第 3 章　智能小车 C 语言编程

3.1　wiringPi 库的介绍

wiringPi 库是由 Gordon Henderson 所编写并维护的一个用 C 语言写成的类库。起初主要是作为 BCM2835 芯片的 GPIO 库。而现在已经非常丰富，除了 GPIO 库以外，还包括 I2C 库、SPI 库、UART 库和软件 PWM 库等。

由于其与 Arduino 的 wiring 系统较为类似，故以此命名。它是采用 GNU LGPLv3 许可证的，可以在 C 或C++ 上使用，而且在其他编程语言上也有对应的扩展。

wiringPi 库包含一个命令行工具 gpio，它可以用来设置 GPIO 引脚，可以用来读写 GPIO 引脚，甚至可以在 Shell 脚本中使用来达到控制 GPIO 引脚的目的。

wiringPi 库的安装方法可参照 1.3.1 节，下面介绍智能小车 C 语言编程使用的 wiringPi 库中的函数。

1. 初始化函数

```
int wiringPiSetup(void);
```

该函数初始化 wiringPi，程序将使用如图 3-1 所示的引脚定义图，具体引脚映射可通过 gpio readall 命令进行查看。程序必须调用初始化函数，否则不能正常工作。树莓派所有的 GPIO 引脚使用 wiringPi 编号。

2. pinMode 函数

```
void pinMode(int pin, int mode);
```

使用该函数可以将某个引脚设置为 INPUT(输入)、OUTPUT(输出)、PWM_OUTPUT(脉冲输出)或者 GPIO_CLOCK(GPIO 时钟)。需要注意的是，仅有引脚 1 支持 PWM_OUTPUT 模式，仅有引脚 7 支持 CLOCK 输出模式。

3. digitalWrite 函数

```
void digitalWrite(int pin, int value);
```

使用该函数可以向指定的引脚写入 HIGH(高)或者 LOW(低)，写入前，需要将

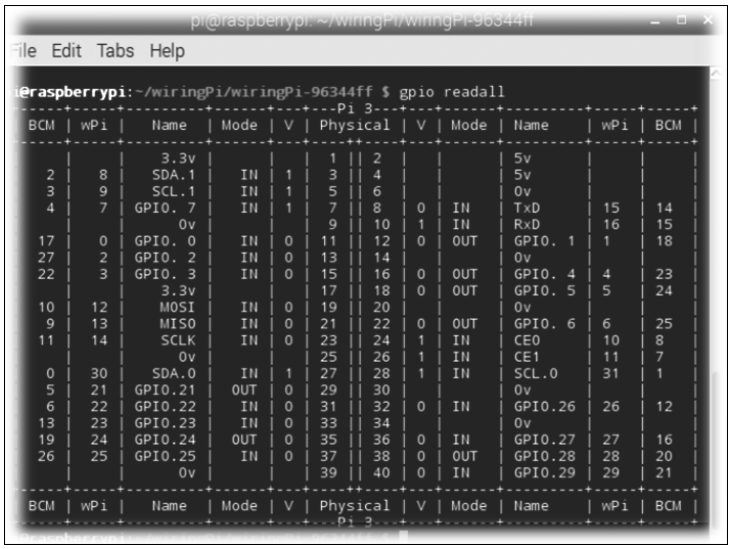

图 3-1　树莓派引脚定义图

引脚设置为输出模式。wiringPi 将任何的非 0 值作为 HIGH(高)来对待,因此,0 是唯一能够代表 LOW(低)的数值。

4. pwmWrite 函数

```
void pwmWrite(int pin, int value);
```

使用该函数可以将值写入指定引脚的 PWM 寄存器中。树莓派板上仅有一个 PWM 引脚,即引脚 1。可设置的值为 0～1024。

5. digitalRead 函数

```
void digitalRead(int pin);
```

使用该函数可以读取指定引脚的值,读取到的值为 HIGH(1)或者 LOW(0),该值取决于该引脚的逻辑电平的高低。

6. softPwmCreate 函数

```
int softPwmCreate(int pin,int initialValue,int pwmRange);
```

该函数将会创建一个软件控制的 PWM 引脚。可以使用任何一个 GPIO 引脚,pwmRange 参数可以为 0(关)～100(全开)。返回值为 0,代表成功,其他值代表失败。

7. softPwmWrite 函数

```
void softPwmWrite(int pin, int value);
```

该函数将会更新指定引脚的 PWM 值。value 参数的范围将会被检查,如果指定

的引脚之前没有通过 softPwmCreate 初始化，将会被忽略。

8. delay 函数

```
void delay(unsigned int howLong);
```

该函数将会中断程序执行至少 howLong 毫秒。因为 Linux 是多任务的，中断时间可能会更长。需要注意的是，最长的延迟值是一个无符号 32 位整数，其大约为 49 天。

9. delayMicroseconds 函数

```
void delayMicroseconds(unsigned int howLong);
```

该函数将会中断程序执行至少 howLong 微秒。因为 Linux 是一个多任务的系统，因此中断时间可能会更长。需要注意的是，最长的延迟值是一个无符号 32 位整数，其大约为 71 分钟。如果延迟低于 100 微秒，将会使用硬件循环来实现；如果超过 100 微秒，将会使用系统 nanosleep()函数来实现。

3.2 智能小车移动控制

3.2.1 固定速度移动控制

C 语言控制智能小车移动使用 pinMode 和 digitalWrite 函数来实现。

1. void pinMode(int pin, int mode)

智能小车电机上的 IN1～IN4 四个引脚分别与树莓派的 wiringPi 编号为 1，4，5，6 这 4 个 GPIO 引脚连接，函数中的 pin 参数对应如表 3-1 所示的 wiringPi 编号；由于树莓派通过 GPIO 引脚向电机引脚输出高电平控制智能小车移动，因此 mode 参数应设置为 OUTPUT。

表 3-1 电机引脚编号

电机引脚	wiringPi 编号	移动方式
IN1	1	左轮前进
IN2	4	左轮后退
IN3	5	右轮前进
IN4	6	右轮后退

2. void digitalWrite(int pin, int value)

树莓派通过 GPIO 引脚向电机引脚输出高电平控制智能小车移动，value 参数为 1 时表示树莓派引脚向电机引脚高电平；为 0 时表示输出低电平。控制智能小车左

转的完整程序如下。

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    wiringPiSetup();
    /*WiringPi GPIO*/
    pinMode (1, OUTPUT);      //IN1
    pinMode (4, OUTPUT);      //IN2
    pinMode (5, OUTPUT);      //IN3
    pinMode (6, OUTPUT);      //IN4
    while(1)
    {
        digitalWrite(1,0);
        digitalWrite(4,0);
        digitalWrite(5,1);          //右轮前进
        digitalWrite(6,0);
    }
    return 0;
}
```

首先添加 wiringPi.h 头文件，使用 wiringPiSetup 函数进行初始化，通过 pinMode 函数将树莓派 wiringPi 编号为 1,4,5,6 的引脚设置为输出模式，最后使用 digitalWrite 函数通过树莓派 5 号引脚向电机的 IN3 引脚输出高电平，从而控制智能小车右轮前进，实现左转。

3.2.2　可变速度移动控制

使用上述实现方式智能小车只能以最大速度移动，为了能够灵活控制智能小车的移动速度需要使用 PWM 控制方式，但是树莓派硬件上支持的 PWM 输出的引脚有限(只有 1 号引脚支持 PWM 输出)。为了突破这个限制，wiringPi 提供了软件实现的 PWM 输出 API。基于 PWM 的智能小车移动控制使用 softPwmCreate 和 softPwmWrite 函数。

1. int softPwmCreate(int pin, int initialValue, int pwmRange)

其中，pin 用来作为软件 PWM 输出的引脚；initalValue 为引脚输出的初始值；pwmRange 为 PWM 值的范围上限。

2. void softPwmWrite(int pin, int value)

其中，value 为 PWM 引脚输出的值，电机的实际速度为：value/pwmRange×最

大速度。

基于 PWM 的控制智能小车左转的完整程序如下。

```
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    wiringPiSetup();
    /* WiringPi GPIO */
    pinMode (1, OUTPUT);      //IN1
    pinMode (4, OUTPUT);      //IN2
    pinMode (5, OUTPUT);      //IN3
    pinMode (6, OUTPUT);      //IN4
    softPwmCreate(1,1,500);
    softPwmCreate(4,1,500);
    softPwmCreate(5,1,500);
    softPwmCreate(6,1,500);
    while(1)
    {
        softPwmWrite(1,0);
        softPwmWrite(4,0);
        softPwmWrite(5,250);          //右轮前进
        softPwmWrite(6,0);
    }
    return 0;
}
```

首先使用 PWM 需要包含头文件 softPwm.h,初始化和引脚模式设置以后,使用 softPwmCreate 函数将 1,4,5,6 四个引脚设置为 PWM 模式,最后通过 softPwmWrite 函数在四个引脚设置输出值,智能小车将以半速(250/500×最大速度)左转。

3.2.3 程序的编译和运行

将控制智能小车左转的程序保存为 left.c,如图 3-2 所示,打开树莓派终端,输入:

```
gcc left.c -o left -lwiringPi -lpthread
sudo ./left
```

GCC(GNU Compiler Collection,GNU 编译器套件)是由 GNU 开发的编程语言编译器。GCC 原本作为 GNU 操作系统的官方编译器,现已被大多数类 UNIX 操作

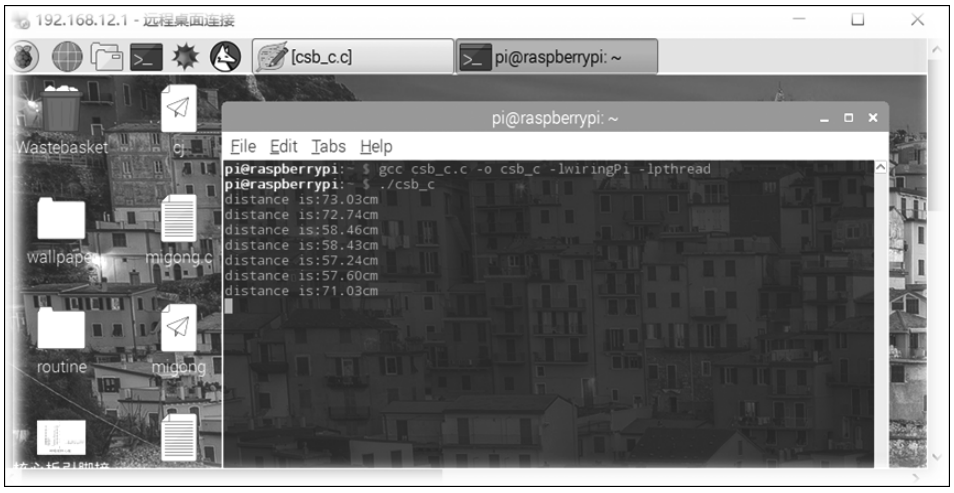

图 3-2　程序的编译和执行

系统(如 Linux、BSD、Mac OS X 等)采纳为标准的编译器,GCC 同样适用于微软的 Windows。GCC 和 g++ 分别是 GNU 的 C & C++ 编译器。GCC/g++ 在执行编译的时候一般有以下 4 步。

(1) 预处理:生成.i 的文件,进行宏的替换、注释的消除和找到相关的库文件,并将 #include 文件的全部内容插入。

(2) 转换成汇编语言:生成.s 文件,.s 文件表示汇编文件,用编辑器打开是汇编指令。

(3) 汇编文件转变为目标代码(机器代码):生成.o 的文件,.o 是 GCC 生成的目标文件,用编辑器打开是二进制机器码。

(4) 连接目标代码:生成可执行程序。

使用 gcc left.c 将编译出一个名为 a.out 的可执行文件;使用"-o"可以指定编译程序的名字,gcc left.c -o left 将编译出一个名为 left 的可执行文件。

Linux 平台下存在着大量库,本质上来说库是一种可执行代码的二进制形式,可以被操作系统载入内存执行。通俗地说就是把常用函数的目标文件打包在一起,提供相应函数的接口,便于程序员使用。按照库的使用方式又可分为动态库和静态库。

(1) 静态库后缀为".a",类似 Windows 平台的.lib 文件,静态库可以简单地看成一组目标文件(.o 文件)的集合,即很多目标文件经过压缩打包后形成的文件。比如在日常编程中,如果需要使用 printf 函数,就需要 stdio.h 的库文件,可是如果直接把对应函数源码编译后形成的.o 文件提供给我们,将会对我们的管理和使用造成极大不便,于是可以使用"ar"压缩程序将这些目标文件压缩在一起,形成"libx.a"静态库文件。其中,"x"为库名。静态库在程序编译时会被连接到目标代码中,相当于将库中的函数加载到程序里,在编译的时候直接编译进去,这样在编译之后执行程序时将

不再需要该静态库。编译之后程序文件大，但加载快，隔离性也好。所以它的优点就显而易见了，即编译后的执行程序不需要外部的函数库支持，因为所有使用的函数都已经被编译进去了。当然，这也会成为它的缺点，因为如果静态函数库改变了，那么程序必须重新编译。

(2) 动态库后缀为".so"，类似 Windows 平台的.dll 文件。动态库在程序编译时并不会被连接到目标代码中，而是仅在编译时引用，体积小，在程序运行到相关函数时才载入函数库里的相应函数，因此在程序运行时还需要动态库的存在。多个应用程序可以使用同一个动态库，启动多个应用程序的时候，只需要将动态库加载到内存一次即可。GCC/g++ 在编译时默认使用动态库。

(3) -l 参数用来指定程序要链接的库，-l 参数紧接着就是库名，那么库名跟真正的库文件名有什么关系呢？拿数学库来说，库名是 m，则动态库文件名是"libm.so"，即把库文件名的头 lib 和尾.so 去掉就是库名了。当使用动态库 libwiringPi.so 时，需要把"libwiringPi.so"复制到/usr/lib 或者/lib 里，编译时加上"-lwringPi"参数即可；同时，使用 wiringPi 库中的函数也需要配套的头文件 wiringPi.h。由于使用到了 PWM 函数，因此需要使用"-lpthread"链接 libthread.so 动态库。

可以在树莓派终端输入 find -name "*libwiringPi*"，验证是否存在 libwiringPi.so 动态链接库；输入 find -name "*libpthread*"，验证是否存在 libthread.so 动态链接库。

3.3 超声波传感器的使用

3.3.1 传感器的连接

如图 3-3 所示，超声波传感器包含 Vcc、Gnd、Trig 和 Echo 四个引脚，其中，Vcc 和 Gnd 引脚对传感器供电，Trig 和 Echo 引脚分别与树莓派的 GPIO 引脚连接。如图 3-4 所示，智能小车前方的超声波传感器的 Trig 引脚与树莓派物理编号为 38 的引脚连接，Echo 引脚与树莓派物理编号为 40 的引脚连接；左方超声波传感器的 Trig 引脚与树莓派的物理编号为 35 的引脚连接，Echo 引脚与树莓派物理编号为 37 的引脚连接；右方超声波传感器的 Trig 引脚与树莓派物理编号为 29 的引脚连接，Echo 引脚与树莓派物理编号为 31 的引脚连接。树莓派 GPIO 引脚的物理编号和 wiringPi 编号的对应关系如表 3-2 所示。

图 3-3 超声波传感器

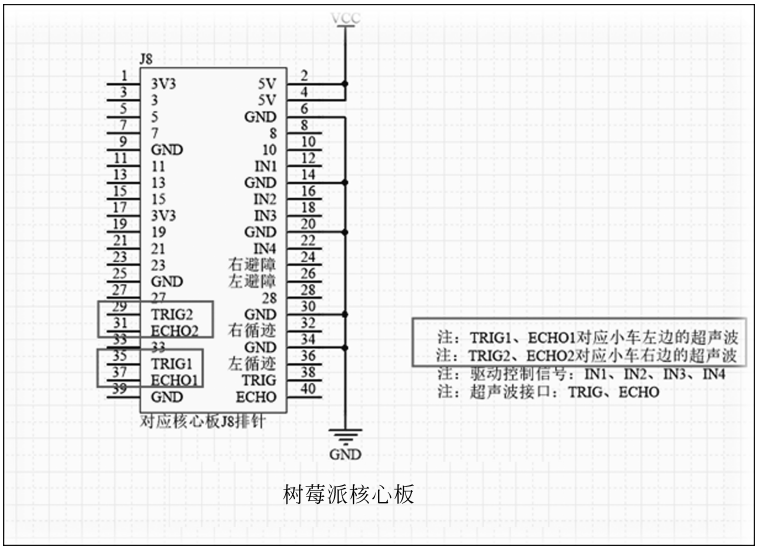

图 3-4　树莓派接线图

表 3-2　编号对应关系

传感器	wiringPi 编号	GPIO 物理编号
红外开关＃1(左)	11	26
红外开关＃2(右)	10	24
超声波传感器(前)	Trig：28， Echo：29	Trig：38， Echo：40
超声波传感器(左)	Trig：24， Echo：25	Trig：35， Echo：37
超声波传感器(右)	Trig：21， Echo：22	Trig：29， Echo：31
循迹传感器＃1(左)	27	36
循迹传感器＃2(右)	26	32
电机(左)	1 4	12 16
电机(右)	5 6	18 22

3.3.2　工作原理

如图 3-5 所示，树莓派向传感器的 Trig 引脚输出一个 10μs 以上的高电平，超声波传感器模块内部自动向外发送 8 个 40kHz 的方波，然后树莓派等待传感器 Echo

引脚的高电平输出，一旦有高电平输出则记录一次时间，当变为低电平时再次记录一次时间，记录的时间差即为超声波往返的时间。根据往返时间可以计算出与障碍物之间的距离。

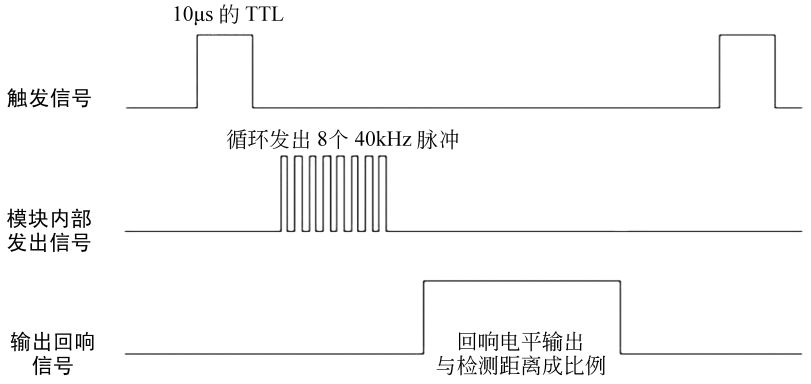

图 3-5　超声波时序图

3.3.3　程序实现

右方超声波传感器测距的程序代码如下。

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#define Trig 21
#define Echo 22
void ultraInit()
{
    pinMode(Echo, INPUT);          //Echo 为与传感器 Echo 引脚连接的树莓派的
                                   //GPIO 引脚
    pinMode(Trig, OUTPUT);         //Trig 为与传感器 Trig 引脚连接的树莓派的
                                   //GPIO 引脚
}
float disMeasure()
{
    struct timeval tv1;
    struct timeval tv2;
    long start, stop;
    float dis;
    digitalWrite(Trig, LOW);
    delayMicroseconds(2);
    digitalWrite(Trig, HIGH);      //树莓派向传感器 Trig 引脚输出高电平
```

```
    delayMicroseconds(10);                          //保持 10μs
    digitalWrite(Trig, LOW);
    while(!(digitalRead(Echo)==1));                 //在树莓派等待传感器 Echo
                                                    //引脚的高电平输出
    gettimeofday(&tv1, NULL);                       //获取当前时间
    while(!(digitalRead(Echo)==0));                 //等待高电平结束
    gettimeofday(&tv2, NULL);                       //获取当前时间
    start=tv1.tv_sec*1000000+tv1.tv_usec;           //微秒级的时间
    stop=tv2.tv_sec*1000000+tv2.tv_usec;
    dis=(float)(stop-start)/1000000*34000/2;        //求出距离,单位为厘米
    return dis;
}
int main()
{

    float dis;
    wiringPiSetup();
    ultraInit();
    while(1)
    {
        dis=disMeasure();
        printf("distance is:% 0.2fcm\n",dis);
        delay(1000);                                //每隔 1s 输出一次
    }
    return 0;
}
```

程序中使用到了 timeval 结构体。

```
struct timeval
{
    long    tv_sec;         /* seconds */
    long    tv_usec;        /* microseconds */
};
```

其中,tv_sec 记录秒,tv_usec 记录微秒,直接使用 gettimeofday 即可获取时间。

3.4　红外和循迹传感器的使用

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#define LEFT 11              //左红外
#define RIGHT 10             //右红外
int main()
{
    wiringPiSetup();
    int SR,SL;
    while(1)
    {
        SR=digitalRead(RIGHT);
        printf("Rightsensor:%d\n",SR);
        SL=digitalRead(LEFT);
        printf("Leftsensor:%d\n",SL);
    }
    return 0;
}
```

智能小车左侧红外传感器与树莓派的 wiringPi 编号为 11 的 GPIO 引脚连接，右侧的红外传感器与树莓派 wiringPi 编号为 10 的引脚连接。左侧循迹传感器与树莓派的 wiringPi 编号为 27 的引脚连接，右侧循迹传感器与树莓派的 wiringPi 编号为 26 的引脚连接。使用 digitalRead 函数获取传感器的状态。

3.5 应 用 案 例

3.5.1 基于超声波传感器的迷宫导航

```
#include <stdio.h>
#include <stdlib.h>
#include <softPwm.h>
#include <time.h>
#include <wiringPi.h>
#define Trig  28             //前方超声波传感器
#define Echo  29
#define BUFSIZE 512
void ultraInit(void)
{
    pinMode(Echo, INPUT);
    pinMode(Trig, OUTPUT);
}
float disMeasure(void)
{
```

```
    struct timeval tv1;
    struct timeval tv2;
    long start, stop;
    float dis;
    digitalWrite(Trig, LOW);
    delayMicroseconds(2);
    digitalWrite(Trig, HIGH);              //树莓派向传感器 Trig 引脚输出高电平
    delayMicroseconds(10);                 //保持 10μs
    digitalWrite(Trig, LOW);
    while(!(digitalRead(Echo)==1));        //在树莓派等待传感器 Echo 引脚的高电
                                           //平输出
    gettimeofday(&tv1, NULL);              //获取当前时间
    while(!(digitalRead(Echo)==0));        //等待高电平结束
    gettimeofday(&tv2, NULL);              //获取当前时间
    start=tv1.tv_sec * 1000000+tv1.tv_usec;     //微秒级的时间
    stop  =tv2.tv_sec * 1000000+tv2.tv_usec;
    dis=(float)(stop-start)/1000000 * 34000/2; //求出距离,单位为厘米
    return dis;
}
void run()                                 //前进
{
    softPwmWrite(4,0);
    softPwmWrite(1,250);                   //左轮前进
    softPwmWrite(6,0);
    softPwmWrite(5,250);                   //右轮前进
}

void brake(int time)                       //停车
{
    softPwmWrite(1,0);
    softPwmWrite(4,0);
    softPwmWrite(5,0);
    softPwmWrite(6,0);
    delay(time * 100);                     //执行时间,可以调整
}
void left(int time)                        //左转(左轮不动,右轮前进)
{
    softPwmWrite(1,0);
    softPwmWrite(4,0);
    softPwmWrite(5,250);                   //右轮前进
    softPwmWrite(6,0);
```

```
    delay(time * 300);
}
void right(int time)                                    //右转(右轮不动,左轮前进)
{
    softPwmWrite(1,250);                                //左轮前进
    softPwmWrite(4,0);
    softPwmWrite(5,0);
    softPwmWrite(6,0);
    delay(time * 300);                                  //执行时间,可以调整
}
void back(int time)                                     //后退
{
    softPwmWrite(4,250);                                //左轮向后移动
    softPwmWrite(1,0);
    softPwmWrite(6,250);                                //右轮向后移动
    softPwmWrite(5,0);
    delay(time * 200);                                  //执行时间,可以调整
}
int main(int argc, char * argv[])
{
    float dis;
    wiringPiSetup();
    /* WiringPi GPIO */
    pinMode (1, OUTPUT);        //IN1
    pinMode (4, OUTPUT);        //IN2
    pinMode (5, OUTPUT);        //IN3
    pinMode (6, OUTPUT);        //IN4
    softPwmCreate(1,1,500);
    softPwmCreate(4,1,500);
    softPwmCreate(5,1,500);
    softPwmCreate(6,1,500);
    while(1)
    {
        dis=disMeasure();
        printf("distance=%0.2f cm\n",dis);      //输出当前超声波测得的距离
        if(dis<30)
        {
            //测得前方障碍的距离小于 30cm 时做出如下响应
            back(4);                                    //后退 800 ms
            left(4);                                    //左转 1200 ms
        }
```

```
        else
        {
            run();                                    //无障碍时前进
        }
    }
    return 0;
}
```

本案例只使用了前方的超声波传感器，当检测与前方障碍物距离小于 30cm 时，先后退 800ms，然后向左旋转 1200ms，否则前进。读者可以进一步完善迷宫导航算法。

3.5.2　基于红外传感器的迷宫导航

```
#include <stdio.h>
#include <stdlib.h>
#include <softPwm.h>
#include <time.h>
#include <wiringPi.h>
#define LEFT 11                               //左红外
#define RIGHT 10                              //右红外
void run()                                    //前进
{
    softPwmWrite(4,0);
    softPwmWrite(1,250);                      //左轮前进
    softPwmWrite(6,0);
    softPwmWrite(5,250);                      //右轮前进
}

void brake(int time)                          //停车
{
    softPwmWrite(1,0);
    softPwmWrite(4,0);
    softPwmWrite(5,0);
    softPwmWrite(6,0);
    delay(time * 100);                        //执行时间,可以调整
}

void left()                                   //左转
{
    softPwmWrite(4,250);                      //左轮后退
    softPwmWrite(1,0);
```

```
    softPwmWrite(6,0);
    softPwmWrite(5,250);                //右轮前进
}
void right()                            //右转
{
    softPwmWrite(4,0);
    softPwmWrite(1,250);                //左轮前进
    softPwmWrite(6,250);                //右轮后退
    softPwmWrite(5,0);

}
void back()                             //后退
{
    softPwmWrite(4,250);                //左轮后退
    softPwmWrite(1,0);
    softPwmWrite(6,250);                //右轮后退
    softPwmWrite(5,0);
}
int main(int argc, char * argv[])
{
    float dis
    int SR;
    int SL;
    wiringPiSetup();
    /*WiringPi GPIO*/
    pinMode (1, OUTPUT);      //IN1
    pinMode (4, OUTPUT);      //IN2
    pinMode (5, OUTPUT);      //IN3
    pinMode (6, OUTPUT);      //IN4
    softPwmCreate(1,1,500);
    softPwmCreate(4,1,500);
    softPwmCreate(5,1,500);
    softPwmCreate(6,1,500);
    while(1)
    {
        //有障碍物为 LOW,没有障碍物为 HIGH
        SR=digitalRead(RIGHT);
        SL=digitalRead(LEFT);
        if (SL==LOW&&SR==LOW)           //前方有障碍物
        {
            printf("BACK");             //前面有物体时小车后退 300ms 再转弯
```

```
            back();
            delay(300);                    //后退 300ms
            left();                        //左转 600ms
            delay(600);
        }
        else if (SL==HIGH&&SR==LOW)        //右方有障碍物
        {
            printf("TURNING LEFT");
            left();
            delay(300);
        }
        else if (SR==HIGH&&SL==LOW)        //左方有障碍物
        {
            printf("TURNING RIGHT");
            right ();
            delay(300);
        }
        else                               //前面没有障碍物则前进
        {
            printf("GO");
            run();
        }
    }
    return 0;
}
```

本案例只使用了智能小车前方的两个红外避障传感器，经查阅表 3-2 可知，智能小车左侧红外传感器的 wiringPi 编号为 11，右侧红外传感器的编号为 10。当红外传感器检测到障碍物时，输出为 LOW，否则为 HIGH。读者可以进一步完善迷宫导航算法。

3.5.3　二路循迹的实现

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <wiringPi.h>
#include <softPwm.h>
#define LEFT  27                   //左循迹
#define RIGHT  26                  //右循迹
void run()                         //前进
```

```
{
    softPwmWrite(4,0);
    softPwmWrite(1,250);
    softPwmWrite(6,0);
    softPwmWrite(5,250);
}
void brake()                    //停车
{
    softPwmWrite(1,0);
    softPwmWrite(4,0);
    softPwmWrite(5,0);
    softPwmWrite(6,0);
}
void left()                     //左转
{
    softPwmWrite(4,250);
    softPwmWrite(1,0);
    softPwmWrite(6,0);
    softPwmWrite(5,250);
}
void right()                    //右转
{
    softPwmWrite(4,0);
    softPwmWrite(1,250);
    softPwmWrite(6,250);
    softPwmWrite(5,0);
}
void back()                     //后退
{
    softPwmWrite(1,250);
    softPwmWrite(4,0);
    softPwmWrite(5,250);
    softPwmWrite(6,0);

}
int main(int argc, char * argv[])
{
    float dis;
    int SR;
    int SL;
    wiringPiSetup();
```

```
    pinMode (1, OUTPUT);      //IN1
    pinMode (4, OUTPUT);      //IN2
    pinMode (5, OUTPUT);      //IN3
    pinMode (6, OUTPUT);      //IN4
    softPwmCreate(1,1,500);
    softPwmCreate(4,1,500);
    softPwmCreate(5,1,500);
    softPwmCreate(6,1,500);
    while(1)
    {
        SR=digitalRead(RIGHT);      //SR为LOW时表明在白色区域,否则表明压在
                                    //黑线上
        SL=digitalRead(LEFT);       //SL为LOW时表明在白色区域,否则表明压在
                                    //黑线上
        if (SL==HIGH&&SR==HIGH)     //在黑线上
        {
            printf("GO");
            run();
        }
        else if (SL==HIGH&&SR==LOW) //左侧在黑线,右侧在白色区域,左转调整
        {
            printf("LEFT");
            left();
        }
        else if (SR==HIGH&&SL==LOW) //右侧在黑线,左侧在白色区域,右转调整
        {
            printf("RIGHT");
            right();
        }
        else                        //都是白色,停止
        {
            printf("STOP");
            brake();
        }
    }

    return 0;
}
```

二路循迹利用位于智能小车前方底部的两个循迹传感器使智能小车沿着如图3-6所示的黑色赛道前进,经查阅表3-2可知,左侧的循迹传感器wiringPi引脚号为

27，右侧循迹传感器引脚号为 26，当引脚状态为 HIGH 时表明循迹传感器下方为黑色地面，否则为白色地面。

图 3-6　循迹赛道