1.概述

使用PIBOT提供的小车已经完成了程序和硬件的调试，如果需要移植到自己的小车，可能会遇到PID调速不正常，解算得到运动结果不一致，反解得到里程有问题等，原因就是单个电机线的方向、电机之间接线、编码器AB相次序，这里需要注意下面几点即可

- a. 给定电机接口函数的输入参数正负与电机方向的关系
- b.电机索引关系
- c.电机转向得到编码器值正负关系

2.PIBOT运动解算和PID

```
void Robot::do_kinmatics(){
    if (!do_kinmatics_flag){
        for(int i=0;i<MOTOR_COUNT;i++){
            pid[i]->clear();
            encoder[i]->get_increment_count_for_dopid();
        }
        return;
    }

    static unsigned long last_millis=0;
    if (Board::get()->get_tick_count()-last_millis>=Data_holder::get()-
>parameter.do_pid_interval){
        last_millis = Board::get()->get_tick_count();

        for(int i=0;i<MOTOR_COUNT;i++){
            feedback[i] = encoder[i]->get_increment_count_for_dopid();
        }
#if PID_DEBUG_OUTPUT
    #if MOTOR_COUNT==2
        printf("input=%ld %ld feedback=%ld %ld\r\n", long(input[0]*1000),
long(input[1]*1000),
                                                    long(feedback[0]),
long(feedback[1]));
    #endif
```

```
    #if MOTOR_COUNT==3
        printf("input=%ld %ld %ld feedback=%ld %ld %ld\r\n", long(input[0]*1000),
long(input[1]*1000), long(input[2]*1000),
                                                        long(feedback[0]),
long(feedback[1]), long(feedback[2]));
    #endif
    #if MOTOR_COUNT==4
        printf("input=%ld %ld %ld %ld feedback=%ld %ld %ld %ld\r\n",
long(input[0]*1000), long(input[1]*1000), long(input[2]*1000),
long(input[3]*1000),
                                                        long(feedback[0]),
long(feedback[1]), long(feedback[2]), long(feedback[3]));
    #endif
#endif
        bool stoped=true;
        for(int i=0;i<MOTOR_COUNT;i++){
            if (input[i] != 0 || feedback[i] != 0){
                stoped = false;
                break;
            }
        }

        short output[MOTOR_COUNT]={0};
        if (stoped){
            for(int i=0;i<MOTOR_COUNT;i++){
                output[i] = 0;
            }
            do_kinmatics_flag = false;
        }else{
            for(int i=0;i<MOTOR_COUNT;i++){
                output[i] = pid[i]->compute(Data_holder::get()-
>parameter.do_pid_interval*0.001);
            }
        }

        for(int i=0;i<MOTOR_COUNT;i++){
            Data_holder::get()->pid_data.input[i] = int(input[i]);
            Data_holder::get()->pid_data.output[i] =  int(feedback[i]);
        }

#if PID_DEBUG_OUTPUT
    #if MOTOR_COUNT==2
        printf("output=%ld %ld\r\n\r\n", output[0], output[1]);
    #endif
    #if MOTOR_COUNT==3
        printf("output=%ld %ld %ld\r\n\r\n", output[0], output[1], output[2]);
    #endif
    #if MOTOR_COUNT==4
        printf("output=%ld %ld %ld %ld\r\n\r\n", output[0], output[1], output[2],
output[3]);
    #endif
#endif
        for(int i=0;i<MOTOR_COUNT;i++){
            motor[i]->control(output[i]);
```

```
        }

        if (Board::get()->get_tick_count()-
last_velocity_command_time>Data_holder::get()->parameter.cmd_last_time){
            for(int i=0;i<MOTOR_COUNT;i++){
                input[i] = 0;
            }
        }
    }
}
```

运动控制是在Robot类中的do_kinmatics函数实现的，大概流程

- 根据解算到各个轮子的速度转换到对应编码器的值和编码器的输出计算PID
- 根据PID结果控制电机
- 超时判断

上面说到的解算就是在Robot类中的update_velocity函数实现的

```
void Robot::update_velocity(){
    short vx = min(max(Data_holder::get()->velocity.v_liner_x, -
(short(Data_holder::get()->parameter.max_v_liner_x))), short(Data_holder::get()-
>parameter.max_v_liner_x));
    short vy = min(max(Data_holder::get()->velocity.v_liner_y, -
(short(Data_holder::get()->parameter.max_v_liner_y))), short(Data_holder::get()-
>parameter.max_v_liner_y));
    short vz = min(max(Data_holder::get()->velocity.v_angular_z, -
(short(Data_holder::get()->parameter.max_v_angular_z))), short(Data_holder::get()-
>parameter.max_v_angular_z));

    float vel[3]={vx/100.0, vy/100.0, vz/100.0};
    float motor_speed[MOTOR_COUNT]={0};
    model->motion_solver(vel, motor_speed);


    for(int i=0;i<MOTOR_COUNT;i++){
        input[i] = motor_speed[i]*short(Data_holder::get()-
>parameter.encoder_resolution)/(2*__PI)*short(Data_holder::get()-
>parameter.do_pid_interval)*0.001;
    }


#if DEBUG_ENABLE
    printf("vx=%d %d motor_speed=%ld %ld\r\n", vx, vz, long(motor_speed[0]*1000),
long(motor_speed[1]*1000));
#endif

    last_velocity_command_time = Board::get()->get_tick_count();
    do_kinmatics_flag = true;
}
```

通过调用运动模型接口调用运动解算(`model->motion_solver(vel, motor_speed)`)，完成从控制的全局速度（下发的角速度和线速度）到各个轮子速度的转换，最终转换为在do_pid_interval时间内编码器的变化值

3.PIBOT电机方向和顺序

3.1电机方向与pwm_value值关系

PIBOT定义所有电机控制给正值时(`motor[i]->control(pwm_value)`)，从电机输出轴方向看电机顺时针转。对应到差分小车apollo, 如

```
motor[0]->control(2000);//控制左电机向后
motor[1]->control(2000);//控制右电机向前
```

对与小车就是在逆时针运动

再如

```
motor[0]->control(-2000);//控制左电机向前
motor[1]->control(2000);//控制右电机向前
```

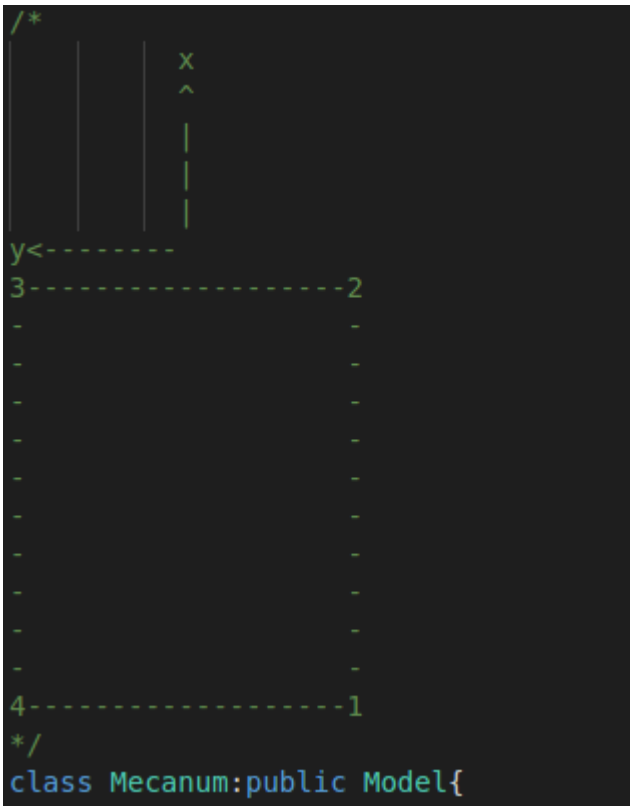
对与小车就是在向前运动

> **总结就是control给定正值，输出轴看电机顺时针转动，反之给定负值逆时针转动**

3.2电机顺序

上面可以看出来`motor[i]`中的索引`i`

apollo中左电机为0，右电机为1 zeus中前电机为0，左电机为1，右电机为2 hades和hera中右后电机为0，右前电机为1，左前电机为2，左后电机为3 具体编号源码`kinematic_models`文件夹中相应文件前右说明



3.3 编码器

这里编码器值是一个程序计数值，一个方向转动编码器值会累加，反方向会减少 PIBOT定义上面2.1中给定正值时，编码器值累加，给定负是编码器值减少

4.验证测试

如果上面有点晦涩，那直接按照如下方式测试即可 apollo为例

4.1 测试方向

do_kinmatics直接motor[0]->control(2000); return;

```
void Robot::do_kinmatics(){
    motor[0]->control(2000);
    return;
    if (!do_kinmatics_flag){
```

> ** 这里2000相对于PWM最大值来的Arduino最大1024 STM32最大设置为5000**

观察左电机电机是否向后

分别查看下面各个控制与实际电机转动情况 motor[0]->control(2000) 左电机向后 motor[0]->control(-2000) 左电机向前 motor[1]->control(2000) 右电机向前 motor[1]->control(-2000) 右电机向后

> 如果得不到相应的结果，根据情况调整: a. motor[0]->control时右电机转动，可能考虑左右电机接线反了 b. motor[1]->control(xx) ,哪个电机是对的，但发现不对。可以调整电机接线，也可以调整程序的方向控制

> PIBOT提供相关的宏MOTORx_REVERSE可以不需要对换电机接线**(PIN1和PIN6),**达到调整方向的目的

4.2测试编码器

恢复会正常程序并新增调试的输出，连接调试串口，打开串口调试工具 对应到程序的输出

```
void Robot::calc_odom(){

    static unsigned long last_millis=0;

    if (Board::get()->get_tick_count()-last_millis>=CALC_ODOM_INTERVAL){
        last_millis = Board::get()->get_tick_count();
#if ODOM_DEBUG_OUTPUT
        long total_count[MOTOR_COUNT]={0};
        for(int i=0;i<MOTOR_COUNT;i++){
            total_count[i] = encoder[i]->get_total_count();
        }

    #if MOTOR_COUNT==2
        printf("total_count=[%ld %ld]", total_count[0], total_count[1]);
    #endif
    #if MOTOR_COUNT==3
        printf("total_count=[%ld %ld %ld]", total_count[0], total_count[1], total_count[2]);
    #endif
    #if MOTOR_COUNT==4
        printf("total_count=[%ld %ld %ld %ld]", total_count[0], total_count[1], total_count[2], total_count[3]);
    #endif
#endif
```

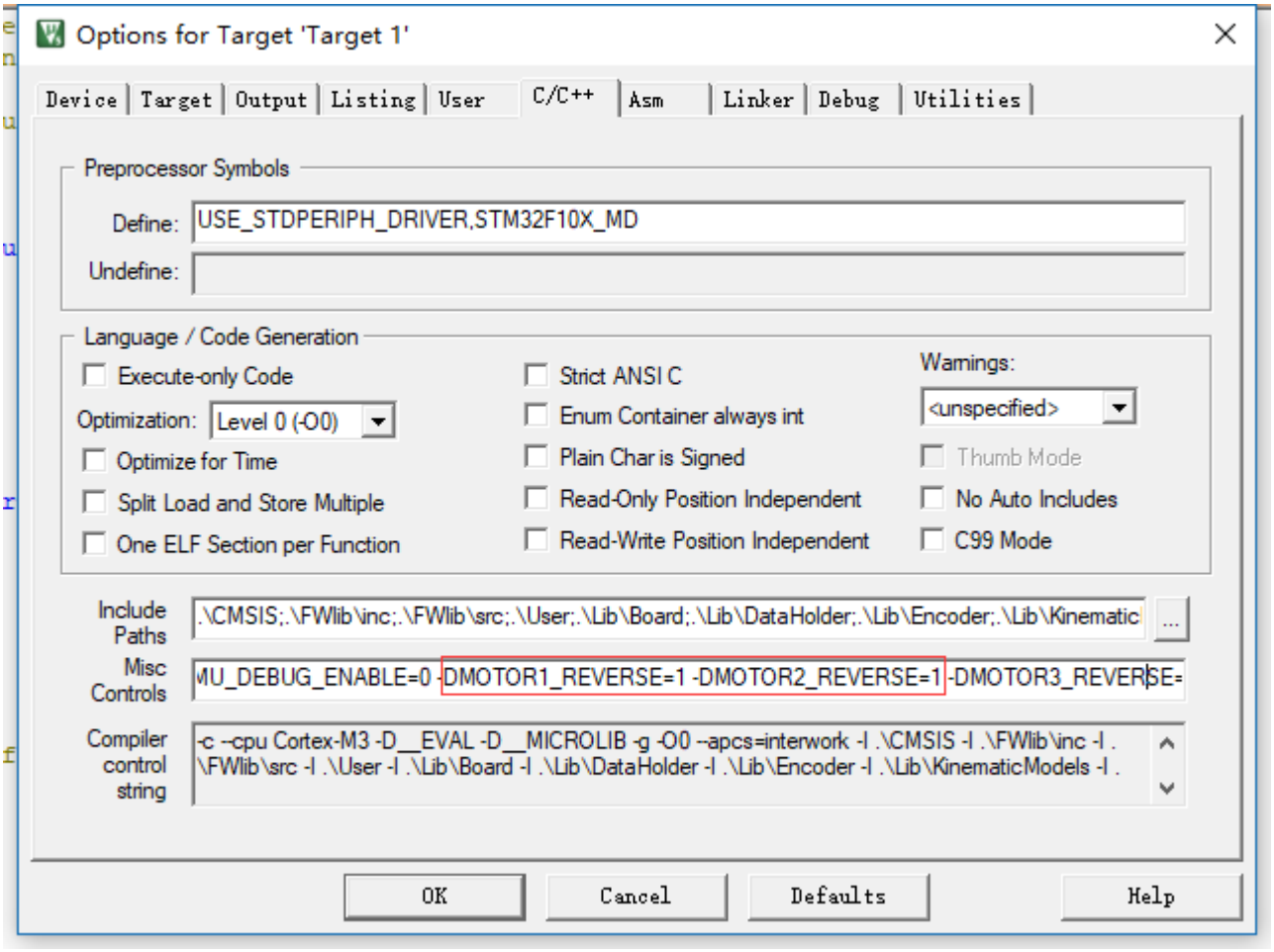
观察串口调试助手中的`total_count=xx yy`输出，`xx`随着左轮的向前转动越来越小，反之越来越大；`yy`随着右轮的向前越来越大，反之越来越小

> 如果得不到相应的结果，根据情况调整: a. 左电机转动`yy`变化或者右电机转动`xx`变化，那应该是2个电机编码器反了，需要调换下 b. 左电机转动`xx`变化，但相反。应该是编码器AB相接线反了；右电机同理

> `PIBOT`提供相关的宏`ENCODERx_REVERSE`可以不需要对换电机编码器接线（**PIN3**和**PIN4**）,达到调整方向的目的

5.程序修改

固件程序提供了直接的宏，针对电机反向或者编码器反向的问题 例如STM32F1电机方向反转宏,修改宏定义



STM32F4编码器方向反转宏(在param.mk文件)

```
param.mk ×
1
2
3   #models_list
4   DDEFS           += -DROBOT_DIFF_2WD=1 -DROBOT_DIFF_4WD=2 -DROBOT_OMNI_3=10 -DROBOT_OMNI_4=11 -DROBOT_MECANUM=12
5
6   #motor_controller_list
7   DDEFS           += -DCOMMON_CONTROLLER=1 -DAF_SHIELD_CONTROLLER=2
8
9   DDEFS           += -DMOTOR1_REVERSE=0 -DMOTOR2_REVERSE=0 -DMOTOR3_REVERSE=0 -DMOTOR4_REVERSE=0
10  DDEFS           += -DENCODER1_REVERSE=0 -DENCODER2_REVERSE=0 -DENCODER3_REVERSE=0 -DENCODER4_REVERSE=0
```