第 9 章
树莓派上利用 TensorFlow 实现对象检测

引　言

现如今人工智能的趋势大火，越来越多的人关注到人工智能的一些有趣的应用，其中 TensorFlow 机器学习系统是谷歌大脑的第二代机器学习系统，它是一个开源软件库，用于各种感知和语言理解任务的机器学习。目前很多商业产品都在使用，如语音识别、Gmail、Google 相册及搜索，还有一个特别有趣的社区——DonkeyCar 社区也在使用 TensorFlow 神经网络框架进行自动驾驶无人小车的项目推广，其中也用到了树莓派。

下面就带领读者通过在树莓派上搭建 TensorFlow 的环境并尝试进行对象检测来完成神经网络学习入门的第一步。

9.1　软硬件环境介绍

9.1.1　硬件准备

◆　树莓派 4B 4GB 版本，其他的树莓派版本也可以使用，只是可

能性能上会有差异。

- 32GB 的 TF 卡。
- MicroHDMI 线。
- 支持 HDMI 接口的显示器一台。
- 鼠标键盘一套。
- 5V/3A、支持 USB-C 接口的电源。
- 树莓派官方摄像头及排线。

9.1.2　软件准备

到官方网站下载最新的 Raspberry Pi OS 系统并烧录到 TF 卡。

当前最新系统的版本信息和内核版本信息如下：

- 操作系统版本信息：Raspbian buster 10。
- 内核版本信息：5.10.17-v8+。

9.2　操作步骤

9.2.1　更新系统及软件仓库

在此，假定你已经烧录了系统并做了初始化设置，完成了联网，开启 SSH 等操作。

在树莓派上打开一个终端，然后在终端中输入：

```
sudo apt-get update
sudo apt-get upgrade -y
```

如果遇到报错，则执行：

```
sudo apt update -allow-releaseinfo-change
sudo apt upgrade -y
```

9.2.2　创建 TensorFlow 的工作目录

为了完整地完成项目的配置，并且避免文件的管理混乱，建议创建一个专门的目录

作为工作目录，然后下载相关的文件并在工作目录中完成项目的配置。继续在终端中执行：

```
mkdir -pv  ~/tf
cd ~/tf
```

9.2.3 安装 TensorFlow 的 Python 库和部分依赖

```
sudo apt-get -y install libatlas-base-dev
pip3 install tensorflow
pip3 install pillow lxml jupyter matplotlib cython
sudo apt-get -y install python-tk
```

9.2.4 安装 OpenCV 视觉框架

因为涉及通过摄像头采集原始数据进行解算和分析，因此使用了最为流行的开源视觉框架 OpenCV，在树莓派上已经可以完美支持。安装 OpenCV 框架前需要先安装一些依赖的软件包：

```
sudo apt-get -y install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
sudo apt-get -y install libavcodec-dev libavformat-dev libswscale-dev
sudo apt-get -y install libv4l-dev
sudo apt-get -y install libxvidcore-dev
sudo apt-get -y install libx264-dev
sudo apt-get -y install qt4-dev-tools
sudo apt-get install -y libqtgui4
sudo apt -y install python3-opencv
sudo apt -y install libqt4-test
```

安装过程中可能会遇到由于网络问题导致的失败，不用担心，重新执行上述的操作即可。最后这些依赖安装完成后，执行下面的命令安装 OpenCV 的 Python 库：

```
pip3 install opencv-python
```

安装完成后，通过在终端执行下面的命令来进行测试，查看安装是否成功：

```
python3
```

然后执行：

```
import cv2
cv2.__version__
```

执行后能看到版本信息就表示安装成功，如果无法看到版本信息，需重新执行之前的命令进行安装。

9.2.5　编译安装 Protobuf 组件

这是整个安装过程中最为耗时的一个步骤，请大家保持耐心，首先安装编译需要的工具。

```
sudo apt-get -y install autoconf automake libtool curl
```

1. 下载最新的源码，编译安装

```
wget https://github.com/protocolbuffers/protobuf/releases/download/
v3.10.0-rc1/protobuf-all-3.10.0-rc-1.tar.gz
```

2. 解压源码

在终端中进入存放压缩包的目录，进行解压缩和编译安装。为了增加编译的速度，使用了参数 -j4，表示调用 4 个核心一起进行编译，速度会快很多。在进行安装时，因为涉及文件系统的一些改动，所以在 make install 命令前要加入 sudo 命令保证权限，如下：

```
tar -xf protobuf-all-3.10.0-rc-1.tar.gz
cd protobuf-3.10.0-rc-1/
./configure
make -j4
sudo make install
```

代码中的 tar 命令是进行压缩包解压的命令，其中的 -xf 是选项，x 是提取，f 是指定文件名。对于 Linux 的源码编译安装，大多时候的执行步骤都是用 configure 命令检查编译环境，也称为预编译。接着 make 进行编译，sudo make install 进行安装，如图 9-1 所示。

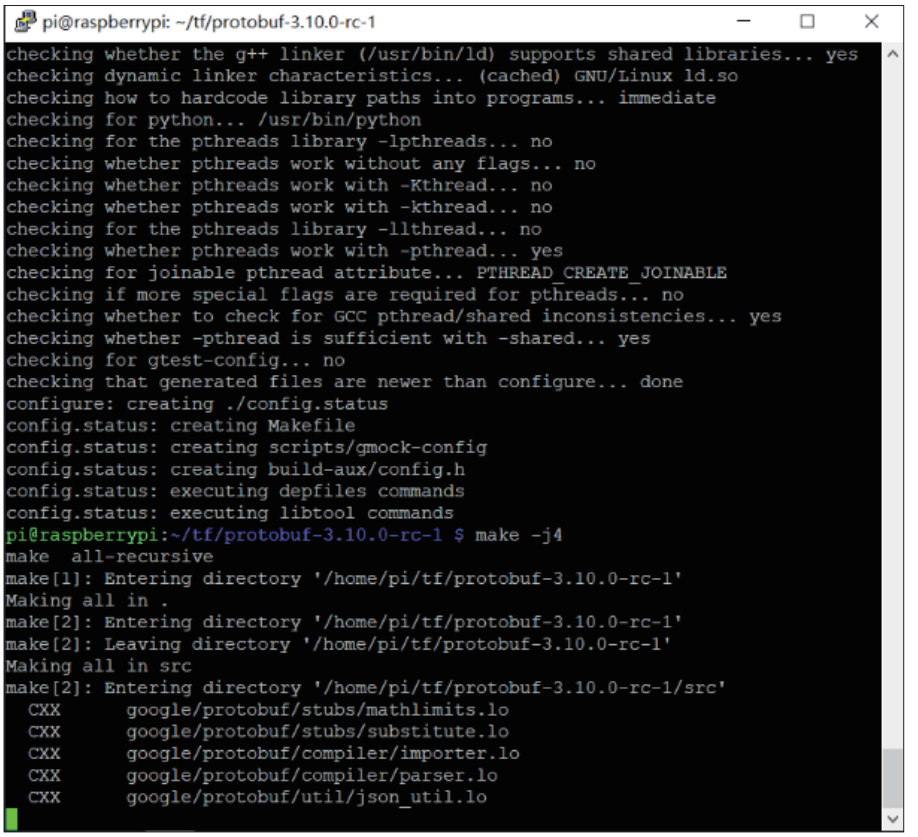

图 9-1　编译 Protobuf

在进行编译时，请大家一定注意 CPU 温度，可以再打开一个终端，然后输入：

```
vcgencmd measure_temp
```

这样执行一次命令就读取一次 CPU 的温度信息，如果需要连续获取，可以在终端中执行一个 Shell 脚本来运行一个死循环来监控 CPU 温度状态，在终端中输入：

```
while true
do
vcgencmd measure_temp
sleep 3
done
```

这样，屏幕上会一直打印当前 CPU 温度的信息，终止它的方法就是按下 Ctrl+C 组合键。如果想查看温度升高的过程，可以利用 sysbench 软件来压测，大家可以直接通过命令安装 sysbench：

```
sudo apt-get update
sudo apt-get -y install sysbench
sysbench --test=cpu --cpu-max-prime=20000 --num-threads=4 run
```

在另一个终端中执行前面的 Shell 脚本循环，就可以实时看到 CPU 温度的信息了。

3. 调整环境变量

为了能够很好地识别 Protobuf 的库文件，需要通过下面的命令调整环境变量：

```
cd python
export LD_LIBRARY_PATH=../src/.libs
python3 setup.py build --cpp_implementation
python3 setup.py test --cpp_implementation
sudo python3 setup.py install --cpp_implementation
export PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION=cpp
export PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION_VERSION=3
sudo ldconfig
```

4. 测试 Protobuf 是否成功安装

为了避免过程中出错，建议大家每一步都进行验证，可以直接在终端中执行：

```
protoc
```

如果提示类似“Usage： Protoc [option] PROTO_FILES”字样说明安装成功了。

9.2.6　重启系统

需要使用以下代码重启系统来清理一下缓存信息。一定要执行这一步骤，否则后续步骤中可能会出现意想不到的问题。

```
sudo reboot
```

9.2.7　重新登录系统并设置 TensorFlow 目录结构

系统重启后，登录进入系统并打开终端，在终端中通过下列命令来创建 TensorFlow 的目录结构。当然也可以根据自己的实际情况创建自己的目录结构，主要是为了方便管理。

```
mkdir tensorflow1
cd tensorflow1
```

9.2.8 下载 TensorFlow 模型

当目录结构创建好后，所有预测的部分都需要通过之前训练好的模型来进行解算，因此使用 GitHub 上其他人已经训练好的模型进行尝试。后期如果有需要，可以自己来训练模型，只是自己训练可能需要更长的时间和更多的训练素材，并且需要更高性能的设备和复杂的建模方法。下面尝试一下解算模型部分是否能够正常工作，在目录中执行下面的命令来下载相关模型，如果由于链路问题失败，需多次尝试或者更改 DNS 服务器地址：

```
git clone --recurse-submodules https://github.com/tensorflow/models.git
```

9.2.9 修改用户初始化配置文件

当模型下载好后，为了让用户每次登录到系统都可以使用相同的环境变量，可以将 tensorflow 的变量信息加入到 .bashrc 文件中，Linux 用户登录时会自动执行属主目录中的 .bashrc 这个初始化脚本文件，它是隐藏文件，文件名前方会有一个“.”。因此需要编辑 /home/pi/.bashrc 文件并添加：

```
export PYTHONPATH=$PYTHONPATH:/home/pi/tensorflow1/models/research:/
home/pi/tensorflow1/models/research/slim
```

保存退出后，关闭当前终端并重新打开终端，查看是否在用户登录时初始化了该文件并已经初始化了变量，如果正常就可以继续操作。有的读者可能会选择直接用 souce 命令加文件名的方式初始化，效果是一样的：

```
souce /home/pi/.bashrc
```

9.2.10 利用 protoc 编译 Protocol Buffer 文件

现在，需要使用 protoc 来编译 Object Detection API 使用的 Protocol Buffer（.proto）文件。.proto 文件位于 / research / object_detection / protos 中，需要从 / research 目录来执行命令：

```
cd /home/pi/tensorflow1/models/research
protoc object_detection/protos/*.proto --python_out=.
```

这条命令是将所有以 .proto 后缀结尾的文件转换成以文件名加 _pb2.py 后缀的文件，注意输出的位置是“out=.”部分指定的，这个“.”不能省略，它表示当前目录。

9.2.11　下载 ssdlite_mobilenet_v2_coco 模型

接下来，需要切换进入 object_detection 目录下载轻量级的模型并解压：

```
    cd /home/pi/tensorflow1/models/research/object_detection

    wget http://download.tensorflow.org/models/object_detection/ssdlite_
mobilenet_v2_coco_2018_05_09.tar.gz

    tar -xzvf ssdlite_mobilenet_v2_coco_2018_05_09.tar.gz
```

现在模型位于 object_detection 目录中，可以配置摄像头并开始使用模型来进行检测了。

9.3　对象检测测试

9.3.1　在树莓派上启用摄像头

在树莓派桌面上打开一个终端，然后在终端中输入：

```
    sudo raspi-config
```

在弹出的 GUI 图形界面中，依次选择“Interface Options”“Camera”“Enable”，并确认后，重启树莓派。

9.3.2　下载检测脚本

在当前目录中执行下载脚本的操作：

```
    wget  https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-
Object-Detection-on-the-Raspberry-Pi/master/Object_detection_picamera.py
```

这个脚本是一个例子，可以通过参考该检测文件来编写自己的检测脚本。

9.3.3 接入摄像头

关闭树莓派，并断电，然后将树莓派官方摄像头（CSI 接口）的 FPC 软排线插入树莓派 CSI 接口中并锁紧，确认排线插入正确。启动树莓派，通过树莓派配置菜单启用摄像头，或者通过 raspi-config 工具启用摄像头，如图 9-2 所示。

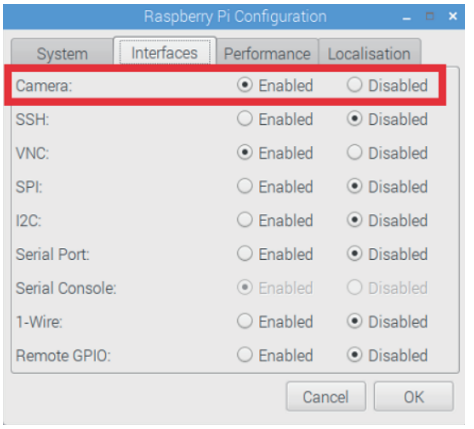

图 9-2 启用摄像头

启用摄像头后，系统会要求重新启动树莓派以便识别摄像头。一定按照系统要求来做，否则摄像头开启不了，检测也会失败。

9.3.4 执行脚本并进行检测

在系统重启完成后，进入之前设置的工作路径，本书的树莓派中当时所在的工作路径是：

```
/home/pi/tensorflow1/models/research/object_detection/
```

通过执行下面的命令来启动对象检测：

```
python3 Object_detection_picamera.py
```

如果使用 PuTTy 或者 Mobaxterm 远程通过 SSH 登录到树莓派时，无法调用桌面的图形环境，因此只能够实现字符界面的操作，在远程终端上，是看不到桌面环境的。由于该实验的程序在执行时会调用图形界面，因此操作时，需要提供显示的桌面信息，在 Linux 系统中，默认桌面的变量 DISPLAY 的值是“:0.0”。在命令的前面加入

DISPLAY=：0.0 参数，来指定特定的桌面并执行相应的命令，如下：

```
DISPLAY=:0.0 python3 Object_detection_picamera.py
```

在执行该命令时，执行的环境为桌面环境中的第一个桌面，Linux 系统可以多个桌面共存，其中 HDMI 接口所接的屏幕显示的内容为第一个桌面的内容，如图 9-3 和图 9-4 所示。

图 9-3　识别效果 1

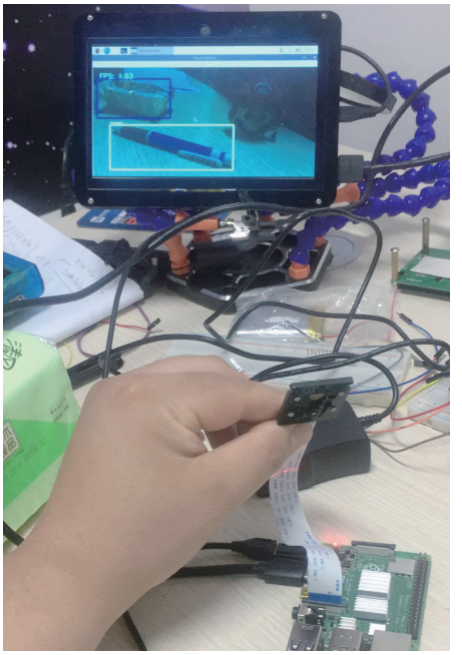

图 9-4　识别效果 2

在识别对象的过程中，可以看到其 FPS 最大帧率是 2 帧左右，可以通过调整分辨率来提高帧率，从而达到快速检测的目的。

9.4 总 结

这个实验的步骤非常多，在操作的过程中经常会因为网络不稳定或者软件包版本变化而无法进行下去，请读者保持耐心，一点点逐步推进，一定能够完成这个神经网络的小项目的！

第 10 章
树莓派扫描仪——树莓派 + OpenCV

引　言

OpenCV 是一个视觉框架，通过该框架，能够完成很多有趣的操作，它使计算机能够通过摄像头来模仿人类观察世界。前面的章节已经介绍了 OpenCV 的一些基础信息，接下来通过实战来更深入了解一些 OpenCV 的有趣之处。

在这一章，通过树莓派结合 OpenCV 的视觉框架，通过一个摄像头来制作一个实时的文档扫描仪，当需要扫描文档时，也许就能派上用场。

10.1　原　理

在树莓派上，可以通过摄像头采集原始数据，并通过 OpenCV 框架提供的各种 API（Application Programming Interface，应用程序接口）调用，快速实现对摄像头采集的图像的处理。例如，通过 OpenCV 库函数中的 imread 方法读取一张图片，然后通过其 getPerspectiveTransform 函数透视变换将图片投影到一个新的视平面

（Viewing Plane），从而将图片通过投影投射为特定需求的图形，也就是将一张看起来倾斜的图片，通过变换，将其变换为正面视图的图片。通俗的说，就是把原本的 2D 图像通过 OpenCV 完成几何变换，不改变图像的内容，只是使像素网格变形并将变形的网络映射到目标图像。

透视变换时需要用到的一个重要方法，也被称为投影映射（Projective Mapping）。以如图 10-1 所示的倾斜文档为例，当通过 OpenCV 进行矩阵变换后，得到的鸟瞰视图效果如图 10-2 所示。

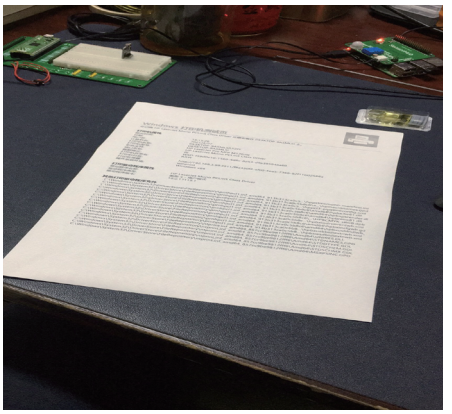

图 10-1　倾斜的文档

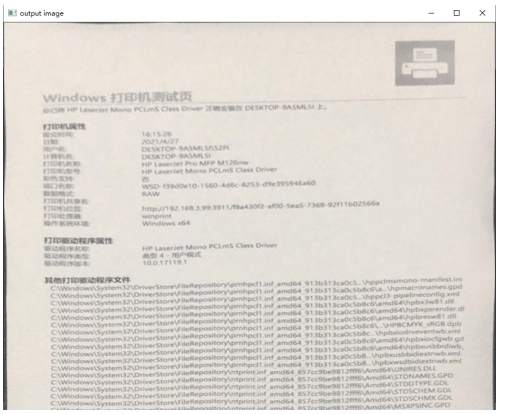

图 10-2　鸟瞰视图

这只是一张静态图的变换，如果结合摄像头采集图片，然后进行批量的转换，那么就可以实现摄像头直接采集并转换文档了，类似一个扫描仪。

10.2　硬件准备

实现这个实验需要准备的硬件如下：

- 一个树莓派开发板，任何型号均可，本书实验环境均为树莓派 4B 4GB 版本。
- 一个树莓派官方摄像头（CSI 接口）或者一个 USB 接口摄像头（要求支持 UVC 驱动）。
- 一张 32GB TF 卡，尽量选择 U3 以上的，以前的 Class10 的速度比较慢，不建议使用。
- 一个 5V/3A 足量的电源（接口要求是 USB Type-C 类型）。
- 一个摄像头支架（可选），主要是为了方便调试。

准备完成后，将树莓派的摄像头 FPC 软排线按照图 10-3 所示的方式接入树莓派的 CSI 接口，并扣紧卡扣。

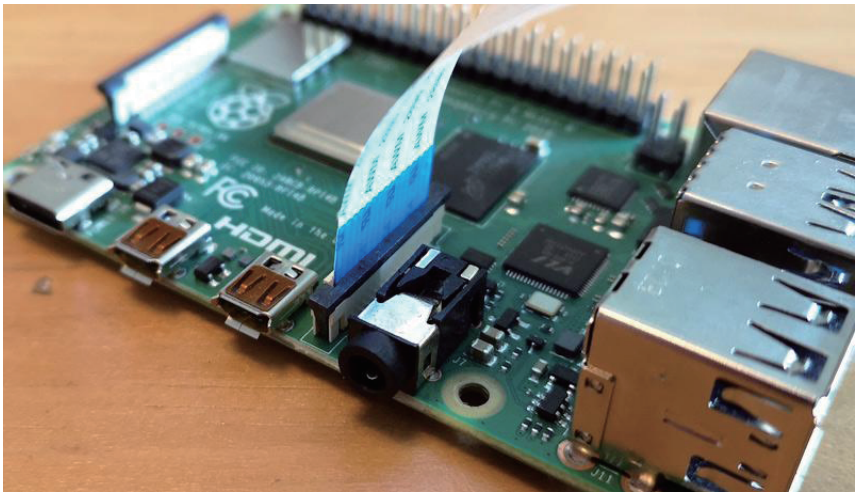

图 10-3　安装树莓派摄像头

10.3　软件准备

10.3.1　软件包版本信息

软件方面需要使用 Raspberry Pi OS，当前使用的操作系统版本如下：

- Raspbian GNU/Linux 10 (buster)；
- virtualenv 版本：15.1.0；
- numpy 版本：1.20.2;
- OpenCV-Python 版本：4.5.1.48；
- Python 版本： 3.7.3。

10.3.2　查看软件版本的方法

如果想要查看当前软件包的版本信息，可以在命令终端中执行下面的命令：

- 查看系统信息：lsb_release –a ;

- 查看 virtualenv 版本：virtualenv –version ;
- 查看 numpy 版本：pip list |grep numpy;
- 查看 OpenCV 版本：pip list |grep opencv-python;
- 查看 Python 版本：python3 –v。

10.4 配置环境

10.4.1 检查网络状态

通过 etcher 烧录好系统后，建议大家完整地按照下列提供的步骤进行配置。在确保树莓派能够很好地连接网络的情况下，打开一个终端窗口，快捷键是 Ctrl+Alt+T。

测试联网的方式是在终端中输入 ping –c 4 www.baidu.com，如果看到类似如下内容，表示通信正常：

```
(venv) pi@RPi8g:~/venv $ ping -c 4 www.baidu.com
PING www.wshifen.com (103.235.46.39) 56(84) bytes of data.
64 bytes from 103.235.46.39 (103.235.46.39): icmp_seq=1 ttl=47 time=74.7 ms
64 bytes from 103.235.46.39 (103.235.46.39): icmp_seq=2 ttl=47 time=74.2 ms
64 bytes from 103.235.46.39 (103.235.46.39): icmp_seq=3 ttl=47 time=74.4 ms
64 bytes from 103.235.46.39 (103.235.46.39): icmp_seq=4 ttl=47 time=74.1 ms

--- www.wshifen.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 4ms
rtt min/avg/max/mdev = 74.150/74.408/74.696/0.315 ms
```

如果提示是 request timeout，则需要检查网络配置并重新检测网络状态。

10.4.2 更新软件仓库并安装软件

在终端中执行下面三条命令，分别是更新软件仓库索引信息、升级软件包、安装 virtualenv 的虚拟环境。

```
sudo apt-get update
sudo apt-get -y upgrade
sudo apt-get -y install virtualenv
```

10.4.3　创建并激活虚拟环境

当安装顺利完成后，在终端中继续执行下面的命令：

```
cd ~
virtualenv -p python3 venv
cd venv
source bin/activate
```

第一条命令表示进入 pi 用户的家目录（home directory），绝对路径就是 /home/pi/ 。第二条命令表示创建一个虚拟环境 venv，这个命令大家可以根据自己项目随便定制，此处使用 venv，比较容易看出来是在自己的虚拟环境中，因此一般创建虚拟环境都使用这个名字； –p python3 的意思是使用 Python3 作为命令解释器。树莓派系统默认支持两个 Python 的版本：一个是 Python2.7.16，另一个是 Python3.7.3，我们需要用 Python3 的环境，因为它也是未来的主流方向。第三条命令表示进入虚拟环境的目录，由于虚拟环境需要激活才可以使用，用 source bin/activate 激活虚拟环境，就会在命令提示符的最前方看到虚拟环境的名字信息，如图 10-4 所示。

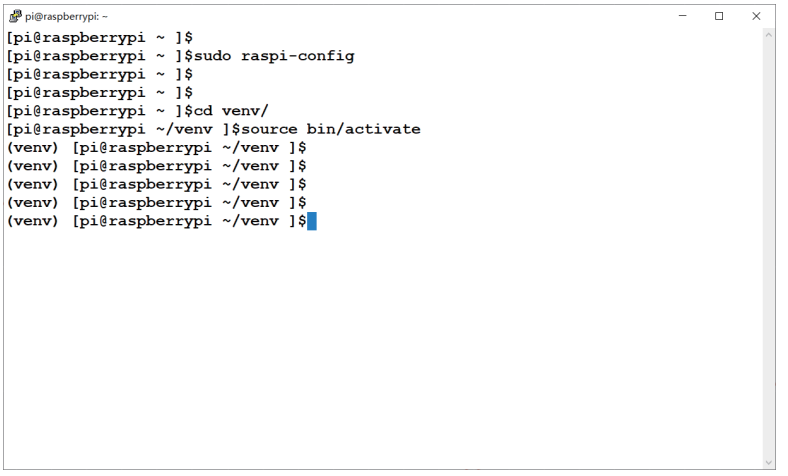

图 10-4　虚拟环境激活状态

10.4.4　启用摄像头

打开摄像头的操作如果在第 9 章已经完成，则跳过该步骤，如果之前没有执行过，则需要按照下面的操作完成，打开一个终端并在终端中输入：

```
sudo raspi-config
```

按 Enter 键后，依次选择如图 10-5~ 图 10-7 所示的选项，完成启用摄像头的配置。

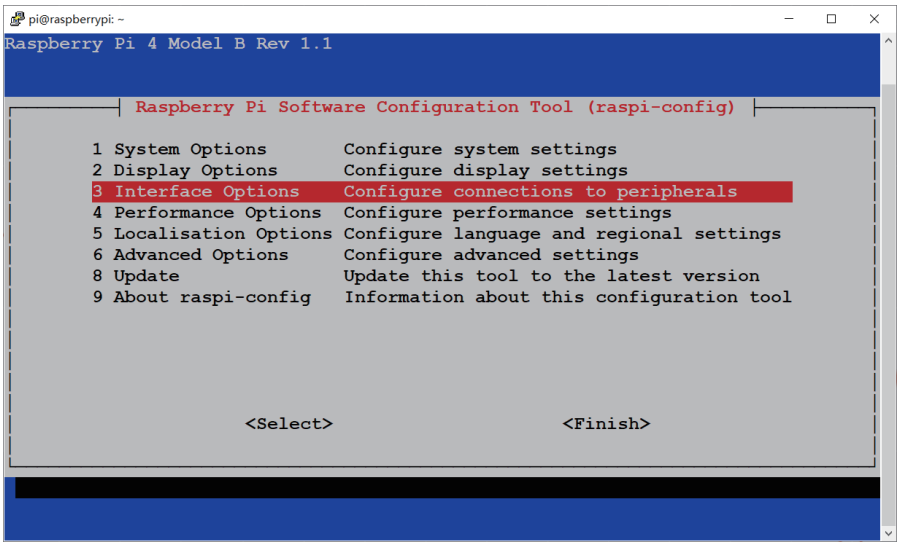

图 10-5　启用摄像头 1

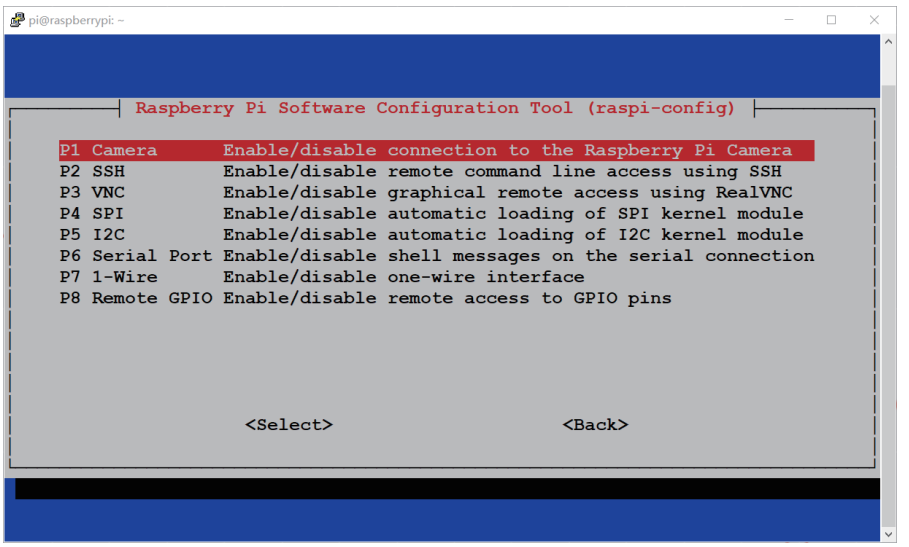

图 10-6　启用摄像头 2

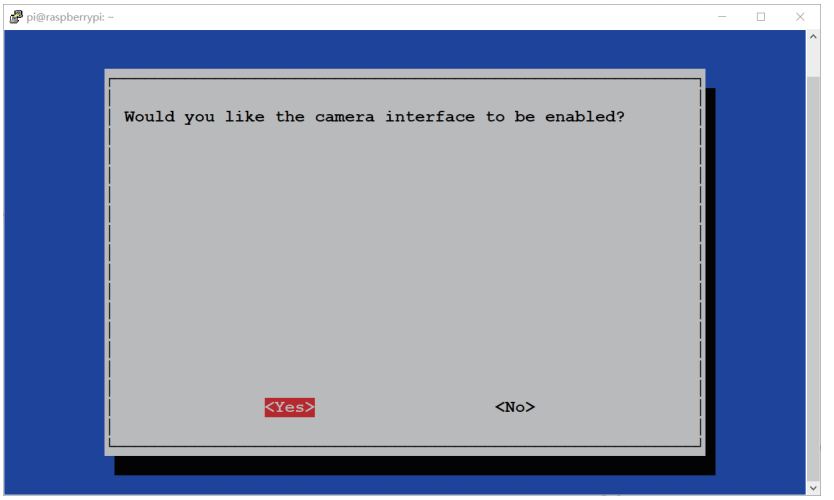

图 10-7　启用摄像头 3

配置完成后，重启树莓派使配置生效。

10.5　代码调试

经过上面的配置，硬件环境就已经搭建好了，接下来就可以启动树莓派并打开一个终端或者直接打开编辑器进行代码编写。初学者可以使用 nano 编辑器，但是本书推荐使用 vim.tiny 编辑器，刚开始可能会觉得比较难，但是一旦入门了这个编辑器以后，基本上在所有的 Linux 发行版上都可以用它顺畅地进行文档编辑了。

首先通过编辑器编写代码如下：

```
import cv2
import numpy as np

circles = np.zeros((4, 2), np.int)      # 用 numpy 创建一个圆的矩阵
counter = 0


def mousePoints(event, x, y, flags, params):    # 定义鼠标单击后获取的位置函数
global counter
if event == cv2.EVENT_LBUTTONDOWN:
print(x, y)
circles[counter] = x, y
counter += 1
```

```
img = cv2.imread('a4.jpg')                    # 读取图片信息

while True:
if counter == 4:                              # 判断鼠标是否单击了四次
width, height = 600, 800
pts1 = np.float32([circles[0], circles[1], circles[2], circles[3]])
                                              # 原图的四个点坐标
pts2 = np.float32([[0, 0], [width, 0], [0, height], [width, height]])
                                              # 转换后的目标图四个角的坐标
matrix = cv2.getPerspectiveTransform(pts1, pts2) # 构建透视变换矩阵
output = cv2.warpPerspective(img, matrix, (width, height)) # 输出转换透视图
cv2.imshow('output img', output)              # 显示转换后的图

for i in range(0, 4):  # 为了方便识别单击的位置，在单击的位置上画实心圆
cv2.circle(img, (circles[i][0], circles[i][1]), 3, (0, 255, 0), cv2.FILLED)
cv2.imshow("origin img", img)                 # 显示原图
cv2.setMouseCallback("origin img", mousePoints)  # 设置鼠标回调函数
if cv2.waitKey(1) & 0xFF == ord('q'):
break

cv2.destroyAllWindows()
```

将文件保存为 mouse_warpperspective.py，然后通过命令行执行：

```
python3  mouse_warpperspective.py
```

执行后，用鼠标在页面的白色区域的四个顶角单击一下，如图 10-8 所示。

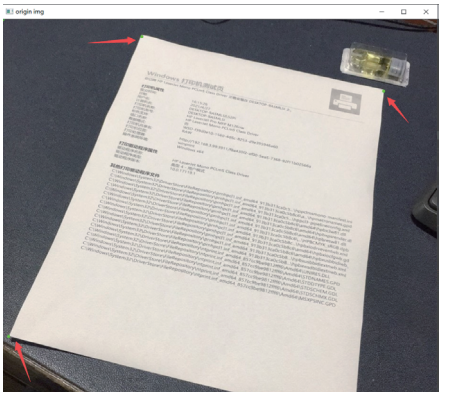

图 10-8　单击位置示意

最后一个点可以单击白纸右下角的位置，大致和右上角的点平行即可。这时就会显示出转换后的图形，如图 10-9 所示。

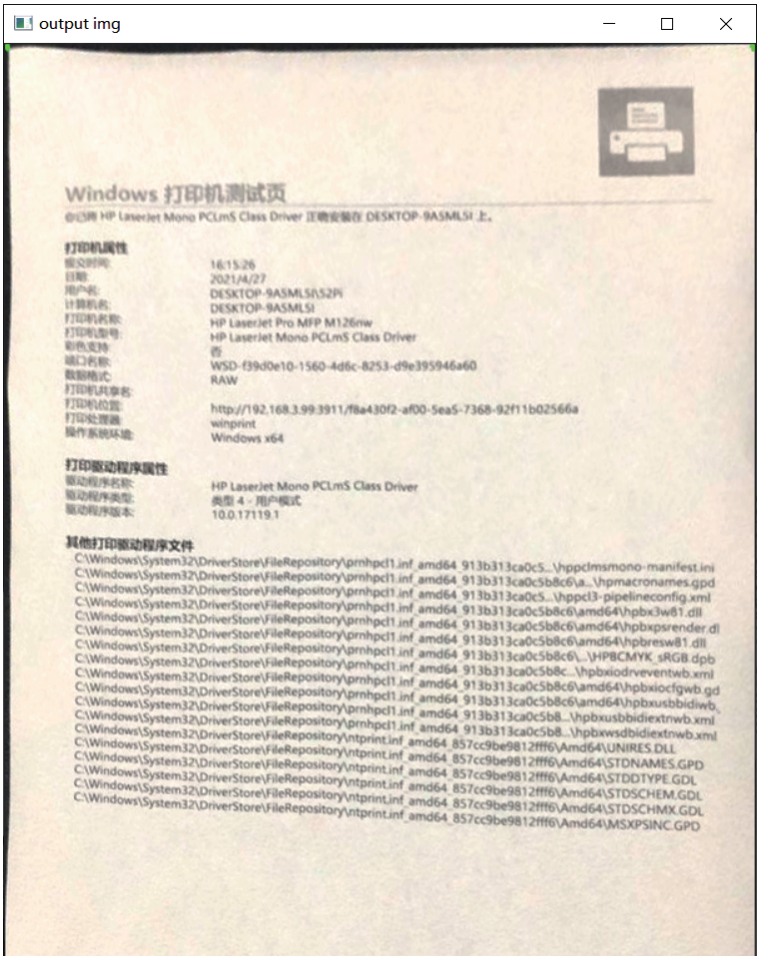

图 10-9　透视变换后的图形

接下来详细分析一下代码的内容：

```
import cv2
import numpy as np
```

导入 cv2 的库和 numpy 的库，它们提供的方法在后面会用到。

```
circles = np.zeros((4, 2), np.int)
counter = 0
```

利用 np.zeros 创建一个 4×2 的全零矩阵，作为鼠标点到屏幕上时的圆点，并且定义一个变量 counter。

```
def mousePoints(event, x, y, flags, params):
global counter
if event == cv2.EVENT_LBUTTONDOWN:
print(x, y)
circles[counter] = x, y
counter += 1
```

创建一个函数 mousePoints，给它传递的参数为鼠标单击时的事件信息。其中，x、y 为鼠标单击时的坐标；flags 和 params 是传递进来的参数。将 counter 变量声明成全局变量，目的是在函数中能够一直访问变量的内容，一般会用 global 声明一次。if 语句中调用了 cv2 的 LBUTTONDOWN 的鼠标单击事件，然后打印了当鼠标单击时在屏幕上的 x、y 坐标，并且将坐标添加到 circles 矩阵中，单击一次 counter 中的计数就会加 1。

```
img = cv2.imread('a4.jpg')
```

利用 cv2 的 imread 方法读取一张图片，这个图片包含着如图 10-1 所示的内容，是一个倾斜的文档。

```
while True:
if counter == 4:
width, height = 1600, 900
pts1 = np.float32([circles[0], circles[1], circles[2], circles[3]])
pts2 = np.float32([[0, 0], [width, 0], [0, height], [width, height]])
matrix = cv2.getPerspectiveTransform(pts1, pts2)
output = cv2.warpPerspective(img, matrix, (width, height))
cv2.imshow('output img', output)

for i in range(0, 4):
cv2.circle(img, (circles[i][0], circles[i][1]), 3, (0, 255, 0), cv2.FILLED)

cv2.imshow("origin img", img)
cv2.setMouseCallback("origin img", mousePoints)
if cv2.waitKey(1) & 0xFF == ord('q'):
break

cv2.destroyAllWindows()
```

这一段进入一个 while 循环，循环中不断地判断 counter 的数字，当 counter 是 4 时，意味着在屏幕上单击了鼠标四次，即顺时针单击了需要转换的图片的四个顶点。首先，定义了输出图片的宽度和高度；然后，定义原始图片的四个顶点的坐标矩阵，并存储到 pts1 变量中，接着定义目标图片的四个顶点坐标的矩阵数据，并存储到 pts2 变量中；接着，利用 cv2 的 getPerspectiveTransform 方法实现一个变换矩阵的参数 matrix；最后，使用 warpPerspective 方法对原图进行转换，转换的参考矩阵利用 matrix，并且根据定义好的宽度和高度进行转换，输出后利用 imshow 显示这张图。

```
for i in range(0, 4):  # 为了方便识别点击的位置，在点击的位置上画实心圆
cv2.circle(img, (circles[i][0], circles[i][1]), 3, (0, 255, 0), cv2.FILLED)
```

在 for 循环里只遍历 0~3 这四个数据，是为了在鼠标按下时，在对应的 x、y 坐标中提供一个实心的圆（cv2.FILLED），这里使用了 circle 方法来绘制圆圈图形，其中的参数为 cv2.circle（绘制到哪里，绘制时的 x、y 坐标，线宽，颜色，实心还是空心圆）。

```
cv2.setMouseCallback("origin img", mousePoints)
```

该语句是设置鼠标的回调函数，当在原图上单击时，调用 mousePoints 函数获取相应的鼠标单击次数和坐标位置。

```
if cv2.waitKey(1) & 0xFF == ord('q'):
break
```

判断用户是否按下了 q 键实现退出。这个 waitKey ( ) 方法是允许用户可以退出程序的交互方法，其中的 1 表示 1s，即循环过程中，会每隔 1s 检测一次是否按下 q 键，等待用户交互。

10.6　应用拓展

在上述代码的基础上，可以利用树莓派定制一个简单的扫描仪，通过摄像头采集图片，然后进行透视变换，大家想一下如何修改代码？

```
import cv2
import numpy as np
```

```
cap = cv2.VideoCapture(0)

circles = np.zeros((4,2), np.int)
counter = 0


def mousePoints(event, x, y, flags, params):
global counter
if event == cv2.EVENT_LBUTTONDOWN:
print(x,y)
circles[counter] = x, y
counter += 1

while True:
ret, frame = cap.read()
if counter == 4:
width, height = 600, 800
pts1 = np.float32([circles[0], circles[1], circles[2], circles[3]])
pts2 = np.float32([[0,0],[width, 0],[0, height],[width, height]])

matrix = cv2.getPerspectiveTransform(pts1, pts2)
output = cv2.warpPerspective(frame, matrix, (width, height))
cv2.imshow('output_img', output)
cv2.imwrite("output_img.jpg", output)

for i in range(0, 4):
cv2.circle(frame, (circles[i][0], circles[i][1]), 3, (255,0,0), cv2.FILL
ED)
cv2.imshow("origin img", frame)
cv2.setMouseCallback("origin img", mousePoints)
if cv2.waitKey(1) & 0xFF == ord('q'):
break

cap.release()
cv2.destroyAllWindows()
```

在新修改的代码中只添加了 4 句，如下：

- cap = cv2.VideoCapture(0)，实例化了一个 cap 对象，调用了摄像头，由于是第一个摄像头，因此参数填写了“0”，这里的 0 是摄像头索引的 id 号。
- ret, frame = cap.read()，读取摄像头采集到的每一帧。

- cv2.imwrite(“output_img.jpg”, output)，将图片保存到 output_img.jpg 存档。
- cap.release()，比较好的习惯就是每次结束程序前释放摄像头对象，便于下一次调用。

10.7 总　结

在生活中经常会遇到需要图形透视变换的场景，通过 Python 和 OpenCV 的简单调用，很快就可以实现一个图形的透视变换操作，并通过摄像头实时调用，生成一个树莓派扫描仪的应用。其最核心的部分就是通过鼠标的单击生成的矩阵创建目标矩阵，然后通过透视变换将图形变化成我们想要的状态，接着保存到相应的文件。这是一个非常简单的基于视觉框架的应用，大家完成后可以思考一下，如果在学校里需要实现阅卷自动化处理，那么数据采集的第一步的操作就是把卷子的内容扫描出来并调整其图像的状态，然后存储在本地磁盘，是不是就可以采用示例的方法去实现呢？