运动控制系统 - 轴移动API使用文档

概述

本文档详细说明了运动控制系统中用于控制各轴移动到指定位置的API函数及其使用方法。

> ⚠️ **重要提示：单位说明**
>
> **位置参数单位：微米（μm）**
>
> - 输入的整数数据单位是**微米（μm）**
> - 1 毫米 (mm) = 1,000 微米 (μm)
> - 例如：移动到 50mm 的位置，应输入 `u32Data = 50000`
>
> **速度参数单位：微米/秒（μm/s）**
>
> - 速度设置函数的单位是**微米/秒（μm/s）**
> - 1 毫米/秒 (mm/s) = 1,000 微米/秒 (μm/s)
> - 例如：设置速度为 10mm/s，应输入 `u32Speed = 10000`

目录

单位说明

📏 位置参数单位：微米（μm）

所有API函数中的位置参数（`u32Data`）单位统一为微米（μm）。

单位换算表

单位	换算关系	微米值	示例
1 微米 (μm)	基本单位	1 μm	`u32Data = 1`
1 毫米 (mm)	= 1000 μm	1,000 μm	`u32Data = 1000`

单位	换算关系	微米值	示例
1 厘米 (cm)	= 10000 μm	10,000 μm	`u32Data = 10000`
1 米 (m)	= 1000000 μm	1,000,000 μm	`u32Data = 1000000`

实际应用示例

```
// 移动到 5mm 的位置
PRtGo_X.u32Data = 5000;       // 5mm = 5000μm

// 移动到 12.5mm 的位置
PRtGo_Y.u32Data = 12500;      // 12.5mm = 12500μm

// 移动到 100mm 的位置
PRtGo_Z.u32Data = 100000;     // 100mm = 100000μm

// 移动 0.5mm (500微米)
PRtGo_X.u32Data = 500;        // 0.5mm = 500μm
```

⚠️ 注意事项

1. **精度**：系统精度可达微米级别，但实际精度取决于机械结构和电机性能
2. **范围**：确保输入值不超过机械行程限制
3. **小数处理**：不支持小数，如需 0.1mm 精度，应输入 100μm
4. **负数表示**：使用 `u8Sign` 字段表示正负，而非负数值

```
// ✅ 正确：使用符号位表示负数
PRtGo_X.u8Sign = 1;           // 1表示负数
PRtGo_X.u32Data = 5000;       // -5mm

// ❌ 错误：不要使用负数值
PRtGo_X.u32Data = -5000;      // 错误！u32不能为负
```

🚀 速度参数单位：微米/秒（μm/s）

所有速度设置函数中的速度参数单位为微米/秒（μm/s）。

速度换算表

单位	换算关系	微米/秒值	示例
1 微米/秒 (μm/s)	基本单位	1 μm/s	`u32Speed = 1`
1 毫米/秒 (mm/s)	= 1000 μm/s	1,000 μm/s	`u32Speed = 1000`
1 厘米/秒 (cm/s)	= 10000 μm/s	10,000 μm/s	`u32Speed = 10000`
1 米/秒 (m/s)	= 1000000 μm/s	1,000,000 μm/s	`u32Speed = 1000000`

速度设置示例

```
// 设置速度为 10mm/s
Run_Set_RunSpeedX(10000);     // 10mm/s = 10000μm/s

// 设置速度为 50mm/s
Run_Set_RunSpeedY(50000);     // 50mm/s = 50000μm/s

// 设置速度为 5mm/s
Run_Set_RunSpeedZ(5000);      // 5mm/s = 5000μm/s

// 使用系统预设速度（推荐）
Run_Set_RunSpeedX(GtSystemSetData.u32WorkSpeedX);
```

核心API函数

1. 设置目标位置函数

1.1 单轴位置设置

```
// X轴位置设置
void WillRunToX_Uniaxial(ref StructSignData tGoPosition)

// Y轴位置设置
void WillRunToY_Uniaxial(ref StructSignData tGoPosition)

// Z轴位置设置
void WillRunToZ_Uniaxial(ref StructSignData tGoPosition)

// X2轴(R轴/旋转轴)位置设置
void WillRunToX2_Uniaxial(ref StructSignData tGoPosition)

// Y2轴位置设置
void WillRunToY2_Uniaxial(ref StructSignData tGoPosition)

// Z2轴位置设置
void WillRunToZ2_Uniaxial(ref StructSignData tGoPosition)
```

参数说明:

- `tGoPosition`: 目标位置数据结构（类型：`StructSignData`）
  - `u8Sign`: 符号 (0为正数，1为负数)
  - `u32Data`: 数值（**单位：微米 μm**）

> ⚠ **重要说明：所有位置参数的单位均为微米（μm）**

---

2. 启动/停止运动函数

```
void RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(
    u8 u8X_Cmd,   // X轴命令
    u8 u8Y_Cmd,   // Y轴命令
    u8 u8Z_Cmd,   // Z轴命令
    u8 u8X2_Cmd,  // X2轴命令
    u8 u8Y2_Cmd,  // Y2轴命令
    u8 u8Z2_Cmd   // Z2轴命令
)
```

参数值定义:

- `1`: 启动该轴运动
- `2`: 停止该轴运动/保持不动

示例:

```
// 启动X和Y轴，其他轴保持不动
RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(1, 1, 2, 2, 2, 2);

// 仅启动Z轴
RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(2, 2, 1, 2, 2, 2);

// 启动所有轴
RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(1, 1, 1, 1, 1, 1);
```

---

3. 运动完成状态检测函数

```
// 检查X轴是否运动完成
u8 CheckRunFinishStatus_Uniaxial_X()

// 检查Y轴是否运动完成
u8 CheckRunFinishStatus_Uniaxial_Y()

// 检查Z轴是否运动完成
u8 CheckRunFinishStatus_Uniaxial_Z()

// 检查X2轴是否运动完成
u8 CheckRunFinishStatus_Uniaxial_X2()

// 检查Y2轴是否运动完成
u8 CheckRunFinishStatus_Uniaxial_Y2()

// 检查Z2轴是否运动完成
u8 CheckRunFinishStatus_Uniaxial_Z2()
```

返回值:

- `0`: 运动已完成/轴已停止
- `非0`: 轴仍在运动中

---

4. 速度设置函数

```
// 设置X轴运动速度
void Run_Set_RunSpeedX(u32 u32Speed)

// 设置Y轴运动速度
void Run_Set_RunSpeedY(u32 u32Speed)

// 设置Z轴运动速度
void Run_Set_RunSpeedZ(u32 u32Speed)

// 设置X2轴(R轴)运动速度
void Run_Set_RunSpeedX2(u32 u32Speed)
```

参数说明:

- `u32Speed`: 速度值（单位：**微米/秒 μm/s**）
  - 例如：设置速度为 10mm/s，应输入 `u32Speed = 10000` (10000μm/s)
  - 通常使用系统配置中的预设速度值，如 `GtSystemSetData.u32WorkSpeedX`

数据结构

StructSignData 结构体

```
public struct StructSignData
{
    public u8 u8Sign;    // 符号: 0为正数, 1为负数
    public u32 u32Data;  // 数值（单位：微米 μm）
}
```

字段说明:

- `u8Sign`: 符号位，0为正数，1为负数
- `u32Data`: 数值部分，**单位为微米（μm）**

使用示例:

```
// 创建目标位置数据
StructSignData targetPos;
targetPos.u8Sign = 0;       // 正数
targetPos.u32Data = 10000; // 位置值：10000微米 = 10毫米

// 或使用已有变量（如 PRtGo_X, PRtGo_Y 等）
PRtGo_X.u8Sign = 0;
PRtGo_X.u32Data = 5000;     // 5000微米 = 5毫米
```

> 📏 **单位换算:**
>
> - 1 毫米 (mm) = 1,000 微米 (μm)
> - 1 厘米 (cm) = 10,000 微米 (μm)

> - 例如：设置 `u32Data = 50000` 表示 50mm 的位置

使用流程

基本流程图

```
开始
  ↓
1. 设置目标速度
  ↓
2. 设置目标位置（调用WillRunToX_Uniaxial等函数）
  ↓
3. 启动运动（调用RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2）
  ↓
4. 等待运动完成（轮询CheckRunFinishStatus_Uniaxial_X等函数）
  ↓
5. 执行后续操作
  ↓
结束
```

完整示例

示例1: 移动X和Y轴到指定位置

```
// 步骤1: 设置速度
Run_Set_RunSpeedX(GtSystemSetData.u32WorkSpeedX);
Run_Set_RunSpeedY(GtSystemSetData.u32WorkSpeedY);

// 步骤2: 设置目标位置（单位：微米）
PRtGo_X.u8Sign = 0;           // 正数
PRtGo_X.u32Data = 10000;      // X轴目标位置：10000μm = 10mm

PRtGo_Y.u8Sign = 0;           // 正数
PRtGo_Y.u32Data = 15000;      // Y轴目标位置：15000μm = 15mm

// 步骤3: 将目标位置发送到运动控制系统
WillRunToX_Uniaxial(ref PRtGo_X);
WillRunToY_Uniaxial(ref PRtGo_Y);

// 步骤4: 启动X和Y轴运动（其他轴保持不动）
RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(
    1,  // X轴: 启动
    1,  // Y轴: 启动
    2,  // Z轴: 不动
    2,  // X2轴: 不动
    2,  // Y2轴: 不动
    2   // Z2轴: 不动
);

// 步骤5: 等待运动完成（在状态机中检测）
// 在case语句或定时器中检测
```

```
if (0 == CheckRunFinishStatus_Uniaxial_X() &&
    0 == CheckRunFinishStatus_Uniaxial_Y())
{
    // 运动完成，执行后续操作
    // ...
}
```

示例2: Z轴下降到指定高度

```
// 设置Z轴速度
Run_Set_RunSpeedZ(GtSystemSetData.u32WorkSpeedZ);

// 设置Z轴目标位置
PRtGo_Z.u8Sign = GtKeyMove.tPoint_Z.u8Sign;
PRtGo_Z.u32Data = GtKeyMove.tPoint_Z.u32Data;

// 发送目标位置
WillRunToZ_Uniaxial(ref PRtGo_Z);

// 启动Z轴运动
RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(
    2,  // X轴: 不动
    2,  // Y轴: 不动
    1,  // Z轴: 启动
    2,  // X2轴: 不动
    2,  // Y2轴: 不动
    2   // Z2轴: 不动
);

// 等待Z轴运动完成
// 在状态机的下一个case中检测
```

示例3: Z轴先抬起，然后XY轴移动（防碰撞）

```
// === 阶段1: Z轴抬起 ===
case 0:
    // Z轴先抬到安全高度
    PRtGo_Z.u8Sign = 0;
    PRtGo_Z.u32Data = 0;  // 抬到零点

    WillRunToZ_Uniaxial(ref PRtGo_Z);
    RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(2, 2, 1, 2, 2, 2);

    PRu8_Run_Step = 1;
    break;

// === 阶段2: 等待Z轴完成 ===
case 1:
    if (0 == CheckRunFinishStatus_Uniaxial_Z())
    {
        // Z轴已抬起，可以安全移动XY轴
        PRtGo_X.u8Sign = GtKeyMove.tPoint_X.u8Sign;
```

```
        PRtGo_X.u32Data = GtKeyMove.tPoint_X.u32Data;

        PRtGo_Y.u8Sign = GtKeyMove.tPoint_Y.u8Sign;
        PRtGo_Y.u32Data = GtKeyMove.tPoint_Y.u32Data;

        WillRunToX_Uniaxial(ref PRtGo_X);
        WillRunToY_Uniaxial(ref PRtGo_Y);

        RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(1, 1, 2, 2, 2, 2);

        PRu8_Run_Step = 2;
    }
    break;

// === 阶段3: 等待XY轴完成 ===
case 2:
    if (0 == CheckRunFinishStatus_Uniaxial_X() &&
        0 == CheckRunFinishStatus_Uniaxial_Y())
    {
        // XY轴已到位
        PRu8_Run_Step = 3;
    }
    break;
```

示例4: 多轴联动（XYR三轴同时运动）

```
// 设置三个轴的速度
Run_Set_RunSpeedX(GtSystemSetData.u32WorkSpeedX);
Run_Set_RunSpeedY(GtSystemSetData.u32WorkSpeedY);
Run_Set_RunSpeedX2(GtSystemSetData.u32WorkSpeedR);

// 设置三个轴的目标位置
PRtGo_X.u8Sign = GtKeyMove.tPoint_X.u8Sign;
PRtGo_X.u32Data = GtKeyMove.tPoint_X.u32Data;

PRtGo_Y.u8Sign = GtKeyMove.tPoint_Y.u8Sign;
PRtGo_Y.u32Data = GtKeyMove.tPoint_Y.u32Data;

PRtGo_R.u8Sign = GtKeyMove.tPoint_R.u8Sign;
PRtGo_R.u32Data = GtKeyMove.tPoint_R.u32Data;

// 发送目标位置
WillRunToX_Uniaxial(ref PRtGo_X);
WillRunToY_Uniaxial(ref PRtGo_Y);
WillRunToX2_Uniaxial(ref PRtGo_R);

// 启动三轴联动
RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(
    1,  // X轴: 启动
    1,  // Y轴: 启动
    2,  // Z轴: 不动
    1,  // X2轴(R轴): 启动
    2,  // Y2轴: 不动
    2   // Z2轴: 不动
```

```
);

// 等待所有轴完成
if (0 == CheckRunFinishStatus_Uniaxial_X() &&
    0 == CheckRunFinishStatus_Uniaxial_Y() &&
    0 == CheckRunFinishStatus_Uniaxial_X2())
{
    // 所有轴运动完成
}
```

注意事项

1. 回原点的两种模式

> 🎯 **快速参考：**
>
> - **推荐方式：** `Let_MCU_Response_PC_KeySec(7);` （一行代码搞定）
> - **备用方式：** 手动移动到(0,0,0) （适用于特殊场景）
> - **回原点标志：** `Gu8_R_HomingReady` （1=已回原点，0=未回原点）

系统提供两种回原点的方式，推荐使用第一种。

方式一：按键触发回原点（⭐ 推荐）

说明： 调用运动卡内部的回原点功能，自动执行完整的回原点流程。

使用方法：

```
// 触发回原点操作
Let_MCU_Response_PC_KeySec(7);  // 编码7代表回原点按键
```

特点：

- ✅ 系统自动控制各轴顺序
- ✅ 内置安全保护机制
- ✅ 回原点速度和加速度已优化
- ✅ 自动检测原点信号
- ✅ 回原点完成后自动设置 `Gu8_R_HomingReady = 1` 标志

推荐理由：

- 此方式经过系统优化，安全可靠
- 无需手动控制各轴运动顺序
- 减少编程复杂度和出错概率

方式二：移动到(0,0,0)坐标

说明： 手动设置各轴目标位置为0，然后控制轴移动到原点。

使用方法：

```
// 设置速度
Run_Set_RunSpeedX(GtSystemSetData.u32WorkSpeedX);
Run_Set_RunSpeedY(GtSystemSetData.u32WorkSpeedY);
```

```
Run_Set_RunSpeedZ(GtSystemSetData.u32WorkSpeedZ);
Run_Set_RunSpeedX2(GtSystemSetData.u32WorkSpeedR);

// 选择Y轴
GtKeyMove.u8_AxisY_Sec = 1; // 选择Y1

// 设置各轴目标位置为0
GtKeyMove.tPoint_X.u8Sign = 0;
GtKeyMove.tPoint_X.u32Data = 0;

GtKeyMove.tPoint_Y.u8Sign = 0;
GtKeyMove.tPoint_Y.u32Data = 0;

GtKeyMove.tPoint_Z.u8Sign = 0;
GtKeyMove.tPoint_Z.u32Data = 0;

GtKeyMove.tPoint_R.u8Sign = 0;
GtKeyMove.tPoint_R.u32Data = 0;

// 执行移动到原点
Let_GoTo_SpecifiedCoordinatesHandEyeMoveOrigin();
```

特点：

- ⚠️ 需要手动控制各轴移动顺序
- ⚠️ 需要注意防撞保护
- ⚠️ 不会触发原点信号检测
- ⚠️ 不会自动设置回原点完成标志

运动顺序（参考 `RunHandEyeMoveOrigin_Task()` 实现）：

```
1. Z轴先抬起到安全高度
2. 等待Z轴完成
3. R轴旋转到0度
4. 等待R轴完成
5. XY轴移动到(0,0)
6. 等待XY轴完成
7. Z轴下降到0位置
```

注意事项：

- 此方式仅移动到机械坐标系的(0,0,0)位置
- 不等同于执行真正的回原点操作
- 通常用于特殊场景或测试用途

两种方式对比

对比项	方式一：按键触发回原点	方式二：移动到(0,0,0)
调用方式	`Let_MCU_Response_PC_KeySec(7)`	设置坐标后调用移动函数
实现复杂度	⭐ 简单（一行代码）	复杂（需要状态机控制）
安全性	⭐ 高（内置保护）	中（需手动保护）

对比项	方式一：按键触发回原点	方式二：移动到(0,0,0)
原点信号检测	⭐ 自动检测	❌ 不检测
回原点标志	⭐ 自动设置	❌ 不设置
适用场景	正常回原点操作	特殊测试场景
推荐指数	⭐⭐⭐⭐⭐	⭐⭐

---

2. 移动到新位置的运动顺序建议

为了避免碰撞，建议按照以下顺序：

移动到新位置时:

```
1. Z轴抬起到安全高度
2. 等待Z轴完成
3. XY轴水平移动
4. 等待XY轴完成
5. R轴旋转（如需要）
6. 等待R轴完成（如需要）
7. Z轴下降到工作高度
```

---

3. 状态检测

- 必须在状态机中轮询检测运动完成状态
- 不要在同一个case中既启动运动又检测完成
- 使用 `0 ==` 判断运动完成

错误示例:

```
// ❌ 错误：在同一case中启动和检测
case 1:
    WillRunToX_Uniaxial(ref PRtGo_X);
    RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(1, 2, 2, 2, 2, 2);

    if (0 == CheckRunFinishStatus_Uniaxial_X())  // 这里会立即返回0，因为还没开始运动
    {
        // 永远不会执行
    }
    break;
```

正确示例:

```
// ✅ 正确：分成两个case
case 1:
    WillRunToX_Uniaxial(ref PRtGo_X);
    RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(1, 2, 2, 2, 2, 2);
    PRu8_Step = 2;
    break;
```

```
case 2:
    if (0 == CheckRunFinishStatus_Uniaxial_X())
    {
        // 运动完成，继续下一步
        PRu8_Step = 3;
    }
    break;
```

---

4. 边界检查

在设置目标位置前，应检查：

- 目标位置是否超出机械行程范围
- 相对移动是否会导致越界
- 移动路径是否安全（避免碰撞）

```
// 示例：检查边界
if (targetPosition > MAX_POSITION || targetPosition < MIN_POSITION)
{
    MessageBox.Show("目标位置超出范围!", "错误",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}
```

---

5. 多轴联动注意事项

- 多轴同时运动时，所有轴必须同时启动（调用一次 `RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2`）
- 检测完成时，必须检测所有参与运动的轴
- 速度需要提前设置好

```
// 三轴联动
WillRunToX_Uniaxial(ref PRtGo_X);
WillRunToY_Uniaxial(ref PRtGo_Y);
WillRunToX2_Uniaxial(ref PRtGo_R);

// 一次性启动所有轴
RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(1, 1, 2, 1, 2, 2);

// 必须检测所有轴
if (0 == CheckRunFinishStatus_Uniaxial_X() &&
    0 == CheckRunFinishStatus_Uniaxial_Y() &&
    0 == CheckRunFinishStatus_Uniaxial_X2())
{
    // 所有轴完成
}
```

---

6. 速度设置时机

- 速度必须在启动运动**之前**设置
- 建议在任务开始时统一设置
- 中途可以修改速度（用于不同工作阶段）

```
// ✅ 正确：先设置速度
Run_Set_RunSpeedX(speed);
WillRunToX_Uniaxial(ref PRtGo_X);
RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(1, 2, 2, 2, 2, 2);

// ❌ 错误：速度设置在启动之后
WillRunToX_Uniaxial(ref PRtGo_X);
RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(1, 2, 2, 2, 2, 2);
Run_Set_RunSpeedX(speed);  // 这时候速度设置已经晚了
```

---

7. 常用全局变量

```
// 当前各轴实时位置（从机返回）
Gt_R_DataX     // X轴当前位置
Gt_R_DataY     // Y轴当前位置
Gt_R_DataZ     // Z轴当前位置
Gt_R_DataR1    // R1轴当前位置
Gt_R_DataR2    // R2轴当前位置

// 运动目标位置（本地设置）
PRtGo_X        // X轴目标位置
PRtGo_Y        // Y轴目标位置
PRtGo_Z        // Z轴目标位置
PRtGo_R        // R轴目标位置

// 系统配置参数
GtSystemSetData.u32WorkSpeedX      // X轴工作速度
GtSystemSetData.u32WorkSpeedY      // Y轴工作速度
GtSystemSetData.u32WorkSpeedZ      // Z轴工作速度
GtSystemSetData.u32WorkSpeedR      // R轴工作速度

// 按键/模板中的目标点
GtKeyMove.tPoint_X                 // 目标X坐标
GtKeyMove.tPoint_Y                 // 目标Y坐标
GtKeyMove.tPoint_Z                 // 目标Z坐标
GtKeyMove.tPoint_R                 // 目标R坐标
GtKeyMove.u8_AxisY_Sec             // Y轴选择 (1=Y1, 2=Y2)
```

---

8. Y1/Y2轴选择

系统支持两个Y轴，需要根据 `GtKeyMove.u8_AxisY_Sec` 选择使用哪个：

```
if (1 == GtKeyMove.u8_AxisY_Sec)  // 使用Y1轴
{
    WillRunToY_Uniaxial(ref PRtGo_Y);
    RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(1, 1, 2, 2, 2, 2);
}
else  // 使用Y2轴
{
    WillRunToY2_Uniaxial(ref PRtGo_Y);
    RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(1, 2, 2, 2, 1, 2);
}
```

常见应用场景

场景1: 回原点操作（推荐方式）

适用于: 开机初始化、换料、紧急停止后复位

示例代码:

```
// 方式一：推荐使用按键触发（简单安全）
private void btnGoHome_Click(object sender, EventArgs e)
{
    // 检查是否已经回过原点
    if (1 == Gu8_R_HomingReady)
    {
        MessageBox.Show("已经完成回原点!", "提示",
                        MessageBoxButtons.OK, MessageBoxIcon.Information);
        return;
    }

    // 触发回原点操作
    Let_MCU_Response_PC_KeySec(7);  // 编码7代表回原点按键

    // 系统会自动执行完整的回原点流程:
    // 1. Z轴抬起
    // 2. R轴回零
    // 3. XY轴回原点
    // 4. 检测原点信号
    // 5. 设置 Gu8_R_HomingReady = 1
}

// 检查回原点是否完成
if (1 == Gu8_R_HomingReady)
{
    // 回原点已完成，可以进行后续操作
}
else
{
    MessageBox.Show("请先执行回原点操作!", "提示",
                    MessageBoxButtons.OK, MessageBoxIcon.Warning);
}
```

场景2: 点对点移动（Pick and Place）

```
// 适用于：拾取物料、放置物料
// 流程：抬起Z → 移动XY → 旋转R → 下降Z
```

场景3: 扫描路径

```
// 适用于：涂胶、绘制图案
// 流程：移动到起点 → 下降Z → 沿路径移动 → 抬起Z
```

场景4: 循环往复运动

```
// 适用于：刷涂、振动送料
// 流程：使用状态机循环，判断传感器信号切换方向
```

场景5: 多工位加工

```
// 适用于：多个加工位置
// 流程：循环遍历各工位坐标，依次移动和加工
```

附录：完整状态机模板

```
private u8 PRu8_MyTask_Step = 0;
private u8 PRu8_MyTask_Start = 0;
private u32 PRu32_MyTask_TimerCnt = 0;

private void MyTask()
{
    // 任务启动控制
    if (0 != PRu8_MyTask_Step && 0 == PRu8_MyTask_Start)
    {
        PRu8_MyTask_Step = 0;
    }

    switch (PRu8_MyTask_Step)
    {
        case 0:  // 等待启动
            if (1 == PRu8_MyTask_Start)
            {
                // 设置速度
                Run_Set_RunSpeedX(GtSystemSetData.u32WorkSpeedX);
                Run_Set_RunSpeedY(GtSystemSetData.u32WorkSpeedY);
                Run_Set_RunSpeedZ(GtSystemSetData.u32WorkSpeedZ);

                PRu8_MyTask_Step = 1;
            }
            break;
```

```
        case 1:  // Z轴抬起
            PRtGo_Z.u8Sign = 0;
            PRtGo_Z.u32Data = 0;

            WillRunToZ_Uniaxial(ref PRtGo_Z);
            RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(2, 2, 1, 2, 2, 2);

            PRu8_MyTask_Step = 2;
            break;

        case 2:  // 等待Z轴完成
            if (0 == CheckRunFinishStatus_Uniaxial_Z())
            {
                PRu8_MyTask_Step = 3;
            }
            break;

        case 3:  // XY轴移动
            PRtGo_X.u8Sign = GtKeyMove.tPoint_X.u8Sign;
            PRtGo_X.u32Data = GtKeyMove.tPoint_X.u32Data;

            PRtGo_Y.u8Sign = GtKeyMove.tPoint_Y.u8Sign;
            PRtGo_Y.u32Data = GtKeyMove.tPoint_Y.u32Data;

            WillRunToX_Uniaxial(ref PRtGo_X);
            WillRunToY_Uniaxial(ref PRtGo_Y);

            RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(1, 1, 2, 2, 2, 2);

            PRu8_MyTask_Step = 4;
            break;

        case 4:  // 等待XY轴完成
            if (0 == CheckRunFinishStatus_Uniaxial_X() &&
                0 == CheckRunFinishStatus_Uniaxial_Y())
            {
                PRu8_MyTask_Step = 5;
            }
            break;

        case 5:  // R轴旋转
            PRtGo_R.u8Sign = GtKeyMove.tPoint_R.u8Sign;
            PRtGo_R.u32Data = GtKeyMove.tPoint_R.u32Data;

            WillRunToX2_Uniaxial(ref PRtGo_R);
            RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(2, 2, 2, 1, 2, 2);

            PRu8_MyTask_Step = 6;
            break;

        case 6:  // 等待R轴完成
            if (0 == CheckRunFinishStatus_Uniaxial_X2())
            {
                PRu8_MyTask_Step = 7;
            }
```

```
            break;

        case 7:  // Z轴下降
            PRtGo_Z.u8Sign = GtKeyMove.tPoint_Z.u8Sign;
            PRtGo_Z.u32Data = GtKeyMove.tPoint_Z.u32Data;

            WillRunToZ_Uniaxial(ref PRtGo_Z);
            RunUniaxial_StartStop_X_Y_Z_X2_Y2_Z2(2, 2, 1, 2, 2, 2);

            PRu8_MyTask_Step = 8;
            break;

        case 8:  // 等待Z轴完成
            if (0 == CheckRunFinishStatus_Uniaxial_Z())
            {
                // 任务完成
                PRu8_MyTask_Start = 0;
                PRu8_MyTask_Step = 0;
            }
            break;
    }
}

// 启动任务
public void StartMyTask()
{
    PRu8_MyTask_Start = 1;
    PRu8_MyTask_Step = 0;
}
```