基于 Arduino 的温度系列实验

摘　　要：基于 Arduino 平台进行了一系列温度相关的实验，并最终实现了一个可交互的温度控制装置。在搭建装置实验的同时进行了并行处理、蓝牙模块调试、PID 算法与参数整定等相关内容的研究。

一、 引言

Arduino 是一种开源的单片机开发平台，可以利用各种各样的传感器来感知环境；控制各种各样的灯光、马达、电机来进行反馈；用自带的编程语言来编写程序，实现复杂的功能。只要程序编写的好，Arduino 可以实现各种用电脑才能实现的功能，在物理实验教学中可以用于自动测量，自动控制等各个领域，在日常生活中也有广泛的应用。本实验是基于 Arduino 平台进行的自动控制系统及其相关的系列实验。

二、 实验装置及内容

实验装置：Arduino Uno 开发板，24V60W 电源，JDY-16 蓝牙芯片，VS1838B 红外接收器，杜邦线、漆包线若干，电热丝，DS18B20 温度传感器，DHT11 温度湿度传感器，IRF540 场效应晶体管继电器，LCD1602 液晶屏，无源蜂鸣器。

实验内容：温度计实验、遥控实验、温控实验。

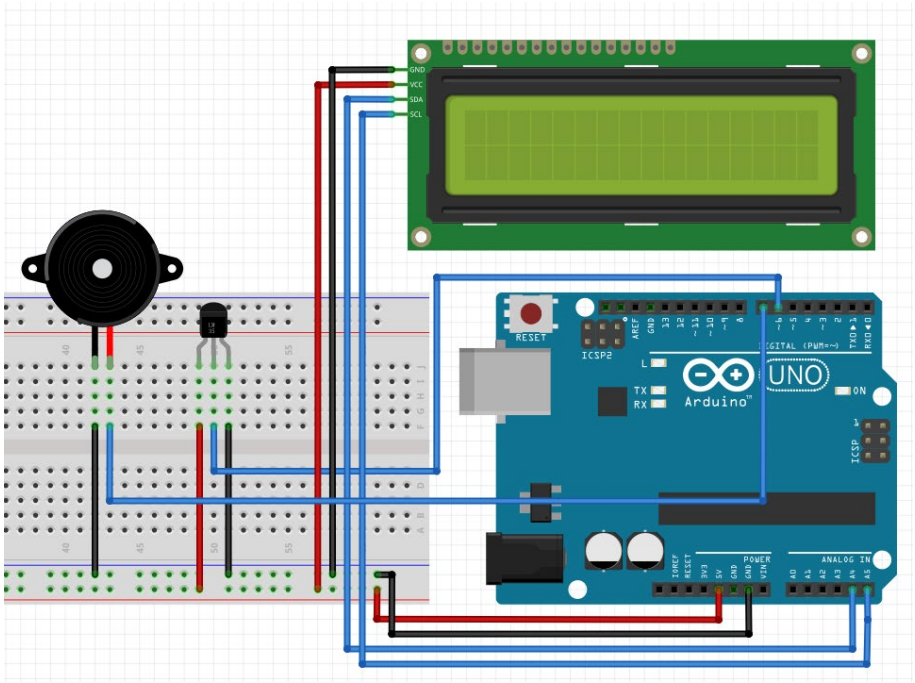

图 1 温度计实验原理图

温度计实验：组装一台具有显示功能的温湿度计，并加入警报功能。

通过 DHT11 温湿度传感器进行测量；液晶显示器通过 I2C 总线与 Arduino 开发板连接。

警报功能通过向无源蜂鸣器周期性的输入方波信号实现，程序中令方波周期与超出的温度成反比，则可模拟出“温度越高，警报频率越高”的效果。

遥控实验：使用 JDY-16 蓝牙模块，将 Arduino 与手机进行互联，并在手机端设计 UI 界面。利用 VS1838B 红外模块实现 Arduino 的红外遥控。

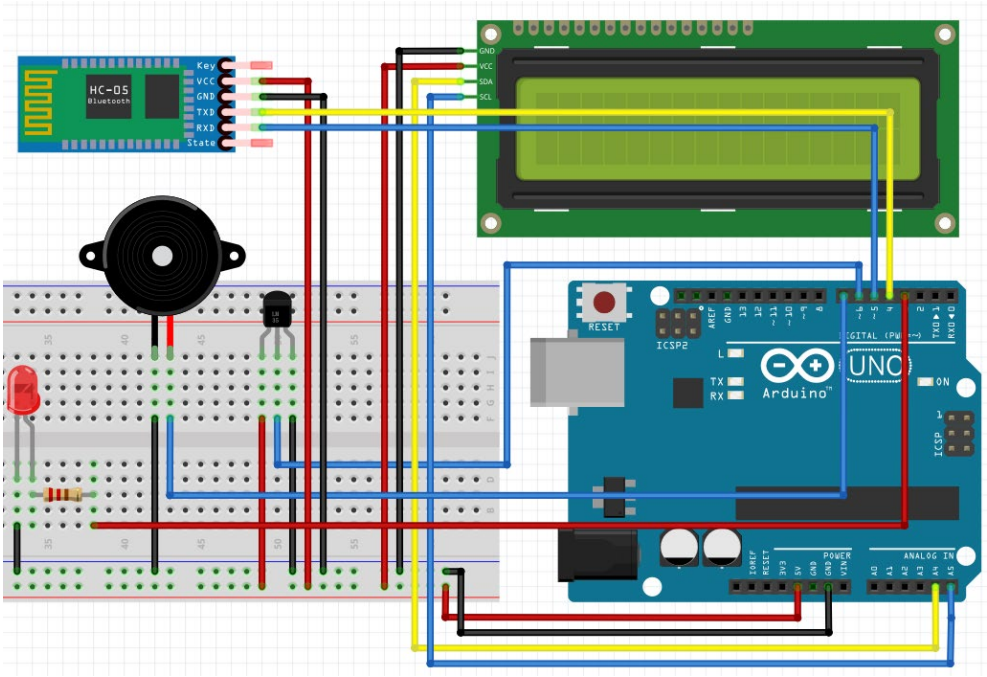

图 2 遥控实验原理图

温控实验：使用 MOS 管驱动加热电路，通过输出的 PWM 信号来调制加热功率，用 PID 算法进行温度控制。

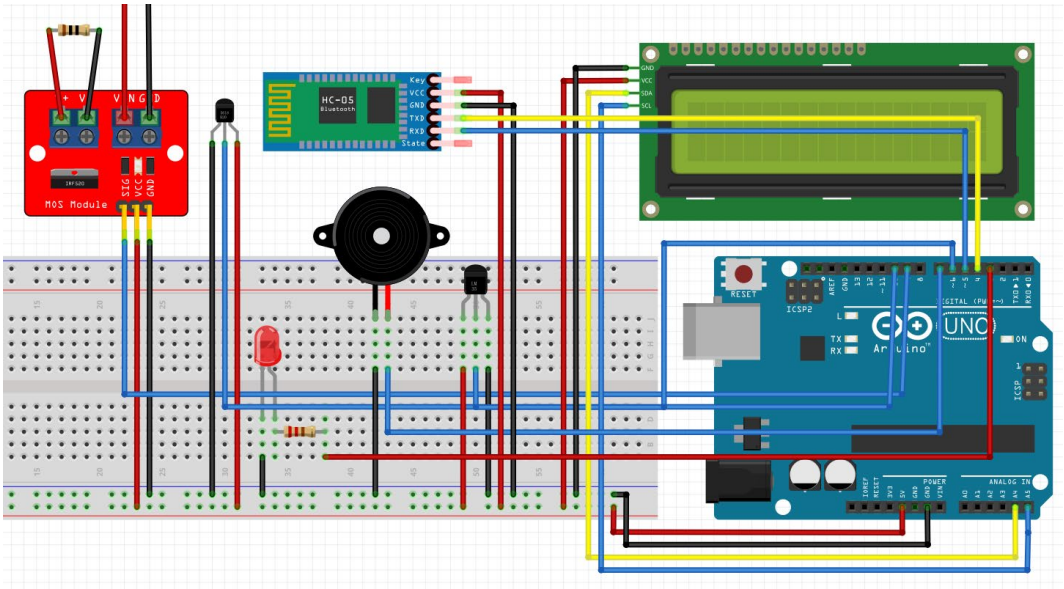

图 3 温控实验原理图

PID 算法简介：

PID 全称为比例积分微分控制，是一种自动控制算法，规律如下：

$$u(t) = K_p\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(\tau)}{d\tau}|_{\tau=t}\right) = K_p e(t) + K_i\int_0^t e(\tau)d\tau + K_d\frac{de(\tau)}{d\tau}|_{\tau=t}$$

其中T_i是积分时间，T_d是微分时间，K_p, K_i, K_d是 PID 的三个参数。

比例环节的作用是对输入进行调节，以温控为例，若温度达不到预计温度则需要加热，温度越接近加热功率越小，高于则需要降温，或至少不加热。

积分环节的作用是消除静差，以温控为例，由于系统本身会向环境中散热，故只用比例作用温度会稳定在目标温度以下，加热与散热平衡。积分环节可以向加热器累计这一偏差，使温度稳定在目标温度。

微分环节的作用是减小振荡，以温控为例，若加热功率过大，温度会迅速升高，通过微分作用可以提前将加热功率降低，来减小振荡。

三、 实验过程记录

1. 温度计实验

该实验电路部分较为简单，但是在程序设计上有一个难点：我将报警系统的警报声编写的比较复杂，一段警报声长好几秒；但是这样一来在警报的长时间内都无法测温。因此，程序在编写时需要多线程处理。

Arduino 本身是单核处理器，无法实现真正的多线程，多线程的原理是利用开发板上自带的 Timer，将 CPU 时间分割成小段，再将每一小段交给不同的任务执行。相比于单线程而言可以更好的控制线程的开关，更好的利用时间。这里我调用了 Scoop 库进行多线程的编写，代码如下：

```
void Warning::setup() {
  pinMode(Buzzer, OUTPUT);
}

void Warning::loop() {
```

图 4 开启多线程

```
if (Temp < upp) Warning.pause();
else Warning.resume();
```

图 5 控制线程开关

2. 遥控实验

1）蓝牙模块

蓝牙模块可以使用 Blinker 库实现与手机的互联，这个库是一个面向对象的库，手机 APP 上每个组件对应代码编写中的一个对象。当操作 APP 中的组件时，蓝牙模块将这一操作用回调函数的形式反馈给 Arduino，同时回调函数会将一些参数用蓝牙传输回手机，这样就实现了手机与 Arduino 的通讯。另外，温度湿度这两个参数不需要调用组件也需要实时向手机反馈，因此需要用到心跳函数，这是一种特殊的回调函数，每隔一段时间两个蓝牙设备要确认通讯是否仍在保持建立，此时可以“借机”将温湿度的数据与手机同步。

图 6 手机 APP 截图

```
BlinkerButton Button_unit("UNIT");
BlinkerNumber Number_temp("temp");
BlinkerNumber Number_humi("humi");
BlinkerSlider Warn_upper("upp");
```

图 7 建立对象

```
void unit_callback(const String & state) {
  BLINKER_LOG("get temp unit: ", state);
  if (state == "unit") {
    if (unit_C == 0)
      unit_C = 1;
    else unit_C = 0;
  }
  Button_unit.print(state);
}

void get_upper(int32_t value) {
  BLINKER_LOG("get upper temp: ", value);
  upp = value;
}
```

图 8 注册回调函数

```
void heartbeat() {
  Number_temp.print(Temp);
  Number_humi.print(Humi);
}
```

图 9 注册心跳函数

2）红外模块

本红外模块使用 NEC 协议，该协议用载波与空闲的时长来负载 0-1 值：560μs载波与560μs空闲代表 0，560μs载波与1680μs空闲代表 1。同时还有较长的引导码与结束码。

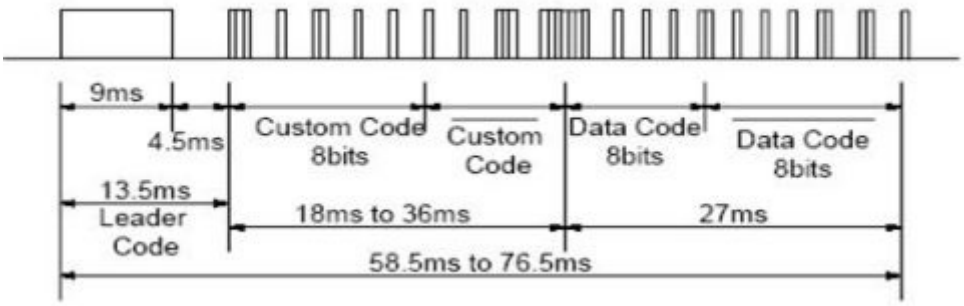

图 10 NEC 协议时序图

```
unsigned int irSignal[] = {9000, 4500, 560, 560, 560, 560, 560, 560, 560, 560, 560, 560, 560, 560, 560, 560, 560, 560,
                           560, 1690, 560, 1680, 560, 1680, 560, 1680, 560, 1680, 560, 1680, 560, 1680, 560, 1680,
                           560, 560, 560, 1680, 560, 560, 560, 1680, 560, 1680, 560, 560, 560, 560, 560, 560,
                           560, 1680, 560, 560, 560, 1680, 560, 560, 560, 560, 560, 1680, 560, 1680, 560, 1680,
                           560, 39416, 9000, 2210, 560};

irsend.sendRaw(irSignal, sizeof(irSignal) / sizeof(irSignal[0]), khz);
```

图 11 发送 NEC 信号

不幸的是，实现以上两个模块的功能用去了 Arduino 82%的存储空间与 85%的动态内存，因此后面的温控实验部分的程序没有与前两部分合并到一起，有些可惜。

3. 温控实验

1）电路搭建与故障排查

第一次电路搭建时，预期加热的对象是一杯水，而水的比热较大，因此我尽可能的将输出电路的功率调大：加热电路选择使用直径 0.6mm 的 Cr20Ni80 合金丝做加热丝，电阻值用分压法测得为$10.6 \pm 0.2\,\Omega$。此时，按照纯电阻电路的功率计算，功率小于额定功率 60W，理应正常工作，但是实验中发现电热丝并没有升温，电路没有接通，并且电源发出蜂鸣声，因此推测此时电路中电流过大，导致电源触发了保护机制，从而停止了输出。

但是改用更大的电阻会导致输出功率过小，因此在第二次电路搭建时我选用 10 个

220Ω电阻并联作为加热丝，并将温度传感器探头直接绑在电阻上，并用导热片来尽量保证温度一致。这次的搭建取得了成功，可以正常进行实验。

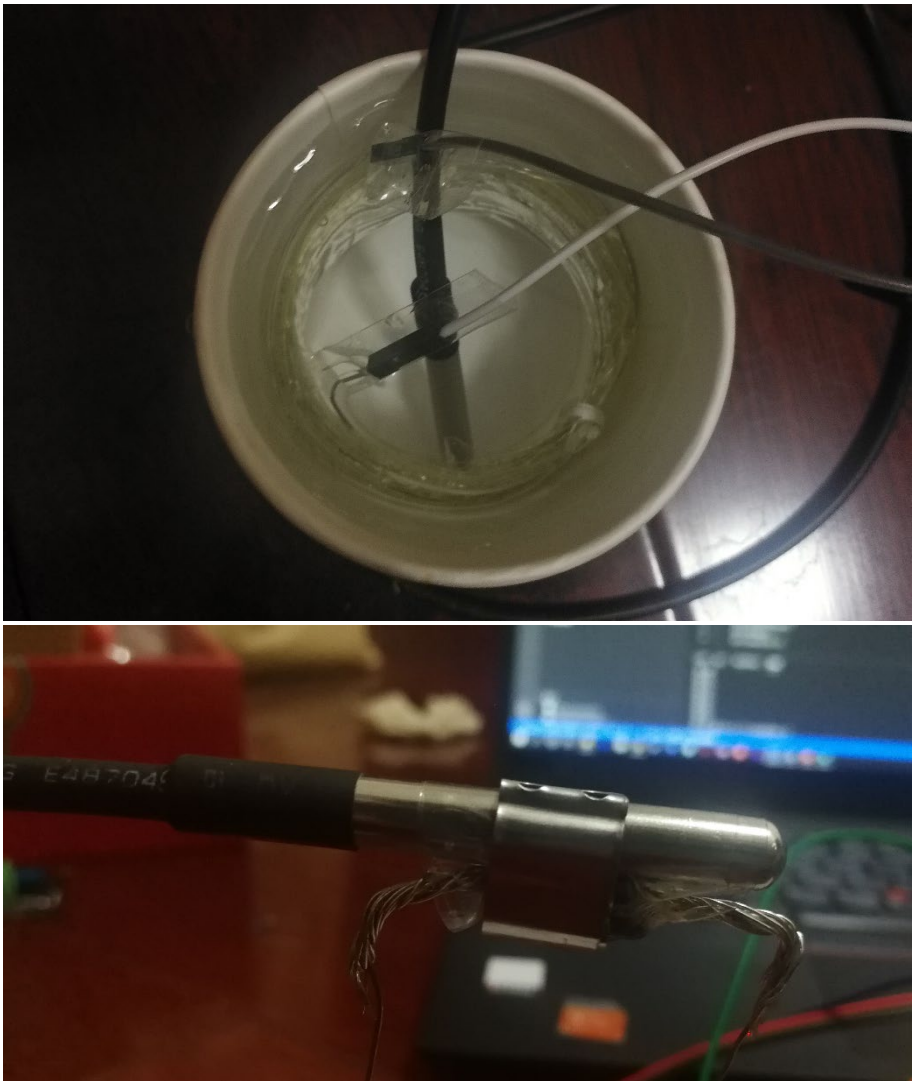

图 12 传感器连接（上图为第一次的方案，下图为第二次的方案）

2）PID 整定

现在，PID 整定已经具有了许多自动化实现的方案，但是我的加热电路与一些常见的场景不同，只能加热而不能降温，降温要依赖环境降温。因此依赖于加正负双向输出的一些自整定方案，比如临界比例法等，都表现不佳，因此我最终选择手动调节参数。

手动调节参数的诀窍在于逐个调节，即先调节 P，再调节 PI，最终是 PID。实验中选取的目标温度为45℃，参数调试记录如下：

调节K_p：

通过比较不同的K_p值，可以发现，K_p较大使得升温有一个很高的过冲峰，而K_p较小

会使得升温变得缓慢，因此拟定K_p选取较为适中的 3。

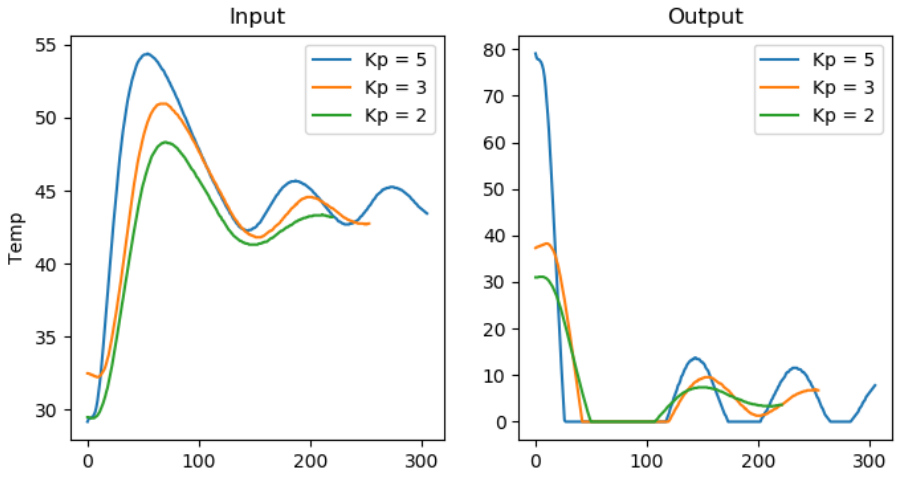

图 13 调节K_p

调节K_i：

通过比较不同的K_i值，可以发现，K_i的引入会导致首次加热时加热功率更大，温度过冲，因此需要恰当的降低一些K_p的值。同时，K_i较大时，温度持续波动，没有衰弱的趋势，这是因为在温度低于目标温度时积分项会持续增加功率，导致温度升的过高。最终选取的参数值为$K_p = 2, K_i = 0.02$。

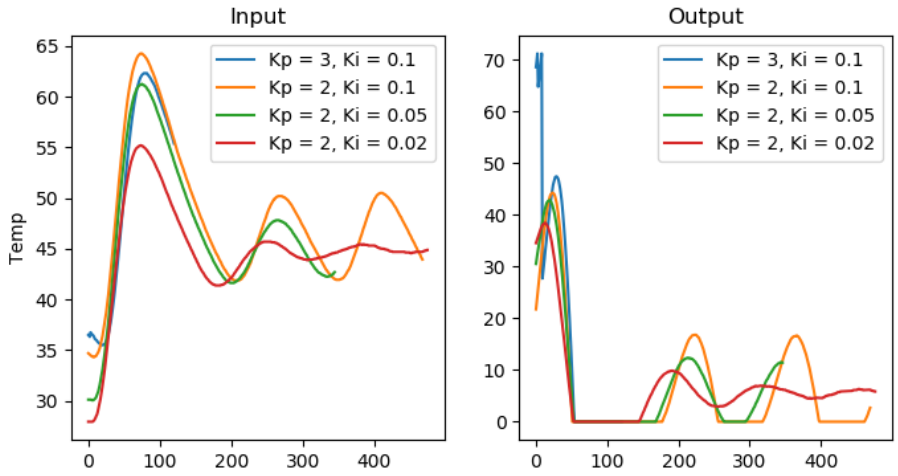

图 14 调节K_i

调节K_d：

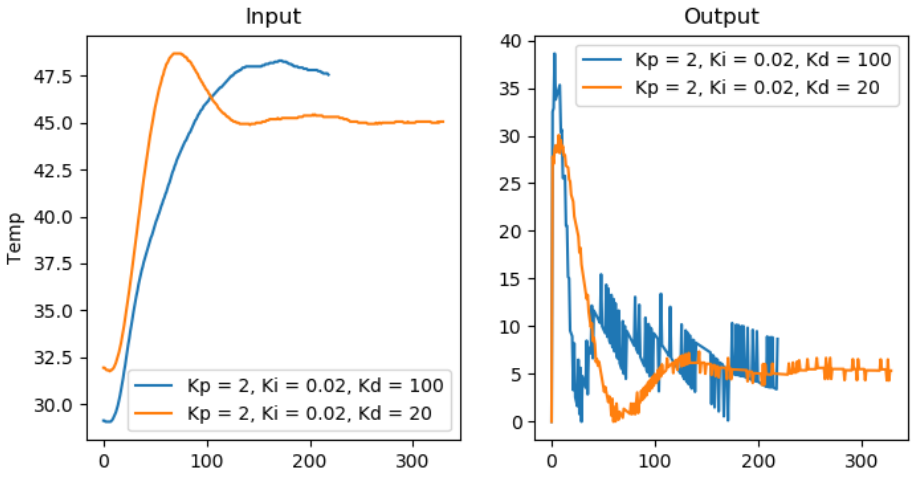

图 15 调节K_d

通过比较不同的K_d值，可以发现，当K_d较大时，输出功率波动明显，同时初次加热用时过长，类似于阻尼振动中“过阻尼”的现象。而适当调小参数之后，可以实现温度的稳定控制。

在参数$K_p = 2, K_i = 0.02, K_d = 20$下，最后 100 个温度值的平均值与方差如下：

$$\mathrm{T} = 45.03 \pm 0.07\ ℃$$

很好的实现了温度控制。

需要注意的是，这组参数依赖于特定的实验环境，在不同的环境下，散热等条件有所不同，参数需要重新整定。

四、 实验结论

1. 制作了基于 Arduino 的温度测量与控制集成平台。
2. 实现了与手机的蓝牙互联。
3. 进行的自动控制系统的 PID 整定，结果$K_p = 2, K_i = 0.02, K_d = 20$。