模块 5

实验：循线迷宫-有限状态机

实验:循线迷宫-有限状态机

5.0 目标

本实验的目的是开发和测试用于机器人的有限状态机

1. 您将学习如何在 C 中使用结构体和指针。
2. 您将了解如何使用 FSM 解决问题。
3. 您将使用 FSM 实现简单的线路跟踪算法。

小知识： 即使您将使用输入开关和输出 LED 实现本实验，FSM 设计过程也可用于机器人控制器。该实验的解决方案将允许机器人循迹一条线（黑色胶带）。

5.1 入门

5.1.1 从下面的软件工程起步

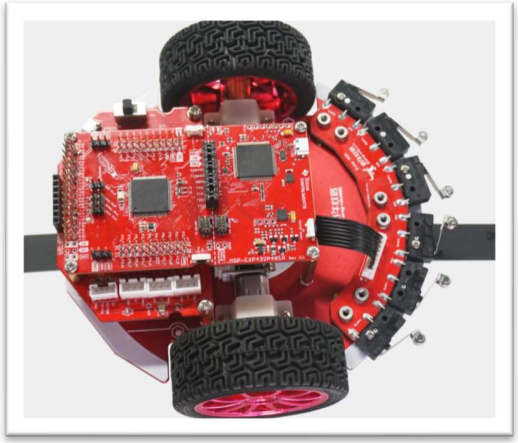
浏览以下 3 个工程：
LineFollowFSM （实现循线的简单 FSM）
Lab08_Track_Maze （main2 对应视频中的程序）
Lab_FSM （本实验的入门项目）

5.1.2 参考资料

Meet the MSP432 LaunchPad (SLAU596*)*
MSP432 LaunchPad User's Guide (SLAU597*)*

5.1.3 阅读材料

Volume 1 Sections 6.1, 6.2, 6.4 and 6.5
Embedded Systems: Introduction to the MSP432 Microcontroller",
or
Volume 2 Section 3.5
Embedded Systems: Real-Time Interfacing to the MSP432 Microcontroller"



5.1.4 本实验所需组件

数量	组件描述	制造商	型号
1	MSP-EXP432P401R LaunchPad	TI	MSP-EXP432P401R

5.1.5 所需实验设备

示波器（1 个至少 10khz 采样的通道）
逻辑分析仪 （4 个至少 10kHz 采样的通道）

5.2 系统设计要求

Lab_FSM 起始项目实现了图 1 中所示的三态 FSM，我们可以使用它来实现循线跟踪机器人。每个状态等待的时间是 500ms。在真实的机器人上，我们将这些延迟时间设置得更短，具体取决于机械机器人对执行器命令的响应速度。但是，在本实验中，选择 500 ms 以便于用眼睛轻松查看输出。

实验:循线迷宫-有限状态机

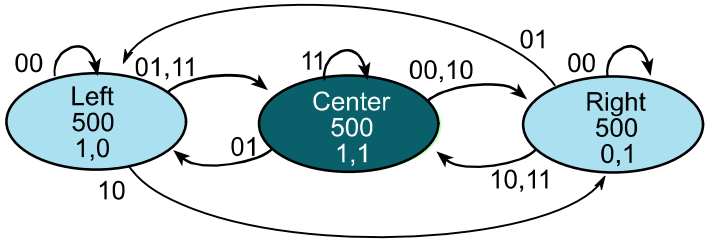


图 1. 摩尔 FSM 状态图实现循线。每个状态的时间以 1ms 为单位显示。

机器人有两个检测循线的传感器，见图 2。如果机器人正确定位在线路上，两个传感器都将读取 1。如果机器人向左或向右稍微偏离，则一个传感器读取 1，另一个传感器读取 0。如果机器人完全脱离线路，两个传感器将读取 0。

机器人有两个电机，如图 2 所示。两个电机和一个被动脚轮允许机器人以差速传动方式运行。如果软件向两个电机输出高电平，则机器人以直线向前移动。如果软件只向一个电机输出高电平，它将转向。如果软件向两个电机输出低电平，它将停止。

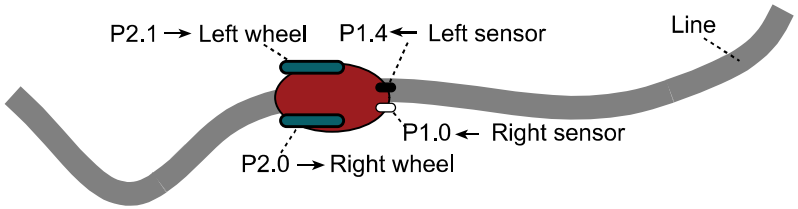


图 2. 带两个线传感器和两个电机轮的机器人。

系统会要求您扩展此 FSM，添加其他状态以实现以下行为。
1) 如果机器人向左偏离一点（输入为 01，机器人在左边状态和中心状态之间摆动），那么图 1 中的 FSM 会混淆（有一个错误），然后完全离开左边的线（ 输入是 00）。在这台机器上，如果当它离开线路时碰巧处于中心状态，即使机器人向左移动，它也会错误地移动到右边状态。 您将通过实现两个左侧状态来解决此问题（因此当稍微离开时它会在两个左侧状态之间摆动）。为了对称，你也将实现两个右边的状态。图 3 显示了部分解决方案。如果输入为 11，则输出应保持为 11。如果输入转到 01（它有点偏左），则输出应切换 1,0↔1,1 引起轻微右转。同样的，如果输入变为 10（它有点偏右），那么输出应该切换 0,1↔1,1 引起轻微左转。

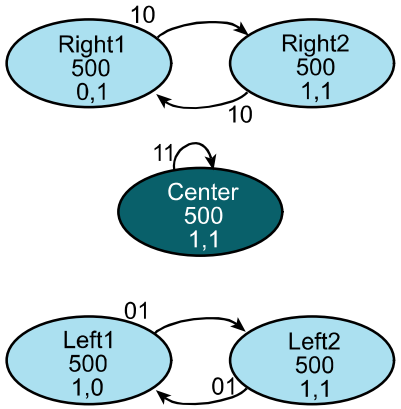


图 3. 扩展的 FSM 状态图。每个状态的时间以 1 毫秒为单位显示。

2) 您需要实现的第二个行为是当机器人完全离开线时会发生什么。如果它偏离到线的右边（在 Right1 或 Right2 中 input=0,0），它应该努力持续 5 秒左转（output=0,1），然后笔直走（output=1,1）持续 5 秒。如果此时它仍然偏离线，它应该停止（output=0,0）。如果找到该线， 应该恢复循线。应该还需要三个状态去实现此行为。

同样的，如果机器人偏离到线的左边（在 Left1 或 Left2 中 input=0,0）， 它应该努力持续 5 秒右转（output=1,0），然后笔直走（output=1,1）持续 5 秒。如果此时它仍然偏离线，它应该停止（output=0,0）。如果找到该线， 应该恢复循线。应该还需要三个状态去实现此行为。

解决方案应该有大约 11 个状态（图 3 中的 5 个状态，右边 3 个状态，左边 3 个状态）。 只要你有 9 个或更多的状态，随意做出假设或改变机器的确切行为。该实验的目的是用状态转换图描述系统的完整行为，然后使用非常简单的 FSM 控制器实现该行为。FSM 控制器应该没有条件分支状态说明。

实验:循线迷宫-有限状态机

5.3 实验准备

您将仅使用 MSP432 Launch Pad 实现此实验，无需其它电路，请参见图 4。
Launch Pad 驱动程序软件将开关输入转换为正逻辑，因此“按下开关”被视为 1，见表 1。
LED 输出为正逻辑，见表 2。

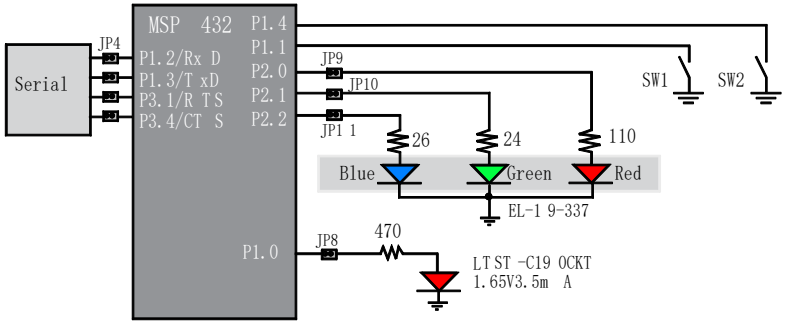


图 4. P1.4 是左传感器，P1.1 是右传感器，P2.1 是左电机，P2.0 是右电机。

LaunchPad_Input 函数（在 LaunchPad.c 中定义）以正逻辑返回开关位置因此推动两个开关会产生 1,1 的输入条件。函数（在 LaunchPad.c 中定义）将数据发送到 3 位彩色 LED。

SW2	SW1	**LaunchPad_Input**	Meaning
pressed	pressed	1,1 = 0x03	On line
pressed	not	1,0 = 0x02	Right of line
not	pressed	0,1 = 0x01	Left of line
not	not	0,0 = 0x00	Off the line

表 1. 开关模拟循线传感器。

P2.1	P2.0	**LaunchPad_Output**	LED	Meaning
off	off	0,0 = 0x00	black	Stop
off	on	0,1 = 0x01	red	Turn left
on	off	1,0 = 0x02	green	Turn right
on	on	1,1 = 0x03	yellow	Straight

表 2. LED 模拟机器人电机

5.4 实验步骤

5.4.1 循线 FSM

第一步是编译，下载并运行如下所示的 **LineFollowFSM** 示例。使用调试器，单步执行控制器(跳过函数) 并观察输入，输出和指针 Spt。 请注意如何定义结构体以及如何使用指针访问结构体中的数据。使用调试器，确定 FSM 所在的内存位置(是在 RAM 还是 ROM 中)?

实验:循线迷宫-有限状态机

```
struct State {

  uint32_t out;                // 2-bit output
  uint32_t delay;              // time to delay in 1ms
  const struct State *next[4]; // Next if input is 0-3
};
typedef const struct State State_t;

#define Center &fsm[0]
#define Left   &fsm[1]
#define Right  &fsm[2]
StateType fsm[3]={
  {0x03, 500, { Right, Left,   Right,  Center }},
  {0x02, 500, { Left,  Center, Right,  Center }},
  {0x01, 500, { Right, Left,   Center, Center }}
};
State_t *Spt;  // pointer to the current state
uint32_t Input;
uint32_t Output;
int main(void){ uint32_t heart=0;
  Clock_Init48MHz();

LaunchPad_Init();
  TExaS_Init(LOGICANALYZER);  // optional
  Spt = Center;
  while(1){
    Output = Spt->out;            // set output from FSM
    LaunchPad_Output(Output);     // output to motors
    TExaS_Set(Input<<2|Output);   // optional
    Clock_Delay1ms(Spt->delay);   // wait
    Input = LaunchPad_Input();    // read sensors
    Spt = Spt->next[Input];       // next
    heart = heart^1;
    LaunchPad_LED(heart);         // optional
  }
}
```

在这个项目中， FSM 反复执行下面 4 步序列：

1) *Output* depends on *State* (LaunchPad LED)
2) *Wait* depends on State
3) *Input* (LaunchPad buttons)
4) *Next* depends on (*Input, State*)

运行程序并观察静态行为。

i) 填写表 3，描述如果输入保持不变，机器会做什么。

SW2	SW1	Input	Meaning	Output behavior
pressed	pressed	1,1	On line	
pressed	not	1,0	Right of line	
not	pressed	0,1	Left of line	

表 3. 简单 FSM 的静态响应表。

当仅按下一个开关时，它表示机器人稍微离线的情况。在这种情况下，一个轮子处于活动状态，另一个轮子振荡开关。这种振动导致这个轮子转动，但速度较慢。如果 P2.1 为高，则左轮转动为 100%。数字波的占空比定义为信号高的时间百分比。如果 P2.0 上的占空比为 n =（高/（高+低）），则右电机旋转 n * 100%，机器人将平缓转动。使用示波器或逻辑分析仪测量端口 P2.0 上的振荡速率和占空比。 见图 5。

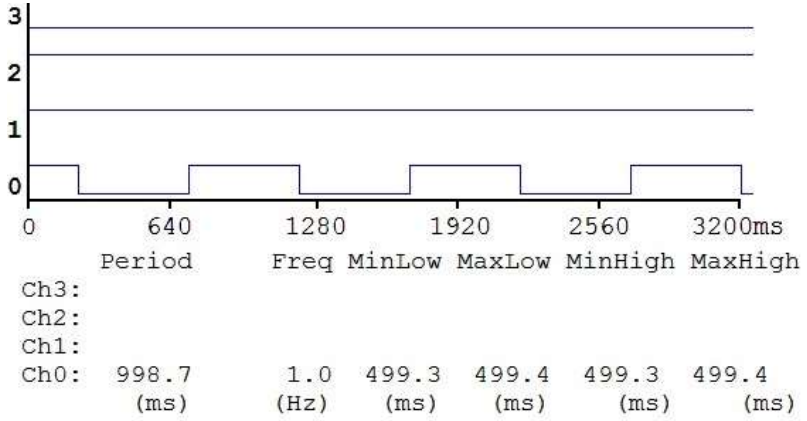


图 5. 显示右轮通道 0 振荡的逻辑分析仪轨迹为 1 Hz，占空比为 50%。

Note：通道 3-2 的 Input =1 (left sensor=0, right sensor =1)，显示状态为有点偏左。 通道 1-0 的 Output (left motor=1, right motor oscillating)，显示一个平缓的右转

最后，您将观察此 FSM 中的问题。
1) Start with both switches pressed (on the line);
2) Release SW2 (the robot is a little off to the left); then
3) Release SW1.

此时您完全偏离到线的左边了。多次重复上面 1-2-3 步骤序列，您会发现它有时会正确结束在左边状态，但有时它会错误结束在右边状态。

5.4.2 设计改进的 FSM

第二步是按照图 3.0 中要求部分的描述设计 FSM。只要您的机器有 9 个或更多状态，随意调整机器的运行方式。在本实验部分，您将：

i) 绘制状态转换图

ii) 创建状态转换表，并输入数据结构的 C 代码。
这三个应该是完全相同的信息（不多也不少）。这种等效性被称为**一对一**，它是良好 FSM 设计的重要特征。如果图形与 C 中的数据结构一一对应，那么我们可以确信系统按图中所描述的那样运行。

iii) 您将使用 5.4.1 节末尾所示的 1-2-3 步骤序列测试您的系统。但是，只要在释放 SW1 之前释放 SW2 至少等待 500 毫秒，那么你应该总是在左边的一个状态中结束。

至少执行此测试十次以验证其是否正常工作。同样对于右侧状态，
1) Start with both switches pressed (on the line);
2) Release SW1 (the robot is a little off to the right); then
3) Release SW2

此时您完全偏离到线的右边了。重复上面 1-2-3 步骤 10 多次应该，你应该总是结束在右边的某一种状态上。

使用逻辑分析仪测试系统的静态行为。假设输入保持不变，请填写表 4。有两个离开线路的条件：偏离左边，偏离右边。

SW2	SW1	Input	Meaning	Output behavior
pressed	pressed	1,1	On line	
pressed	not	1,0	Right of line	
not	pressed	0,1	Left of line	
not	not	0,0	Off the line	
not	not	0,0	Off the line	

表 4. FSM 实验的静态响应表。

5.5 疑难解答

无法编程 LaunchPad：

- 检查 LaunchPad 开发板上的电缆，跳线。
- 检查 Windows 驱动程序以查看操作系统是否识别该板卡。
- 在此计算机上尝试其他 LaunchPad。
- 在另一台计算机上尝试此 LaunchPad。

严重故障：

- 验证 **Spt** 总是指向 FSM 的一个条目。

时间延迟太慢或太快：

- 验证计算机运行在 48 MHz。
- 返回并确保模块 6GPIO 中的实验仍然有效。

5.6 请思考

在这节中，我们列出了完成本实验后要考虑的思考问题。这些问题旨在测试您对本实验中概念的理解。

- 可以有两个具有相同输出的状态吗？ 为什么？
- FSM 如何创建 50％占空比输出方波？改变什么让它变成 75％（更平缓的转弯）？改变什么让它变成 25％（急转弯）？
- 重要的是，状态转换图和 C 中的数据结构是一对一的。一对一是什么意思并解释它是怎么一回事？
- 本实验使用以 **LaunchPad_**开头的四个函数中的 I / O 抽象。头文件 LaunchPad.h 中有哪些信息？代码文件 LaunchPad.c 中有什么？这种抽象概念有什么好处？
- FSM 有 2 个输入。如果有 3 个输入会有什么变化？ 4 个输入呢？
- FSM 有 2 个输出。如果有 3 个输出会有什么变化？ 4 个输出呢？
- FSM 是如何测试的？

5.7 其它挑战

在这节中，我们列出了您可以做的其它活动，以进一步探索此模块的概念。您可以扩展系统或提出完全不同的东西。 例如：

- 使用实验 6 中接口的实际循线传感器替换开关输入。如果使用循线传感器，则可以将输入从 2 位扩展为 4 位。
- 使用 FSM 方法解决类似问题，如交通信号灯控制器或步进电机控制器。
- 此 FSM 使用指针定义当前状态。您可以使用索引来实现 FSM 以访问状态参数。 例如，**Output = fsm[index].out**；

5.8 接下来是哪些模块?

FSM 是用于解决复杂系统的强大设计工具。许多机器人挑战的有效解决方案将包括 FSM。

模块 2)　将实际开关和 LED 连接到微控制器。
　　这将允许更多的输入和输出增加系统的复杂性。
模块 6)　实现机器人的循线迷宫挑战。

5.9 您应该已经学会

在这节中，我们将回顾您应该在本单元中学到的重要概念：

- 使用 ***struct*** 来组织数据。
- 使用指针访问数据。
- 在项目中使用多个文件来实现抽象。
- 设计一个简单的 FSM 绘制状态转换图。
- 将状态转换图转换为 C 语言的数据结构体。
- 使用逻辑分析仪测量输入/输出之间的时序。
- 调试 FSM 并验证其正确行为。