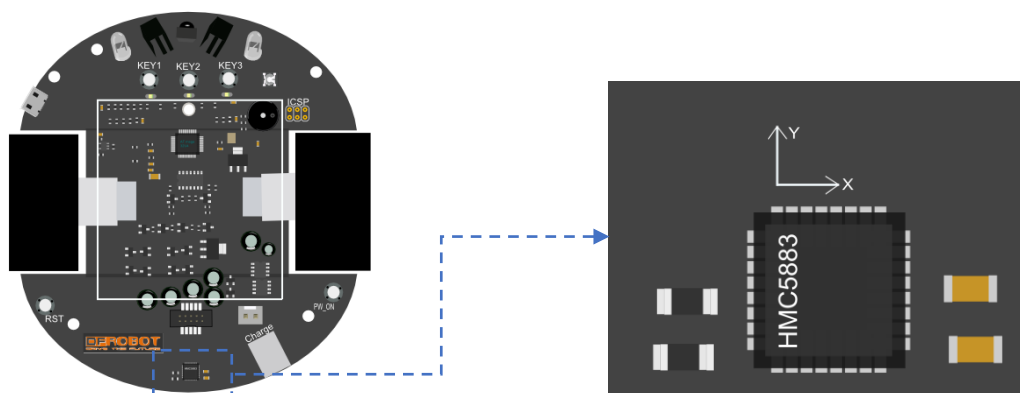


第八课 电子罗盘的使用



一． 本节要点

电子罗盘可以干的事很多：水平孔和垂直孔测量、水下勘探、飞行器导航、科学研究、教育培训、建筑物定位、设备维护、导航系统、测速、仿真系统、GPS 备份、汽车指南针、虚拟现实。

随着智能终端的快速普及，地理方向的获取已经变得十分容易，遥想当年航海者经验高于一切仪器的大航海时代初期（当时没有很可靠的仪器），我们是不是该庆幸自己生处一个不会迷路的时代，并且，我们身处一个可以自己制作数字指南针的时代！

我们这次要完成的任务有：

- ◆ 使用小车上的电子罗盘制作指南针
- ◆ 测出小车方向并用电脑端显示出来

二 . 程序下载

STEP1： 首先是要安装库文件 HMC5883L

我的电脑 > 本地磁盘 (C:) > arduino-1.0.4-windows > arduino-1.0.4 > libraries		
名称	修改日期	类型
Ethernet	2013/3/11 15:29	文件夹
FFT	2013/8/14 11:03	文件夹
Firmata	2013/3/11 15:29	文件夹
GSM	2013/3/11 15:29	文件夹
Hexapod_Servo	2013/9/16 16:29	文件夹
HMC5883L	2014/4/4 18:26	文件夹
Hx711	2014/1/21 16:00	文件夹
I2Cdev	2014/1/22 11:12	文件夹
LCD4884	2012/11/23 15:55	文件夹
LiquidCrystal	2013/3/11 15:29	文件夹
LiquidCrystal_I2C	2013/7/8 13:16	文件夹

图 1 库文件安装目录

STEP2： 打开代码文件夹下 hmc 文件夹里的 hmc.ino 文件，编译下载到 miniQ 上，

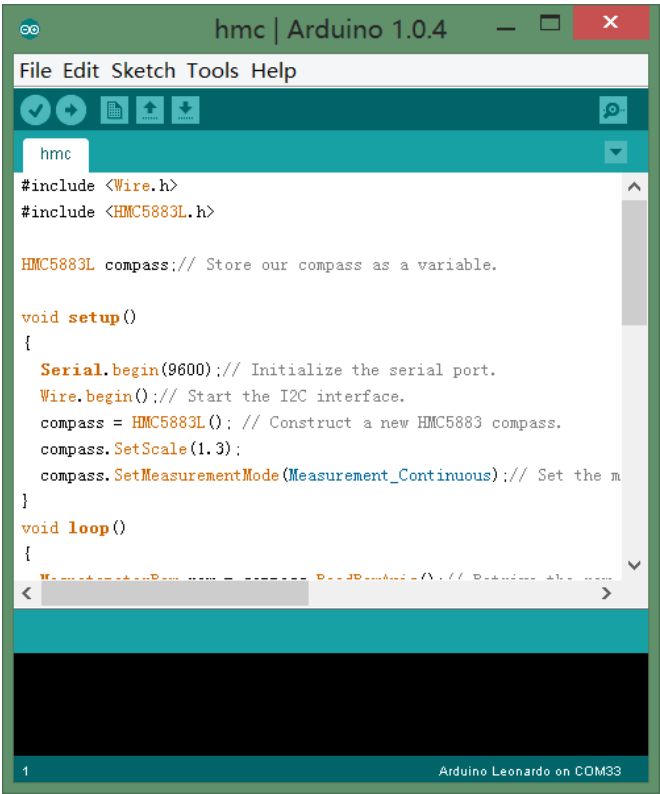


图 2 程序截图

STEP3：打开串口助手

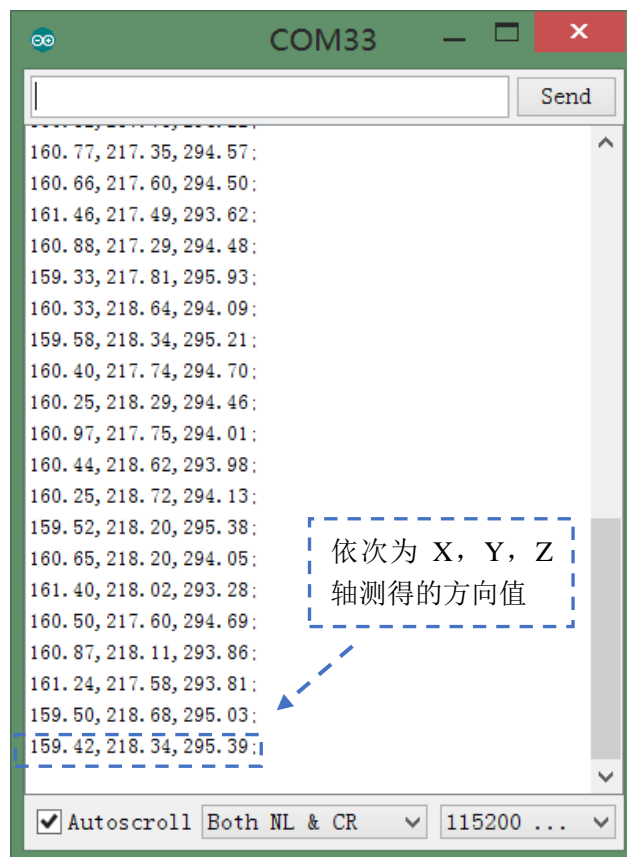


图 3 电脑端串口输出结果截图

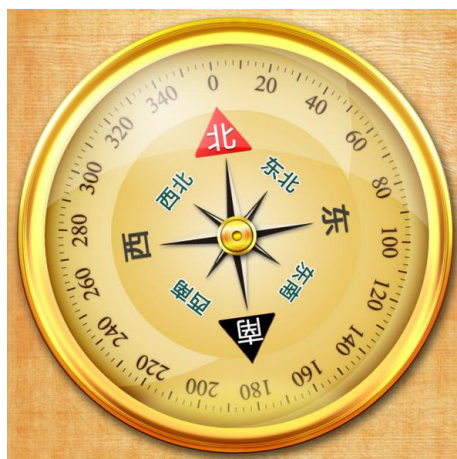


图 4 角度与方向的换算关系

如上，小车测得的数据约为 160，所以小车方向为南偏东 20 度。但是由于传感器安装方向问题，小车实际正面朝向需要减去 90 度。所以小车朝向为 50 度，即北偏东 50 度。转动小车，观察小车输出数据变化吧。



图 5 小车电子罗盘测定的方向

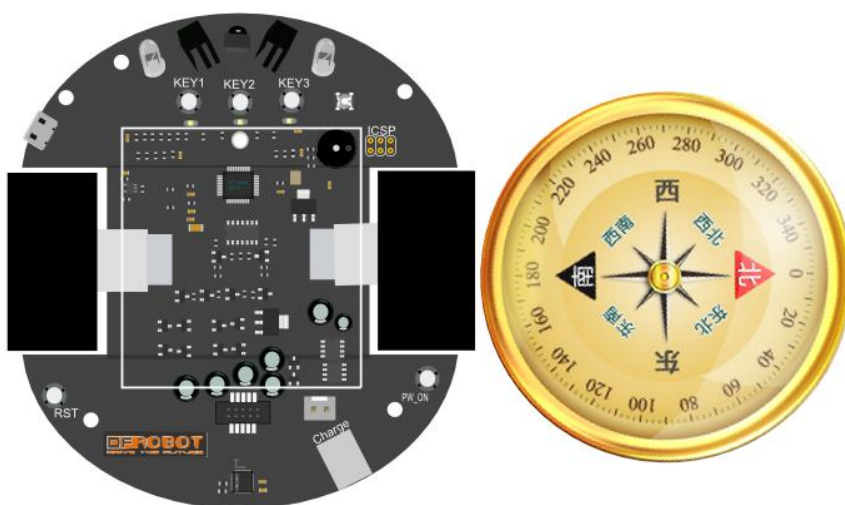


图 6 小车 X 轴值为 0 时的与标准指南针的比对

好了，现在旋转一下你的小车，看看效果吧。

注意事项

(1) 由于电子罗盘的是基于地磁感应的，所以传感器周围尽量不要有磁性物品，如果测量数据不准确可以检查工作台或者手表等磁性物品。

(2) 要是您足够细心，一定会发现小车水平转动时 X 值是和方向对应的，但是如果不是水平的数据就不对了。



图 7 智能机翻转示意图

这时，你如果身边有个智能手机的话，打开指南针，看看指针会不会在手机翻转时指方向。当然，测试表示 Iphone 用户无需担心这一点，因为其内部软件结合了加速度计已经做好了补偿。

常见问题

问： 使用 HMC5883L 能不能不用加速度传感器就能计算出航向？

答： 是的，不用加速度传感器就可以计算出航向。但是，传感器必须保证是水平的，也就是，俯仰角与滚转角都为零。这样，在水平面上的三个轴中只有两个会被用到。

问： 为什么需要加速度传感器？

答： 如果罗盘不是水平的，就需要加速度传感器来纠正由于倾斜引起的误差。或者，Honeywell 提供了一体化解决方案 HMC6343 作为另外一种选择。HMC6343 内置 3 轴加速度传感器，具备倾斜补偿的航向输出。

三． 软件解析

(1) 库文件导入函数

```
#include <Wire.h>//导入 IIC 库文件
#include <HMC5883L.h>//导入 HMC5883 库文件
```

(2) 定义

```
HMC5883L compass;// Store our compass as a variable.
```

(3) 初始化 HMC5883

```
compass = HMC5883L(); // Construct a new HMC5883 compass.
compass.SetScale(1.3);
compass.SetMeasurementMode(Measurement_Continuous);// Set the
measurement mode to Continuous
```

(4) 值获取

```
MagnetometerRaw raw = compass.ReadRawAxis();// Retrive the raw
values from the compass
MagnetometerScaled scaled = compass.ReadScaledAxis();// Retrived
the scaled values from the compass (scaled to the configured scale).
```

(5) 值处理 (从两个轴向的分量上计算出方向)

```
float xHeading = atan2(scaled.YAxis, scaled.XAxis);
float yHeading = atan2(scaled.ZAxis, scaled.XAxis);
float zHeading = atan2(scaled.ZAxis, scaled.YAxis);
```

(6) 值处理 (将取值调整到 0-2PI)

```
if(xHeading < 0)    xHeading += 2*PI;
if(xHeading > 2*PI) xHeading -= 2*PI;
if(yHeading < 0)    yHeading += 2*PI;
if(yHeading > 2*PI) yHeading -= 2*PI;
if(zHeading < 0)    zHeading += 2*PI;
if(zHeading > 2*PI) zHeading -= 2*PI;
```

(7) 值处理 (将角度值从弧度换成角度)

```
float xDegrees = xHeading * 180/M_PI;
float yDegrees = yHeading * 180/M_PI;
float zDegrees = zHeading * 180/M_PI;
```

由上 ,HMC5883 的库文件提供了读出角度值的函数。我们只要调用这些函数就可以了。

xHeading 是根据反三角函数计算出的 x 的角度 ,再判断 xHeading 是不是在 0-360 度以内 ,不是的话就加上或减去 360 度 ,这样我们就可以得到 0-360 度这个范围内的角度值了。

四． 硬件原理

HMC5883 是一种磁阻效应传感器 ,磁阻效应传感器是根据磁性材料的磁阻效应制成的。磁性材料(如坡莫合金)具有各向异性 ,对它进行磁化时 ,其磁化方向将取决于材料的易磁化轴、材料的形状和磁化磁场的方向。如图所示 ,当给带状坡莫合金材料通电流 I 时 ,材料的电阻取决于电流的方向与磁化方向的夹角。如果给材料施加一个磁场 B (被测磁场) ,就会使原来的磁化方向转动。如果磁化方向转向垂直于电流的方向 ,则材料的电阻将减小 ;如果磁化方向转向平行于电流的方向 ,则材料的电阻将增大。磁阻效应传感器一般有四个这样的电阻组成 ,并将它们接成电桥。在被测磁场 B 作用下 ,电桥中位于相对位置的两个电阻阻值增大 ,另外两个电阻的阻值减小。在其线性范围内 ,电桥的输出电压与被测磁场成正比。

测得的数据最后经过 HMC5883 内的处理部分的处理即可有外部读取。由于没有集成 3 轴加速度计 ,所以 HMC5883 测量的角度是不带有倾角补偿的。

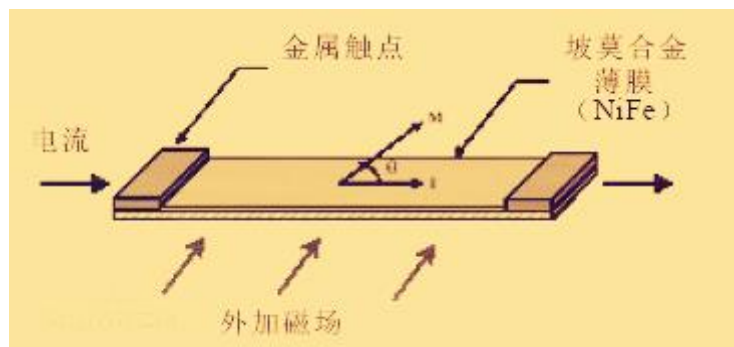


图 8 磁阻效应传感器示意图

五． 电路设计

由于电子罗盘是一个很集成化的解决方案，所以并不需要我们在你外部加上过多的电路。

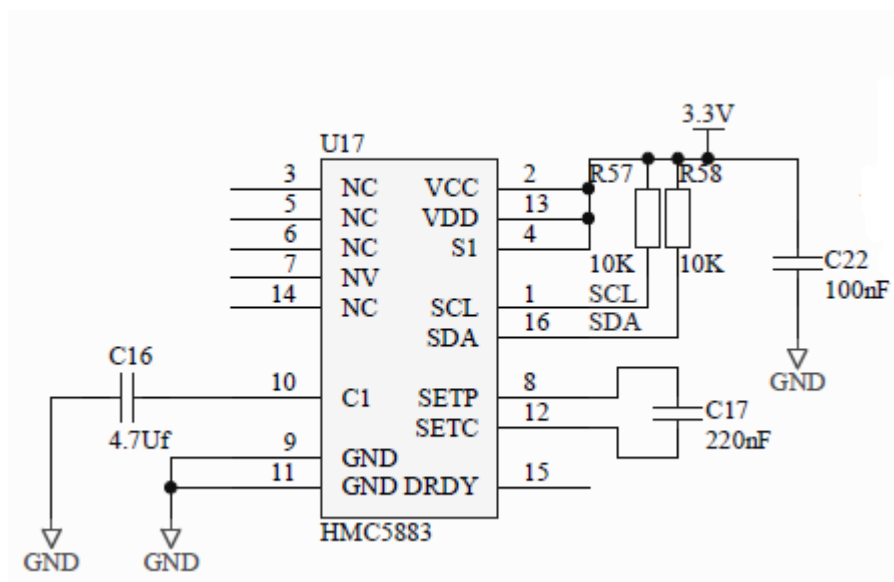


图 9 HMC5883 电路图

如图，和 miniQ 上控制器相连的只有第一脚（SCL）和第 16 脚（SDA），其它的都是芯片自身的电源电路。两个 10K 的电阻是把 SCL 和 SDA 的初始电压拉高为高电平的，可以防止引脚状态的不确定性。Arduino 端的引脚号 D2—SDA，D3—SCL。

好了，有了电子罗盘，让你的小车朝这一个方向一直前进吧。

六． 扩展部分

（1）有了电子罗盘，我们可以使小车按照固定的方向行驶了，试试写个程序，使小车自动朝着东方行驶，过程中旋转小车方向，看看小车反应吧。

（2）尝试使用 3 轴加速度完成小车的倾角补偿，同时学习 3 轴加速度的使用，进一步了解数字式传感器的使用。

七．充电库

miniQ 上用来测量方向的传感器为 HMC5883，霍尼韦尔 HMC5883 是一种表面贴装的高集成模块，并带有数字接口的弱磁传感器芯片，应用于低成本罗盘和磁场检测领域。小车尾部的电子罗盘传感器可以帮助我们更好的完成对小车行进方向的获取。

由于传感器通讯格式为 IIC(一种通讯协议)，所以我们只要在程序中对所要数据的地址进行询问就可以了。通过询问得到的数据，对小车方向进行控制，使小车航向保持不变。

我们经常在各种拓展板的介绍中看见，支持 UART，SPI，IIC 通信协议等等，这些协议在 Arduino 里面都非常简单，可能是一个简单的 `Serial.print()`，`SPI.transfer()`，`Wire.write()`就解决了问题。但是那么这些函数在硬件层次到底是如何实现的呢？想了解这个，首先要了解一些关于微控制器（Arduino MCU 等）对于电路的控制方式。现代控制器的输入输出也就是数字 IO 口（某些带有 DA 功能的口除外）通常只能输出高低电平。所谓高低电平也就是 0V 和 5V 的一个电压信号（也有控制器是 3.3V（如 Flymaple），Arduino Uno 和 miniQ 的电压啊都是 5V），那么只通过低和高如何能发送复杂的信息呢？

为什么需要协议？

这个时候数学解决了这个问题，传说是起源与周易的二进制数学，就只有 2 个数，0 和 1。正好这一对兄弟分别对应低电平和高电平。于是这样就好像可以传信息了嘛，收到高电平（5V）就当作收到 1，收到低电平就当作收到 0，于是这个时候接收这个信号的芯片就好像在做一个电报接收员一样把一连串的 010101 信号对应着一个信号表翻译成具体的指令。等等，说到这，喜欢思考的读者应该已经发现，之前我所说的高电平，低电平发送数据有个致命的毛病，如果我要连续发送 2 个 0 信号怎么发？我发送 2ms（实际可能要快得多）的低电平，到底是想发多少个 0？

这时候，各位聪明的人类很快就想出了 2 个办法：

- (1) 限定每个信号的时间，比如规定高低电平持续时间为 1ms，那么 2ms 低电平就代表 2 个 0。(UART 基本是用的这个原理)
- (2) 多添加一个信号线，当这个信号线上的电平发生变化的时候读入之前信号线上的信号(后来这个新加的信号线就叫时钟线，大部分协议都是用的这种方式)

这两种方式各有利弊，第一种只需要 1 根线，同样由于只有一根线，那么为了减少误传输只能降低传输速度，第二个需要 2 根线，但可以以很快的速度进行数据传输。也可以再加一组线组成全双工模式(发送数据的同时还可以接收数据)。

什么是 IIC 协议？

了解了以上这些，我们就可以回到正题，IIC 是什么呢，IIC 是以第二种方式运行的一套协议。他由 3 根线组成分别叫 SDA，SCL，GND，SDA 为数据线，SCL 为时钟线，GND 为参考电平，就是 0 电平，这个线的主要功能是提供一个参考的电压，告诉 IC 到底多少为 0V 电势(通常我们说的多少 V 是说的电势差及电压)。

下面我要说的东西就比较复杂了，对于第一次接触通信协议的童鞋可能有点难以理解，我会尽量写的简单清晰一点。

先说说为什么要用协议，有人可能会想，按照之前所说，当时钟线上电平发生改变(我们形象的称之为上升沿和下降沿)时读入数据线的 0 或 1 不就好了，为什么还要什么协议呢？

说说我个人理解，之所以有协议是为了更加稳定也更加的可靠。举个例子，家里敲门，大家可能都有这样的经历，有时候听到敲门然后出门，发现门外并没有人，这就有 2 个可

能，一个是外部原因：外面有什么人或者物不小心敲到了门。或者是内因：自己不小心听错了。无论是什么原因，都造成了我们信息的错误获取（认为有人敲门），所以我们想到一个方法，以后每次敲门都先敲 1 下，然后连续敲 3 下，以此表示我要进来这个信号，这样里面的人听到有人敲门后再听到 3 次短促的敲门才去开门，这样外因或者内因产生的错误都将大大减少，我认为这是发明协议的最初原因。

好了言归正传，现在我们可以说说什么叫 IIC 协议：

这种总线类型是由飞利浦半导体公司在八十年代初设计出来的，主要是用来连接整体电路，IIC 是一种多向控制总线，也就是说**多个芯片可以连接在一起**，同时每个芯片都可以对其他的芯片进行数据传输。这种方式简化了信号传输总线。但是这种方式也增加了一个传送数据的步骤，就是需要对信号线上的每个芯片定义一个逻辑地址，让芯片知道这个信息是发给他的（类似电话号码的功能），这个之后再说。

IIC 协议包含 START（类似于第一下敲门）、ACK（里面的人给予回应）、NACK（另一种回答）、STOP（表示数据发送结束）这几个东西，当然还少不了普通的数据发送。尽管协议中规定 START 必须，其他几个非必须，但实际上其他三个仍旧非常重要。

IIC 协议的具体形式

IIC 发送数据是 8 位数据 (即 8 个 0 或 1) 为一个小单元一起发送。

整个协议发送的顺序如下：通常我们为了方便把设备分为主设备和从设备，我的划分方法是谁控制时钟线 (即控制 SCL 的电平高低变换)，谁就是主设备。。。

主发从收： 主机发送 START 信号 -> 主机发送地址 -> 从机发送 ACK 应答 -> (主机发送数据 -> 从机发送 ACK 应答 (循环) -> 主机发送 STOP 信号 或 主机发送 START 信号启动下一次传输

主收从发： 主机发送 START 信号 -> 从机发送发地址 -> 主机发送 ACK 应答 -> (从机发送数据 -> 主机发送 ACK 应答 (循环)) -> 接受至最后一个字节时，主机发送 NACK -> 主机发送 STOP 信号 或 主机发送 START 信号，启动下一次传输

还是回到大家熟悉的 Arduino 编程，看下面一段代码：

```
Wire.beginTransmission(0x04);  
Wire.write(0xaa);  
Wire.endTransmission();
```

0x 代表的是 16 进制数转换为 2 进制为：

```
0x04=0100b  
0xaa=10101010b
```

以上是一段主发从收的代码。

```
Wire.beginTransmission(0x04);
```

执行了 3 步：主机发送 START 信号 -> 主机发送地址 -> 从机发送 ACK 应答，后面的 0x04 就是从机的地址。

```
Wire.write(0xaa);
```

执行了 2 步：主机发数据 -> 从机发送 ACK 应答（循环），参数 0xaa 就是发送的数据。

```
Wire.endTransmission();
```

执行了最后一步：主发送 STOP 信号。

IIC 为什么需要地址？

这个问题其实很好理解，IIC 总线上很多器件是连在一起的，就像学校里的广播一样，全校各个班级都和这个广播播放的地方连接，如果有一天广播播报说请 3 班同学到教务处报到，那么很多个 3 班的同学都会去，而只要加上 2 年级 3 班，就会只有一个班的人。这个就是地址的用处，可以指定整个学校里的特定的一个班不会让其它的班级误解。而且不用每个班级都使用一条单独的广播线路。同理，IIC 总线上每一个器件都需要有一个地址，而且不能重复。