智能仓储系统的开发研究

先进计算与机器人研究所

2023 年 7 月 9 日

目录

1 第一章: 压力测量模块

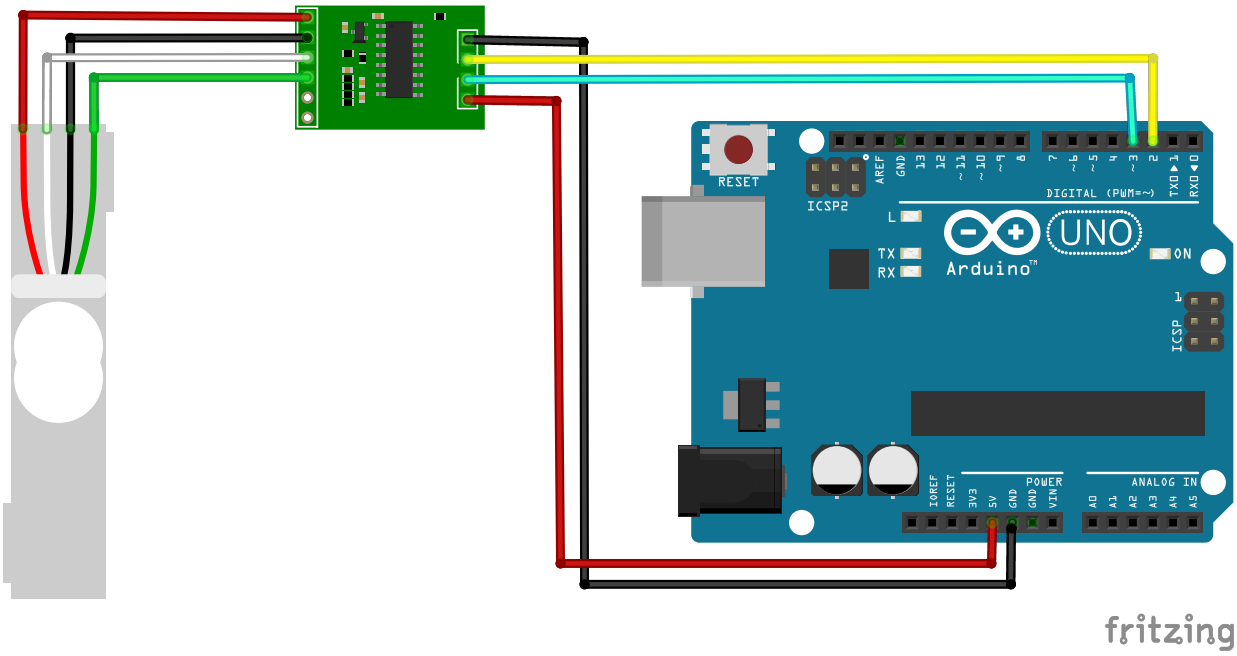

图 1: 压力测量模块流程

压力测量模块的连线：

- 称重传感器的红线代表电源正极, 连接 HX711 模块的 E+ 引脚
- 称重传感器的黑线代表电源负极, 连接 HX711 模块的 E- 引脚
- 称重传感器的白线代表信号输出, 连接 HX711 模块的 A- 引脚
- 称重传感器的绿线位保留线, 连接 HX711 模块的 A+ 引脚
- HX711 模块的 GND 引脚连接 arduino 的 GND 引脚
- HX711 模块的 VCC 引脚连接 arduino 的 5V 引脚
- HX711 模块的 DT 引脚连接 arduino 的任意数字引脚 pin_DT(下文为数字引脚 3), 在构造函数里会将 Arduino 中的该引脚设置为输入模式, 用于接收 HX711 传来的数字信号。
- HX711 模块的 SCK 引脚连接 arduino 的任意数字引脚 pin_SCK(下文为数字引脚 2), 在构造函数里会将 Arduino 中的该引脚设置为输出模式, 用于发送脉冲, 然后 HX711 会按数位输出数据到 pin_DT。

初始秤的计算步骤

- 1. 获得空秤时的 AD 值, 并除以 Gapvalue;
- 2. 获得放了物体后的 AD 值, 并除以 Gapvalue;
- 2. 相减后即得到理论质量;

在上述计算步骤中, 最常用到的就是读取 AD 值这个函数, 而且在得到空秤后的 AD 值, 需要储存便于之后相减的运算。所以我们需要一个类来封装上述读取的方法和读取后的变量。

- 在基类 HX711 中只有读取 AD 值的方法, 也就是第一二步中的前半部分;
- 在基于 HX711 的派生类 Surface 中还有计算 Gapvalue 和储存 AD 值相关的方法, 也就是第一二步中的后半部分;

- 在基于 Surface 的派生类 PressureSensor 中有计算步骤中第三步的方法。

这样三个类刚好实现了这三步初始秤的计算, 并且每一个类的使用都有简洁的 ino 文件来演示。

在这三个类中的方法取名主要是"Get", "Set", "Output" 这三种, "Get" 是用来计算返回一个值, "Set" 是用来将 Get 返回的值保存到类中的某一个变量中, "Output" 是用来将某一变量的值打印在串口。

1.1 HX711

HX711 类的核心变量是 HX711value, 因为后续继承不必继承这个变量, 所以设为私有成员。

```
#include <Arduino.h>

class HX711{
private:
  unsigned long HX711value;

public:
  int pin_SCK, pin_DT;

  HX711(){};
  HX711(int p1, int p2);
  unsigned long Get_HX711value();
  void Set_HX711value();
  void Output_HX711();
};
```

从这一章开头的硬件连接图可以看出来, 压力测量模块除了 5v 和接地要连线, 要设置 SCK 和 DT 连接的数字引脚。设置 SCK 和 DT 的数字引脚是放在 HX711 构造函数 HX711(int p1, int p2) 里,

```
HX711::HX711(int p1, int p2){
  pin_SCK = p1;
  pin_DT = p2;
  pinMode(pin_DT, INPUT);
  pinMode(pin_SCK, OUTPUT);
};
```

HX711 类的中 Get_HX711value 是用开发商的方法返回 AD 值, 具体原理放在了附录前两节。

```
unsigned long HX711::Get_HX711value() {
  unsigned long AD_value=0;
    digitalWrite(pin_DT, HIGH);
    delayMicroseconds(1);
    digitalWrite(pin_SCK, LOW);
    delayMicroseconds(1);
  while(digitalRead(pin_DT));

  for(unsigned char i=0;i<24;++i) {
      digitalWrite(pin_SCK, HIGH);
        delayMicroseconds(1);
      AD_value <<= 1;
        digitalWrite(pin_SCK, LOW);
        delayMicroseconds(1);
      if(digitalRead(pin_DT)) ++AD_value;
    // Serial.println(AD_value);
    }
    // AD_value ^= 0x800000;

    digitalWrite(pin_SCK, HIGH);
    delayMicroseconds(1);
    digitalWrite(pin_SCK, LOW);
```

```
    delayMicroseconds(1);

  // Serial.println(AD_value);
  return AD_value;
};
```

Set_HX711value 是通过 Get_HX711value 来更新 HX711value。

```
void HX711::Set_HX711value() {
  HX711value = Get_HX711value();
}
```

Output_HX711 是将 HX711value 打印在串口。

```
void HX711::Output_HX711() {
  Serial.print("HX711value:");
  Serial.println(HX711value);
}
```

1.1.1 HX711.ino

演示代码是实现将 HX711value 打印在串口。

```
#include "HX711.h"

HX711 hx711(4, 5);

void setup() {
  Serial.begin(9600);
}

void loop() {
  hx711.Set_HX711value();
  hx711.Output_HX711();
  delay(500);
}
```

1.2 Surface

Surface 类的核心变量是 Gapvalue 和 Surfacevalue, 因为后续要继承这个变量, 所以设为公共成员。

```
#include "HX711.h"
#include <EEPROM.h>

class Surface : public HX711 {
union coeffience {
  unsigned long offboard;
  byte onboard[4];
};

public:
  unsigned long Surfacevalue;
  float Gapvalue;
  int range;

  Surface(){};
  Surface(int p1, int p2, int r);
  unsigned long Get_Surfacevalue();
  float Get_Gapvalue();
  void Set_Surfacevalue();
```

```
  void Set_Gapvalue();
  void Output_Surfacevalue();
  void Output_Gapvalue();
};
```

Surface 继承 HX711 后, 构造函数和私有成员不能继承, 所以在 Surface 中重新写了构造函数 Surface(int p1, int p2)。与 HX711 的构造函数不同的是, 这里实现了在 arduino 已储存了 Surfacevalue 时, 会直接读取。

```
Surface::Surface(int p1, int p2, int r) : HX711(p1, p2){
  coeffience memory;
  for (int i=0; i<4; ++i)
    memory.onboard[i]=EEPROM.read(i);
  if (memory.offboard!=0)
    Surfacevalue = memory.offboard;

  range = r;
};
```

这个实现中有一个细节就是创建了一个共用体 coeffience, 当某一共用体 coeffience 变量保存了任意格式的数据, 可以是 unsigned long 格式, 也可以是 byte* 格式, 可以通过访问不同成员 offboard 和 onboard 来切换。

因为 arduino 板上由于位数限制不能直接储存 unsigned long 格式的数据, 需要 4 个地址来储存, 只能按位分成 byte[4] 进行储存, 所以这个共用体对 arduino 的数据储存十分重要。

关于存储 Surfacevalue, 一种是存储 Surface 的 AD 值, 一种是存储除以 Gapvalue 后的 Surfacevalue。因为 AD 值会一直跳动, 即使不放物体, 波动也一直存在。所以存储 AD 值没有什么意义, 一有不同就会需要覆盖重写。除以 Gapvalue 后的值比之前小很多, 波动不会特别明显, 偶尔也需要覆盖重写, 但次数少多了, 节约了 arduino 的重写次数 (arduino 重写次数有限)。

```
float Surface::Get_Gapvalue(int range) {
  return 128 * 16.777215 / range;
};
```

Gapvalue 的计算公式详见附录前两节, range 表示传感器的量程, 当前用的是 20。

Set_Gapvalue 是通过 Get_Gapvalue 来更新 Gapvalue。

```
void Surface::Set_Gapvalue() {
  Gapvalue = Get_Gapvalue();
};
```

Surface 类的中 Get_Surfacevalue 是计算十次 Surface 的 AD 值的平均值, 除以 Gapvalue 后返回计算结果。

```
unsigned long Surface::Get_Surfacevalue() {
  unsigned long AD_Sum=0, AD_Read=0;
  for (int i=0; i<10; ++i){
    AD_Read = Get_HX711value();
    AD_Sum+=AD_Read;
  }
  unsigned long value = AD_Sum/(10*Gapvalue);

  return value;
};
```

Set_Surfacevalue 是通过 Get_Surfacevalue 来更新 Surfacevalue, 并判断是否与 arduino 已储存的相同, 相同就不再覆盖写入了, 反之覆盖重写不同的地方。

```
void Surface::Set_Surfacevalue() {
  Surfacevalue = Get_Surfacevalue();
  coeffience buffer_Surface;
  buffer_Surface.offboard = Surfacevalue;
```

```
5
6     for(int i=1; i<4; i++) //判断是否与上次储存的AD_Surface相同
7       if (!(EEPROM.read(i)==buffer_Surface.onboard[i])) {
8         Serial.println("OVERWRITE");
9         EEPROM.write(i, buffer_Surface.onboard[i]);
10      }
11  };
```

Output_Surfacevalue 是将 Surfacevalue 打印在串口。

```
1  void Surface::Output_Surfacevalue() {
2    Serial.print("Surfacevalue:");
3    Serial.println(Surfacevalue);
4  }
```

1.2.1 Surface.ino

演示代码先将 Gapvalue 打印在串口, 然后将 Surfacevalue 打印在串口。

```
1  #include "Surface.h"
2
3  Surface s(4,5,20);
4
5  void setup() {
6    Serial.begin(9600);
7    s.Set_Gapvalue();
8    s.Output_Gapvalue();
9  }
10
11 void loop() {
12   s.Set_Surfacevalue();
13   s.Output_Surfacevalue();
14   delay(500);
15 }
```

1.3 PressureSensor

PressureSensor 类的核心变量是 Weight, 因为后续要继承这个变量, 所以设为公共成员。

```
1  #include "Surface.h"
2
3  class PressureSensor : public Surface {
4  public:
5    unsigned long Weight;
6
7    PressureSensor(){};
8    PressureSensor(int p1, int p2, int r);
9    unsigned long Get_Weight();
10   void Set_Weight();
11   void Output_Weight();
12 };
```

PressureSensor 继承 Surface 后, 构造函数和私有成员不能继承, 所以在 PressureSensor 中重新写了构造函数 PressureSensor(int p1, int p2)。除了设置 SCK 和 DT 连接的数字引脚, 还要更新好 Gapvalue 和 Surfacevalue。

```
1  PressureSensor::PressureSensor(int p1, int p2, int r) : Surface(p1, p2, r){
2    Set_Gapvalue();
3    Set_Surfacevalue();
4  }
```

PressureSensor 类的中 Get_Weight 是先调用 Get_HX711value(), 除以 Gapvalue 后, 与在构造函数 PressureSensor(int p1, int p2) 里更新了的 Surfacevalue 相减, 返回计算结果。通过 if 来避免这两种情形: 在没放东西的时候, 有可能 AD 值比一开始测得还小, 这时候就会因为出现负数的原因, 显示一个加一取反后的很大的数; 也有可能 AD 值比一开始测得大一点, 显示为 1, 2 等较小的数。

```
unsigned long PressureSensor::Get_Weight() {
  unsigned long difference = Get_HX711value()/Gapvalue - Surfacevalue;
  if (difference > 20000 || difference < 4) difference = 0;
  return difference;
};
```

Set_Weight 是通过 Get_Weight 来更新 Weight。

```
void PressureSensor::Set_Weight() {
  Weight = Get_Weight();
};
```

Output_Weight 是将 Weight 打印在串口。

```
void PressureSensor::Output_Weight() {
  Serial.print("Weight:");
  Serial.println(Weight);
};
```

1.3.1 PressureSensor.ino

演示代码将 Weight 打印在串口。

```
#include "PressureSensor.h"

PressureSensor ps(4,5,20);

void setup() {
  Serial.begin(9600);
  ps.Set_Surfacevalue();
}

void loop() {
  ps.Set_Weight();
  ps.Output_Weight();
  delay(500);
}
```

1.4 上位机采集称重结果

1.4.1 上位机

在上一节得到的结果中, 我们还需要验证一下和真实重量的差别有多大。为了方便保存已放置的砝码的质量数值, 用到了上位机。

- 用户在上位机输入放入砝码的总次数, 即数组长度;
- 用户在上位机输入当前放在秤上的真实质量数值;
- 按下回车后, 等待 arduino 接收到信号后返回称重结果给上位机;
- 上位机在接收到后, 将之前输入的真实质量数值和对应称重结果分别保存到两个数组中, 然后画图显示。

```python
import serial # 导入模块
import re
import matplotlib.pyplot as plt
import numpy as np

weights = []
realweights = []
def read_serial_data(port):
    ser = serial.Serial(port, 9600, timeout=1)
    n = int(input("数组长度:"))
    while 1:
        user_input = input("请输入一个数字(输入'q'退出):")

        if user_input.lower() == 'q':
            break

        try:
            ser.write(user_input.encode())
            while 1:
                data = ser.readline().decode('utf-8')
                if data:
                    break
            number = re.findall("\d+", data)
            if number and number[0] != '0':
                print(number[0])
                weights.append(int(number[0]))
                realweights.append(int(user_input))
            if len(weights) == n:
                print(weights)
                print(realweights)
                break

        except ValueError:
            print("输入无效,␣请输入一个有效的数字!␣")
    ser.close()

read_serial_data("/dev/ttyUSB0")

k1 = 0
k2 = 0
n = len(realweights)
for i in range(n):
    k1 += realweights[i] * weights[i]
    k2 += realweights[i] * realweights[i]
mean1 = sum(realweights) / n
mean2 = sum(weights) / n
k = (k1 - n * mean1 * mean2) / (k2 - n * mean1 * mean1)
b = mean2 - mean1 * k
print(k, b)
x = np.linspace(500, 600, 1000)
y = k*x+b

plt.plot(x, y)
plt.scatter(realweights, weights)
plt.xlabel('realweights')
plt.ylabel('weights')
plt.show()
```

得到的数组 realweights 和 weights 分别作为横纵坐标画散点图。理想情况下, 画出的散点图用直线 $y = kx + b$ 拟合就能基本覆盖。所以后续用最小二乘法来进行校准秤。为了方便日常使用的校准, 这里是在上位机中显示散点图和最小二乘法得到的直线图。

1.4.2 DataCollect

DataCollect 类的核心作用就是接收到上位机发来的信号后称重, 并将结果返回给上位机。因为要称重, 所以肯定要用到 PressureSensor 类, 后续没有这个这个类相关的继承, 所以直接将 ps 设为私有成员。

```
#include "PressureSensor.h"

class DataCollect{
private:
  PressureSensor ps;

public:
  DataCollect() {};
  DataCollect(int p1, int p2, int r);
  void Output_data();
};
```

在构造函数 DataCollect(int p1, int p2) 里, 对 ps 进行初始化, 并更新 ps 中 Surfacevalue。

```
DataCollect::DataCollect(int p1, int p2, int r) {
  ps = PressureSensor(p1, p2, r);
  ps.Set_Surfacevalue();
};
```

由于串口通信是一个字节一个字节传输, 所以当检测到串口有消息传来时, 需要先把消息用 Serial.read() 读完, 再进行称重, 打印结果。否则, 上位机发送” 一个” 数值 (其实是个字符串), 会收到多个返回值。

```
void DataCollect::Output_data() {
  if (Serial.available()>0) {
    while (Serial.available()>0)
      Serial.read();
    ps.Set_Weight();
    ps.Output_Weight();
  }
}
```

1.4.3 DataCollect.ino

演示代码实现等待接收上位机信号, 称重一次, 并返回一次结果。

```
#include "DataCollect.h"

DataCollect dc(4, 5, 20);

void setup() {
  Serial.begin(9600);
}

void loop() {
  dc.Output_data();
  delay(500);
}
```

1.5 附录

1.5.1 常用函数 HX711_Read(part 1)

HX711_Read() 是用来得到 HX711AD 模块读到的电压信号。HX711AD 模块在连接压力传感器后, 可以将压力传感器所传出的电压信号 (一种模拟信号) 转换成 AD 值 (一种数字信号)。其内部实现是通过建立一个电压值 (模拟信号) 与 24 位的二进制数 (数字信号) 的对应关系表。在前面我们指定了与 HX711_SCK

和 HX711_DT 相连的引脚 pin_SCK 和 pin_DT, 在 HX711_Read() 中, pin_SCK 主要是输出脉冲到 HX711_SCK, pin_DT 主要是输入 HX711_DT 传来的数字信号。其中前 24 次脉冲分别使得 pin_DT 读到二进制数的所有数位上的数。在此之后, 可以选择再进行第 25 次或第 26 次或第 27 次脉冲, 其作用用于确认压力传感器输入到 HX711AD 模块的输入通道种类 (分为 AB 两种) 以及增益倍数, 具体规律如下表。

SCK 脉冲数	输入通道	增益
25	A	128
26	B	32
27	A	64

硬件连接上也有对应的要求, 当压力传感器的输出信号线选择 A 通道 (即白线和绿线连接 HX711AD 模块的 A-和 A+ 引脚时), 如果只发送 25 次脉冲并成功发送, 压力传感器输出的信号会得到 128 倍增益, 此时 HX711AD 模块接收到的模拟信号为:$\frac{x}{A} \times VAVDD \times 128 \times 1mv/V$。HX711AD 模块的引脚 E+ 上的输出电压就是 VAVDD, E-上的输出电压为 VGND(因为接地了, VGND 值为 0)。在每次测量压力时, 压力传感器收到 HX711AD 模块发送的 VAVDD 作为激励电压后, 会返回 $\frac{x}{A} \times VAVDD \times$ 灵敏度 (这里压力传感器的灵敏度为 1mv/V, 具体值的大小可能存在误差, 但下文用此数值代替)。考虑到信号增益倍数 128 以及 VAVDD 与 HX711AD 模块的最大储存值 $2^{24}-1$, 可以得到 AD 最大值

$$\begin{aligned} AD_{max} &= \frac{x}{range} \times VAVDD \times 0.001 \times 128 \times \frac{2^{24}-1}{VAVDD} \\ &= 0.128 \times (2^{24}-1) \end{aligned} \tag{1}$$

```
unsigned long PressureSensor::HX711_Read() {
    unsigned long AD_value=0;
    digitalWrite(pin_DT, HIGH);
    delayMicroseconds(1);
    digitalWrite(pin_SCK, LOW);
    delayMicroseconds(1);
    while(digitalRead(pin_DT));

    for(unsigned char i=0;i<24;++i) {
        digitalWrite(pin_SCK, HIGH);
        delayMicroseconds(1);
        AD_value <<= 1;
        digitalWrite(pin_SCK, LOW);
        delayMicroseconds(1);
        if(digitalRead(pin_DT)) ++AD_value;
    }

    digitalWrite(pin_SCK, HIGH);
    delayMicroseconds(1);
    digitalWrite(pin_SCK, LOW);
    delayMicroseconds(1);

    return AD_value;
};
```

HX711 的读取是整个压力测量模块用到的最多的函数。可以分成三大部分来看:
第一步: 准备开始读取信号。这一步也要做四件事情:

I. 局部变量 AD_value 赋值为 0。这里选用 unsigned long 类型是因为, 设压力传感器的量程为 range, 量程对应的 AD 值即为 AD 值的最大值, 经过计算可以得到, $AD_{max} = 128 \times 0.001 \times (2^{24}-1) = 0.128 \times (2^{24}-1)$ 。由于 AD 值理论上只有正数, 并且在 Arduino 中,unsigned long 型的变量可以存储的值的范围为 $0 \sim 2^{32}-1$, 因此这里选择用 unsigned long 型变量进行存储;

```
unsigned long AD_value=0;
```

II. pin_DT 引脚的初始电平设置为高电平, 若与外部器件连接, 电平会发生改变;

```
digitalWrite(pin_DT, HIGH);
delayMicroseconds(1);
```

III. pin_SCK 对应的 Arduino 上数字信号引脚 2 写入低电平, 并和 pin_DT 一样稍微延迟 1 微秒, 确保成功写入。由于 HX711_SCK 接收到 pin_SCK 传来的脉冲上升沿时才开始输出信号, 所以 pin_SCK 要先写入低电平;

```
digitalWrite(pin_SCK, LOW);
delayMicroseconds(1);
```

IV. 未连接之前一直是高电平, 直到连接后 pin_DT 恢复到了低电平与 HX711_DT 保持一致, 跳出循环。低电平表示准备好开始传输了。

```
while(digitalRead(pin_DT));
```

1.5.2 常用函数 HX711_Read(part 2)

第二步: 开始读取信号。这里也要分四步

```
for(unsigned char i=0;i<24;++i) {
    digitalWrite(pin_SCK, HIGH);
    delayMicroseconds(1);
    pin_value<<=1;
    digitalWrite(pin_SCK, LOW);
    delayMicroseconds(1);
    if(digitalRead(pin_DT)) ++pin_value;
}
```

I. 写入高电平, 产生上升沿;

```
digitalWrite(pin_SCK, HIGH);
delayMicroseconds(1);
```

II. pin_value 移位准备储存最高位数字;

```
pin_value<<=1;
```

III. pin_SCK 恢复到低电平, 为下一次传递信号做准备;

```
digitalWrite(pin_SCK, LOW);
delayMicroseconds(1);
```

IV. 如果是高电平, 意味着这一数位上的数为 1, 否则为 0, 不更新。

```
if(digitalRead(pin_DT)) ++AD_value;
```

第三步: 设置下一次 HX711 读取的增益模式, 这里还是选的 A 输入通道, 增益 128。

```
digitalWrite(pin_SCK, HIGH);
delayMicroseconds(1);
digitalWrite(pin_SCK, LOW);
delayMicroseconds(1);
```

1.5.3 最小二乘法拟合直线公式

$$k = \frac{\sum xy - n\overline{x}\overline{y}}{\sum x^2 - n\overline{x}^2} \tag{2}$$

$$b = \overline{y} - k\overline{x} \tag{3}$$