# AGV Programming Guide

# 版本历史

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 作者 | 日期 | 备注 |
| V1.0 | 王咏斌 |  |  |
|  |  |  |  |
|  |  |  |  |

# 目录

[AGV Programming Guide 1](#_Toc3794611)

[版本历史 1](#_Toc3794612)

[目录 2](#_Toc3794613)

[1. 引言 3](#_Toc3794614)

[2 AGV开发板 4](#_Toc3794615)

[2.1 AGV控制板 4](#_Toc3794616)

[2.2 接近开关 4](#_Toc3794617)

[2.3 继电器 5](#_Toc3794618)

[2.4 超声波避障 6](#_Toc3794619)

[3 IO接口定义 6](#_Toc3794620)

[4 代码 7](#_Toc3794622)

[4.1 Main.C 7](#_Toc3794623)

[4.2 Input.C 9](#_Toc3794624)

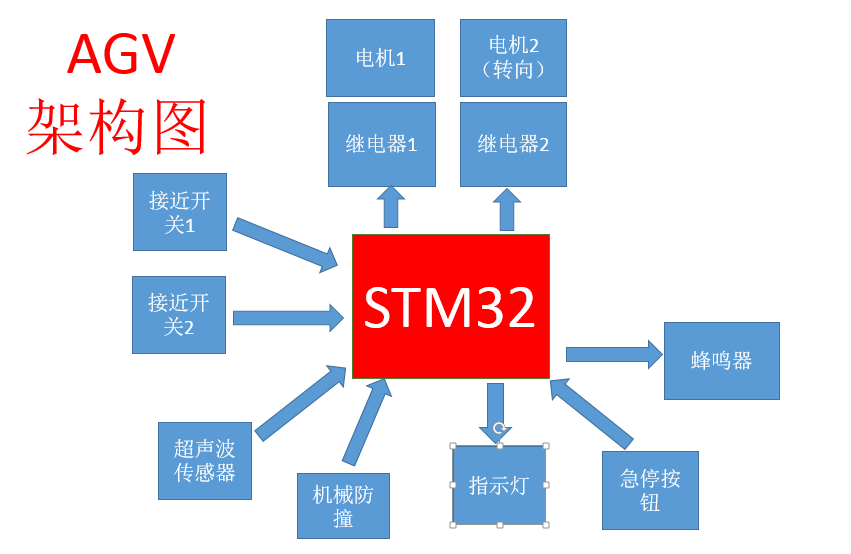
[4.3 Output.C 10](#_Toc3794625)

# 引言

为了实现智能制造，机器换人，加快物流运输效率以提高公司利润，所以立项研发AGV小车。

AGV小车优势：

1. 工作效率高
2. 成本费用较低
3. 节省管理精力
4. 可靠性高
5. 较好的柔性和系统拓展性
6. 安全性高
7. AGV小车整体框架图



# AGV开发板

## AGV控制板

采用基于STM32F407VET6芯片的单片机进行AGV小车的研发。目前STM32控制板是从淘宝上购买。

单机片具有以下优势：

1. 有优异的性能价格比
2. 集成度高，体积小，可靠性好
3. 控制能力强
4. 低功耗，低电压
5. 易扩展

## 接近开关

目前采用接近开关进行导航，分别为左接近开关和右接近开关。

当左接近开关检测到轨道，表示小车偏右，需要左转。同理，当右接近开关检测到轨道，表示小车偏左，需要右转。

## 继电器

继电器负责控制舵轮的转向。

当接近开关检测到信号，信号传输到STM32开发板，再经过开发板处理，输出到继电器，通过继电器控制舵轮的转向。

## 超声波避障

# IO接口定义

## 



# 代码

## Main.C

Main.C 根据输入的信号，进行控制分析，最后输出信号，控制相关外设。

int main(void)

{

//// PE0 IN1 左接近开关

//// PE1 IN2 右接近开关

//// PE2 IN3 停车按钮

//// PC9 OUT1 左转继电器

//// PC8 OUT2 右转继电器

//// PD15 OUT3 前进继电器

////关于输入 0 1的问题

////GPIO口接三极管，上拉电阻，GPIO输入口信号是低电平 0

////当in口接收到低电平(检测到信号)，GPIO输入口信号是高电平 1

////当in口接收到高电平，GPIO输入口信号是低电平 0

////关于输出 0 1的问题

////GPIO口接三极管，上拉电阻，out口默认是高电平

////当GPIO 输出 1，三极管导通，out口输出低电平 继电器闭合

////当GPIO 输出 O，三极管截止，out口输出高电平 继电器断开

//read\_PE0 ()//左边接近开关信号

//read\_PE1 ()//右边接近开关信号

//read\_PE2 () //启动按钮信号

u8 read\_PE0,read\_PE1,read\_PE2;

//Stm32\_Clock\_Init(336,8,2,7); //SYSCLK = 168MHZ

delay\_init(168); //延时初始化

LED\_Init(); //LED初始化

INPUT\_Init(); //输入口初始化

OUTPUT\_Init(); //输出口初始化

//uart\_init(9600);

while (1)

{

//获取输入信号

read\_PE0 = GPIO\_ReadInputDataBit(GPIOE,GPIO\_Pin\_0); //获取GPIO E 第0 PIN信号

read\_PE1 = GPIO\_ReadInputDataBit(GPIOE,GPIO\_Pin\_1);

read\_PE2 = GPIO\_ReadInputDataBit(GPIOE,GPIO\_Pin\_2);

if (read\_PE0 == 1 & read\_PE1 == 0) //当左边接近开关收到信号 左转继电器闭合

{

GPIO\_SetBits(GPIOC,GPIO\_Pin\_9); //置1 PC9 左转继电器闭合

GPIO\_ResetBits(GPIOC,GPIO\_Pin\_8);//置0 PC8 右转继电器断开

}

else if (read\_PE1 == 1 & read\_PE0 == 0) //当右边接近开关收到信号 右转继电器闭合

{

GPIO\_ResetBits(GPIOC,GPIO\_Pin\_9);//置0 PC9 左转继电器断开

GPIO\_SetBits(GPIOC,GPIO\_Pin\_8); //置1 PC8 右转继电器闭合

}

else //没有信号的话，2个继电器断开

{

GPIO\_ResetBits(GPIOC,GPIO\_Pin\_9);//置0 PC9 左转继电器断开

GPIO\_ResetBits(GPIOC,GPIO\_Pin\_8);//置0 PC8 右转继电器断开

}

if (read\_PE2 == 1) ///当收到启动按钮信号，前进继电器闭合

{

GPIO\_SetBits(GPIOD,GPIO\_Pin\_15); //置1 PD15 前进继电器闭合

}

else GPIO\_ResetBits(GPIOD,GPIO\_Pin\_15); //置0 PD15 前进继电器断开

}

}

## Input.C

Input.C 作用是输入口的参数初始化:

1. GPIO口时钟使能
2. GPIO初始化，选择GPIO组，pin角，模式

void INPUT\_Init(void)

{

///PE0代表 GPIO\_E\_Pin0

/// PE0 IN1 左接近开关

/// PE1 IN2 右接近开关

/// PE2 IN3 启动按钮

GPIO\_InitTypeDef GPIO\_InitStructure;

//IO口时钟使能

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOE, ENABLE);

//IO PE0 初始化

GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_0; //第0 PIN

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_IN; //模式选择为输入模式

GPIO\_Init(GPIOE,&GPIO\_InitStructure); // GPIO E

//IO PE1 初始化

GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_1;

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_IN;

GPIO\_Init(GPIOE,&GPIO\_InitStructure);

//IO PE2 初始化

GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_2;

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_IN;

GPIO\_Init(GPIOE,&GPIO\_InitStructure);

///可以写成 PIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_0|GPIO\_Pin\_1|GPIO\_Pin\_2;

}

## Output.C

output.C 作用是输入口的参数初始化:

1. GPIO口参数初始化
2. GPIO初始化，选择GPIO组，pin角，模式和输出速度

void OUTPUT\_Init(void)

{

///PC9代表 GPIO\_C\_Pin9

//// PC9 OUT1 左转继电器

//// PC8 OUT2 右转继电器

//// PD15 OUT3 前进继电器

GPIO\_InitTypeDef GPIO\_InitStructure;

//IO口时钟使能

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOC, ENABLE); //使能时钟GPIOC

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOD, ENABLE); //使能时钟GPIOD

//PC9 初始化

GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_9; //第9Pin

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_OUT; //通用输出模式

GPIO\_InitStructure.GPIO\_OType=GPIO\_OType\_PP; //推挽输出

GPIO\_InitStructure.GPIO\_Speed=GPIO\_Speed\_50MHz; //速度控制

GPIO\_Init(GPIOC,&GPIO\_InitStructure);

//PC8 初始化

GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_8;

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_OUT;

GPIO\_InitStructure.GPIO\_OType=GPIO\_OType\_PP; //推挽输出

GPIO\_InitStructure.GPIO\_Speed=GPIO\_Speed\_50MHz;

GPIO\_Init(GPIOC,&GPIO\_InitStructure);

//PD15 初始化

GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_15;

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_OUT;

GPIO\_InitStructure.GPIO\_OType=GPIO\_OType\_PP; //推挽输出

GPIO\_InitStructure.GPIO\_Speed=GPIO\_Speed\_50MHz;

GPIO\_Init(GPIOD,&GPIO\_InitStructure);

}