2017331200012 郑文彬 基于树莓派的DIY气象站

【原文对照报告-大学生版】

报告编号：e5f4ffc9b1971e14　　检测时间：2021-05-17 20:07:39　　检测字符数：15750

作者姓名：无　　所属单位：浙江理工大学

检测结论：

全文总相似比 **7.37%** = 复写率 7.37% + 他引率 0.0% + 自引率 0.0% + 专业术语 0.0%

其他指标：自写率：92.63%

高频词：数据,通过,实现,感器,传感器

典型相似文章： 无

指标说明：复写率：相似或疑似重复内容占全文的比重　他引率：引用他人的部分占全文的比重

自引率：引用自己已发表部分占全文的比重　自写率：原创内容占全文的比重

典型相似性：相似或疑似重复内容占全文总相似比超过30%　专业术语：公式定理、法律条文、行业用语等占全文的比重

相似片段：

总相似片段 37

期刊：9　博硕：18　综合：0

外文：0　自建库：0　互联网：10

检测范围：

中文科技期刊论文全文数据库　中文主要报纸全文数据库　中国专利特色数据库

博士/硕士学位论文全文数据库　中国主要会议论文特色数据库　港澳台文献资源

外文特色文献数据全库　维普优先出版论文全文数据库　互联网数据资源/互联网文档资源

高校自建资源库　图书资源　古籍文献资源

个人自建资源库　年鉴资源　IPUB原创作品

时间范围：1989-01-01至2021-05-17

原文对照

颜色标注说明：

- 自写片段
- 复写片段（相似或疑似重复）
- 引用片段（引用）
- 专业术语（公式定理、法律条文、行业用语等）

第1章绪论

1.1研究目的与意义

天气是由多种复杂因素影响的结果，直接影响人类活动以及农作物生长的重要因素。世界各国的气象局提供的天气预报是可以实现天气的实时获取以及预测未来天气的状况。而我想通过这次的DIY气象站的研究与设计课题，来学习气象站的工作原理以及熟悉课题的一般研究方法与过程。通过这次的学习与研究,将要实现以下目标：其一了解气象观测的原理研究，其二学习课题的一般研究方法及熟悉该研究过程；并且通过这次的课题研究了解我知识体系的不足之处。

1.2国内外研究现状

气象站是帮助人类测量天气信息的重要手段。气象站根据使用场景有气象卫星、海上气象站以及地表气象站，根据场景的不同，这些气象站也有不同的侧重点，本次课题研究的方向是地表气象站。主要获取实时的常用天气数据，例如温度、湿度、风速、风向、降雨量、紫外线强度等。在地表气象站中可以简单划分为人工观测与自动观测两大类。本次设计的目标是自动观测方向。在国内，从1945年中国共产党在延安建立第一个气象站，到1999年自主生产气象站，再到2018年年底，自动气象站的覆盖率达到了95.6%，跃居世界第一。在自主观测类气象站的观测过程中数据获取主要依靠传感器自主的数据采集、其次借助互联网技术实现本地与云端的数据传输、云端数据持久化存储、然后经过数据分析与数据可视化，直观的显示各类气象数据信息。以下将开始介绍各阶段技术的研究现状。

1.2.1 传感器的发展

传感器是一种将物理信号转化为易于处理的电信号（包含数字信号）的检测装置。自从1947年贝尔实验研制出晶体管以及后续的快速发展，各类传感器的体积不断变小，其成本也不断减小，种类也不断增加。在[3],[4],[5],[6],[11],[12]都讲述了传感器的选择与使用，例如[6]中讲述了利用标准阻值来描述温度、以及利用电压信号描述湿度参数等，在[11]中具体描述了使用DHT11来获取温度、与湿度信息，这也是我在本次设计中使用到的一款传感器。在[12]中还讲解了HC-SR501用于检测人体接近的信号，等等。这些传感器是采集数据的重要组成部分，选择合适的传感器，也是影响采集数据准确度的关键。

1.2.2 云端数据存储

云计算是近年来的技术热点，免维护、即取即用的优势吸引人们不断探索该领域。现如今稳定、快速的4G与廉价的Wi-Fi以及更快速的5G网络，更为其提供发展基础。例如阿里云、腾讯云、Google云、AWS等提供廉价、优质的云服务平台，可以作为本次研究的云端平台。其中数据可以通过HTPP等技术实现本地到云端的传递。通过这些基础技术可以实现将本地数据传递到云端处理。但是在这之前，需要核心控制芯片需要正确接收到传感器采集的数据，在[3-12]均有描述具体的数据传输的过程与方法。其中[3]描述的是利用树莓派的GPIO进行接受传感器传递的数据，[4]中采用RS-485来获取传感器的数据，在[5]中采用了GPIO、ADC、USART、IIC来获取传感器数据，通过这些将传感器的数据获取到，就可以同一发送到云端平台进行存储、处理可视化等，在[4-12]提供了以下技术手段实现该功能：借助云平台提供的接口，例如阿里云物联网平台提供的Link Kit可以快速实现；大部分云平台都提供快速接入的API，可以快速部署实现。

1.2.3数据的可视化

数据的可视化是数据计算结果的最终体现，也是整套方案的价值体现，在[14]中提供了利用Python实现数据可视化的方法，同时[13]也提供了基于微信小程序的数据可视化实现方案，以及最普遍的提供网页展示的实现方案，通过数据的可视化，可以直观、明确的展示采集的数据，同时也是体现这一套系统整体价值的直观表现。经过数据采集、数据传输、数据存储、数据可视化的整套方案，及可以实现一个DIY气象站的设计，当然在查阅资料的过程中，发现已有许多的实现方案可供参考，这也为了该课题的研究提供的指导性的作用。

1.3 研究内容与论文结构

这篇文章主要讲述了一个DIY气象站的设计过程。主要通过五个章节来讲述这一方案。第1章主要介绍本次设计的研究目的与意义，并且介绍传感器的数据获取、云端数据存储以及数据可视化的研究现状。第2章主要介绍通过设计合适的传感器实现天气数据的获取过程。第3章详细介绍本地数据到云端的数据传输以及在云端的存储过程。第4章主要介绍云端数据的可视化过程。第5章对本文进行总结，并对气象站将来的发展进行展望。

第2章天气数据的采集

本章节主要描述如何对天气信息的数据进行采集。对于气象站的设计需要采集温度、湿度、风速、风向、降雨量、紫外线强度等。首先在本次设计中只摄取温度、湿度、风速以及风向的数据，并在王城设计的基础上可以快速增加其他天气信息的获取。在获取温度、湿度的过程中，根据查阅的资料，可以采用DHT11这款传感器，该传感器可以直接获取到温度与湿度的数据，并通过串口通信将数据传递给树莓派；风速传感器需要借助一款反射传感器CNY70，将采集风叶的转动角度，并通过该角度计算出风速的指数；而风向传感器需要借助TOPVR的无极电位器，将风向指数转化为标准阻值，并通过ADC模块将数据传递给树莓派。以下将具体描述给各类数据获取的具体过程。

2.1风速数据的获取

风速数据是大气运动距离对时间的微分，该物理量表示每秒空气移动的距离。该数据不能直接观测，树莓派也无发直接测量。需要借助CNY70这款反射传感器。

图2-1 CNY70反射传感器

该款传感器有一枚二极管与晶体管组成，二极管是一枚红外发射器，晶体管是光电晶体管。该晶体管可以根据反射面的不同，光电晶体管产生不同的电压响应。通过以下电路将实现对数据的采集。

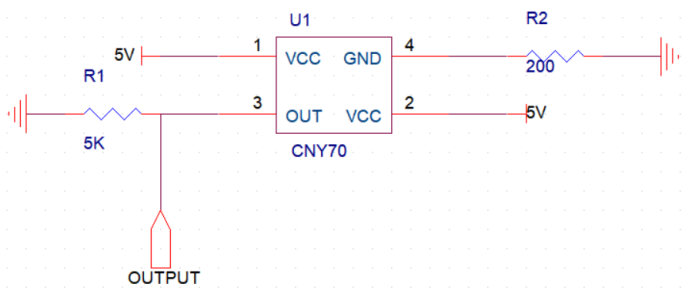

图 2-2 CNY70采集电路

在采集之前还需要借助编码盘对风叶进行编码，这里采用30° 的黑白编码盘。

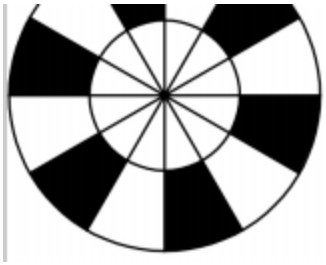

图 2-3 黑白编码盘

通过风叶的转动，编码盘的黑白区域在经过CNY70时会产生不同的电压，该电压的值与反射面与反射面与传感器的距离有关。

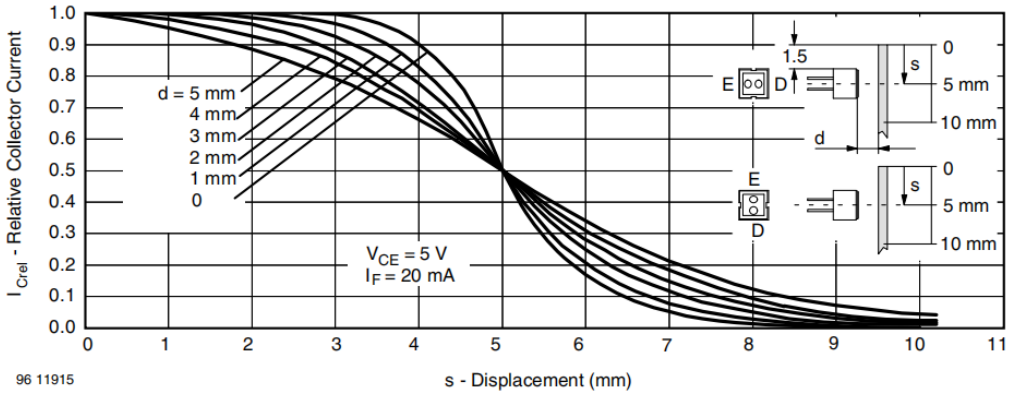

图 2-4 CNY70的距离与输出电流的关系

通过资料与实现得知，在距离为5mm时，电路在白色区域输出电压为4.5V左右，在黑色区域输出0.4V左右。通过以上的方案可议实现风速指数的电信号转化。先通过风叶可以将运动距离转化为转动角度；之后通过光电传感器读取风叶转盘上的编码盘从而记录一段时间的转动的角度，再根据角速度与线速度之间的线性关系得出风速。

2.1.1 数据转换原理

在风速传感器中，风速物理量到数字量之间的转换经历两次转换，第一次是利用风叶将距离对时间的微分转化为角度对时间的微分，之后通过光电传感器CNY70检测风叶编码盘，将转动角度转化为高低点电平，之后通过树莓派的GPIO对电平转变进行计数。这样就是实现了物理量到电信号的转变。在通过编码盘时CNY70在不同的区域会产生不通过的电压信号，通过电压信号的高低从而判断经历了多少的区域，将转动角度实现电信号的量化，该数据的嘴角记录值是30°，也就是电平的每一次变化代表风叶转动了30°。这样就实现了风速到电信号的转变。

2.1.2 数据通信协议

在该传感器中没有使用特定的通信协议，传感器直接转化为高低电平信号转递给树莓派。树莓派通过GPIO口来感知电平的变化，并记录电平的变化次数。

```
GPIO.setup(port,GPIO.IN)
VALUE = GPIO.input(port)
START = time.time()
while True:
NEW_VALUE = GPIO.input(port)
if ( VALUE != NEW_VALUE ):
# print(VALUE, NEW_VALUE)
COUNT += 1
# print(VALUE, NEW_VALUE, COUNT)
VALUE = NEW_VALUE
if (time.time() - START >= 1 ):
speed =(float)("{:.2f}".format(COUNT*30*0.04))
Break
```

通过以上方法可以实现传感器的数据收集，将风叶的转动角度获取到。

2.2.3 数据处理算法

树莓派接受到风速传感器的电信号只是连续的高低电平，这需要经过一系列的转化才能转化为五我们所需要的风速数据，首先通过一秒内电平的变化次数，按照编码盘可知每次电平变化风叶转动30°，从而可计算出风叶一秒转动的角度；再接着利用角速度与线速度之间的正比关系得出我们需要的风速信息。通过获取的的电压变换次数，以每次变化的次数代表风叶转动30°可以计算出该变化次数内风叶转动的角度：

$$转动角度row = N \times 30 \quad (2\text{-}1)$$

通过转动角度与记录该段次数的时间，本次设计采用的是记录每一秒内的电平变化次数，所以得到的就是每一秒风叶的转动角度row。通过风叶的转动半径约为4cm，则通过线速度与角速度的关系可以计算出风速的指数。

$$风速speed = row \times 0.4 \times 10^{-2} \quad (2\text{-}2)$$

通过以上方案将测量的风速的理论值，由于编码盘的最小精度是30°，所以测量的风速指数与实际风速理论上会有1.2m/s的误差。该误差一般的日常生活中可以允许，则实现风速指数的测量方案。

2.2温湿度数据的获取

温湿度传感器是使用DHT11传感器直接将物理量转化为数字量，不需要经过其他转换。但是该传感器需要满足特定的通信协议才能获取到正确的测量指数，该传感器有四个引脚，其中只用到三个音引脚：VCC、GND以及DATA三个引脚。所获取的温度数据以及湿度数据都是通过一条数据线来传递的，属于半双工的方式来传递信息。除此之外。还携带有校验数据，用于检验测量数据的有效性。并且传递的数据固定为40比特，其中8位温度整数值、8位湿度整数值、8位温度小数值、8位湿度小数值以及8位校验位。树莓派通过特殊的信号格式传递给传感器，传感器接收到后将开始传数据。

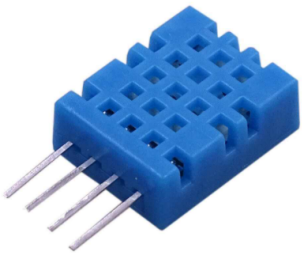

图 2-5 DHT11 温湿度传感器

2.2.1 数据转换原理

温度与湿度的数据是通过DHT11传感器直接转化，转化过程被封装在传感器内部。对外部不可见，该类传感器是将温度变化为标准阻值，湿度转化为标准电压，这些信号都是需要经过ADC采样转化为数字信号。该数字信号是固定为40比特的数据，其中包含8位温度整数值、8位温度小数值、8位湿度小数值、8位湿度小数值以及8位校验值，并且数据的输出端使用了5K的上拉电阻。

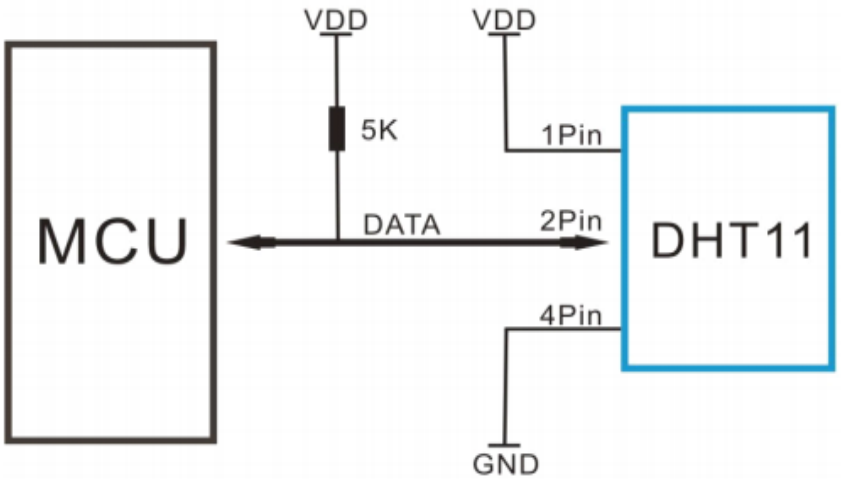

图 2-6 DHT11的应用电路

在该传感器上电之后，需要等待1秒，保证电路处于稳定状态，当然最好在VDD与GND再增加一个100nF的电容起到去耦滤波的作用，该传感器的温度测量范围再0~50℃，误差是2℃；湿度的测量范围是20~90%，误差是5%。

2.2.2 数据通信协议

在DHT11中，是通过串口将数据发送给树莓派的。在这一过程中，传感器与树莓派需要满足DHT11的通信协议，才能正确采集数据。采用的是单总线的数据格式，每次采集间隔在4ms左右，数据是固定的40位，其具体的数据格式是：8位湿度整数值+8位湿度小数值+8位温度整数值+8位温度小数值+8位数据校验值，当树莓派发送开始采集命令后DHT11从低功耗模式转换到高速模式，在接受完树莓派的开始信号后，DHT11将开始传递已采集的40比特数据。发送完成后，如果没有接受到再次采集的信号，则进入低功耗模式。整个数据通信的时序如下图：

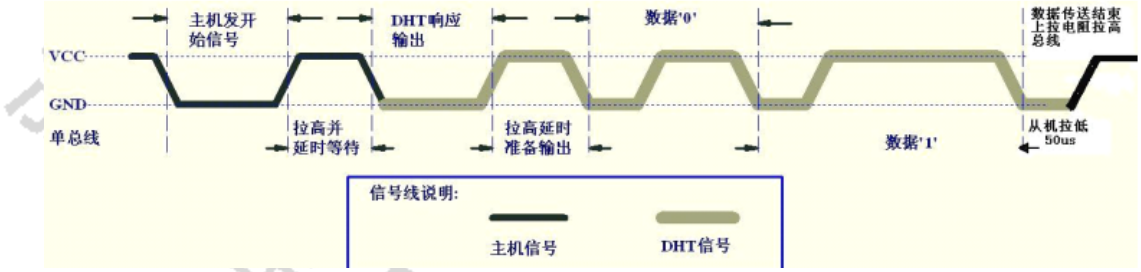

图 2-7 DHT11 数据采集的时序图

首先总线在高电平时处于空闲状态，当需要采集信号时，树莓派将信号拉低至少18ms，保证DHT11收到采集信号，之后树莓派将总线拉高结束采集指令（延时20~40us），并将GPIO口设置是为输入，这时DHT11会发送一个80us的响应信号。紧随其后DHT11将总线拉高，准备开始传递信号，每一比特的传递都是以50us的低电平为间隔，高低电平的数据是以高电平的保持时间来反应的。

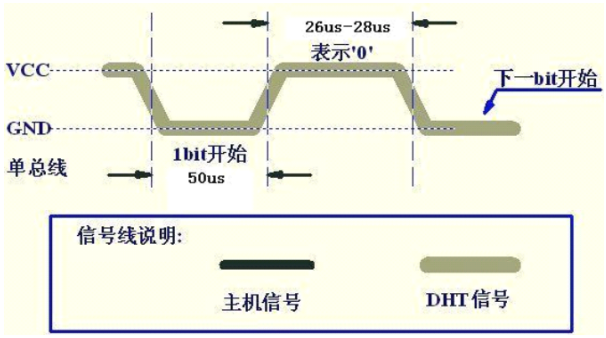

图 2-8 数据0的表示方法

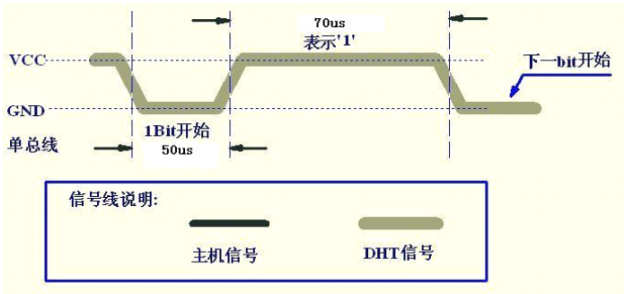

图 2-9 数据1的表示方法

通过以上协议树莓派将获取到湿度与温度的指数的获取，该获取到的数据需要经过转换与检验。

2.2.3 数据处理算法

在使用DHT11传感器获取的40比特固定的数据中，其中温度数据有8位整数，8位小数，湿度数据有8位整数，8位小数以及8位校验位，在正确获取到传感器的数据之后，需要通过以下算法的转换与判断才能获取到有效的数据。首先数据的格式是8位湿度整数值、8位湿度小数值、8位温度整数值、8位温度小数值以及8位数据校验值。

```
    if (self._last_called == 0 or (time.monotonic() - self._last_called) > delay_between_readings):
self._last_called = time.monotonic()
new_temperature = 0
new_humidity = 0
if self._use_pulseio:
pulses = self._get_pulses_pulseio()
else:
pulses = self._get_pulses_bitbang()
if len(pulses) < 10:
# Probably a connection issue!
     raise RuntimeError("DHT sensor not found, check wiring")
if len(pulses) < 80:
# We got *some* data just not 81 bits
     raise RuntimeError("A full buffer was not returned. Try again."
buf = array.array("B")
for byte_start in range(0, 80, 16):
     buf.append(self._pulses_to_binary(pulses, byte_start, byte_start + 16)
if self._dht11:
# humidity is 1 byte
new_humidity = buf[0]
# temperature is 1 byte
new_temperature = buf[2]
else:
# humidity is 2 bytes
     new_humidity = ((buf[0] << 8) | buf[1]) / 10
# temperature is 2 bytes
# MSB is sign, bits 0-14 are magnitude)
     new_temperature = (((buf[2] & 0x7F) << 8) | buf[3]) / 10
# set sign
if buf[2] & 0x80:
new_temperature = -new_temperature
# calc checksum
chk_sum = 0
for b in buf[0:4]:
chk_sum += b
# checksum is the last byte
if chk_sum & 0xFF != buf[4]:
# check sum failed to validate
     raise RuntimeError("Checksum did not validate. Try again.")
if new_humidity < 0 or new_humidity > 100:
# We received unplausible data
     raise RuntimeError("Received unplausible data. Try again.")
self._temperature = new_temperature
self._humidity = new_humidity
```

通过以算法将获取到湿度与温度的具体指数，这一指数，与实际的温度、湿度指数的误差分别在2℃与5%之间，在日常生活中，该误差不影响对天气数据的认知。通过以上方案将实现温度与湿度的数据获取。

2.3 风向数据的获取

风向的数据据获取没有适合的用于直接转换的传感器，需要借助对风向标角度的判断来测量风向的数据。首先通过查阅资料可知，一般生活中风向的取值有16组值，分别是E、N、S、W、NE、SE、SW、NE、以及NNE、ENE、SSE、ESE、SSW、WSW、WNW、NNW，这些角度均是相差22.5°该数据的测量可以借助无极电位器来测量。无极电位器是在一圆周中具有不同的阻值的电位器，并且该阻值可以与转动的角度一一对应，通过该阻值的测量，就可以获得该风向的角度从而判断出此时的风向。根据采集的角度为整数值，可以将0~360的圆周划分为16等分，如果测量的角度满该区域，则可以近似该为风向值。具体来说，0作为N的起点，则可以通过，11.25°作为另一个区间的开始，这样将360划分为32个区域，两相邻的两个区域的中心代表该两个中心的风向。

2.3.1 数据转换原理

风向数据是通过无极电位器实现的数据采集的，利用该传感器实现将风向转化为电位器阻值的变化，并通过模拟电压信号获取该数据，通过ADC模块实现电压到数字的转换，并通过串口将数据传输给树莓派，利用采集的电压数值与无极电位器的线性相关，将电位器的角度计算出，并通过角度判断出风向信息。由于该测量值需要借助ADC模块，而树莓派中没有该模块，所以在此处借助Arduino模块将ADC采集的数据通过串口发送给树莓派。该数据是采用10k的无极电位器，通过中间引脚将电位器此时的电压值输出到Arduino的ADC模块的入口，然后通过ADC模块的转化，将模拟电压值转化为具体的数值。在Arduino的函数库中已封装好函数，仅需要通过调用即可完成数据的转换，该电压值在0~5V之间线性的变化，此时可以与0~360之间进行一一对应，从而得到角度的数值，在通过决策得到风向的数据。

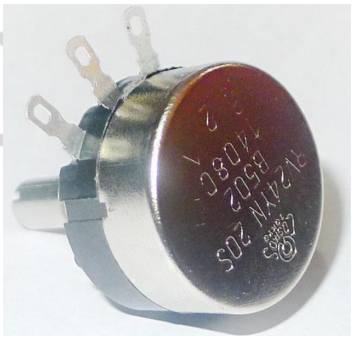

图 2-10无极电位器

2.3.2 数据通信协议

在无极电位器到树莓派之间的数据传输需要借助一个ADC模块进行转化，在本次方案中，采用Arduino来代替该模块实现模拟型号的传递过程。首先由Arduino的ADC模块采集到模拟电压信号，之后通过串口将数据传递到树莓派，在本次方案中指定串口的波特率为115200，通过串口不断传递采集电压信息。在树莓派端通过Python的第三方模块Serial将数据不断采集，从而实现数据从传感器到树莓派的传递。在这里需要注意波特率与每一次传递的间隔。在传递完成后还需要继续数据的处理，采集到的仅是电压信号，需要将其转化为角度信号。这一步是在树莓派中完成的。

图 2-11 Arduino模块

2.3.3 数据处理算法

通过以上数据的采集与传递，树莓派获取到与风向数据一一对应的电压信号，然后经过对应公式的转化将电压信号转化为角度信息：

$$角度\mathrm{row} = voltage \times \frac{5}{360} \quad (2\text{-}3)$$

转化为角度信息之后，完后决策函数，实现角度到风向值的判断：

```
def angle_to_direction(angle):
if( angle >=349 or angle <= 11 ):
direction = "N"
elif( angle >= 12 and angle <= 33 ):
direction = "NNE"
elif( angle >= 34 and angle <= 56 ):
direction = "NE"
elif( angle >= 57 and angle <= 78 ):
direction = "ENE"
elif( angle >= 79 and angle <= 101 ):
direction = "E"
elif( angle >= 102 and angle <= 123 ):
direction = "ESE"
elif( angle >= 124 and angle <= 146 ):
direction = "SE"
elif( angle >= 147 and angle <= 168 ):
direction = "SSE"
elif( angle >= 169 and angle <= 191 ):
direction = "S"
elif( angle >= 192 and angle <= 213 ):
direction = "SSW"
elif( angle >= 214 and angle <= 236 ):
direction = "SW"
elif( angle >= 237 and angle <= 258 ):
direction = "WSW"
elif( angle >= 259 and angle <= 281 ):
direction = "W"
elif( angle >= 282 and angle <= 303 ):
direction = "WNW"
elif( angle >= 304 and angle <= 326 ):
direction = "NW"
elif( angle >= 327 and angle <= 348 ):
direction = "NNW"
else:
direction = "UN"
return direction
```

通过以上方案实现风向的测量，在该方案中由于测量的角度取整，所以最小角度为1°，所以该决策函数是采用近似决策的，每一个风向值有小于1°的误差，在日常生活中，该误差可以忽略。

通过以上三个方案将实现天气数据的风速、风向、温度以及湿度的获取。也完成信息采集的基本任务，该功能

的具体实现过程，在附录源码station.py中可以找到（采用模块化的设计方法实现）。数据的采集是整个设计过程的第一步，接下来将实现数据的传输与存储的实现方案。该方案是实现整个设计的第二阶段。

第3章 天气数据的存储

本章将描述天气数据从本地树莓派端到云端的传递过程。在这一功能中需要涉及到数据如何从树莓派传递到云端，数据在云端的存储方式以及如何存储到云端数据库中。首先按照数据的先后顺序，需要先了解数据如何从树莓派传递到云端，其次了解数据在数据中的存储结构，最后实现如何存储到数据库中。在此次设计方案中，结合之后的页面数据可视化的设计方案，采用统一的后端API接口实现数据的存储，通过Django后端框架实现数据模型的涉及以及管理数据库的API。然后对外提供一个数据存储与数据获取的接口。Django是通用的web框架，采用Python语言开发，与MVT模型，即Model、View、Template的设计模型。采用Model封装数据的处理，视图实现API的功能，模板实现样式的统一。在本次实现方案装，仅使用Django实现后端框架，实现使用HTTP的POST方法将数据传递给云端存储。

在数据的传输过程中，数据是以表单的形式进行提交的。需要携带的信息了气象站的身份信息，温度、湿度、风速以及风向的信息。这些信息在push接口中会通过request.POST.get(key)的方法获取到，然后查询该气象站是否在数据库中注册，如果已被注册，则通过获取该气象站对象的接口信息，通过station.weather_set.create()方法将数据存储在该气象站对象中，并通过station.save()实现数据的持久化存储。

3.1 数据结构的设计

数据库是管理数据的系统，在主流的数据库中，MySQL是一种关系型数据库，可以满足本次开发的设计要求。在存储数据之间，我们首先需要设计适当的数据结构来方便我们存储数据。数据是以气象站为核心的，每一个气象站必须拥有不同的身份来标识，并通过该身份识别是哪一个气象站传递的数据，所以针对每一个注册的气象站，需要存储对应的天气信息，每一个天气数据拥有温度、湿度、风速、风向等数据，并且每一个天气记录必须对应一个气象站身份，来表明是哪一个气象传递的数据，并记录采集的时间，所以每一条记录都是有气象站信息与采集时间唯一的确定，所以在MYSQL中将存在两张表，一张存储气象站的数据、另一张存储天气数据信息、并通过气象站的身份实现两张表的对应。这一设计方案将采用Django的模型来实现，即采用类定义实现气象站信息与天气数据的管理。

```
class Station(models.Model):
    station_id = models.CharField('sid', primary_key=True, max_length=10)
    station_name = models.CharField("name", max_length=40)
def __str__(self):
    string = "%s,%s"%(self.station_id,self.station_name)
return string
class Weather(models.Model):
    station = models.ForeignKey(Station, on_delete=models.CASCADE)
date = models.DateTimeField("datetime")
speed = models.FloatField('speed')
    direction = models.CharField('direction',max_length=10)
temperature = models.FloatField('temperature')
humidity = models.FloatField('humidity')
def __str__(self):
    string = "<{}>[{}] {}/{} {}/{}".format(
self.station,
self.date,
self.speed,
self.direction,
self.temperature,
self.humidity
)
return string
```

通过以上模型的设计将借助Django实现数据模型的设计,之后可以通过统一的API接口来管理数据库,此次数据库

采用的是MySQL关系型数据。在Django框架中需要指定数据库的接口。

3.2 网络中数据传输

在设计完存储的数据结构之后，我们接下来确定数据在网络中的传输方案，在网络中我们可以使用Socket来实现云端与树莓派的数据的通信，但是由于该方式对于本次设计较于复杂，所以本次设计我们采用更为方便的HTTP来实现数据的传输，HTTP可以通过GET与POST两种方式来传递数据，GET方式是将数据直接添加在URL中传递，而POST方式是将数据放在请求头中发送给云端。而且这可以和之后的数据可视化复用同一套数据库操作接口。该方案采用Django提供的View来实现，通过HTTP的POST实现。这里通过以下视图来实现：

```
def push(request):
if request.method == 'POST':
sid = request.POST.get("station_id")
new_date = request.POST.get("date")
new_speed = request.POST.get("speed")
new_direction = request.POST.get("direction")
new_temperature = request.POST.get("temperature")
new_humidity = request.POST.get("humidity")
type_str_date = type(new_date).__name__
if( type_str_date == "str" ):
new_date = float(new_date)
    new_date = datetime.datetime.utcfromtimestamp(new_date)
ret = len(Station.objects.filter(station_id=sid))
if(ret == 0 ):
    message = "This station is not in database, please regeister it."
return JsonResponse(message,safe=False)
station = Station.objects.get(station_id=sid)
station.weather_set.create(
date =new_date,
speed = new_speed,
direction = new_direction,
temperature = new_temperature,
humidity = new_humidity,
)
    message = "This weather has benn recorded to database."
return JsonResponse(message, safe=False)
```

在实现该接口之后，可以通过运行服务，通过树莓派端的程序将采集的信息传递到云端数据库中。，通过Django提供的及可视化页面可以查看到数据库中的数据。

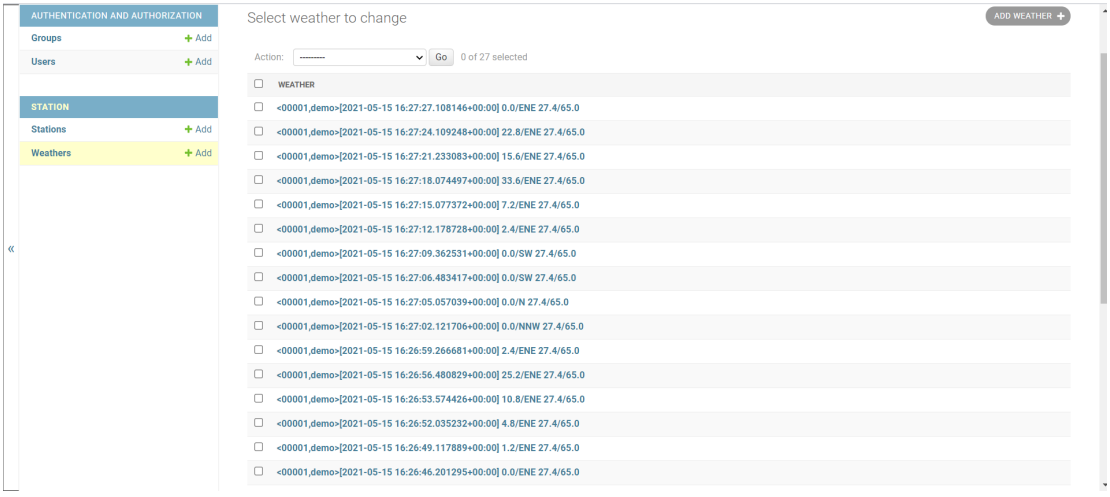

图 3-1 云端存储的数据

在完成数据的传输之后，将开始讲述数据的存储过程。将数据在数据库中持久化存储。

3.3 数据云端的存储

在云端接受到有效的数据结构之后，需要通过调用设计好的数据库接口将数据存储在数据库中，本次设计方案使用的是MySQL数据库。通过Python第三方提供的数据库接口，通过定义合适的数据模型，可以快速实现数据库的操作。

首先通过指定数据库的接口，然后设计好模型Station与Weather后。可以通过：

Weather.objects.filter(station_id = value)来查询该气象站是否在云端注册，气象站检测通过后，可以采用Weather.objects.get(station_id = value)获取气象站对象，之后通过获取的气象站对象station调用station.weather_set.create(weather_data_list)来创建一条天气数据记录；然后通过station.save()将数据存储到数据中去。这一过程的实现，直接在前面的数据传输中实现，无须独立完成。通过以上方案将实现数据从树莓派端到云端传输及存储的过程。到这一步，以实现天气数据的采集、数据的上传、数据的存储，接下来将实现数据的可视化。

第4章天气数据的可视化

数据的可视化在[14]中进行了详细的讲解，在本次方案中采用React+Echarts来实现。在Echarts项目中提供适用于React的模块来实现数据的可视化，而采用React前端框架实现云端数据的请求并为Echarts提供实时数据。首先在Django中提供一个获取天气数据的接口pull(),该接口采用的是HTTP的GET方法，返回一个json格式的数据，其中包含默认第一个气象站的所有天气数据。然后通过React中采用axios访问后端提供的接口获取数据，之后将该数据传递给Echarts做数据显示，通过React的组件的componentDidMount方法实现数据的动态刷新。

图4-1 数据可视化的初步显示方案

4.1 可视化数据的接口实现

通过Echarts项目为reacts提供的模块，可以快速搭建数据可视化页面，通过Django提供的pull()接口，可以为可视化页面提供显示的数据。

```
def pull(request):
stations = Station.objects.all()
station = stations[0]
_weathers_ = station.weather_set.all()
data_size=len(_weathers_)
weathers = [{
'station_id':weather.station.station_id,
'station_name':weather.station.station_name,
'date': weather.date.timestamp(),
```

```
'speed': weather.speed,
'direction': weather.direction,
'temperature': weather.temperature,
'humidity':weather.humidity,
} for weather in _weathers_
]
return JsonResponse(weathers, safe=False)
```

通过以上的接口提供React可以获取数据库中的天气信息，React是提供前端的页面框架，Echarts在React中提供一个组件模块，可以快速实现数据可视化的开发。具体实现过程是，定义一个类组件，在componentDidMount()方法中通过axios中的get方法获取后端提供的pull数据接口，将天气信息获取到本地，然后通过将该数据传递到可视化组件中，在该组件中通过props参数获取到所有数据，由于该数据是以每一条记录为单位的json数组，需要借助数据的map方法，将数据进行提取，将当前的数据显示到当前的天气信息模块，然后通过组件对象的setOption()方法来将数据填充到可视化模型中。在这里需要可以指定分别指定x坐标的值与y坐标的值。该值是以数组的形式关联。

```
componentDidMount() {
const _this_ = this;
axios({
method: 'GET',
url: 'http://192.168.1.6:8000/station/api/pull',
}).then(function (response) {
_this_.setState({
weathers: response.data,
status: true,
})
}).catch(function (error) {
_this_.setState({status: false,})
})
}
```

以上为实现前端通过后端接口获取数据的实现方案。在这一步如果前端页面与后端使用不同的域，可能会出现跨域无法访问的问题，这一不需要在后端服务开启跨域访问，在服务的全局设置中增加以下配置。

```
INSTALLED_APPS = [
'corsheaders',
]
MIDDLEWARE = [
'corsheaders.middleware.CorsMiddleware',
]
CORS_ORIGIN_ALLOW_ALL = True
CORS_ALLOW_CREDENTIALS = True
```

这样就开启了后端服务的跨域访问的功能，在返回的数据是一个json格式的数组，我们需要的每一个天气信息的数组，这需要通过数组的map方法，将数据解析成一个个的指数指数数组，方便后可视化的数据绑定。

4.2数据可视化与动态刷新

该数据的可视化会获取数据库中最新的数据，然后通过对比如果数据发生变化，则实时更新到可视化页面中，实现数据的及时刷新。通过该步骤的设计方案。DIY气象站的初步设计方案将完成，完成数据的采集、传输、存储以及可视化的过程。在显示的组件Weatherchart中通过以下过程来实现数据分提取：

```
const date = [];
const speed = [];
const direction = [];
const temperature = [];
```

```
const humidity = [];
console.log(this.props.weathers)
this.props.weathers.map((weather) => {
     date.push(moment(weather.date).format('YYYY-MM-DD HH:mm:ss'));
speed.push(weather.speed);
direction.push(weather.direction);
temperature.push(weather.temperature);
humidity.push(weather.humidity);
});
```

通过以上实现数据的提取之后，将通过setOption中为可视化模型下指定显示的数据。

```
chart_1.setOption({
graid: [{
width: '100%',
height: '100%',
}],
xAxis: [{
name: "时间",
type: 'category',
data: date,
}],
yAxis: [ {
name: "风速（m/s）",
type: 'value',
} ],
legend:{
orient: 'vertical',
left: "center",
data:['风速'],
},
series: [{
name: "风速",
type: 'line',
data: speed,
}],
tooltip: {
trigger: 'item',
formatter: function(params, ticket, callback) {
// var color = params.color;
     var string = '<div>'+params.name+'<br/><span></span>'+'风速:'+params.value+'<br/>风
向:'+direction[params.dataIndex]
return string
}}
```

之后的处理与上面的是实现方案类似最终呈现出图4-1的的显示效果。完成数据可视化的设计方案。

第5章总结与展望

写道这里，本文也要结束了。通过这次的课题研究，我基本完成了课题研究的目标，但是，我知道我本次设计还有许多的不足需要改进。不过我也在这次的研究中学习了很多新的知识点，例如利用Python开发网站服务，实现后端的接口设计，实现前端的页面数据可视化，这也是我第一实现此类需求，难免会遇到许多的问题，例如数据访

问是的跨域问题等，当然，学习的路上总免不了遇到各类问题，在解决问题之后的喜悦也是让我不断前行的动力。接下来，开始我本次课题设计的总结与展望。

5.1总结

通过本学期的学习与研究，已基本完成毕业设计的任务。完成从天气数据的采集，到数据从树莓派端到远端的传递，以及云端数据的是持久化存储，最后完成数据的获取以及可视化的过程。虽然在这个过程中还有许多过程需要完善。但是也基本完成DIY气象站的初步目标。通过本次的课题设计的研究，我首先了解了气象站的一般工作原理，其次，我熟悉了课题研究的一般方法。同时也学习了Python在生活的应用以及一个基本需求的的方法。在本次的设计过程中，需要设计到传感器的原理及应用、树莓派的原理及应用，云端平台环境的搭建，以及数据在网络的中的传递等只是。虽然，或许我在专业知识学习的旅途中还有许多路要走，但是通过这一次的学习，我了解我的知识体系的不足。这为我未来的学习方向提供了导向。同时，也让我学习到了Python在网页服务后端与前端的基本设计方法以及Echarts、React以及Django的使用方法。

5.2展望

气象站是我们人类活动了解天气信息必不可少的手段。通过这次的课题研究，我完成了DIY气象站的初步研究。但是在研究的过程中，我也发现了许多需要改进的不足，例如，天气信息中需要增加日常生活所需的紫外线强度的信息采集以及PM2.5的测量。不过这些数据可以在初步的设计方案中进行扩展。当然还需要改进的部分是实现传感器与树莓派数据交互的优化，比如采用蓝牙传输可以实现近距离多点覆盖，从而使采集的数据更加准确、合理，同时，在未来可以实现更多传感器的快速扩展。在树莓派与云端数据的传递过程中，接口太过简单，结果可能是出现安全性问题，后续可以采用https传递数据，或者采用websocket来实现。当然在考虑到后续维护与扩展的问题，需要将模块之间的耦合性降低，实现 气象站与云端，甚至数据库之前的分离，通过统一的接口实现该解决方案，最后直接展示采集的数据有些过于简单，后续最好通过一些简单的算法实现更加具体直观的数据显示，例如通过风速、温度、湿度、紫外线强度等数据实现穿衣指数以及出门必带物品的提醒。从而更加人性化和智能化。当然，后续的问题不只这些，这需要在后续的开发过程中，不断的探索与实验。寻找最佳的解决方案。希望我在未来通过对该方案的升级，让该方案真正的服务与人们的生活。从而实现自我价值的体现。