2020 人工智能课程报告（总）

课题：Task1-智能小车

姓名：车永祥

学号：2018202179

日期：2020/12/20

组号：26

目录

一、 项目总述

1. 实验内容

在本次课题中，我们基本完成了现有硬件支持的所有基本功能，其中包括：

1) 智能小车底盘安装之电机固定
2) 智能小车底盘安装之万向轮固定、电池盒固定和底板简单调试
3) 智能小车底盘安装之电机线接法与 Arduino 主板及转接板安装
4) 智能小车前进实验
5) 智能小车前后左右综合实验
6) 智能小车按键启动和蜂鸣器报警
7) 智能小车黑线循迹实验
8) 智能小车红外避障实验（基础避障+跟随障碍物）
9) 智能小车超声波测距实验
10) 智能小车超声波避障实验(无舵机+有舵机)
11) 智能小车红外遥控实验

其中 1）,7）,11）由组员车永祥单独实现，8）,9）,10）由组员周珂单独实现，其余 2）,3）,4）,5）,6）由组内两人共同实现。

2. 实验进度

1）我们首先完成了小车元件的安装和基本调试，尝试了像前进、后退。旋转这样的一些简单功能。
2）接下来，为了实现一些高级功能。我们先花了两周时间认真学习了小车的一些硬件知识，学习了 Arduino 小车编程的代码结构和一些函数。
3）然后逐步去使用提供的硬件，安装按键和蜂鸣器，利用红外探测功能实现黑线循迹和红外避障。
4）最后，安装好超声波模块和舵机，实现了超声波测距、超声波避障（有舵机+无舵机）以及红外遥控实验。

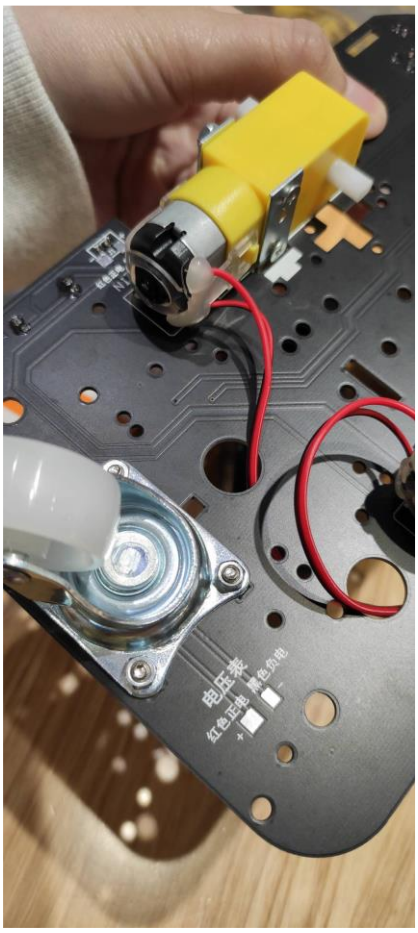
3. 小组情况

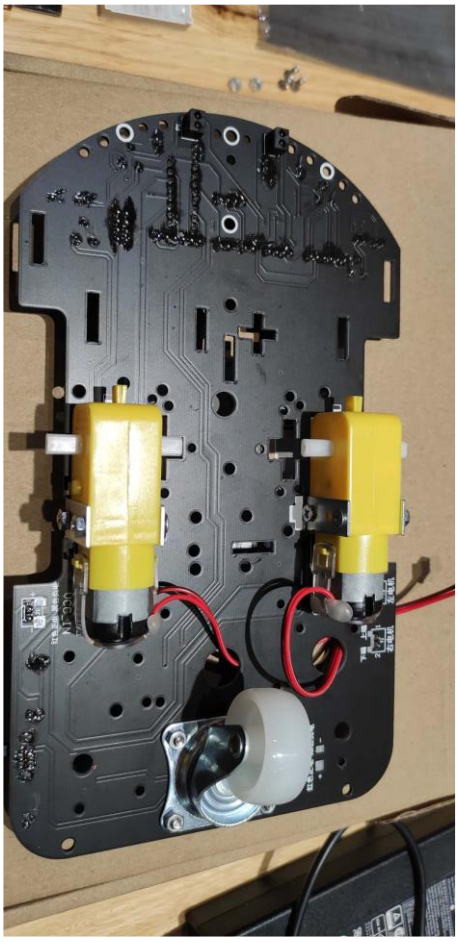
组号：26
组员：周珂 2018202070
车永祥 2018202179
组员之间合作非常愉快，大家都能负责的完成自己的任务。

二、 小车的组装及调试

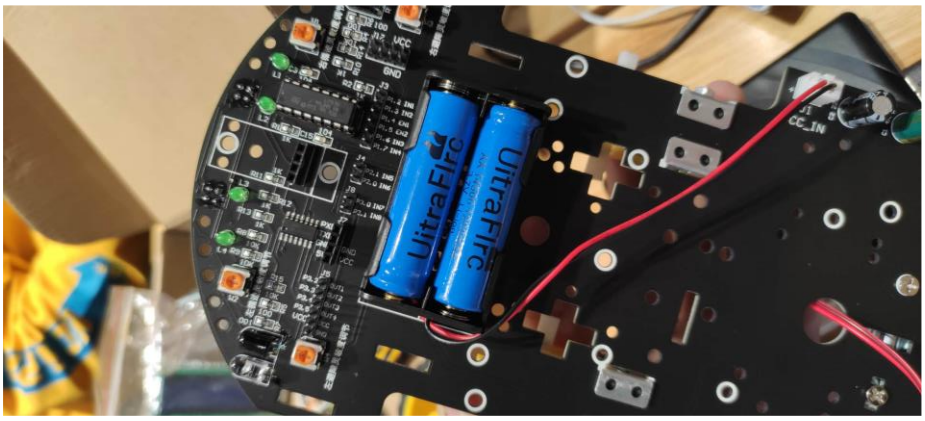
1. 硬件组装

A. 步骤一：左右电机的安装和固定以及万向轮的安装固定

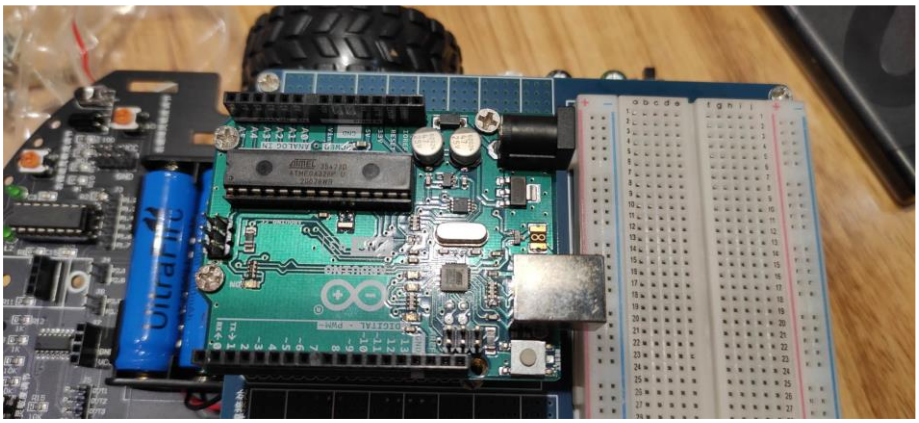


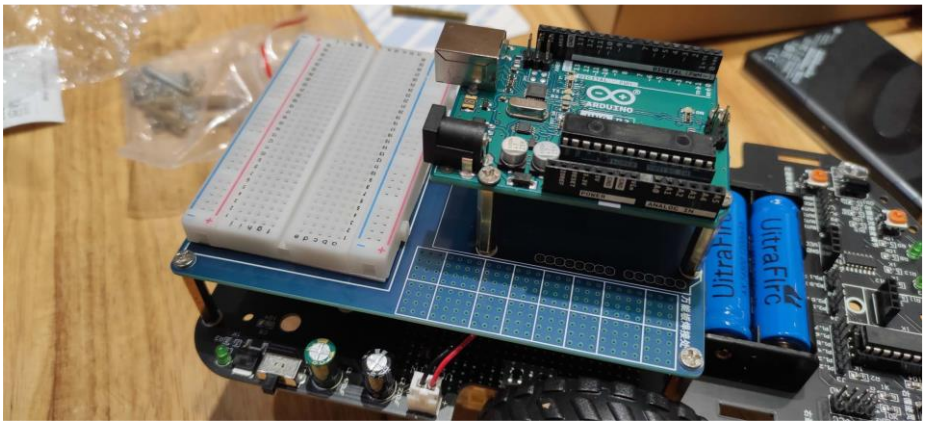


B. 步骤二：电池的固定

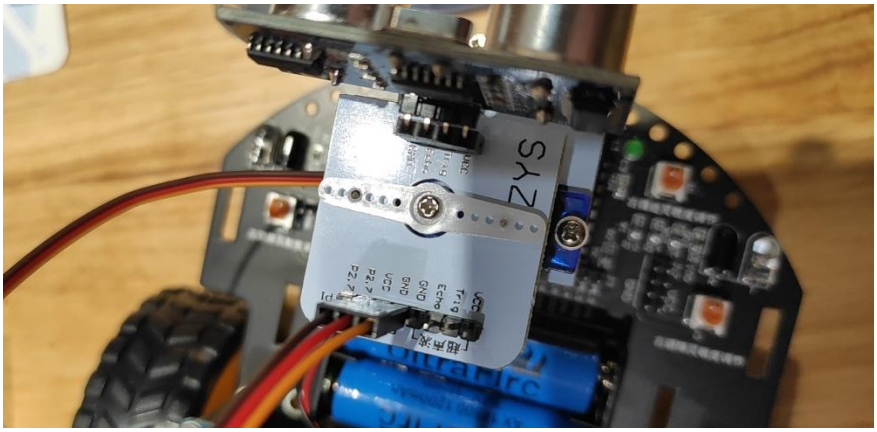


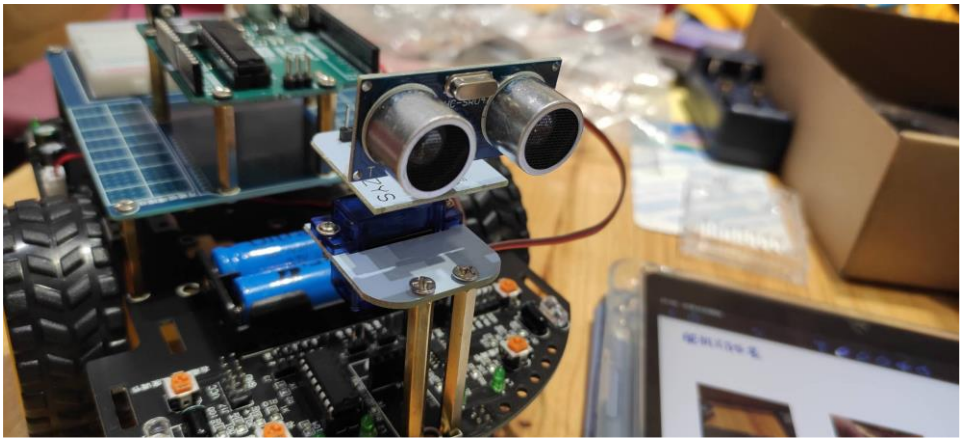
C. 步骤三：主板的安装和固定，基本线路的连接





D. 步骤四：舵机及超声模块的安装

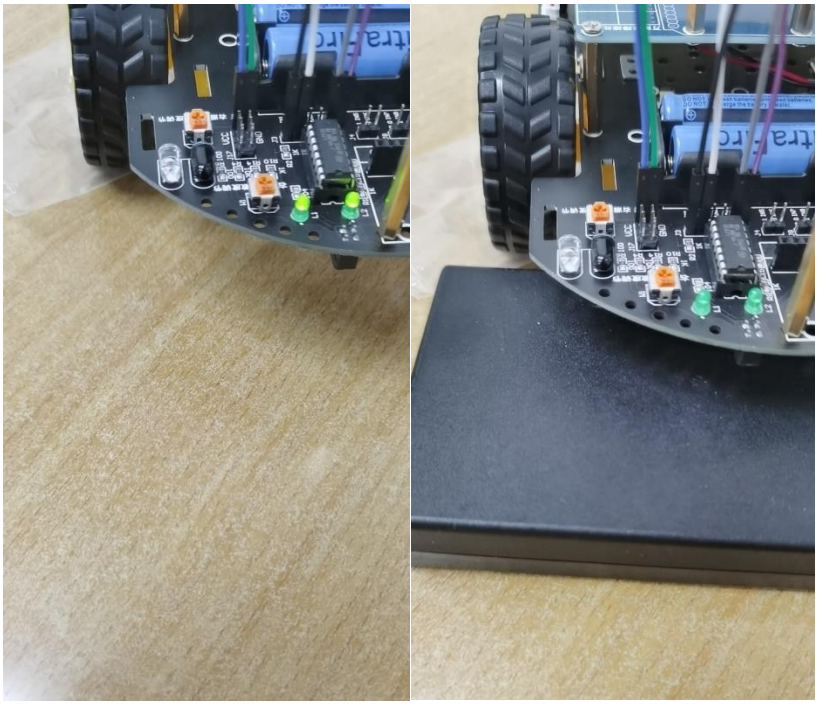




2. 探头调试

A. 底板上避障探头的测试

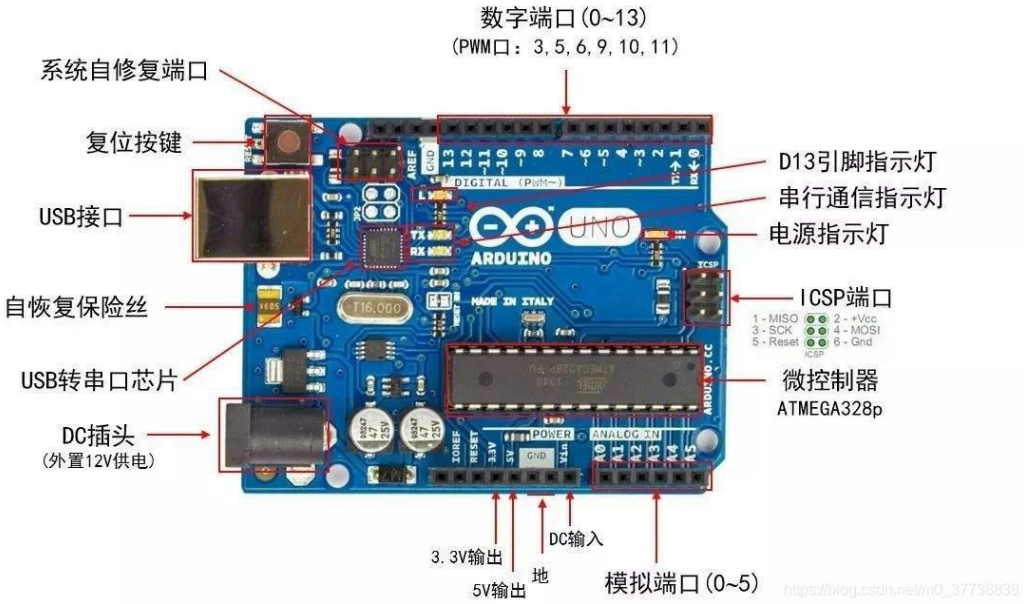
B. 底板上随机探头的测试



三、 硬件、函数和程序结构等的学习

1. 硬件学习

1) Arduino UNO板



2) Arduino单片机

性能：

Digital I/O 数字输入/输出端口 **0—13**。

Analog I/O 模拟输入/输出端口 **0-5**。

支持 **ICSP** 下载，支持 **TX/RX**。

输入电压：**USB** 接口供电或者 **5V-12V** 外部电源供电。

输出电压：支持 **3.3V** 级 **5V DC** 输出。

处理器：使用 **Atmel Atmega168 328** 处理器，因其支持者众多，

已有公司开发出来 **32** 位的 **MCU** 平台支持 **arduino**。

3) Arduino板上几个特殊的端口

VIN 端口： **VIN 是 input voltage** 的缩写，表示有外部电源时的输入端口。

AREF:Reference voltage for the analog inputs(模拟输入的基准电压）。使用 **analogReference()**命令调用。

ICSP：也有称为 **ISP（In System Programmer)**，就是一种线上即时烧录，目前比较新的芯片都支持这种烧录模式，包括大家常听说的 **8051** 系列的芯片，也都慢慢采用这种简便的烧录方式。我们都知道传统的烧录方式，都是将被烧录的芯片，从线路板上拔起，有的焊死在线路板上的芯片，还得先把芯片焊接下来才能烧录。为了解决这种问题，发明了 **ICSP** 线上即时烧录方式。只需要准备一条 **R232** 线（连接烧录器），以及一条连接烧录器与烧录芯片针脚的连接线就可以。电源的**+5V**， **GND**，两条负责传输烧录信息的针脚，再加上一个烧录电压针脚，这样就可以烧录了。

2. Arduino 开发语言

Arduino 使用 C/C++编写程序，虽然 C++兼容 C 语言，但这是两种语言，C 语言是一种面向过程的编程语言，C++是一种面向对象的编程语言。早期的 Arduino 核心库使用 C 语言编写，后来引进了面向对象的思想，目前最新的 Arduino 核心库采用 C 与 C++混合编写而成。

通常我们说的 Arduino 语言，是指 Arduino 核心库文件提供的各种应用程序编程接口（Application Programming Interface，简称 API）的集合。这些 API 是对更底层的单片机支持库进行二次封装所形成的。例如，使用 AVR 单片机的 Arduino 的核心库是对 AVR-Libc（基于 GCC 的 AVR 支持库）的二次封装。

传统开发方式中，需要通过配置多个寄存器来实现相应功能，而在 Arduino 中，繁杂的寄存器被封装成简单的 API，能进行直观控制，增强程序的可读性的同时也提高了开发效率。

Arduino 语言是建立在 C/C++基础上的，其实也就是基础的 C 语言，Arduino 语言只不过把 AVR 单片机（微控制器）相关的一些参数设置都函数化，不用我们去了解他的底层，让我们不了解 AVR 单片机（微控制器）的朋友也能轻松上手。

3. Arduino 编程

(1)常量设置（C 语言部分）

HIGH | LOW 表示数字 IO 口的电平，HIGH 表示高电平（ 1),LOW 表示低电平（ 0)。

INPUT | OUTPUT 表示数字 IO 口的方向， INPUT 表示输入（高阻态）， OUTPUT 表示输出（ AVR 能提供 5V 电压 40mA 电流)。

（2）Arduino 语言

结构:

声明变量及接口名称（ int val;int ledPin=13;)。

setup()——函数在程序开始时使用，可以初始化变量、接口模式、启用库等（例如：pinMode(ledPin,OUTUPT);)。

loop（） ——在 setup()函数之后，即初始化之后， loop() 让你的程序循环地被执行。使用它来运转 Arduino。

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

(3)一些基本函数理解

A. pinMode(接口名称,OUTPUT 或 INPUT)将——接口定义为输入或输出接口，用在 setup()函数里。pinMode(pin, mode)将数位脚位(digital pin)指定为输入或输出。

范例 :

pinMode(7,INPUT); // 将脚位 7 设定为输入模式

B. digitalWrite(接口名称, HIGH 或 LOW)——将数字接口值至高或低。digitalWrite(pin, value)将数位脚位指定为开或关。脚位必须先通过 pinMode 明示为输入或输出模式 digitalWrite 才能生效。
范例 :
digitalWrite(8,HIGH); //将脚位 8 设定输出高电位

C. digitalRead（接口名称） ——读出数字接口的值。int digitalRead(pin)将输入脚位的值读出，当感测到脚位处于高电位时时回传 HIGH，否则回传 LOW。
范例 :
val = digitalRead(7); // 读出脚位 7 的值并指定给 val

D. analogWrite(接口名称, 数值)——给一个接口写入模拟值（ PWM 波）。对于 ATmega168 芯片的 Arduino（包括 Mini 或 BT），该函数可以工作于 3, 5, 6, 9, 10 和 11 号接口。老的 ATmega8 芯片的 USB 和 serial Arduino 仅仅支持 9, 10 和 11 号接口。
analogWrite(pin, value)改变 PWM 脚位的输出电压值，脚位通常会在 3、5、 6、 9、 10 与 11。Value 变数范围 0-255，例如：输出电压 2.5 伏特（ V），该值大约是 128。
范例 :
analogWrite(9,128); // 输出电压约 2.5 伏特（ V）

E. analogRead(接口名称)——从指定的模拟接口读取值， Arduino 对该模拟值进行 10-bit 的数字转换，这个方法将输入的 0-5 电压值转换为 0 到 1023 间的整数值。
int analogRead(pin)读出类比脚位的电压并回传一个 0 到 1023 的数值表示相对应的 0 到 5 的电压值。
范例 :
val = analogRead(0); //读出类比脚位 0 的值并指定给 val 变数

F. delay()——延时一段时间， delay(1000)为一秒。
delay(ms)暂停晶片执行多少毫秒
范例:
delay(500); //暂停半秒（ 500 毫秒）
delay Microseconds(us)暂停晶片执行多少微秒
范例:
delayMicroseconds(1000); //暂停 1 豪秒

G. Serial.begin(波特率)——设置串行每秒传输数据的速率（波特率）。在同计算机通讯时，使用下面这些值：300, 1200, 2400, 4800,9600, 14400, 19200, 28800, 38400, 57600 或 115200。你也可以在任何时候使用其它的值，比如，与 0 号或 1 号插口通信就要求特殊的波特率。用在 setup()函数里

H. Serial.read()——读取持续输入的数据。

I. Serial.print(数据,数据的进制)——从串行端口输出数据。Serial.print(数据)默认为十进制等于 Serial.print(数据， DEC)。
Serial.println(数据，数据的进制)——从串行端口输出数据，跟随一个回车和一个换行符。这个函数所取得的值与 Serial.print()一样

J. unsigned long pulseIn(pin, value)设定读取脚位状态的持续时间，例如使用红外线、加速度感测器测得某一项数值时，在时间单位内不会改变状态。
范例 :
```
time = pulsein(7,HIGH); // 设定脚位 7 的状态在时间单位内保持为 HIGH
```

K. shiftOut(dataPin, clockPin, bitOrder, value)把资料传给用来延伸数位输出的暂存器，函式使用一个脚位表示资料、一个脚位表示时脉。 bitOrder 用来表示位元间移动的方式（ LSBFIRST 最低有效位元或是 MSBFIRST 最高有效位元），最后 value 会以 byte 形式输出。此函式通常使用在延伸数位的输出。
范例 :
```
shiftOut(dataPin, clockPin, LSBFIRST, 255);
```

L. 时间函数控制与计算晶片执行期间的时间 unsigned long millis()回传晶片开始执行到目前的毫秒
范例:
```
duration = millis()-lastTime; // 表示自"lastTime"至当下的时间
```

四、 前进、后退、左转、右转、蜂鸣器等基本功能尝试

1、 前进、后退

A. Run（）函数

前行即给左右电机都传入high信号，这个run函数在这里设置了一个较低的速度，否则在小车避障时会出现不能及时停止的状况。Right_motor_go，Right_motor_back，Left_motor_go，Left_motor_back信号设置为1,0,0,1

```
void run()     // 前进
{
  digitalWrite(Right_motor_go,HIGH);  // 右电机前进
  digitalWrite(Right_motor_back,LOW);
  analogWrite(Right_motor_go,130);//PWM比例0~255调速，左右轮差异略增减
  analogWrite(Right_motor_back,0);
  digitalWrite(Left_motor_go,LOW);  // 左电机前进
  digitalWrite(Left_motor_back,HIGH);
  analogWrite(Left_motor_go,0);//PWM比例0~255调速，左右轮差异略增减
  analogWrite(Left_motor_back,130);
  //delay(time * 100);   //执行时间，可以调整
}
```

B. Break（）函数、Back（）函数

刹车即所有的信号都设为low，Right_motor_go，Right_motor_back，Left_motor_go，Left_motor_back信号设置为0,0,0,0

后退即将 后退的信号设为low，Right_motor_go，Right_motor_back，Left_motor_go，Left_motor_back信号设置为0,1,1,0

```
void brake(int time)  //刹车，停车
{
  digitalWrite(Right_motor_go,LOW);
  digitalWrite(Right_motor_back,LOW);
  digitalWrite(Left_motor_go,LOW);
  digitalWrite(Left_motor_back,LOW);
  delay(time * 100);//执行时间，可以调整
}
void back(int time)          //后退
{
  digitalWrite(Right_motor_go,LOW);  //右轮后退
  digitalWrite(Right_motor_back,HIGH);
  analogWrite(Right_motor_go,0);
  analogWrite(Right_motor_back,150);//PWM比例0~255调速
  digitalWrite(Left_motor_go,HIGH);  //左轮后退
  digitalWrite(Left_motor_back,LOW);
  analogWrite(Left_motor_go,150);
  analogWrite(Left_motor_back,0);//PWM比例0~255调速
  delay(time * 100);     //执行时间，可以调整
}
```

2、 左转、右转

A. 左转

右轮的速度大于左轮才能实现左转，所以左轮的设置有很多种，这里写了了左轮不前进也不后退的左转和左轮向后旋转的左转。

左轮不动，右轮前进时，Right_motor_go，Right_motor_back，Left_motor_go，Left_motor_back信号设置为1,0,0,0

左轮后退，右轮前进时，Right_motor_go，Right_motor_back，Left_motor_go，Left_motor_back信号设置为1,0,1,0

```
void left()         //左转(左轮不动，右轮前进)
{
   digitalWrite(Right_motor_go,HIGH);   // 右电机前进
  digitalWrite(Right_motor_back,LOW);
  analogWrite(Right_motor_go,130);
  analogWrite(Right_motor_back,0);//PWM比例0~255调速
  digitalWrite(Left_motor_go,LOW);   //左轮后退
  digitalWrite(Left_motor_back,LOW);
  analogWrite(Left_motor_go,0);
  analogWrite(Left_motor_back,0);//PWM比例0~255调速
  //delay(time * 100);  //执行时间，可以调整
}

void spin_left(int time)         //左转(左轮后退，右轮前进)
{
   digitalWrite(Right_motor_go,HIGH);   // 右电机前进
  digitalWrite(Right_motor_back,LOW);
  analogWrite(Right_motor_go,130);
  analogWrite(Right_motor_back,0);//PWM比例0~255调速
  digitalWrite(Left_motor_go,HIGH);   //左轮后退
  digitalWrite(Left_motor_back,LOW);
  analogWrite(Left_motor_go,130);
  analogWrite(Left_motor_back,0);//PWM比例0~255调速
  delay(time * 100);    //执行时间，可以调整
}
```

B. 右转

左轮的速度大于右轮才能实现左转，所以右轮的设置也有很多种，这里写了了右轮不前进也不后退的右转和右轮向后旋转的右转。

右轮不动，左轮前进时，Right_motor_go，Right_motor_back，Left_motor_go，Left_motor_back信号设置为0,0,0,1

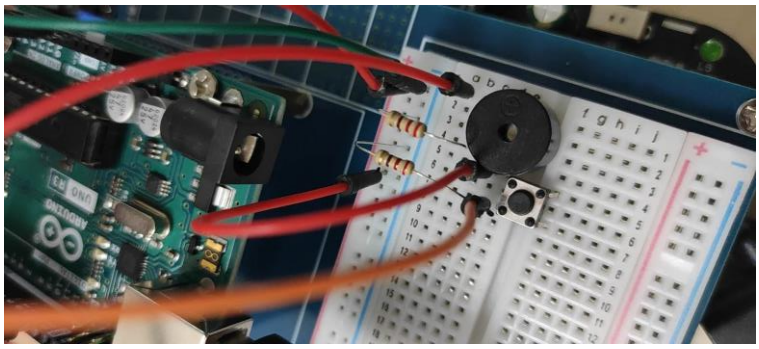
右轮后退，左轮前进时，Right_motor_go，Right_motor_back，Left_motor_go，Left_motor_back信号设置为0,1,0,1

```
void right()        //右转(右轮不动，左轮前进)
{
   digitalWrite(Right_motor_go,LOW);   //右电机后退
  digitalWrite(Right_motor_back,LOW);
  analogWrite(Right_motor_go,0);
  analogWrite(Right_motor_back,0);//PWM比例0~255调速
```

```
  digitalWrite(Left_motor_go,LOW);//左电机前进
  digitalWrite(Left_motor_back,HIGH);
  analogWrite(Left_motor_go,0);
  analogWrite(Left_motor_back,130);//PWM比例0~255调速
  //delay(time * 100);  //执行时间，可以调整
}

void spin_right(int time)        //右转(右轮后退，左轮前进)
{
  digitalWrite(Right_motor_go,LOW);   //右电机后退
  digitalWrite(Right_motor_back,HIGH);
  analogWrite(Right_motor_go,0);
  analogWrite(Right_motor_back,130);//PWM比例0~255调速
  digitalWrite(Left_motor_go,LOW);//左电机前进
  digitalWrite(Left_motor_back,HIGH);
  analogWrite(Left_motor_go,0);
  analogWrite(Left_motor_back,130);//PWM比例0~255调速
  delay(time * 100);     //执行时间，可以调整
}
```

3、 蜂鸣器



```
接受到高电平的时候，蜂鸣器发出声音，低电平时不发出声音
void keysacn()
{
         int val;
     val=digitalRead(key);//读取数字7 ，将电平值赋给val
     while(!digitalRead(key))//当按键没被按下时，一直循环
    {
         val=digitalRead(key);//此句可省略，可让循环跑空
     }
     while(digitalRead(key))//当按键被按下时
     {
        delay(10);  //延时10ms
        val=digitalRead(key);//读取数字7 电平值赋给val
        if(val==HIGH)  //第二次判断按键是否被按下
        {
            digitalWrite(beep,HIGH);         //蜂鸣器响
            while(!digitalRead(key))     //判断按键是否被松开
            digitalWrite(beep,LOW);      //蜂鸣器停止
         }
        else
            digitalWrite(beep,LOW);//蜂鸣器停止
        }
}
```

五、 红外避障实验

1、 实验原理

利用小车的两个红外避障探头，把红外避障探头探测到的信号作为输入，然后根据返回信号的不同来调节小车的行动。如果前方有障碍物，避障指示灯亮，返回信号标记为low，表示低电平；如果前方没有障碍物，避障指示灯不亮，返回信号记为high，表示高电平。

自然地就有，左边有障碍物，左避障指示灯亮，小车向右转；右边有障碍物，右避障指示灯亮，小车向左转；前面有障碍物，左右避障指示灯都亮，小车先停止，然后后退，试探左右转动绕开障碍物。

2、 参数设置

```
int Left_motor_go=8;     //左电机前进(IN1)
int Left_motor_back=9;     //左电机后退(IN2)

int Right_motor_go=10;    // 右电机前进(IN3)
int Right_motor_back=11;    // 右电机后退(IN4)

int key=7;//定义按键 数字7 接口
int beep=12;//定义蜂鸣器 数字12 接口

const int SensorRight = 3;      //右循迹红外传感器(P3.2 OUT1)
const int SensorLeft = 4;       //左循迹红外传感器(P3.3 OUT2)

const int SensorRight_2 = 5;    //右红外传感器(P3.4 OUT3)
const int SensorLeft_2 = 6;     //左红外传感器(P3.5 OUT4)

int SL;    //左循迹红外传感器状态
int SR;    //右循迹红外传感器状态
int SL_2;    //左红外传感器状态
int SR_2;    //右红外传感器状态
```

3、 调用基础函数实现避障

```
void loop()
{
  keysacn();       //调用按键扫描函数
  while(1)
  {
    //有信号为LOW  没有信号为HIGH
    SR_2 = digitalRead(SensorRight_2);
    SL_2 = digitalRead(SensorLeft_2);
    if (SL_2 == HIGH&&SR_2==HIGH)
      run();   //前方没有障碍物时，调用前进函数
    else if (SL_2 == HIGH & SR_2 == LOW)// 右边探测到有障碍物，有信号返回，向左转 ，一定要向左转，否则会陷入死循环
        left();
    else if (SR_2 == HIGH & SL_2 == LOW) //左边探测到有障碍物，有信号返回，向右转
      right();
    else // 都是有障碍物，后退
    {
      back(4.5);//后退
      spin_right(4.5);//有旋转，调整方向
    }
  }
}
```

4、 变形应用——跟随障碍物

跟随障碍物和躲避障碍物恰好是对相同的信号两种完全不同的处理方式，躲避障碍物是返回来低电平则躲避或者停止运动，而躲避障碍物则是遇到低电平继续前行。所以小车的基础行为函数都不用变化，和避障代码完全相同，只在调用时将high和low调换。

```
void loop()
{
  keysacn();        //调用按键扫描函数
  while(1)
  {
    //有信号为LOW  没有信号为HIGH
    SR_2 = digitalRead(SensorRight_2);
    SL_2 = digitalRead(SensorLeft_2);
    if (SL_2 == LOW&&SR_2==LOW)
      run();   //此时前面有障碍物，调用run函数，继续前行
    else if (SL_2 == HIGH & SR_2 == LOW)// 右边探测到有障碍物，向右转，即实现跟随
      right();
    else if (SR_2 == HIGH & SL_2 == LOW) //左边探测到有障碍物，向左转 ，即实现跟随
      left();
    else // 没有障碍物，停
    brake();
  }
}
```

六、 超声波测距实验

1、HC-SR04（超声波）模块介绍

（1）模块介绍

VCC引脚：接+5V。

GND引脚：接GND。

Trig引脚：实际是trigger，有触发引发的意思，到时候只需要给这个引脚一个持续10us的高电平，HC-SR04就会自动地发射8个40KHz的方波（即为超声波）。

Echo引脚：echo实际是是回声回波的意思，当HC-SR04成功的向外发射超声波的时刻开始，这个引脚就会变成高电平，高电平会一直持续到HC-SR04接收到回波为止。

注意：由上面的各引脚的功能我们可知道Trig和Echo必须接在arduino的D口（即数字端口）

（2）本产品特点

A. 感应角度：不大于15 度。
B. 探测距离：2cm-400cm
C. 高精度：可达0.3cm。
D. 盲区（2cm）超近。

2、超声波测距原理

（1）理论解释

超声波发射器向某一方向发射超声波，在发射的同时开始计时，超声波在空气中传播，途中碰到障碍物就立即返回来，超声波接收器收到反射波就立即停止计时。声波在空气中的传播速度为340m/s，根据计时器记录的时间t，就可以计算出发射点距障碍物的距离s，即：s=340m/s×t / 2 。简单的来说就是利用了时间差测距法。

（2）具体处理

A. 单位换算问题

在智能小车的实验中，我们得到的时间，距离得到的单位分别是us,cm，所以要注意单位的换算，换算后的公式为：

S = (1/2) * 340 * (time /10^6) * 100 = time / 58

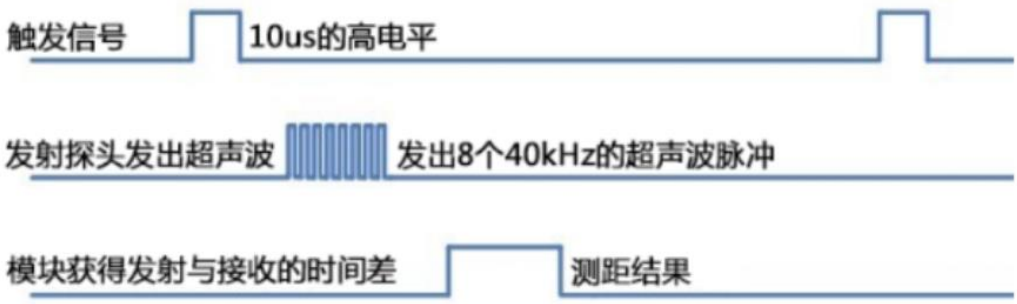
B.对超声波模块返回信号处理问题

在测距函数中调用了函数pulseIn，这个函数的作用是把Echo引脚高电平的持续时间测出，并返回，返回值的单位是微秒us。下面介绍一些这个函数的要点，我们在第一次的文档中已经记录过。

pulseIn函数知识要点：

pulseIn()：用于检测引脚输出的高低电平的脉冲宽度。
pulseIn(pin, value)
pulseIn(pin, value, timeout)
Pin---需要读取脉冲的引脚
Value---需要读取的脉冲类型，HIGH或LOW

Timeout---超时时间，单位微秒，数据类型为无符号长整型。
使用方法及时序图：



- 使用Arduino采用数字引脚给SR04的Trig引脚至少10μs的高电平信号，触发SR04模块测距功能；
- 触发后，模块会自动发送8个40KHz的超声波脉冲，并自动检测是否有信号返回。这步会由模块内部自动完成。
- 如有信号返回，Echo引脚会输出高电平，高电平持续的时间就是超声波从发射到返回的时间。此时，我们能使用pulseIn()函数获取到测距的结果，并计算出距被测物的实际距离。

3、代码解释

Echo，Trig分别为超声波模块的输入输出脚，给超声波模块的输入输出信息定义接口

```
int Echo = A1;  // Echo回声脚(P2.0)
int Trig =A0;  //  Trig 触发脚(P2.1)
```

Distance_test()函数时超声波测距的关键函数，主要利用了前面提到的函数pulseIn。pulseIn函数其实就是一个简单的测量脉冲宽度的函数，默认单位是us。也就是说pulseIn测出来的是超声波从发射到接收所经过的时间。　对除数58的理解是，声音在干燥、摄氏 20度的空气中的传播速度大约为343米/秒，我们作一下单位换算为34,300厘米/秒。所以实际距离就是1厘米，对应58.3微秒。实际上整个测距过程是测的发出声波到收到回波的时间，程序里的时间单位为us。所以换成距离cm，要除以58。当然除以58.3可能更精确。

```
void Distance_test()   // 量出前方距离
{
  digitalWrite(Trig, LOW);   // 给触发脚低电平2μs
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);  // 给触发脚高电平10μs，这里至少是10μs，触发测距功能
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);    // 持续给触发脚低电
  float Fdistance = pulseIn(Echo, HIGH);  // 读取高电平时间(单位：微秒)
  Fdistance= Fdistance/58;
  Serial.print("Distance:");      //输出距离（单位：厘米）
  Serial.println(Fdistance);         //显示距离
  Distance = Fdistance;
}
```

七、 超声波避障实验

1. 超声波避障实验（无舵机）

（1）超声波避障原理

和红外避障原理一致，只是探测障碍物的元件发生了变化。由超声波模块发射超声波，计算前方物体的距离，障碍物距离设置为2cm-30cm之间，如果距离处于这个之间，那么就需要调用后退函数，然后调用左转函数改变运动状态。

但是超声波模块探测障碍物是有局限的，一方面是距离的局限，一方面是时间的限制。就目前的实验来看，超声避障的效果是不如红外避障的效果的。这是因为超声波模块探测的范围很窄，时常还会探测到一些非前行道路上的障碍物造成干扰。

（2）代码解释

接口设置，与红外避障相比，只是把输出输出脚由原来的红外传感器变为了超声波模块

```
void setup()
{
  Serial.begin(9600);     // 初始化串口
  //初始化电机驱动IO为输出方式
  pinMode(Left_motor_go,OUTPUT); // PIN 8 (PWM)
  pinMode(Left_motor_back,OUTPUT); // PIN 9 (PWM)
  pinMode(Right_motor_go,OUTPUT);// PIN 10 (PWM)
  pinMode(Right_motor_back,OUTPUT);// PIN 11 (PWM)
  // pinMode(SensorRight, INPUT); //定义右循迹红外传感器为输入
  // pinMode(SensorLeft, INPUT); //定义左循迹红外传感器为输入
  //pinMode(SensorRight_2, INPUT); //定义右红外传感器为输入
  //pinMode(SensorLeft_2, INPUT); //定义左红外传感器为输入
  //初始化超声波引脚
  pinMode(Echo, INPUT);    // 定义超声波输入脚
  pinMode(Trig, OUTPUT);   // 定义超声波输出脚
  pinMode(key,INPUT);//定义按键接口为输入接口
  pinMode(beep,OUTPUT);

  lcd.begin(16,2);       //初始化1602液晶工作                              模式
  //定义1602液晶显示范围为2行16列字符
}
```

主函数，每个主函数都调用了按键扫描函数，小车的行动是由按键控制的。

```
void loop()
{
  keysacn();        //调用按键扫描函数
  while(1)
  {
    Distance_test();//测量前方距离
    Distance_display();//液晶屏显示距离
    if(Distance < 30)//数值为碰到障碍物的距离，可以按实际情况设置
      while(Distance < 30)//再次判断是否有障碍物，若有则转动方向后，继续判断
      {
        back(10);//后退
        right(1);//右转
        //brake(1);//停车
```

```
        Distance_test();//测量前方距离
        Distance_display();//液晶屏显示距离
      }
```

```
    else
      run();//无障碍物，直行
  }
}
```

2. 超声波避障实验（有舵机）

（1）舵机的作用

加上舵机后解决了超声波探测范围很窄的问题，大大拓宽了超声波的探测范围。

（2）代码解释

对于超声波测量的距离设置了三个变量

```
int Front_Distance = 0;//距前面障碍物的距离
int Left_Distance = 0;//距左边障碍物的距离
int Right_Distance = 0;//距右边障碍物的距离
```

接口设置函数，增加了设定舵机接口为输出接口，pinMode(servopin,OUTPUT);

```
void setup()
{
  Serial.begin(9600);     // 初始化串口
  //初始化电机驱动IO为输出方式
  pinMode(Left_motor_go,OUTPUT); // PIN 8 (PWM)
  pinMode(Left_motor_back,OUTPUT); // PIN 9 (PWM)
  pinMode(Right_motor_go,OUTPUT);// PIN 10 (PWM)
  pinMode(Right_motor_back,OUTPUT);// PIN 11 (PWM)
  pinMode(key,INPUT);//定义按键接口为输入接口
  pinMode(beep,OUTPUT);

  //初始化超声波引脚
  pinMode(Echo, INPUT);    // 定义超声波输入脚
  pinMode(Trig, OUTPUT);   // 定义超声波输出脚
  lcd.begin(16,2);       //初始化1602液晶工作                    模式
  //定义1602液晶显示范围为2行16列字符
  pinMode(servopin,OUTPUT);//设定舵机接口为输出接口
}
```

定义一个脉冲函数，用来模拟方式产生PWM值舵机的范围是0.5MS到2.5MS 1.5MS 占空比是居中周期是20MS

```
void servopulse(int servopin,int myangle)
{
  pulsewidth=(myangle*11)+500;//将角度转化为500-2480 的脉宽值 这里的myangle就是0-180度所以180*11+50=2480  11是为了换成90度的时候基本就是1.5MS
  digitalWrite(servopin,HIGH);//将舵机接口电平置高
   90*11+50=1490uS  就是1.5ms
  delayMicroseconds(pulsewidth);//延时脉宽值的微秒数  这里调用的是微秒延时函数
  digitalWrite(servopin,LOW);//将舵机接口电平置低
 // delay(20-pulsewidth/1000);//延时周期内剩余时间  这里调用的是ms延时函数
  delay(20-(pulsewidth*0.001));//延时周期内剩余时间  这里调用的是ms延时函数
}
```

这是前方，左方，右方的距离探测函数。程序中电脑打印数值部分都被屏蔽了，打印会影响小车遇到障碍物的反应速度，调试时可以打开屏蔽内容Serial.print，打印测到的距离。

```
void front_detection()
{
  //此处循环次数减少，为了增加小车遇到障碍物的反应速度
  for(int i=0;i<=5;i++) //产生PWM个数，等效延时以保证能转到响应角度
  {
    servopulse(servopin,90);//模拟产生PWM
  }
  Front_Distance = Distance_test();
  //Serial.print("Front_Distance:");      //输出距离（单位：厘米）
 // Serial.println(Front_Distance);         //显示距离
 //Distance_display(Front_Distance);
}

void left_detection()
{
  for(int i=0;i<=15;i++) //产生PWM个数，等效延时以保证能转到响应角度
  {
    servopulse(servopin,175);//模拟产生PWM
  }
  Left_Distance = Distance_test();
  //Serial.print("Left_Distance:");      //输出距离（单位：厘米）
  //Serial.println(Left_Distance);         //显示距离
}

void right_detection()
{
  for(int i=0;i<=15;i++) //产生PWM个数，等效延时以保证能转到响应角度
  {
    servopulse(servopin,5);//模拟产生PWM
  }
  Right_Distance = Distance_test();
  //Serial.print("Right_Distance:");      //输出距离（单位：厘米）
  //Serial.println(Right_Distance);         //显示距离
}
```

主函数

```
void loop()
{
  keysacn();        //调用按键扫描函数
  while(1)
  {
    front_detection();//测量前方距离
    if(Front_Distance < 30)//当遇到障碍物时
    {
      back(2);//后退减速
      brake(2);//停下来做测距
      left_detection();//测量左边距障碍物距离
      Distance_display(Left_Distance);//液晶屏显示距离
      right_detection();//测量右边距障碍物距离
      Distance_display(Right_Distance);//液晶屏显示距离
      if((Left_Distance < 30 ) &&( Right_Distance < 30 ))//当左右两侧均有障碍物靠得比
较近
```

```
        spin_left(0.7);//旋转掉头
      else if(Left_Distance > Right_Distance)//左边比右边空旷
      {
        left(3);//左转
        brake(1);//刹车，稳定方向
      }
      else//右边比左边空旷
      {
        right(3);//右转
        brake(1);//刹车，稳定方向
      }
    }
    else
    {
      run(); //无障碍物，直行
    }
  }
}
```