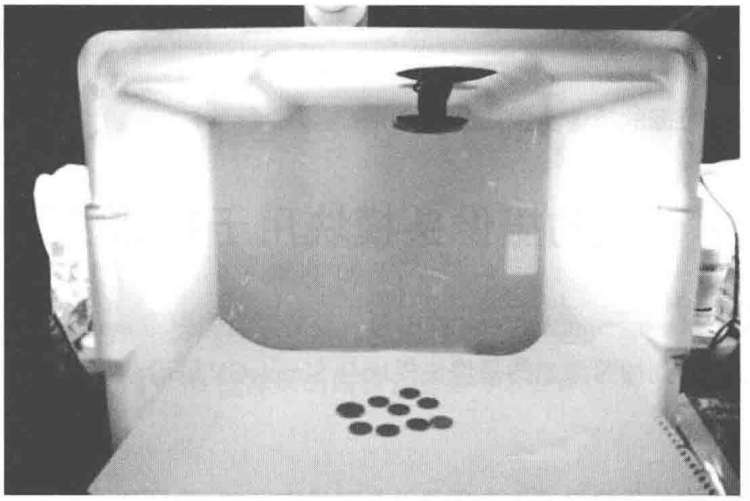

图 8-1　利用自制“摄影灯箱”提供均匀照明

此外，你还可以购买专门用于摄影的商品化的摄影灯箱，它们通常更加好用。

为了使得照明系统明亮而均匀，你可能需要费些时间来反复试验。此外，阴影也会是一个棘手的问题。

进一步探讨

你可以通过 SimpleCV 控制台来测试自己的 USB 摄像头。启动 SimpleCV，然后输入下面粗体显示的命令。

```
SimpleCV:1> c = Camera()
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument

SimpleCV:2> i = c.getImage()
SimpleCV:3> i
SimpleCV:3: <SimpleCV.Image Object size:(640, 480), filename: (None),
at memory location: (0x2381af8)>
SimpleCV:4> i.show()
```

你不用理会无效的参数消息。

当你执行 i.show()命令时，会打开另外一个窗口，并且会在其中显示摄像头刚才捕获的图像。

虽然树莓派的摄像头模块（见 8.3 节）可以与 SimpleCV 联合使用，但是该模块到树莓派的导线太短了，所以最好还是使用一款高品质的摄像头。

参考资料

要想联合使用树莓派的摄像头模块和 SimpleCV，请参考 8.3 节。

8.3 将树莓派的摄像头模块用于机器视觉

面临问题

你希望将直接连接到树莓派上的摄像头模块与 SimpleCV 联合使用。

解决方案

树莓派的摄像头模块无法自动作为摄像设备而显露出来。为了让该模块与 SimpleCV 协同工作，最简单的方法就是借助 Python 的 picamera 模块，通过该摄像头捕获图像。

下面的代码片段会利用 picamera 来捕获一张图像，并将其保存到一个临时文件中，然后作为适合 SimpleCV 使用的 image 对象进行加载。

```
import picamera
from SimpleCV import *

def get_camera_image():
    with picamera.PiCamera() as camera:
        camera.capture('tmp.jpg')
    return Image('tmp.jpg')
```

在 8.4 节中的程序假定使用 USB 摄像头。该程序的第二个版本会将上面的函数用于树莓派的摄像头模块，具体代码位于文件 coin_count_pi_cam.py 中。

进一步探讨

像这样每当捕获一张图像都要写文件的做法貌似效率不高，事实上也确实如此，不过对于树莓派来说，使用 SimpleCV 对该图像所做的任何处理的耗时，很可能都会比加载和保存图像的时间要长得多。

你可以通过 picamera 模块来设置摄像头的各种功能，最常用的有控制解析度、自动曝光和白平衡。这样做的好处在于可以使机器视觉的效果更加易于保持一致性。

摄像头的最佳解析度是 2592 × 1944 像素，不过这个解析度会严重降低图像处理的速度，所以你可能想要将解析度设置在 1024 × 768 像素附近，并且关闭自动白平衡。为此，你可以使用 get_camera_image 函数来修改相应的设置，具体代码如下所示。

```
import picamera
```

```
from SimpleCV import *

def get_camera_image():
    with picamera.PiCamera() as camera:
        camera.resolution = (1024, 768)
        camera.awb_mode = 'off'
        camera.capture('tmp.jpg')
    return Image('tmp.jpg')
```

参考资料

关于树莓派摄像头的安装方法，请参考 1.14 节。

关于 Python 的 picamera 模块的详细信息，请访问 http://picamera.readthedocs.org/。

关于通过 SimpleCV 使用 USB 摄像头的内容，请参考 8.2 节。

8.4 数硬币

面临问题

你想要利用机器视觉来统计摄像头下面硬币的数目。

解决方案

利用 SimpleCV 及其 findCircle 函数，你可以实时计算放到摄像头下面的硬币的数目。

对于这个机器视觉应用来说，你需要提供良好的灯光照明，以及位置固定的摄像头。我使用的配置如图 8-1 所示。

在开始编写可以计算树莓派看到的硬币数量的程序之前，你得先通过 SimpleCV 控制台来进行必要的实验，从而获得正确识别圆形所需的参数。为此，先利用命令 simplecv 来启动 SimpleCV，并通过下面给出的命令来启动摄像头，捕获图像，然后在单独的窗口中显示出来。

```
SimpleCV:1> c = Camera()
SimpleCV:2> i = c.getImage()
SimpleCV:3> i.show()
```

上面的代码将会显示硬币的图像，具体见图 8-2。

圆检测要求对图像进行反转处理，或者使用黑色背景也行。

你可以通过下面所示的命令来实现图像的反转，并显示结果。

```
SimpleCV:4> i2 = i.invert()
SimpleCV:5> i2.show()
```

上面的命令将建立 i 反转后的副本，具体如图 8-3 所示。

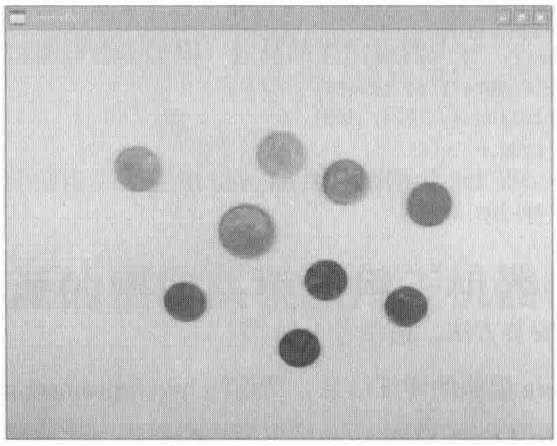

图 8-2　一些硬币的基本图像

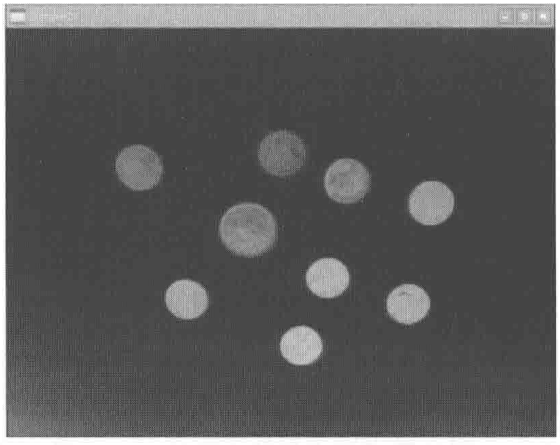

图 8-3　硬币反转后的图像

到目前为止，图像已经准备好了，接下来要做的是利用让 SimpleCV 通过 findCircle 命令在图像中寻找圆形。这个命令需要 3 个参数，并且为了防止出现错误辨识的情况，你需要对这些参数进行相应的调整。这些参数分别介绍如下。

canny

这是边缘检测的阀值。按照机器视觉的术语来讲，边缘就是图像像素颜色发生重大变化处的分隔线。这个参数的默认值是 100，并且当降低这个值的时候，会检测到更多的边缘。当然，这不会导致检测到更多的圆形，因为这些额外的边缘存在，有可能会破坏本来很圆的形状。就硬币来说，这些边缘既可以是硬币的文字方面的，也可能是图像方面的。

thresh

找到边缘之后，圆检测需要确定边缘表示圆所需的长度。当降低这个值的时候，会检测到更多的圆形。

distance

这个参数用来设置相邻圆形之间的间隔距离（以像素为单位）。

为了寻找圆形，可以使用下列所示的命令。

```
SimpleCV:6> coins = i2.findCircle(canny=100, thresh=70, distance=15)
SimpleCV:7> coins
 SimpleCV.Features.Detection.Circle at (237,297),
 SimpleCV.Features.Detection.Circle at (307,323),
 SimpleCV.Features.Detection.Circle at (373,305),
 SimpleCV.Features.Detection.Circle at (305,261),
 SimpleCV.Features.Detection.Circle at (385,253),
 SimpleCV.Features.Detection.Circle at (243,231),
 SimpleCV.Features.Detection.Circle at (307,383),
 SimpleCV.Features.Detection.Circle at (407,371),
 SimpleCV.Features.Detection.Circle at (235,373)]
```

如果一个硬币也没找到的话，可以尝试降低参数 canny 和 thresh 的取值。如果找的硬币过多的话，可以增加 thresh 的值。你可以通过下列命令，将硬币外圈叠加到原始图像上面，从而检查 SimpleCV 是否真正找到了这些硬币。

```
SimpleCV:8> coins.draw(width=4)
SimpleCV:9> coins.show()
```

这将显示叠加在实际硬币上的圆圈（见图 8-4）。

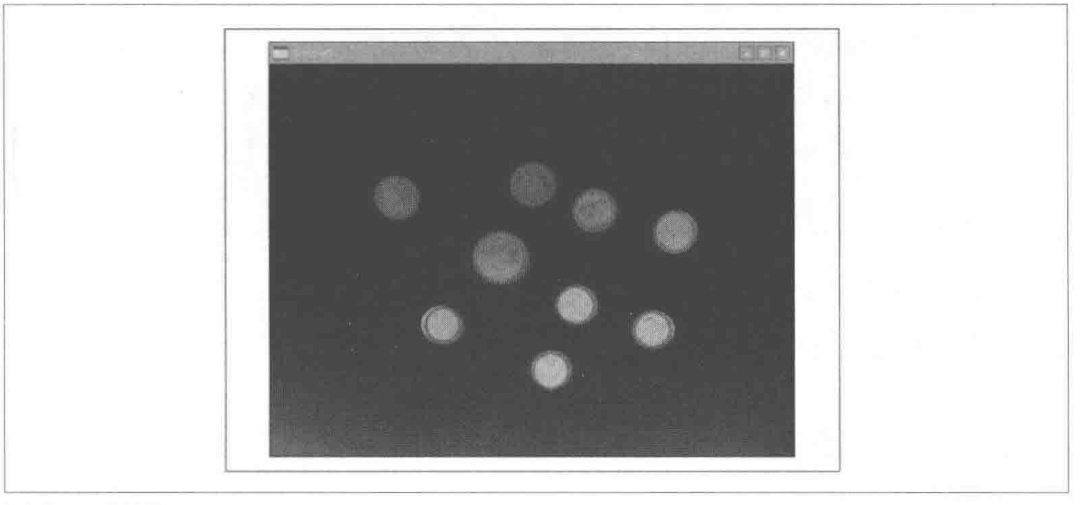

图 8-4　发现硬币

你可以到处移动硬币，或者加减硬币数量，然后重新拍照，重复上述过程，以便确认识别过程的可靠性。此外，你还可以调整各个参数，直到达到令人满意的效果为止。

我们可以把上面在 SimpleCV 控制台中使用的命令打包到一个 Python 程序中去，让它（以树莓派最快的速度）打印出检测到的硬币数量。这个程序可以从本书相关资源中（http://www.raspberrypicookbook.com）下载，相应的文件名为 coin_count.py。

```
from SimpleCV import *

c = Camera()

while True:
    i = c.getImage().invert()
    coins = i.findCircle(canny=100, thresh=70, distance=1)
    print(len(coins))
```

就像你看到的那样，在导入 SimpleCV 库后，程序中的命令跟在控制台中输入的命令完全一样。唯一的区别在于它不是显示硬币，而是通过 len 函数显示硬币数量。

```
$ sudo python count_coins.py
9
9
9
10
10
```

你可以通过四处移动硬币，或通过添加硬币来检查该项目的工作情况是否良好。

进一步探讨

当我使用 B+版本的树莓派的时候，在库加载和相机设置而导致最初的延迟之后，每秒会出现大约“计数”两次。如果使用的是树莓派 2 的话，计数的次数会增加到每秒 5 次左右。

虽然我们没有将这里的东西应用到自动售货机的打算，但是进一步利用硬币的直径来确定币值并加总桌面上的硬币币值本身就是一个非常有趣的项目。

你可以使用 diameter 方法来确定某个硬币的直径，具体代码如下所示。

```
SimpleCV:10> coins[0].diameter()
SimpleCV:11> 60
```

参考资料

关于 SimpleCV 的安装信息，请参考 8.1 节。

关于配置摄像头的相关内容，请参考 8.2 节。

8.5 人脸检测

面临问题

你希望找出人脸在照片或摄像头图像中的坐标位置。

解决方案

你可以使用 SimpleCV 中的 Haar-like 特征检测功能来分析图像，并识别其中的人脸。

如果你还没有做过这类实验的话，请先安装 SimpleCV（参见 8.1 节）。首先，你需要打开 SimpleCV 控制台，并加载一幅含有人脸的图像。实际上，你可以从本书的下载资源中（http://www.raspberrypicookbook.com）找到合适的人脸图像文件，该文件名为 faces.jpg。

然后，运行下列命令：

```
SimpleCV:1> i=Image("faces.jpg")
SimpleCV:2> faces = i.findHaarFeatures('face.xml', min_neighbors=5)
SimpleCV:3> faces.draw(width=4)
SimpleCV:4> i.show()
```

这样就会打开一个图像浏览窗口，其中的人脸已经使用矩形做了标注，如图 8-5 所示。

图 8-5　检测人脸

进一步探讨

像使用摄像头进行交互一样，你还可以将现成的文件加载到 SimpleCV 中。在上面的例子中，图像 i 是从文件 faces.jpg 中载入的。方法 findHaarFeatures 有一个强制性的文件，它描述待搜索的特征的类型。这些特征被称作 haar 特征，并且由一个 XML 文件给出具体的描述。

这些文件是由 SimpleCV 来预加载的，不过你可以通过搜索互联网来了解 haar 文件的详细说明。

本例用到的第二个参数（min_neighbors）的作用是调节 haar 函数，随着 min_neighbors 数值的降低，假阳性的检测结果就会随之增多。如果观察假阳性结果的话，会发现图片上通常都有面部的正面器官（嘴巴、鼻子和眼睛）。

另外还有许多内置的 haar 特征，你可以通过下列命令列出它们。

```
SimpleCV:5> i.listHaarFeature()
SimpleCV:4> fullbody.xml', 'face4.xml', 'face.xml',
'upper_body.xml', 'right_ear.xml', 'eye.xml', 'lower_body.xml',
'two_eyes_small.xml', 'nose.xml', 'face2.xml', 'lefteye.xml',
'right_eye.xml', 'two_eyes_big.xml', 'face3.xml', 'mouth.xml',
'glasses.xml', 'profile.xml', 'left_ear.xml', 'left_eye2.xml',
'upper_body2.xml', 'right_eye2.xml', 'face_cv2.xml'
```

就像你所看到的，它们都是与身体部位有关的。

检测 haar 特征的时候，通常需要几秒时间，即使在树莓派 3 上面也是如此。

参考资料

关于 SimpleCV 的安装信息，请参考 8.1 节。

关于配置摄像头的相关内容，请参考 8.2 节。

此外，你可以在 https://github.com/Itseez/opencv/tree/master/data/haarcascades 页面找到许多有趣的 Haar 文件。

8.6 运动检测

面临问题

你想利用连接到树莓派上面的摄像头检测其视野内的移动物体。

解决方案

你可以使用 SimpleCV 来检测源自摄像头的连续帧之间的变化情况。

下面的程序代码会将每次捕获到的图像与之前的图像进行比较。然后，它会检查差值图像中的所有斑块（颜色相近的区域），如果找到大于 MIN_BLOB_SIZE 的，就会打印输出一则消息，声明检测到移动现象。

```
from SimpleCV import *

MIN_BLOB_SIZE = 1000

c = Camera()
old_image = c.getImage()
```

```
while True:
    new_image = c.getImage()
    diff = new_image - old_image
    blobs = diff.findBlobs(minsize=MIN_BLOB_SIZE)
    if blobs :
        print("Movement detected")
    old_image = new_image
```

进一步探讨

图像的连续帧大致如图 8-6 和图 8-7 所示。当第二幅图像减去第一幅之后，将得到类似于如图 8-8 所示的图像。然后，对其进行斑块检测（blob detection），将得到如图 8-9 所示的由轮廓线勾勒出的斑块图。

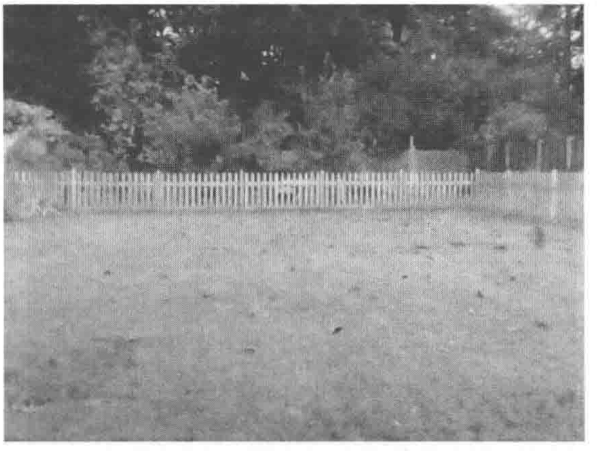

图 8-6　运动检测帧 1

图 8-7　运动检测帧 2

图 8-8　运动检测，差值图像

图 8-9　运动检测斑块

如果使用树莓派 2 的话，上面的运动检测程序每秒可以处理 5 帧。

参考资料

关于 SimpleCV 的安装方法，请参考 8.1 节。关于配置摄像头的相关内容，请参考 8.2 节。

检测运动的另一种方法是使用被动红外（PIR）传感器，具体参考 12.9 节。

8.7　光学字符识别

面临问题

你想把包含文字的图像转换为真正的文本。

解决方案

你可以使用光学字符识别（OCR）软件 tesseract 从图像中提取文本。

要想安装 tesseract，可以使用如下所示的命令。

```
$ sudo apt-get install python-distutils-extra tesseract-ocr tesseract-ocr-eng
libopencv-dev libtesseract-dev libleptonica-dev python-all-dev swig
libcv-dev python-opencv python-numpy python-setuptools
build-essential subversion
$ sudo apt-get install tesseract-ocr-eng tesseract-ocr-dev libleptonica-dev
python-all-dev swig libcv-dev
$ sudo svn checkout
http://python-tesseract.googlecode.com/svn/python-tesseract-0.7.4/
$ cd python-tesseract-0.7.4
$ sudo python setup.py build
$ sudo python setup.py install
```

为了试用 tesseract 软件，你需要提供一张含有文字的图像文件。为此，你可以从本书提供的下载资源中下载一个名为 ocr_example.png 的文件，地址 http://www.raspberrypicookbook.com。

要想把图像转换成文本，可以使用如下所示的代码。

```
$ tesseract ocr_example.png found
Tesseract Open Source OCR Engine v3.02 with Leptonica
$ more found.txt
The quick brown fox jumped over
the lazy dogs back.
$
```

进一步探讨

库 tesseract 适用于大部分的图像类型，包括 PDF、PNG 和 JPG 文件。

参考资料

若要进一步了解 tesseract 库，请访问 https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract。