目　录

1.实验内容与要求

1.1 实验内容

(1)掌握 TI-RSLK 学习套件的硬件构成及工作原理,完成小车的硬件搭建;

(2)掌握基于 TI 的 Cortex M4 嵌入式系统相关知识,以 MSP432P401R 作为小车的微控制器;

(3)设计电机驱动电路，实现电机变速、启动、反转及停止;

(4)采用红外循迹传感器识别黑色轨道线，实现小车的智能循迹;

(5)采用红外测距传感器检测障碍物，实现小车的避障（避障功能）;

(6)设计防撞开关，实现对小车的保护以及防碰撞（碰撞功能）;

(7)设计控制小车行走的程序，使小车可以在随机搭建的轨道上智能循迹自动前进，且实现防碰撞避障功能。

1.2 实验要求

（1）检测开关性能和 LED 灯的性能:

a）点亮 LED 灯，可以是跑马灯的形式闪烁;

b）能够用两个开关分别控制 LED 灯，控制点亮的时间间隔或灯的颜色。

（2）测试轮子转动的功能，小车可以开动，最好速度方向可以由开关调节.

（3）测试循线功能、避障功能和碰撞转向功能，全部完成并通过迷宫测试。

2. 实验原理

2.1 检测开关与 LED 跑马灯

2.1.1 硬件原理

利用功能选择寄存器 PxSEL0 和 PxSEL1 来选择引脚的功能：I/O 功能或者 3 个可能的外设模块功能中的一个，具体配置方法如下表所示：

PxSEL1	PxSEL0	引脚功能
0	0	选中通用 I/O 功能
0	1	选中主外设模块功能
1	0	选中第二外设模块功能
1	1	选中第三外设模块功能

方向寄存器 PxDIR 中的每一位都将反映相应 I/O 引脚的方向，位 0 表示引脚为输入方向，位 1 表示引脚为输出方向。

PxREN 寄存器中的每一位都用于使能或者禁用相应 I/O 引脚的上拉或下拉电阻。位 0 表示禁用上拉/下拉电阻；位 1 表示使能上拉/下拉电阻。

输出寄存器 PxOUT 当引脚的功能配置为 I/O 端口且方向为输出时，PxOUT 寄存器中的每一位将对应相应引脚的输出值，位 0 表示输出为低，位 1 表示输出为高；若将引脚功能配置为 I/O 端口且方向为输入，在使能上拉或下拉电阻时，则 PxOUT 寄存器中的相应位用于选择上拉或下拉电阻。位 0 表示下拉；位 1 表示上拉。

综上，开关和 LED 显示的 I/O 初始化程序如下：

```
1.  void IniIO(void){
2.  P1->SEL0 &= ~0x13;
3.  P1->SEL1 &= ~0x13;
4.  P2->SEL0 &= ~0x07;
5.  P2->SEL1 &= ~0x07;
6.  P1->DIR &= ~0x12;
7.  P1->DIR |= 0x01;
8.  P2->DIR |= 0x07;
9.  P1->REN |= 0x12;
10. P1->OUT |= 0x12;
11. P1->OUT &= ~0x01;
12. P2->OUT &= ~0x07;
13. P2->DS |= 0x07;
14. }
```

2.1.2 软件原理

通过开关控制流水灯闪烁的时间,两个开关分别连接P1.1，P1.4管脚，开关摁下，输入低电平。通过读取P1口的状态，读取开关状态。而流水灯的颜色通过 REGLED 灯的 BLUE、GREEN、RED 三个组合产生不同的颜色,其中 BLUE、GREEN、RED 灯分别连接在 P2.2、P2.1、P2.0 上，管脚设置为输出，当管脚输出电平为低，对应的 LED 灯熄灭，当管脚电平为高，对应 LED 灯点亮，从而实现灯的颜色的变化。LED 原理图如下所示：

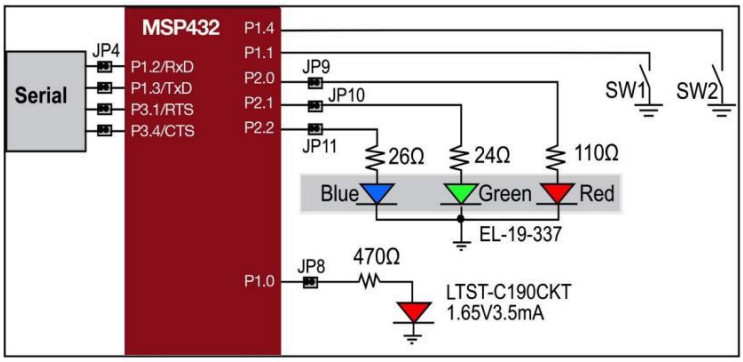


图 1：LED 原理图

2.2 检测轮子转动与速度调节

2.2.1 硬件原理

编码减速电机组所使用是减速直流电机，与之相匹配的是 TI 的 DRV8838 驱动器的 H 桥电机驱动器，连接到 CPU 的 P1.6、P2.6、P3.6、P1.7、P2.7、P3.7 的六个 GPIO 口，该驱动器允许向前或向后转动每个电机。驱动器通过 MSP432P401R 产生的 PWM 波控制电机的状态，实现小车的启动、左右转弯、前进后退、掉头、停止等功能。

DRV8838 可以驱动一个直流电机或其他诸如螺线管的器件。输出驱动器块由一个配置为 H 桥的 N 通道功率 MOSFE 组成，以驱动电机绕组。一个内部电荷泵生成所需的栅极驱动电压，其接口如下图所示：

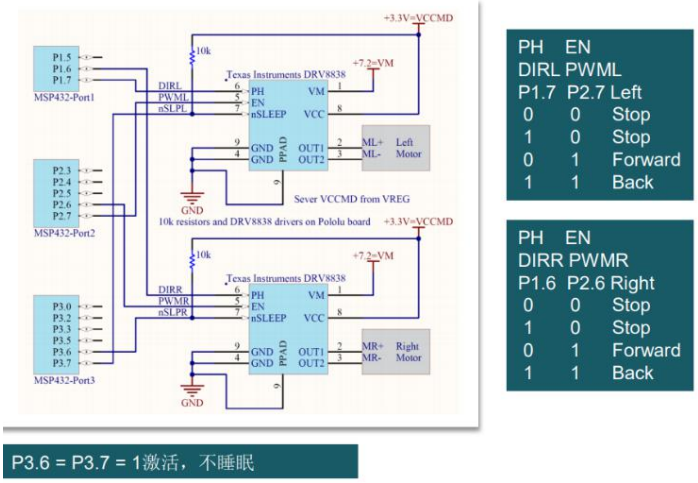


图 2：电机驱动模块硬件连接图

2.2.2 软件原理

实验 2 中主要是产生 PWM 波，PWM 波由电机输出，脉冲宽度调制指按照一定的规律改变脉冲序列的脉冲宽度（占空比值），调节输出量和波形的一种调制方式，常用来输出矩形 PWM 信号。在本控制系统中 PWM 用于电机转速的控制，通过改变 MSP432 控制程序中的 GPIO 口 P2.6、P2.7 输出的 PWM 的占空比就可以调节两轮的转速。在一个周期内，增大占空比，就会增大高电平的比例，输出在电机上的电压会随之增大，电机转速也随之加快。

MSP432 的控制程序一般是通过自有的定时器 A（Timer A）来输出 PWM。Timer_A 为 16 位定时器，具有 7 个捕获比较器。Timer_A 支持多路捕获/比较、PWM 输出和定时计数。Timer_A 也具有丰富的中断能力，当定时时间到或满足捕获/比较条件时，将可触发 Timer_A 中断。而在本控制系统中只需要两组不同的 PWM 值来控制左右两轮的转速从而控制小车的运动状态。

通过查阅 TI-RSLK 操作手册，控制电机运动状态的共有 6 个 GPIO 输出口：P1.6、P2.6、P3.6、P1.7、P2.7、P3.7。

LaunchPad	MDPDB	DRV8838	描述
P1.6	DIRR	PH	Right Motor Direction
P3.6	nSLPR	nSLEEP	Right Motor Sleep
P2.6	PWMR	EN	Right Motor PWM
P1.7	DIRL	PH	Left Motor Direction
P3.7	nSLPL	nSLEEP	Left Motor Sleep
P2.7	PWML	EN	Left Motor PWM

图三：GPIO 口与对应功能描述

对于电机运动可以分为若干种情况。通过 Motor.c 中的五个函数：

void Motor_Forward

void Motor_Stop

void Motor_Right

void Motor_Left

void Motor_Backward

控制小车前进、停止、右转、左转、后退，同时可以通过改变左右电机输出波形占空比的大小来控制转弯的速度（幅度）。

2.3 检测碰撞转向功能

2.3.1 硬件原理

为了保护小车，套件中包含了 6 个防撞开关，防撞开关从左到右分别连接 CPU 的 P4.7、P4.6、P4.5、P4.4、P4.3、P4.2 的六个 GPIO 口。防撞开关的作用除了可以在碰撞时起缓冲作用，还可以通过对 MSP432P401R 返回信息，使 MSP432P401R 做出相应的响应，修正运动线路。本设计利用边缘触发中断来检测碰撞传感器的碰撞。小车与物体的碰撞触发边缘中断，在中断服务程序中读取防撞开关的状态，一旦检测到小车发生碰撞，MSP432P401R 则做出相应的相应，改变小车的前进方向。

图 4-1：防撞开关实物图

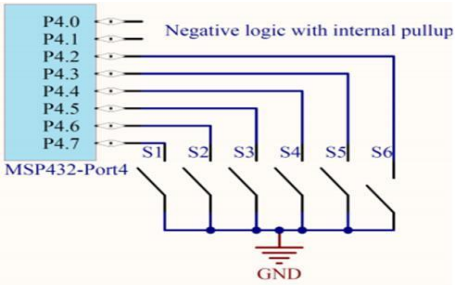


图 4-2：防撞开关接口图

2.3.1 软件原理

在小车的前部放置 6 个碰撞传感器，并将它们与微处理器 MSP432 相连。碰撞传感器允许软件知道哪里与前方障碍物体相碰。根据小车操作手册可以得知 MSP432 微处理器的 P4.7、P4.6、P4.5、P4.3、P4.2、P4.0 是从左到右 6 个碰撞传感器的 GPIO 口。将编写一个函数来初始化碰撞传感器 Bump_Init( )。该函数只需设置适当的端口引脚，并根据需要启用内部电阻。

在 uint8_t Bump_Read( )函数中，当左边两个传感器 P4.7 和 P4.6 发生碰撞时，返回值 3；当中间两个传感器 P4.5 和 P4.3 发生碰撞时，返回值 2；当右边两个传感器 P4.2 和 P4.0 发生碰撞时，返回值 1；不发生碰撞时，返回值 0。并由返回值的大小在主程序中控制小车的运动。在 HandleCollision（）函数中，当左边三个传感器发生碰撞时，小车先后退然后向左转；当右边三个传感器发生碰撞时，小车先后退然后向右转。

2.4 检测红外测距转向功能

2.4.1 硬件原理

在防撞开关的基础上，本设计还扩展了红外测距传感器以避开障碍物。该套件使用的为 3 个 Sharp DP2Y0A21YK0F 传感器，分别连接 CPU 的 P9.1、P9.0、P4.1 的三个 GPIO 口（如下图所示），探测范围 20cm-150cm，其输出电压与距离成反比。探测原理为三角测量原理：红外发射器以一定的角度发射红外光束，当遇到物体以后，光束会被反射回来。反射回来的红外光线被 CCD 检测器检测到以后，会获得一

个偏移值L，利用三角关系，在知道了发射角度a，偏移距L，中心矩X，以及滤镜的焦距f以后，传感器到物体的距离D就可以通过几何关系计算出来了，从而判别是否进行避障改变前进方向。

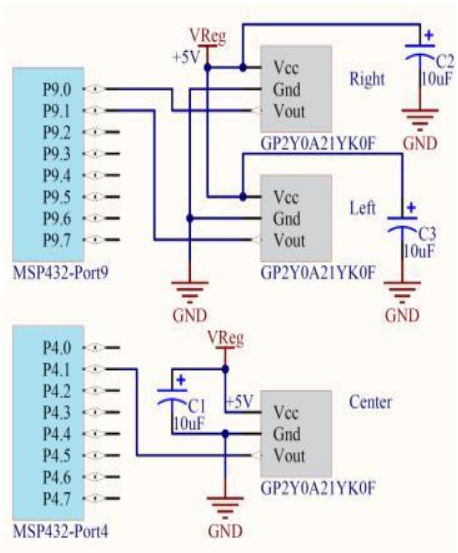


图 5-1：红外测距模块接口图

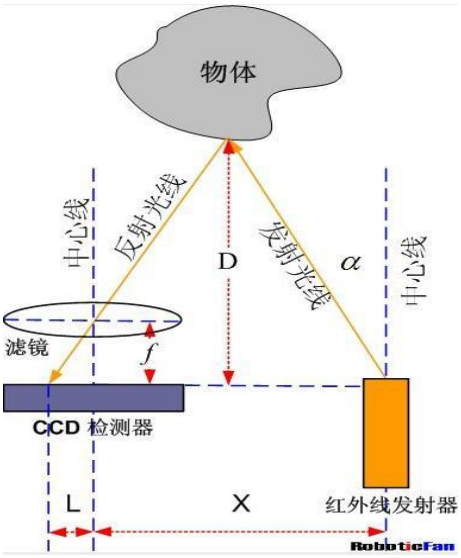


图 5-2：测距传感器三角测量原理图

2.4.2 软件原理

红外测距使用夏普的 DP2Y0A21YK0F 红外距离传感器，将距离的物理信号转换为电压信号。但电压信号属于模拟信号，微处理器不能直接识别处理，此时需要利用 ADC 处理 IR 传感器输出的模拟信号。其中输出电压与距离的关系如下图所示：

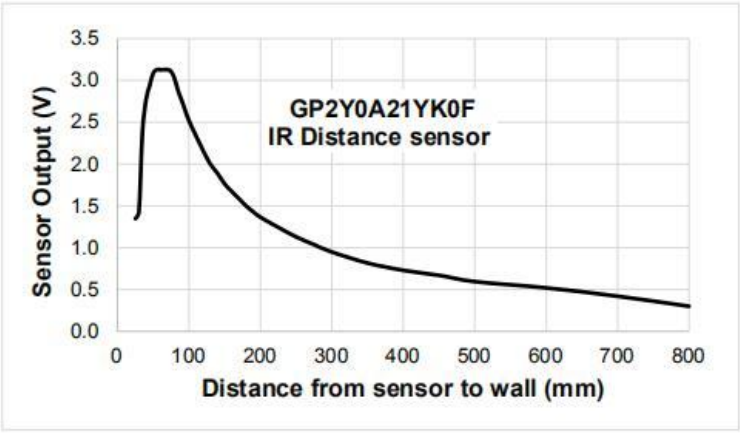


图 6：L-U 关系图

模/数转换器（Analog to Digital Converter，简称 ADC）是电子设备中必备的重要器件，用于将模拟信号转变为微控制器可以识别的数字信号，包括：采样、保持、量化和编码 4 个过程。

MSP432 包含一个 14 位的 ADC14 模块，支持快速 14 位模/数转换。ADC 的核心是将模拟输入转换为 14 位数字表示的数字信号，使用两个可编程电压（VR+和 VR-）来定义转换的上下限。下式为 ADC 转换公式：

$$N_{ADC} = 16384 \times \frac{V_{in+} - V_{R-}}{V_{R+} - V_{R-}}$$

但是由于红外测距传感器的测量值不太稳定，以一次测量的结果来判别控制小车的运行偶然性太大，所以以 1ms 为一个测量周期，取 16 个周期的平均值作为输出的 ADC 的采样值，即主程序中的 loop 函数。这样使控制系统更加稳定、更加准确。 设 n 为 14 位 ADC 的采样值（从 0 到 16383），且 D 为距离，单位是 mm。二者之间存在如下的非线性转换关系：

$$D = \frac{1195172}{n - 1058}$$

其中 1195172 和-1058 是校准系数的经验值，通过 ADC 实验取得的。得到准确的距离之后，就可以通过测量得到的距离值来判断小车的运行状态。

2.5 检测循迹功能

2.5.1 硬件原理

智能循迹小车的核心在于循迹，本设计采用 JN_LSA_2RSLKB01

线阵传感器，包含 8 个传感器，每个传感器都对应二进制数，如果看到黑色则返回 1，如果看到白色则返回 0。

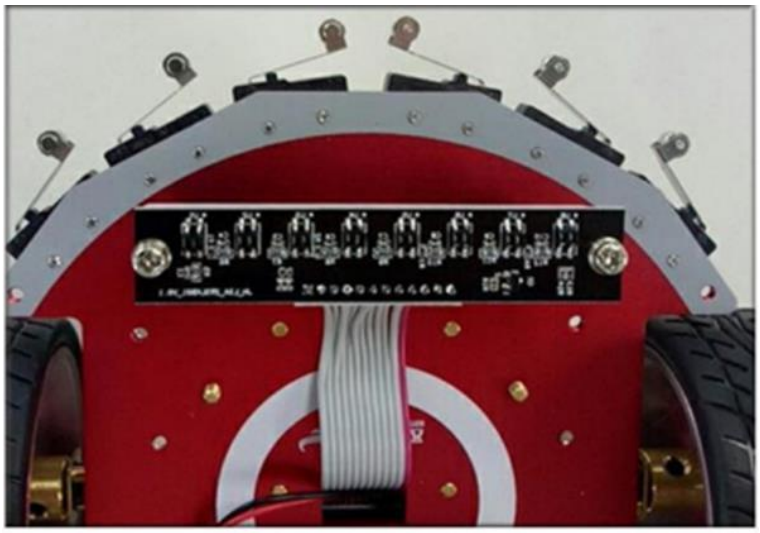


图 7-1：JN_LSA_2RSLKBO1 线阵传感器

如果小车正确地定位在线上，中间传感器将看到黑色；如果小车向左或向右偏离，则一个或多个外侧传感器将看到黑色；如果小车完全脱离轨迹，则所有传感器都将看到白色。MSP432P401R 通过对传感器返回的数据进行分析，从而控制小车的行进状态，修正小车运动路径。红外循迹传感器对路面进行实时检测，当其检测到黑色时返回 1，检测到白色时返回 0，将八个传感器返回的 8 位二进制数组合成一个位置参数。红外传感器依次连接 CPU 的 P7.7 至 P7.0 的八个 GPIO 口。

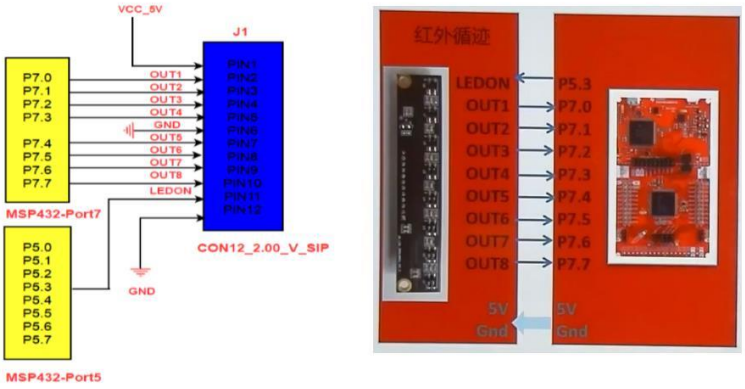


图 7-2：红外循迹模块接口图

2.5.2 软件原理

我们将小车的位置定义为从传感器到线的距离，我们使用的循迹模块宽约 66 毫米，每两个传感器间距为 9.5mm，小车中心在传感器 4 和 5 之间对齐，以 0.1mm 为单位定义 8 个距离值，从 U1 到 U8 依次定义为 332、237、142、47、-47、-142、-237、 -332,最后根据返回的 8 位二进制数分析计算得出偏离距离。因此我们能够估计距离线中心-33 到+33 mm 的机器人位置。

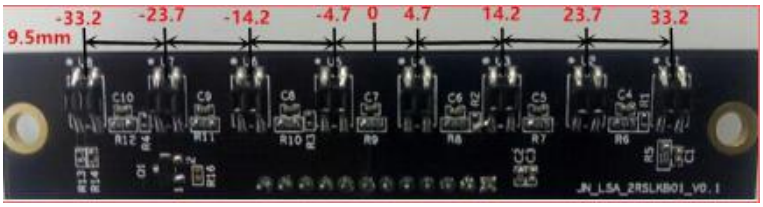


图 7-3：传感器距离标定

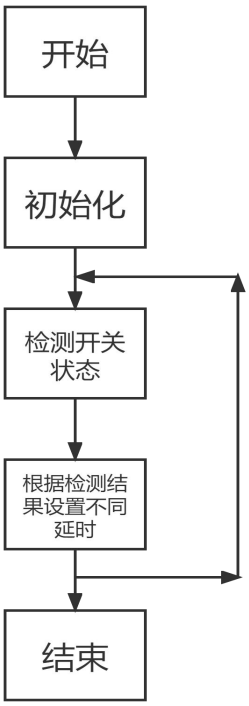
W = {332， 237， 142， 47， -47， -142， -237， -332}

对于 Reflectance_Read（）函数返回的每个二进制位，将b_i定义为 0（白色）或 1（黑色）。一种可能的传感器集成功能是计算 8 个传感器读数或二进制结果的加权平均值。假设至少有一个黑色读数，估计距离 Reflectance_Position 为：

$$d = \frac{\sum_{i=0}^{7} b_i W_i}{\sum_{i=0}^{7} b_i}$$

3. 软件流程与结果

3.1 实验一：流水灯实验

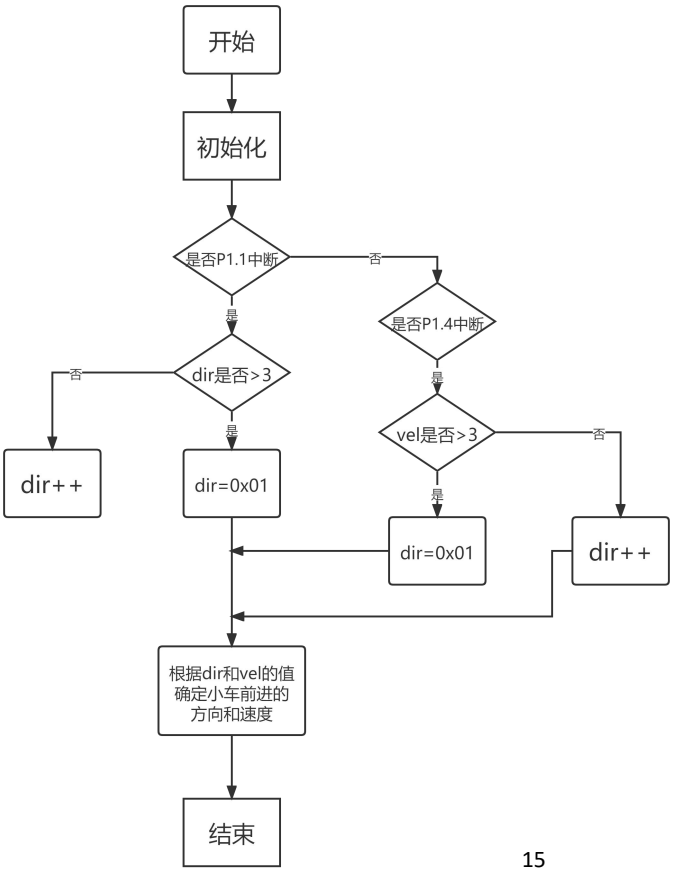


1. 打开电源开关，LED 以红灯、绿灯、蓝灯、粉灯交替闪烁；

2. 两个开关都不按，此时流水灯的闪烁速度最快。

3. 单独按下 SW2，闪烁速度变慢，单独按下 SW1，闪烁速度更慢，两个开关一同按下，此时灯闪烁时间最慢。

3.2 实验二：调节轮子速度与转动



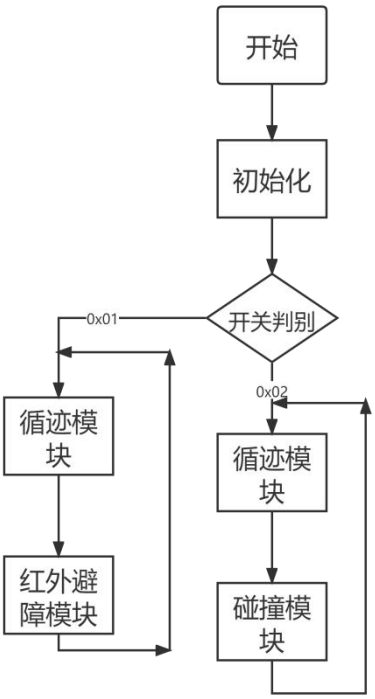
1. P1.1 改变行进方向，P1.4 改变轮子速度。

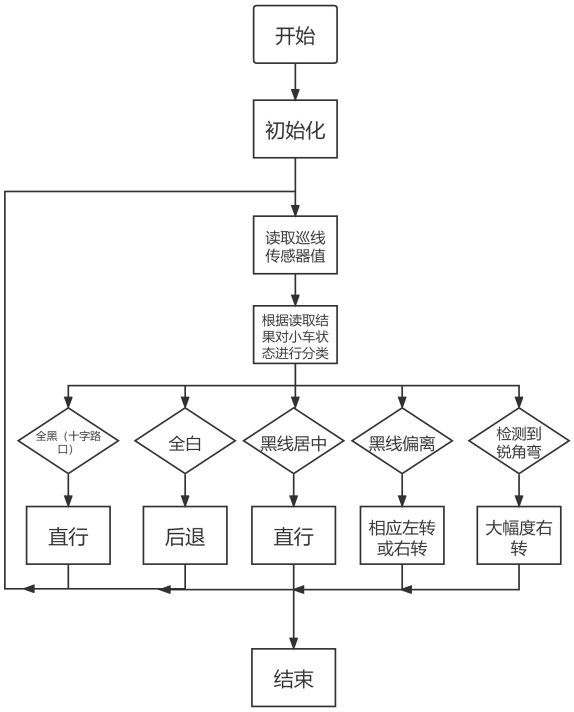
2. 首先测试 P1.1 功能，初始时，小车前进，第一次按下 P1.1 后，小车后退，第二次按下 P1.1 后，小车左转，由于持续左转，从而实验现象时小车向左原地自转，第三次按下

P1.1，小车右转，由于持续右转，从而实验现象时小车向右原地自转，第四次按下 P1.1，小车回到最初状态，前进。

3. 测试 P1.4 功能，初始状态是速度最慢的情况，每按一次 P1.4 的开关，小车速度变快，第四次按下时，小车回到最初的速度。

3.3 实验三：测试碰撞避障、红外避障与循迹





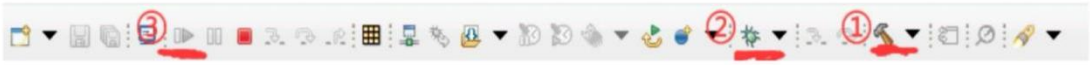
1. 打开电源，电机转动，此时小车只具备全速前进功能。

2. 按下左边开关 P1.1，小车进入循迹+红外避障模式，此时指示灯为蓝色。

3. 按下复位按钮 reset，小车重回初始化模式

4. 按下右边开关 P1.4，小车进入循迹+碰撞避障模式，此时指示灯为绿色。

4. 调试方法



利用 CCS 软件对小车功能进行实现，代码编译完成后，按上图所示①键进行代码调试，若无报错，用数据线将小车与电脑相连。再按图示②键进行代码烧录，最后按图示③键完成。此外，可以用 load 按键将代码烧录至小车中，不用连接数据线即可运行程序

5. 实验结果

5.1 流水灯实验

打开电源开关，LED 以红灯、绿灯、蓝灯、粉灯交替闪烁；两个开关都不按，此时流水灯的闪烁速度最快。单独按下 SW2,闪烁速度变慢，单独按下 SW1，闪烁速度更慢，两个开关一同按下，此时灯闪烁时间最慢。

5.2 轮子转动检测实验

P1.1 改变行进方向，P1.4 改变轮子速度。初始时，小车前进，第一次按下 P1.1 后，小车后退，第二次按下 P1.1 后，小车左转，由于持续左转,从而实验现象时小车向左原地自转,第三次按下P1.1，小车右转，由于持续右转，从而实验现象时小车向右原地自转，第四次按下 P1.1，小车回到最初状态，前进。

P1.4 改变行进速度,初始状态是速度最慢的情况,每按一次 P1.4 的开关，小车速度变快，第四次按下时，小车回到最初的速度。

5.3 碰撞避障、红外避障与循迹

碰撞避障：初始时小车行进方向为前进，当撞上障碍物，小车转向，改变前进方向，当再次撞上障碍物，小车转向，改变前进方向，从而实现碰撞转向。

红外避障：将障碍物置于小车前方，在小车距离障碍物一定距离时，触发红外避障模块，小车转向，改变前进方向，从而实现红外避障。

循迹：小车在黑色直线上沿黑色直线行走；遇到左转或者右转弯道可以顺利转向；遇到锐角拐弯可以快速转向并正确沿黑线行走；检测到十字路口后，向前直行；检测到全白，小车后退。

6. 实验感想

为期六个星期的的微处理器原理与应用综合实验终于落下了帷幕，回首过去的几周时间和三次验收，从面对一堆代码资料无从下手到带领学弟学妹参加南京理工大学第十六届机器人大赛“循迹智能车竞速赛”项目获奖，于我个人而言收获颇多。

这次实验分三次验收，前两次较为简单的基础功能为了让我们熟悉 CCS 软件的使用以及对小车编程的基本方法有所了解；第三次验收的要求较为复杂，在理解了硬件工作原理后，如何用代码语言实现功能成为了我们组攻坚克难的重点。

相较于大物实验的动手操作和模拟电子线路 EDA 的软件仿真，微机小车的实验既要我们对于微机原理中寄存器，中断以及各类逻辑运算有着熟练的掌握，同时也需要了解 C 语言的基本语法，是一门理论

与实践结合比较紧凑的实验课。

通过代码而对小车硬件进行控制，在我看来新奇又有趣。虽然我也曾不明白 GPIO 口为什么要这样初始化，也曾搞不懂为什么有时按下开关小车没有反应，也曾为了线路图中的各种不同 case 而头疼，但当我看到跑马灯按照我设置的颜色依次亮起，看到小车轮子的转速按照我的设置而有快有慢，看到小车在白底黑线的地图上自动循迹，我内心的成就感告诉我，这就是知识带来的快乐。

实验过程中小组成员的头脑风暴和集思广益也让我感受颇多。作为组长，对团队成员的分工安排各取所长，对团队进度的协同调整与统筹规划，对我个人是一种很强的锻炼，也让我们对本专业所能从事的工作有了一定的了解与认知。

最后，感谢胡老师三次验收时候对我们组工作的批评指正，虽然老师的有些问题我当时回答不上来，但经过我回去之后的思考与学习，我对于微处理器原理的相关知识又有了更深一层的了解与体会。

7. 小组成员分工

实验 3 程序编写，视频拍摄与制作

小车组装，辅助程序编写与调试

实验 1 程序编写，资料查询整理

实验 2 程序编写，资料查询整理

附录

实验一：

```
#include "msp.h"
#include "../inc/Clock.h"
#include <stdio.h>
#include "..\inc\LaunchPad.h"
#include "msp432p401r.h"

#define RED 0x01
#define GREEN 0x02
#define BLUE 0x04
#define OFF 0x00
#define yellow 0x03
#define sky_blue 0x06
#define white 0x07
#define pink 0x05
void Port1_Init(void){
P1->SEL0 = 0x00;
P1->SEL1 = 0x00;
P1->DIR = 0x01;
P1->REN = 0x12;
P1->OUT = 0x12;
}
void Port2_Init(void){
P2->SEL0 = 0x00;
P2->SEL1 = 0x00;
P2->DS = 0x07;
P2->DIR = 0x07;
P2->OUT = 0x00;
```

```
28. }
29.
30.
31. void Port1_Output(uint8_t data){
32.  P1->OUT = (P1->OUT&0xFE)|data;
33. }
34. void Port2_Output(uint8_t data){
35.  P2->OUT = data;
36. }
37. uint8_t Port1_Input(void){
38.  return (P1->IN&0x12);
39. }
40. int main(void){
41.     uint8_t status;
42.     int delay;
43.     Port1_Init();
44.     Port2_Init();
45.     Clock_Init48MHz();
46.     uint32_t i;
47.  while(1){
48.     status = Port1_Input();
49.     Clock_Delay1ms(100);
50.     switch(status){
51.      case 0x10:
52.          delay = 300;
53.          for(i=0;i<5;i++){
54.              Port2_Output(RED);
55.              Clock_Delay1ms(delay);
56.              Port2_Output(yellow);
57.              Clock_Delay1ms(delay);
```

```
58.                 Port2_Output(GREEN);
59.                 Clock_Delay1ms(delay);
60.                 Port2_Output(BLUE);
61.                 Clock_Delay1ms(delay);
62.                 Port2_Output(sky_blue);
63.                 Clock_Delay1ms(delay);
64.                 Port2_Output(pink);
65.                 Clock_Delay1ms(delay);
66.                 Port2_Output(white);
67.                 Clock_Delay1ms(delay);
68.             }
69.             Port2_Output(OFF);
70.             Clock_Delay1ms(200);
71.             Port1_Output(1);
72.             Clock_Delay1ms(1000);
73.             break;
74.         case 0x02: // SW2 pressed,流水灯
75.             delay = 800;
76.             for(i=0;i<5;i++){
77.                 Port2_Output(BLUE);
78.                 Clock_Delay1ms(delay);
79.                 Port2_Output(OFF);
80.                 Clock_Delay1ms(delay);
81.                 Port1_Output(1);
82.                 Clock_Delay1ms(delay);
83.                 Port1_Output(0);
84.                 Clock_Delay1ms(delay);
85.             }
86.             break;
87.         case 0x00: // both switches pressed
```

```
            Port2_Output(GREEN);
            Clock_Delay1ms(2000);
            Port2_Output(OFF);
            Clock_Delay1ms(2000);
            Port1_Output(1);
            Clock_Delay1ms(2000);
            break;
        case 0x12: // neither switch pressed
            Port2_Output(0);
            Port1_Output(0);
            break;
        }
    }
}
```

实验二：

```
#include "msp.h"
#include "../RSLK_base/inc/Clock.h"
#include "../RSLK_base/inc/Motor.h"
#include "../RSLK_base/inc/PWM.h"
#include "../RSLK_base/inc/LaunchPad.h"
#include <stdio.h>
uint8_t dir,vel;
void Port1_Motor_Init(void)
{
    P1->SEL0 &= ~0xD2;
    P1->SEL1 &= ~0xD2;
    P1->DIR |= 0xC0;
    P1->OUT &= ~0xC0;
    P1->DIR &= ~0x12;
```

```
15.      P1->REN |= 0x12;
16.      P1->OUT |= 0x12;
17.      P1->IFG &= ~0x12;
18.      P1->IE |= 0x12;
19.      P1->IES |= 0x12;
20.      P3->SEL0 &= ~0xC0;
21.      P3->SEL1 &= ~0xC0;
22.      P3->DIR |= 0xC0;
23.      P3->OUT &= ~0xC0;
24.  }
25.  void PORT1(void)
26.  {     uint8_t  InFlag;
27.       InFlag=P1->IFG;
28.       if(InFlag & 0x02 )
29.      {
30.           if(dir>0x04)dir=0x01;
31.           else
32.               dir++;
33.           P1->IFG &=~0x02;
34.      }
35.      if(InFlag & 0x10 )
36.      {
37.          if(vel>0x03)vel=0x01;
38.          else
39.              vel++;
40.          P1->IFG &=~0x10;
41.      }
42.      P1->IFG=0x00;
43.  }
```

```
uint8_t Port1_Input(void){
    return (P1->IN&0x12);  //read P1.4,P1.1 inputs
}
void main (void){
    Clock_Init48MHz();
    Port1_Motor_Init();
    int right=3500;
    int left=3500;
    dir=0x01;
    vel=0x01;//速度
    while(1){
        PORT1();
        switch(dir){
        case 0x02:
            Motor_Forward(left*vel,right*vel);
            break;
        case 0x03:
            Motor_Backward(left*vel,right*vel);
            break;
        case 0x04:
            Motor_Left(left*vel,right*vel);
            break;
        case 0x05:
            Motor_Right(left*vel,right*vel);
            break;
        case 0x01:
            Motor_Stop();
            break;
        default: break;
        }
```

```
    }
}
```

实验三：

```
#include <stdint.h>
#include "msp.h"
#include "..\inc\BumpInt.h"
#include "..\inc\Clock.h"
#include "..\inc\SysTick.h"
#include "..\inc\CortexM.h"
#include "..\inc\LaunchPad.h"
#include "..\inc\Motor.h"
#include"..\inc\PWM.h"
#include"..\inc\Reflectance.h"
#include "..\inc\ADC14.h"
#include "..\inc\IRDistance.h"
#include "..\inc\LPF.h"
#include "..\inc\TimerA0.h"
#include "..\inc\UART0.h"

//测距
void TimedPause(uint32_t time)
{
    Clock_Delay1ms(time); // 小车停止
    Motor_Stop();
}
int32_t position;
uint8_t Data;
uint16_t L = 10000;
uint16_t R = 10000;
```

```
27. void reflactance(void) //循迹功能
28. {
29.     Data = Reflectance_Read(500);  //5000ms 扫描一次
30.     position = Reflectance_Position(Data);
31.     switch (Data)
32.     {
33.     case 0x06:
34.     case 0x03:
35.     case 0x07:
36.     case 0x1a:
37.     case 0x1c:
38.     case 0x1d:
39.     case 0x1e:
40.     case 0x0f:
41.     case 0x1f:
42.     case 0x3f:
43.     case 0x01:
44.     case 0x7f:Motor_Left(L, R);break;
45.     case 0x1b:  //0001 1011
46.     case 0x19:Motor_Left(0, R);break;  //0001 1001
47.     case 0x98:
48.     case 0x58:
49.     case 0xd8:
50.     case 0xf8:
51.     case 0xb8:
52.     case 0xf0:
53.     case 0xfc:
54.     case 0x80:
55.     case 0xfe:
```

```
56.      case 0x38:
57.      case 0x60:
58.      case 0xa0:
59.      case 0xe0: Motor_Right(L, R); break;
60.      case 0x00: Motor_Backward(L + 2000, R + 2000); Clock_
   Delay1ms(100); break;
61.      default:Motor_Forward(L, R); break;
62.      }
63.  }
64.
65.  //碰撞
66.  uint32_t CollisionData, CollisionFlag; // mailbox
67.  uint32_t ch1, ch2, ch3;
68.  int32_t z = 1195172;
69.  int32_t e = 1058;
70.  uint8_t data, cnt_collision = 0;
71.
72.  void HandleCollision(uint8_t bumpSensor)
73.  {
74.      Motor_Stop();
75.      CollisionData = bumpSensor;
76.      CollisionFlag = 1;
77.      if (CollisionData == 0x1f || CollisionData == 0x2f ||
    CollisionData == 0x37)
78.      {
79.          Motor_Backward(5500, 5500);
80.          Clock_Delay1ms(210);
81.          Motor_Right(5000, 5000);
82.          Clock_Delay1ms(270);
```

```
83.             Motor_Stop();
84.         }
85.         else if (CollisionData == 0x3b || CollisionData == 0x
    3d || CollisionData == 0x3e)
86.         {
87.             Motor_Backward(5500, 5500);
88.             Clock_Delay1ms(210);
89.             Motor_Left(5000, 5000);
90.             Clock_Delay1ms(270);
91.             Motor_Stop();
92.         }
93.  }
94.  void main(void) {
95.         uint8_t t;
96.         uint32_t distance;
97.         UART0_Initprintf();
98.         ADC0_InitSWTriggerCh17_12_16();
    //初始化 AD 采样
99.         CollisionFlag = 0;
100.        Clock_Init48MHz();
    //时钟初始化
101.        LaunchPad_Init();
    //初始化 GPIO 接口
102.        Motor_Init();
    // PWM 初始化
103.        Motor_Forward(6000,6000);
    //周期 10ms，左轮高电平，右轮高电平
104.        PWM_Init34(10000, 5000, 5000);
105.        Reflectance_Init();
```

```
106.     while (LaunchPad_Input() == 0);
   // wait for touch 读取按键
107.     t = LaunchPad_Input();
108.     switch (t) {
109.     case 0x01:
   //按键 1，进行避障循迹
110.         while (1) {
111.             LaunchPad_Output(4);
   //理论蓝灯
112.             ADC_In17_12_16(&ch1, &ch2, &ch3);
113.             distance = z / (ch2 - e);
114.             if (distance != 0 && distance < 150)
115.             {
116.                 Motor_Right(5000, 5000);
117.                 Clock_Delay1ms(1000);
118.                 Motor_Stop();
119.             }
120.             else
121.             {
122.                 reflactance();
   //循迹
123.             }
124.         }
125.         break;
126.     case 0x02:
   //按键 2，进行碰撞循迹
127.         while (1)
128.         {
129.             BumpInt_Init(&HandleCollision);
```

```
130.                LaunchPad_Output(2);
    //默认绿灯
131.                reflactance();
    //循迹
132.            }
133.            break;
134.        }
135. }
```