实验报告 3

1-Wire 总线与聚类算法

范皓年 1900012739　信息科学技术学院

2020 年 10 月 31 日

目录

1 实验目的

1. 了解 1-Wire 单总线通信与信号时序的基本原理
2. 了解温度传感器 DS18B20 的使用，以及树莓派中温度传感器的基本使用方法。
3. 掌握树莓派 1-Wire 总线接口的编程和使用方法
4. 学习 K 均值聚类算法并通过一个两类的实例理解算法思想

2 实验原理

2.1 1-Wire 总线简介

1-Wire 总线是一种特殊的串行通信方式。与目前多数标准串行数据通信方式 SPI/I2C/MICROWIRE 不同，它采用单根信号线，既传输时钟又传输数据，而且数据传输是双向的。1-Wire 总线接口具有节省 I/O 资源、结构简单、成本低廉、便于总线扩展和维护等诸多优点。

1-Wire 总线由一个总线主节点以及一个或多个从节点组成系统组成，主节点通过一根信号线对从节点进行数据读取。主节点一般是微控制器，在本实验中是树莓派 MPU；从节点一般是 1-Wire 器件，在本实验中是温度传感器 DS18B20。

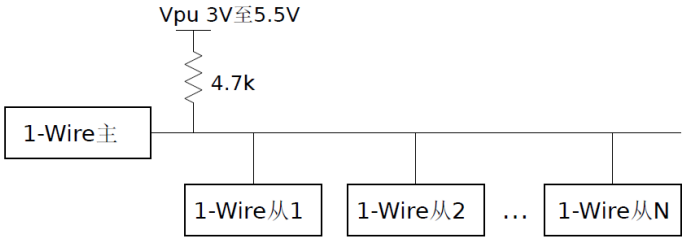


图 1: *1-Wire 总线通信系统模式图*

1-Wire 总线只用一条线进行双向通信，所以对时序的要求比较严格。基本的时序包括如下几种：复位与应答，写一位，读一位。在复位及应答时序中，主器件发出复位信号后，要求从器件在规定的时间内送回应答信号；在位读和位写时序中，主器件要在规定的时间内读回或写出数据。

在本实验中我们只需要从指定的文件夹中读取温度传感器传回的数据即可。更多的内容在温度传感器的部分中说明。

2.2 温度传感器 DS18B20 及其使用方法

硬件准备 实验器材中的 Pioneer 600 扩展板利用树莓派 GPIO 接口扩展了 1-Wire 总线接口，可以方便地接入各种 1-Wire 器件。将三脚的 DS18B20 插入 1-Wire 接口，我们即完成了实验的硬件准备。

驱动准备 树莓派提供了 2 个驱动模块 w1-gpio 与 w1-therm，分别用于通过 GPIO 扩展 1-Wire 总线接口及提供对温度传感器 DS18B20 的读写控制。其中，w1-gpio 提供了 1-Wire 总线的 I/O 操作方法，w1-therm 提供了对温度传感器 DS18B20 的内部操作方法。

在通过 1-Wire 总线访问温度传感器之前，需要确认系统已加载 w1-gpio 与 w1-therm 模块。具体地，需要使用如下命令：

```
pi@raspberrypi:~# lsmod
```

命令之后将输出已经载入系统的模块。如果发现其中有此实验必需的 w1-gpio 与 w1-therm 模块则表示可以正常使用此接口。否则我们需要使用 modprobe 命令加载这两个模块：

```
pi@raspberrypi:~# sudo modprobe w1-gpio
pi@raspberrypi:~# sudo modprobe w1-therm
```

其中，sudo 是 Linux 系统管理指令，允许系统管理员让普通用户执行一些或者全部 root 权限命令。

确认完成 w1-gpio 与 w1-therm 模块的加载后，进入文件系统/sys/bus/w1/devices 目录，并显示当前目录下的所有文件即可。

```
pi@raspberrypi:~# cd /sys/bus/w1/devices
pi@raspberrypi:~# ls
```

执行 ls 命令后在/sys/bus/w1/devices 目录下会发现一个以 28-XXXX 开头的文件夹 (如果接多个 DS18B20，将会看到多个 28-xxxx 的文件，分别对应各个 DS18B20)。进入以 28-XXXX 开头的文件夹，显示文件夹会发现一个名为 w1_slave 的文件，其中存储着温度传感器的有效信息。

利用 cat 命令读取文件 w1_slave 的内容会返回带有温度传感器当前的温度值的 2 行字符。一个示例如下：

```
70 01 4b 46 7f ff 10 10 e1 : crc=e1 YES
70 01 4b 46 7f ff 10 10 e1 t=23000
```

第一行显示了 CRC 校验结果，最后的 YES 表示 CRC 校验成功，数据有效；第二行输出结果中的 t=23000 就是当前的温度值，换算成摄氏度需要除以 1000，即当前温度为 23000/1000=23 摄氏度。

2.3 Python 文件操作

上面的温度值只能通过屏幕显示，为了让树莓派能够自动读取温度传感器 DS18B20 中的温度数据，我们还需要使用 Python 的文件操作。

Python 通过 open 函数可以构造一个文件对象，然后就可以进行文件的读取、写入等操作。open 操作返回一个文件类，后续对这个文件的操作都通过这个类来进行。open 函数的定义如下：

```
open(name[, mode[, buffering]])
```

其中 name 参数为文件的名称，mode 为文件的打开方式，buffering 表示缓冲区的大小。最常用的 mode 参数为”r”和”w”，分别代表读和写的方式，当用”w”的方式打开时，如果文件已经存在，则里面的内容会被清空。还有一种”a”方式用来在文件后面添加新的内容。另外”r+”、”w+”和”a+”都以读写方式打开文件，但”w+”会清空文件，”a+”会将文件指针放到文件末尾。

在完成这个文件的所有操作之后，应当使用 close 方法主动关闭这个文件，这是一个好习惯，可以防止后续的意外读写。close 方法可以调用多次，但如果一个文件对象关闭过后，发生读写操作会抛出 ValueError 错误。

以下是一个文件读取操作完整实例：

```
    f = open(filename, 'r')
    lines = f.readlines()
    f.close()
```

虽然写入文件在本实验中不需使用，我们仍然给出如下的写入文件操作：

```
    f = open(filename, 'a')
    f.write('Last line')
    f.close()
```

为了更加简便且防止忘记关闭文件对象，我们可以使用 with open 语句：

```
with open filename as f:
    pass
```

仅在 with open 缩进区域文件被打开。离开后自动关闭。

如果要查看目录中的文件，可以使用 python 的标准库 glob，如下面的示例打印出 dir 目录下的全部文件：

```
import glob
for name in glob.glob('dir/*'):
    print(name)
```

在 glob 的参数中，可以使用通配符等正则表达式来匹配文件名称。

2.4 聚类算法简介与 K 均值聚类算法

机器学习的核心思想之一，就是将具体的事物的**特征**抽象出来，并借此形成合理的标签，从而能将新的具体事物按特征重新匹配到各个类别中。

表现在聚类算法中，就是将数据集中的样本分成若干个通常不相交的子集，每个子集称为一个簇（cluster）。通过这样的划分，每个簇就可能对应于一个潜在的概念，例如可以把生物分成植物和动物，把书籍分成文史类和科技类等。这样的分类就可以使得机器具有一定的认识具体事物的能力，或者说智能。

基于不同的策略，可以设计出多种类型的聚类算法，这里介绍一种非常简单的 K 均值聚类算法（K-means clustering）。

K 均值聚类算法是一种迭代求解的聚类分析算法，其步骤如下：

(a) 获取原始数据

(b) 随机选取 K 个位置作为初始的聚类中心

(c) 计算每个样本与各个聚类中心之间的距离，把每个样本分配给距离它最近的聚类中心，聚类中心以及分配给它们的样本就代表一个聚类

(d) 每分配完成后，聚类的聚类中心会根据聚类中现有的样本被重新计算，获得新的聚类中心

(e) 这个过程将不断重复直到满足某个终止条件。

(f) 终止条件可以是没有（或最小数目）样本被重新分配给不同的聚类，或者是聚类中心不再发生变化。

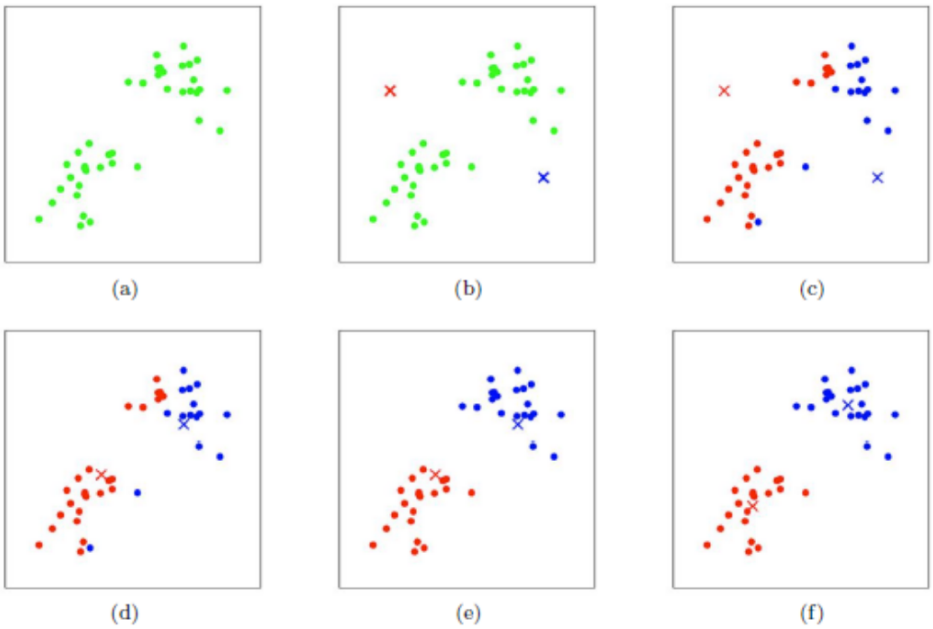


图 2: K 均值聚类算法实现

2.5 Python 科学计算库的使用

NumPy 是 Python 中科学计算的基础软件包。它是一个提供多了维数组对象，多种派生对象（如：掩码数组、矩阵）以及用于快速操作数组的函数及 API，它包括数学、逻辑、数组形状变换、排序、选择、I/O 、离散傅立叶变换、基本线性代数、基本统计运算、随机模拟等等。

NumPy 包的常用数据类型是 ndarray 对象。它与标准 Python Array（数组）之间有几个重要的区别：

1. NumPy 数组在创建时具有固定的大小，与 Python 的原生数组对象（可以动态增长）不同。更改 ndarray 的大小将创建一个新数组并删除原来的数组。
2. NumPy 数组中的元素都需要具有相同的数据类型，因此在内存中的大小相同。例外情况：Python 的原生数组里包含了 NumPy 的对象的时候，这种情况下就允许不同大小元素的数组。
3. NumPy 数组有助于对大量数据进行高级数学和其他类型的操作。通常，这些操作的执行效率更高，比使用 Python 原生数组的代码更少。

以下为使用示例：

```
import numpy as np
mean = 5
sigma = 1.3
x=mean+sigma*np.random.randn(10) # 生成均值为5，方差为1.3的正态分布的10个数
# 也可以直接使用normal
# x = np.random.normal(mean, sigma, 10)
y = np.append(np.random.normal(mena+3, sigma*2, 10)) # 添加元素
m = np.mean(y)
y1 = np.delete(y, 10) # 删除第十个元素
```

Matplotlib 是一个 Python 2D 绘图库，可以生成各种格式和跨平台交互式环境的出版物质量图片数据。它的语法格式类似于 matlab，例如下面代码画出 x 数据的直方图。

```
import matplotlib.pyplot as plt
plt.hist(x,80,histtype='bar',facecolor='yellowgreen',alpha=0.75)
plt.show()
```

Scipy 是一个用于数学、科学、工程领域的常用软件包，可以处理插值、积分、优化、图像处理、常微分方程数值解的求解、信号处理等问题。它用于有效计算 Numpy 矩阵，使 Numpy 和 Scipy 协同工作，高效解决问题。这里我们在最终的数据分析时，对直方图进行正态拟合，使得聚类的效应看起来更加显著。[1]

我们将 matplotlib 作出的图像返回参数承接，随后利用 stats 模块进行正态拟合。利用如下三步可以作出直方图以及对应的正态拟合曲线。

```
n1, bins1, patches1 = plt.hist(
    t1, 80, histtype='bar', density=1, facecolor='blue', alpha=0.75)
y1 = stats.norm.pdf(bins1, np.mean(t1), np.std(t1))
plt.plot(bins1, y1, 'r--')
```

3 实验内容

3.1 读取温度传感器的值

首先利用实验二中的方法初始化 GPIO，本次实验当中的 LED 闪烁利用 PWM 串口实现，所以也要进行 PWM 串口的相关初始化。

主要逻辑是不断地读入温度传感器的数据文件，从中清洗出实际温度，然后判断温度值是否满足达到 30 度，确定是否报警。

```
# 导入GPIO相关操作所需要的库
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)

# 初始化PWM相关串口
LED = 26
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)
pwm = GPIO.PWM(LED, 50)
pwm.start(0)
pwm.ChangeDutyCycle(0)

```

[1] 同时这条曲线意外地帮助我们找到了均值偏离的原因，详见图 4(b)

```
# 准备查找28开头的数据文件
import glob

if __name__ == "__main__":
    while True: # 主要循环，保证温度判断始终进行
        for name in glob.glob('/sys/bus/w1/devices/28*'):
            name+='/w1_slave'
            with open(name, 'r') as f:
                contents = f.readlines()
                s = contents[1].find('t=') + 2 # 温度信息在字符串中的位置
                temp = int(contents[1][s:]) / 1000
                print('current temperature:', temp)
            # 判断是否需要报警
            if temp > 30:
                pwm.ChangeDutyCycle(100)
            else:
                pwm.ChangeDutyCycle(0)
```

3.2 一个聚类算法实例

实例说明：

假定某共用办公室有两个人使用，他们使用办公室的时候都会用空调遥控器设置房间的温度。但空调遥控器显示面板坏掉了，只能通过加减温度的方式盲调整。已知他们习惯的温度不同，办公室的温度计有记录功能，每天记录 12 小时的温度，每 30 分钟记录一次，根据一个月所记录的数据，通过聚类算法，算出这两个人的喜好温度，并估算他们各使用办公室多少时间。可以认为这两个人使用办公室的时候室内温度是不同方差和不同均值的正态分布随机数，采用 randn 函数生成模拟数据，并用 matplotlib 画出数据的直方图。使用 K 均值聚类算法将模拟数据分成两个簇，解答前面的问题。使用温度传感器读入温度，根据温度值，判断是谁在使用办公室

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import glob

k1, k2 = 0, 0


def cluster():
    half_total_times = 360 # 温度记录总数的一半。
    # 为了让结果看起来更加优美，可以适当增大这个值。

    # 绘制图像
```

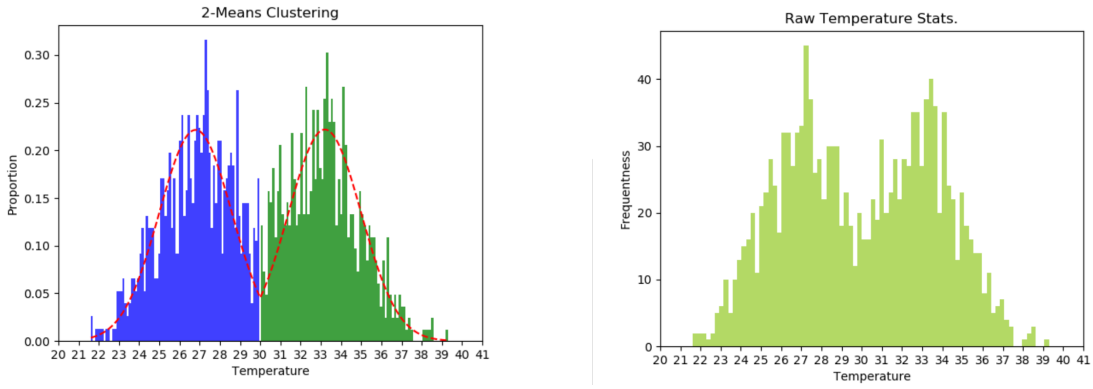
```
    log = np.append(np.random.normal(33, 2, half_total_times),
                    np.random.normal(27, 2, half_total_times))
    plt.hist(log, 80, histtype='bar', facecolor='yellowgreen', alpha=0.75)
    plt.ylabel('Frequentness')
    plt.xlabel('Temperature')
    plt.xticks(np.arange(20, 42, 1))
    plt.title("Raw Temperature Stats.")
    plt.show()

    np.random.shuffle(log)
    # 将预先分堆的两组数据打乱后再分堆，以免聚类算法直接退出。
    t1, t2 = log[:half_total_times], log[half_total_times:]
    global k1
    global k2
    k1, k2 = np.mean(t1), np.mean(t2) # 建立初始聚类中心

    while True:
        flag = True
        i, j = 0, 0
        while i < len(t1): # 对第一个类进行遍历
            tmp = t1[i]
            if abs(tmp-k1) > abs(tmp-k2): # 如果发现类中样本偏离中心就换类
                t2 = np.append(t2, tmp)
                t1 = np.delete(t1, i)
                flag = False # 标记再聚类过程还在进行
                print(tmp, "has been moved to 2")
                continue
            i = i + 1
        while j < len(t2): # 进行和类1一样的操作
            tmp = t2[j]
            if abs(tmp-k1) < abs(tmp-k2):
                t1 = np.append(t1, tmp)
                t2 = np.delete(t2, j)
                flag = False
                print(tmp, "has been moved to 1")
                continue
            j = j + 1
        if flag: # 当聚类已经完成就退出循环
            break
        k1 = np.mean(t1) # 重新设定聚类中心
        k2 = np.mean(t2)
```

```
        print(k1, k2)
    print("final k-means:", k1, k2)
    # 以下开始作图
    n1, bins1, patches1 = plt.hist(
        t1, 80, histtype='bar', density=1, facecolor='blue', alpha=0.75)
    n2, bins2, patches2 = plt.hist(
        t2, 80, histtype='bar', density=1, facecolor='green', alpha=0.75)
    # 利用scipy的组件绘制两个分类的图样并绘制
    y1 = stats.norm.pdf(bins1, np.mean(t1), np.std(t1))
    y2 = stats.norm.pdf(bins2, np.mean(t2), np.std(t2))
    plt.plot(bins1, y1, 'r--')
    plt.plot(bins2, y2, 'r--')
    # 补充图样
    plt.ylabel('Proportion')
    plt.xlabel('Temperature')
    plt.xticks(np.arange(20, 42, 1))
    plt.title("2-Means Clustering")
    plt.show()
```

聚类完成之后，我们将两组数据已经分成两堆具有正态特性的温度值。作出图像如下，并同先前随机生成的数据集作对照，发现确实将以不同中心所正态分布的两类温度值有效地分成具有显著特征的两类：



(a) 聚类结果：温度值分成具有特征的两类

(b) 第一次作图：原始数据记录

图 3: 聚类结果与原始数据对照

```
final k-means: 26.863610073071495 33.09840875548246
```

接下来为判断函数。借用 3.1中的温度读取方法，将温度值和两个聚类中心进行对照。

```
def judge():
    try:
        while True:
            for name in glob.glob('/sys/bus/w1/devices/28*'):
```

```
                name += '/w1_slave'
                with open(name, 'r') as f:
                    contents = f.readlines()
                    s = contents[1].find('t=') + 2
                    temp = int(contents[1][s:]) / 1000
                    print('current temperature:', temp)
                    if abs(temp - k1) < abs(temp - k2):
                        print("It's person 1 here")
                    else:
                        print("It's person 2 here")
    except KeyboardInterrupt:
        pass


if __name__ == "__main__":
    cluster()
    judge()
```

3.3 思考题

在上面的实例当中，我们也看到，得出的两个聚类中心并不等于我们所设定的值：

```
final k-means: 26.863610073071495 33.09840875548246
```

当数据量较小时，这个差距是由数据集本身的特性决定的。由于数据集过小，随机性使得平均值并不能较完美地趋近中心值。事实上，选定 half_total_times 为一个 360 时，每一次的结果都会有较大的差异。

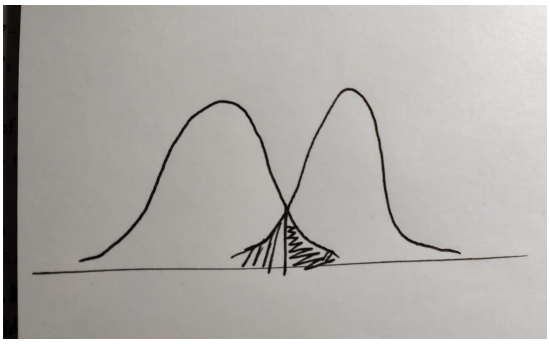
于是我们猜测当数据量较大时，这个值将会排除随机因素，趋于一个稳定的值。依次将数据量扩大，作如下的试验：

```
# htt : half_total_times
final k-means: 33.12662153907564 26.9236373559599 # htt = 5000
final k-means: 33.12400746682427 26.867736591964416 # htt = 20000
final k-means: 26.89201367317691 33.13533873227883 # htt = 50000
final k-means: 33.13308044620618 26.87871264214395 # htt = 100000
final k-means: 26.873664094790183 33.113832074382046 # htt = 150000
final k-means: 26.87391229842333 33.104500152812 # htt = 200000
final k-means: 33.11234652005665 26.885121276955825 # htt = 250000
final k-means: 26.876981068264236 33.11720891682733 # htt = 300000
final k-means: 26.89036390407023 33.112480044788235 # htt = 350000
final k-means: 26.88380554470867 33.121317671847606 # htt = 400000
final k-means: 26.886840666914292 33.120665211354314 # htt = 450000
final k-means: 33.118645760649365 26.87833700669375 # htt = 500000
final k-means: 33.11972276310472 26.883882190093534 # htt = 550000
```

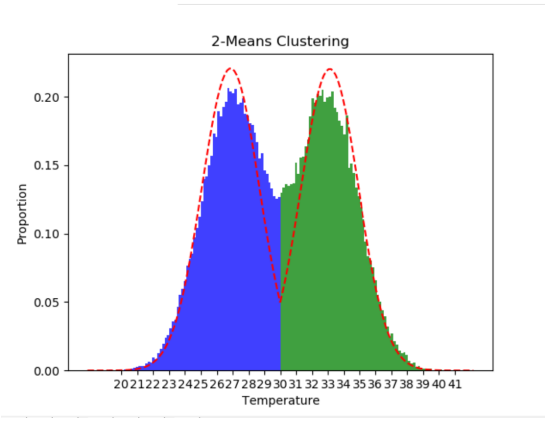
```
final k-means: 26.882009336811468 33.11476481580636 # htt = 600000
final k-means: 26.885940661245822 33.11899334301187 # htt = 650000
final k-means: 33.122877982248134 26.884929683501547 # htt = 700000
final k-means: 26.882989502619292 33.11483313992061 # htt = 750000
final k-means: 26.880544838547046 33.112846759715936 # htt = 800000
```

我们不难看到，k 均值也并没有收敛到正态的中心设定值。高温集合的均值偏大，低温集合的均值偏小。

这一点从原始数据其实是不难理解的。两个正态曲线有一部分交错，由于聚类算法过程只按距离分配，低温的正态的波浪线部分分配给高温类，高温的正态的斜线部分分配给低温类，如图 4(a)，所以两类当中各自并不是一条完整的正态曲线的数据集，如图 4(b)。直方图和红色的正态曲线有较大的差异。聚



(a) 聚类偏离原理示意图



(b) 实际聚类图样：沿中线分成两组非正态数据

类算法使得交错的部分不能有效地归入实际上的两类类，而是数学上的两类。从而导致了最终的系统性的稳定偏离，收敛到 33.112(3) 和 26.882(4)，而不是 33 和 27。

A 附录代码

这里的代码不带有行号，易于复制并试运行。

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)

LED = 26
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)
pwm = GPIO.PWM(LED, 50)
pwm.start(0)
pwm.ChangeDutyCycle(0)

import glob

if __name__ == "__main__":
    while True:
        for name in glob.glob('/sys/bus/w1/devices/28*'):
            name+='/w1_slave'
            with open(name, 'r') as f:
                contents = f.readlines()
                s = contents[1].find('t=')
                s = s + 2
                temp = int(contents[1][s:]) / 1000
                print('current temperature:', temp)
            if temp > 30:
                pwm.ChangeDutyCycle(100)
            else:
                pwm.ChangeDutyCycle(0)
```

kmeans.py

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import glob

k1, k2 = 0, 0


def cluster():
```

```
    half_total_times = 5000
    log = np.append(np.random.normal(33, 2, half_total_times),
                    np.random.normal(27, 2, half_total_times))
    plt.hist(log, 80, histtype='bar', facecolor='yellowgreen', alpha=0.75)
    plt.ylabel('Frequentness')
    plt.xlabel('Temperature')
    plt.xticks(np.arange(20, 42, 1))
    plt.title("Raw Temperature Stats.")
    plt.show()

    np.random.shuffle(log)
    t1, t2 = log[:half_total_times], log[half_total_times:]
    global k1
    global k2
    k1, k2 = np.mean(t1), np.mean(t2)

    while True:
        flag = True
        i, j = 0, 0
        while i < len(t1):
            tmp = t1[i]
            if abs(tmp-k1) > abs(tmp-k2):
                t2 = np.append(t2, tmp)
                t1 = np.delete(t1, i)
                flag = False
                print(tmp, "has been moved to 2")
                continue
            i = i + 1
        while j < len(t2):
            tmp = t2[j]
            if abs(tmp-k1) < abs(tmp-k2):
                t1 = np.append(t1, tmp)
                t2 = np.delete(t2, j)
                flag = False
                print(tmp, "has been moved to 1")
                continue
            j = j + 1
        if flag:
            break
        k1 = np.mean(t1)
        k2 = np.mean(t2)
```

```
        print(k1, k2)
    print("final k-means:", k1, k2)
    n1, bins1, patches1 = plt.hist(
        t1, 80, histtype='bar', density=1, facecolor='blue', alpha=0.75)
    n2, bins2, patches2 = plt.hist(
        t2, 80, histtype='bar', density=1, facecolor='green', alpha=0.75)
    y1 = stats.norm.pdf(bins1, np.mean(t1), np.std(t1))
    y2 = stats.norm.pdf(bins2, np.mean(t2), np.std(t2))
    plt.plot(bins1, y1, 'r--')
    plt.plot(bins2, y2, 'r--')
    plt.ylabel('Proportion')
    plt.xlabel('Temperature')
    plt.xticks(np.arange(20, 42, 1))
    plt.title("2-Means Clustering")
    plt.show()


def judge():
    try:
        while True:
            for name in glob.glob('/sys/bus/w1/devices/28*'):
                name += '/w1_slave'
                with open(name, 'r') as f:
                    contents = f.readlines()
                    s = contents[1].find('t=') + 2
                    temp = int(contents[1][s:]) / 1000
                    print('current temperature:', temp)
                    if abs(temp - k1) < abs(temp - k2):
                        print("It's person 1 here")
                    else:
                        print("It's person 2 here")
    except KeyboardInterrupt:
        pass


if __name__ == "__main__":
    cluster()
    judge()
```