电机驱动实验

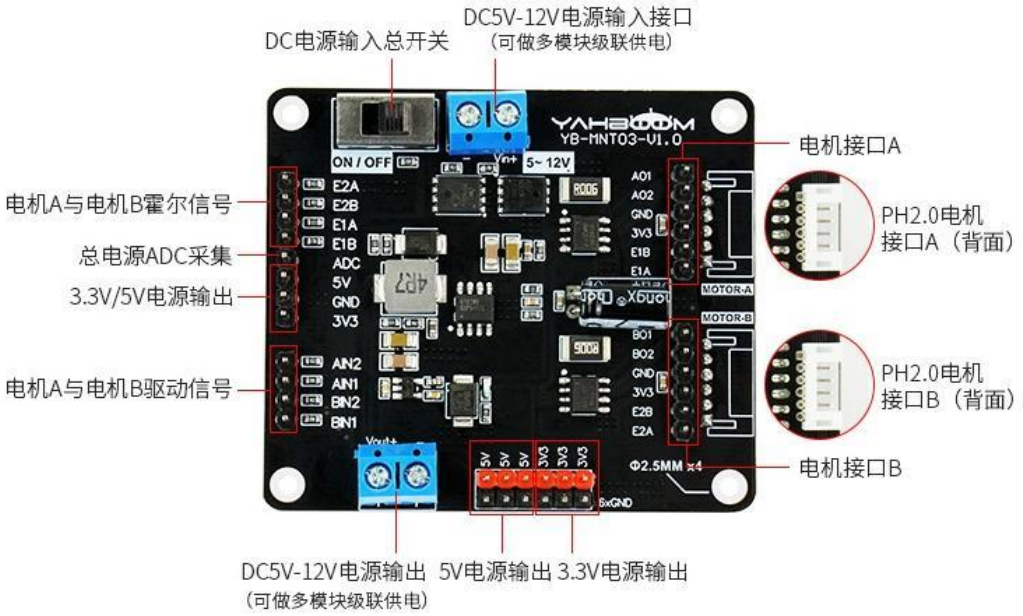
1、实验目的

烧录电机驱动代码，将stm32f103rct6主控板上电后，启动小车的电机驱动功能，实验效果为小车前进1s，后退1s，左旋1s， 右旋1s， 停止，然后循环以上步骤。

2、实验原理

对于4路编码器电机310的控制我们采用的是两个双路电机驱动板来驱动电机。通过控制驱动芯片的AIN1,AIN2,BIN1,BIN2的电平高低来控制电机的正转，反转，停止；通过驱动板上的E1A，E1B，E2A，E2B来获取电机的编码值，进而实现pid调速。

双路电机驱动模块：



引脚详细说明					
接口类型	引脚名称	引脚说明	接口类型	引脚名称	引脚说明
单片机/主控端接口	E1A	电机1霍尔信号A	电机端接口	AO1	电机1电源+
	E1B	电机1霍尔信号B		AO2	电机1电源-
	E2A	电机2霍尔信号A		GND	GND
	E2B	电机2霍尔信号B		3V3	电机1霍尔供电
	ADC	采集VM输入电压		E1B	电机1霍尔信号B
	5V	输出5V3A电源		E1A	电机1霍尔信号A
	GND	GND		BO1	电机2电源+
	3V3	输出3.3V电压		BO2	电机2电源-
	AIN1	电机1驱动信号1		GND	GND
	AIN2	电机1驱动信号2		3V3	电机2霍尔供电
	BIN1	电机2驱动信号1		E2B	电机2霍尔信号B
	BIN2	电机2驱动信号2		E2A	电机2霍尔信号A

双路电机驱动方式：

输入管脚IN1/IN2控制H桥的输出状态，下表是输入输出间的逻辑关系：

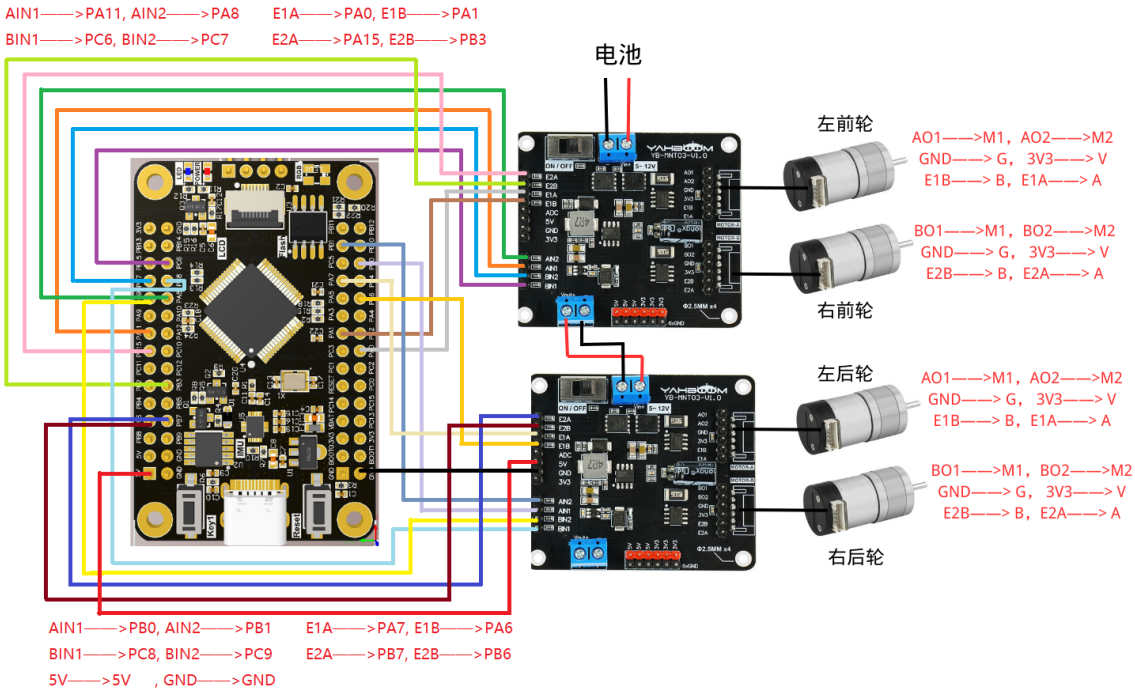
IN1	IN2	OUT1	OUT2	功能
0	0	Z	Z	滑行，休眠
1	0	H	L	正向
0	1	L	H	反向
1	1	L	L	刹车

当使用PWM控制来实现调速功能时，H桥可以操作在两种不同的状态，快衰减或者慢衰减。在快衰减模式，H桥是被禁止的，续流电流流经二极管；在慢衰减模式，输出H桥的两个下管都是打开的。

IN1	IN2	功能
PWM	0	正转PWM，快衰减
1	PWM	正转PWM，慢衰减
0	PWM	反转PWM，快衰减
PWM	1	反转PWM，慢衰减

3、实验步骤

将两个双路电机驱动板与STM32主控板按下图连接起来，只是控制电机不使用pid调速可以不连接两路驱动板的E1A，E1B，E2A，E2B接口，但若是要pid调速则必须全部按图连接。



注意：310电机的接线需要按图中引脚标注连接，不可以随意连接，也不可自行更换成排插，否则会烧毁驱动板！

接线完成之后，若只想驱动小车行走不用pid调速，则烧录无pid调速代码；若想实现pid调速则需要烧录有pid调速的电机驱动代码。烧录代码成功之后，小车则前进1s，后退1s，左旋1s， 右旋1s， 停止，然后重复循环以上步骤。

4、主要代码展示

app_motion.c代码：

```
#include "app_motion.h"
// 编码器10毫秒前后数据
int g_Encoder_All_Now[MAX_MOTOR] = {0};
int g_Encoder_All_Last[MAX_MOTOR] = {0};

int g_Encoder_All_Offset[MAX_MOTOR] = {0};

uint8_t g_start_ctrl = 0;

car_data_t car_data;
motor_data_t motor_data;

uint8_t g_yaw_adjust = 0;
car_type_t g_car_type = CAR_FOURWHEEL; //CAR_MECANUM;

static float Motion_Get_Circle_Pulse(void)
{
    float temp = 0;
    switch (g_car_type)
    {
    case CAR_MECANUM:
        temp = ENCODER_CIRCLE_330;
        break;
    case CAR_MECANUM_MAX:
        temp = ENCODER_CIRCLE_205;
        break;
    case CAR_FOURWHEEL:
        temp = ENCODER_CIRCLE_330;
        break;
    case CAR_ACKERMAN:
        temp = ENCODER_CIRCLE_550;
        break;
    default:
        temp = ENCODER_CIRCLE_330;
        break;
    }
    return temp;
}

#if ENABLE_REAL_WHEEL
// 实际轮子转的圈数，单位：XX转/分钟
int real_circle[MAX_MOTOR] = {0};

// 实际轮子的速度，单位：m/s
float real_circle_speed[MAX_MOTOR] = {0};

// 实际小车角速度
float real_motion_angular = 0.0;

void* Motion_Real_Circle_Speed(uint8_t index)
{
```

```
    if (index == 1) return (int*) real_circle;
    if (index == 2) return (float*) real_circle_speed;
    return NULL;
}
#endif

// 仅用于添加到调试中显示数据。
void* Motion_Get_Data(uint8_t index)
{
    if (index == 1) return (int*)g_Encoder_All_Now;
    if (index == 2) return (int*)g_Encoder_All_Last;
    if (index == 3) return (int*)g_Encoder_All_Offset;
    return NULL;
}

// 获取电机速度
void Motion_Get_Motor_Speed(float* speed)
{
    for (int i = 0; i < 4; i++)
    {
        speed[i] = motor_data.speed_mm_s[i];

    }
}

// 设置偏航角状态，如果使能则刷新target目标角度。
void Motion_Set_Yaw_Adjust(uint8_t adjust)
{
    if (adjust == 0)
    {
        g_yaw_adjust = 0;
    }
    else
    {
        g_yaw_adjust = 1;
    }
    if (g_yaw_adjust)
    {
        #if ENABLE_INV_MEMS
        PID_Yaw_Reset(ICM20948_Get_Yaw_Now());
        #elif ENABLE_MPU9250
        PID_Yaw_Reset(MPU_Get_Yaw_Now());
        #endif
    }
}

// 返回偏航角调节状态。
uint8_t Motion_Get_Yaw_Adjust(void)
{
    return g_yaw_adjust;
}

// 控制小车运动，Motor_X=[-3600, 3600]，超过范围则无效。
void Motion_Set_Pwm(int16_t Motor_1, int16_t Motor_2, int16_t Motor_3, int16_t
Motor_4)
```

```
{
    if (Motor_1 >= -MOTOR_MAX_PULSE && Motor_1 <= MOTOR_MAX_PULSE)
    {
        Motor_Set_Pwm(MOTOR_ID_M1, Motor_1);
    }
    if (Motor_2 >= -MOTOR_MAX_PULSE && Motor_2 <= MOTOR_MAX_PULSE)
    {
        Motor_Set_Pwm(MOTOR_ID_M2, Motor_2);
    }
    if (Motor_3 >= -MOTOR_MAX_PULSE && Motor_3 <= MOTOR_MAX_PULSE)
    {
        Motor_Set_Pwm(MOTOR_ID_M3, Motor_3);
    }
    if (Motor_4 >= -MOTOR_MAX_PULSE && Motor_4 <= MOTOR_MAX_PULSE)
    {
        Motor_Set_Pwm(MOTOR_ID_M4, Motor_4);
    }
}

// 小车停止
void Motion_Stop(uint8_t brake)
{
    Motion_Set_Speed(0, 0, 0, 0);
//      PID_Clear_Motor(MAX_MOTOR);
    g_start_ctrl = 0;
    g_yaw_adjust = 0;
    Motor_Stop(brake);
}
    // speed_mX=[-1000, 1000], 单位为：mm/s
    void Motion_Set_Speed(int16_t speed_m1, int16_t speed_m2, int16_t speed_m3,
int16_t speed_m4)
    {
        g_start_ctrl = 1;
        motor_data.speed_set[0] = speed_m1;
        motor_data.speed_set[1] = speed_m2;
        motor_data.speed_set[2] = speed_m3;
        motor_data.speed_set[3] = speed_m4;
        for (uint8_t i = 0; i < MAX_MOTOR; i++)
        {
            PID_Set_Motor_Target(i, motor_data.speed_set[i]*1.0);
        }
    }
```

```
// 从编码器读取当前各轮子速度，单位mm/s
void Motion_Get_Speed(car_data_t* car)
{
    int i = 0;
    float speed_mm[MAX_MOTOR] = {0};
    float circle_mm = Motion_Get_Circle_MM();
    float circle_pulse = Motion_Get_Circle_Pulse();
    float robot_APB = Motion_Get_APB();

    Motion_Get_Encoder();

    // 计算轮子速度，单位mm/s。
```

```
    for (i = 0; i < 4; i++)
    {
        speed_mm[i] = (g_Encoder_All_Offset[i]) * 100 * circle_mm /
circle_pulse;
    }

        car->Vx = (speed_mm[0] + speed_mm[1] + speed_mm[2] + speed_mm[3]) / 4;
    car->Vy = -(speed_mm[0] - speed_mm[1] - speed_mm[2] + speed_mm[3]) / 4;
    car->Vz = -(speed_mm[0] + speed_mm[1] - speed_mm[2] - speed_mm[3]) / 4.0f /
robot_APB * 1000;

    if (g_start_ctrl)
    {
        for (i = 0; i < MAX_MOTOR; i++)
        {
            motor_data.speed_mm_s[i] = speed_mm[i];
        }

        #if ENABLE_YAW_ADJUST
        if (g_yaw_adjust)
        {
            #if ENABLE_INV_MEMS
            Mecanum_Yaw_Calc(ICM20948_Get_Yaw_Now());
            #elif ENABLE_MPU9250
            Mecanum_Yaw_Calc(MPU_Get_Yaw_Now());
            #endif
        }
        #endif
        PID_Calc_Motor(&motor_data);

        #if PID_ASSISTANT_EN
        if (start_tool())
        {
            int32_t speed_send = car->Vx;
            // int32_t speed_send = (int32_t)speed_m1;
            set_computer_value(SEND_FACT_CMD, CURVES_CH1, &speed_send, 1);
        }
        #endif
    }
}

// 返回当前小车轮子轴间距和的一半
float Motion_Get_APB(void)
{
//    if (g_car_type == CAR_MECANUM) return MECANUM_APB;
//    if (g_car_type == CAR_MECANUM_MAX) return MECANUM_MAX_APB;
//    if (g_car_type == CAR_MECANUM_MINI) return MECANUM_MINI_APB;
//    if (g_car_type == CAR_ACKERMAN) return AKM_WIDTH;
//    if (g_car_type == CAR_FOURWHEEL) return FOURWHEEL_APB;
    return MECANUM_APB;
}

// 返回当前小车轮子转一圈的多少毫米
float Motion_Get_Circle_MM(void)
{
```

```
//    if (g_car_type == CAR_MECANUM) return MECANUM_CIRCLE_MM;
//    if (g_car_type == CAR_MECANUM_MAX) return MECANUM_MAX_CIRCLE_MM;
//    if (g_car_type == CAR_MECANUM_MINI) return MECANUM_MINI_CIRCLE_MM;
//    if (g_car_type == CAR_ACKERMAN) return AKM_CIRCLE_MM;
//    if (g_car_type == CAR_FOURWHEEL) return FOURWHEEL_CIRCLE_MM;
    return MECANUM_CIRCLE_MM;
}

// 设置当前控制的小车类型。
void Motion_Set_Car_Type(car_type_t car_type)
{
    if (car_type >= CAR_TYPE_MAX) return;
    if (g_car_type == car_type) return;
    g_car_type = car_type;
    // Flash_Set_CarType(g_car_type);
}

// 返回当前控制的小车类型。
uint8_t Motion_Get_Car_Type(void)
{
    return (uint8_t)g_car_type;
}

// 获取编码器数据，并计算偏差脉冲数
void Motion_Get_Encoder(void)
{
    Encoder_Get_ALL(g_Encoder_All_Now);

    for(uint8_t i = 0; i < MAX_MOTOR; i++)
    {
        // 记录两次测试时间差的脉冲数
        g_Encoder_All_Offset[i] = g_Encoder_All_Now[i] - g_Encoder_All_Last[i];
        // 记录上次编码器数据
        g_Encoder_All_Last[i] = g_Encoder_All_Now[i];

    #if ENABLE_REAL_WHEEL
        // 计算每分钟转多少圈，10毫秒测到的脉冲数*100变秒*60变分钟/每一圈的脉冲数
        real_circle[i] = g_Encoder_All_Offset[i] * 60 * 100 /
Motion_Get_Circle_Pulse();
        // 计算轮子速度，单位m/s。每分钟转的圈数*转一圈运动的距离/60得到每秒的米数
        real_circle_speed[i] = real_circle[i] * (Motion_Get_Circle_MM() / 1000.0)
/ 60.0;
    #endif
    }
}

// 控制小车运动
void Motion_Ctrl(int16_t V_x, int16_t V_y, int16_t V_z, uint8_t adjust)
{
     Mecanum_Ctrl(V_x, V_y, V_z, adjust);
}

// 控制小车的运动状态
void Motion_Ctrl_State(uint8_t state, uint16_t speed, uint8_t adjust)
{
```

```
    uint16_t input_speed = speed * 10;

        Mecanum_State(state, input_speed, adjust);
}

// 运动控制句柄，每10ms调用一次，主要处理速度相关的数据
void Motion_Handle(void)
{
    Motion_Get_Speed(&car_data);

    if (g_start_ctrl)
    {
        Motion_Set_Pwm(motor_data.speed_pwm[0], motor_data.speed_pwm[1],
motor_data.speed_pwm[2], motor_data.speed_pwm[3]);
    }
}
```

app_mecanum.c代码
#include "app_mecanum.h"

static float speed_lr = 0;
static float speed_fb = 0;
static float speed_spin = 0;

```
static int speed_L1_setup = 0;
static int speed_L2_setup = 0;
static int speed_R1_setup = 0;
static int speed_R2_setup = 0;

static int g_offset_yaw = 0;
uint16_t g_speed_setup = 0;

// X轴速度(前正后负：±1000)，Y轴速度(左正右负：±1000)，旋转速度(左正右负：±5000)
// X3 PLUS X轴速度(前正后负：±700)，Y轴速度(左正右负：±700)，旋转速度(左正右负：±3200)
void Mecanum_Ctrl(int16_t V_x, int16_t V_y, int16_t V_z, uint8_t adjust)
{
    float robot_APB = Motion_Get_APB();
    speed_lr = -V_y;
    speed_fb = V_x;
    speed_spin = (-V_z / 1000.0f) * robot_APB;
    if (V_x == 0 && V_y == 0 && V_z == 0)
    {
        Motion_Stop(STOP_BRAKE);
        return;
    }

    speed_L1_setup = speed_fb + speed_lr + speed_spin;
    speed_L2_setup = speed_fb - speed_lr + speed_spin;
    speed_R1_setup = speed_fb - speed_lr - speed_spin;
    speed_R2_setup = speed_fb + speed_lr - speed_spin;

    if (Motion_Get_Car_Type() == CAR_MECANUM_MAX)
```

```
    {
        if (speed_L1_setup > CAR_X3_PLUS_MAX_SPEED) speed_L1_setup =
CAR_X3_PLUS_MAX_SPEED;
        if (speed_L1_setup < -CAR_X3_PLUS_MAX_SPEED) speed_L1_setup = -
CAR_X3_PLUS_MAX_SPEED;
        if (speed_L2_setup > CAR_X3_PLUS_MAX_SPEED) speed_L2_setup =
CAR_X3_PLUS_MAX_SPEED;
        if (speed_L2_setup < -CAR_X3_PLUS_MAX_SPEED) speed_L2_setup = -
CAR_X3_PLUS_MAX_SPEED;
        if (speed_R1_setup > CAR_X3_PLUS_MAX_SPEED) speed_R1_setup =
CAR_X3_PLUS_MAX_SPEED;
        if (speed_R1_setup < -CAR_X3_PLUS_MAX_SPEED) speed_R1_setup = -
CAR_X3_PLUS_MAX_SPEED;
        if (speed_R2_setup > CAR_X3_PLUS_MAX_SPEED) speed_R2_setup =
CAR_X3_PLUS_MAX_SPEED;
        if (speed_R2_setup < -CAR_X3_PLUS_MAX_SPEED) speed_R2_setup = -
CAR_X3_PLUS_MAX_SPEED;
    }
    else
    {
        if (speed_L1_setup > 1000) speed_L1_setup = 1000;
        if (speed_L1_setup < -1000) speed_L1_setup = -1000;
        if (speed_L2_setup > 1000) speed_L2_setup = 1000;
        if (speed_L2_setup < -1000) speed_L2_setup = -1000;
        if (speed_R1_setup > 1000) speed_R1_setup = 1000;
        if (speed_R1_setup < -1000) speed_R1_setup = -1000;
        if (speed_R2_setup > 1000) speed_R2_setup = 1000;
        if (speed_R2_setup < -1000) speed_R2_setup = -1000;
    }
    Motion_Set_Speed(speed_L1_setup, speed_L2_setup, speed_R1_setup,
speed_R2_setup);
}

// 通过偏航角计算当前的偏差值，校准小车运动方向。
void Mecanum_Yaw_Calc(float yaw)
{
    float yaw_offset = PID_Yaw_Calc(yaw);
    g_offset_yaw = yaw_offset * g_speed_setup;

    // 用于调试打印偏差数据
    // static int aaaaaaaaa = 0;
    // aaaaaaaaa++;
    // if (aaaaaaaaa > 5)
    // {
    //     aaaaaaaaa = 0;
    //     DEBUG("Yaw Calc:%.5f, %d", yaw_offset, g_offset_yaw);
    // }

    int speed_L1 = speed_L1_setup - g_offset_yaw;
    int speed_L2 = speed_L2_setup - g_offset_yaw;
    int speed_R1 = speed_R1_setup + g_offset_yaw;
    int speed_R2 = speed_R2_setup + g_offset_yaw;

    if (speed_L1 > 1000) speed_L1 = 1000;
    if (speed_L1 < -1000) speed_L1 = -1000;
```

```
    if (speed_L2 > 1000) speed_L2 = 1000;
    if (speed_L2 < -1000) speed_L2 = -1000;
    if (speed_R1 > 1000) speed_R1 = 1000;
    if (speed_R1 < -1000) speed_R1 = -1000;
    if (speed_R2 > 1000) speed_R2 = 1000;
    if (speed_R2 < -1000) speed_R2 = -1000;
    Motion_Set_Speed(speed_L1, speed_L2, speed_R1, speed_R2);
}

// 控制麦克纳姆轮小车运动状态。
// 速度控制：speed=0~1000。
// 偏航角调节运动：adjust=1开启，=0不开启。
void Mecanum_State(uint8_t state, uint16_t speed, uint8_t adjust)
{
    Motion_Set_Yaw_Adjust(adjust);
    g_speed_setup = speed;
    switch (state)
    {
    case MOTION_STOP:
        g_speed_setup = 0;
        Motion_Stop(speed==0?STOP_FREE:STOP_BRAKE);
        break;
    case MOTION_RUN:
        Mecanum_Ctrl(speed, 0, 0, adjust);
        break;
    case MOTION_BACK:
        Mecanum_Ctrl(-speed, 0, 0, adjust);
        break;
    case MOTION_LEFT:
        Mecanum_Ctrl(0, speed, 0, adjust);
        break;
    case MOTION_RIGHT:
        Mecanum_Ctrl(0, -speed, 0, adjust);
        break;
    case MOTION_SPIN_LEFT:
        Motion_Set_Yaw_Adjust(0);
        Mecanum_Ctrl(0, 0, speed*5, 0);
        break;
    case MOTION_SPIN_RIGHT:
        Motion_Set_Yaw_Adjust(0);
        Mecanum_Ctrl(0, 0, -speed*5, 0);
        break;
    case MOTION_BRAKE:
        Motion_Stop(STOP_BRAKE);
        break;
    default:
        break;
    }
}
```