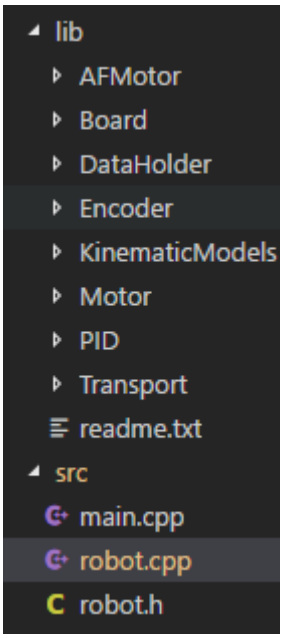
1.包列表



包	说明	备注
AFMotor	Adafruit shield电机驱动板驱动	
Board	板子资源接口及Mega2560中的实现	
DataHolder	关键数据存储类	
Encoder	编码器接口及Mega2560中编码器实现	
KinematicModels	机器人模型类	针对不同型号机型的解算
Motor	电机驱动接口类及AF电机驱动与一般驱动板的实现	
PID	PID运算类	
Transport	通讯接口类与实现	
robot	robot调度类	

2.main分析

直接贴源码，无需赘述

```
#include "Arduino.h"
#include "robot.h"
void setup()
{
    Robot::get()->init();
}
void loop()
{
    Robot::get()->check_command();
    Robot::get()->do_kinmatics();
    Robot::get()->calc_odom();
}
```

3.robot文件分析

3.1robot.h

```
#ifndef PIBOT_ROBOT_H_
#define PIBOT_ROBOT_H_

#include "dataframe.h"

//根据机器人模型选择相应的解算类头文件
#if ROBOT_MODEL == ROBOT_MODEL_DIFF//差分轮
#include "differential.h"
#endif

#if ROBOT_MODEL == ROBOT_OMNI_3//全向三轮
#include "omni3.h"
#endif

class MotorController;//电机控制器接口类
class Encoder;                  //编码器接口类
class PID;                              //PID运算接口类
class Transport;                //通讯接口类
class Dataframe;                //通讯协议数据包接口类
class Model;                    //机器人模型接口类

class Robot:public Notify{
    public:
            //单例模式
        static Robot* get(){
            static Robot robot;
```

```
            return &robot;
        }

        //初始化
        void init();

                //检测上位机命令
        void check_command();

                //运动学控制，下发的速度转换为各个轮子速度平且进行PID控制
        void do_kinmatics();

                //根据个轮子编码器反馈给出机器人位置姿态及实时速度信息
        void calc_odom();

                //Notify接口实现， 针对某些消息关注的回调函数
        void update(const MESSAGE_ID id, void* data);
    private:
        Robot(){}
        void init_motor();//初始化电机相关
        void init_trans();//初始化通讯相关
    private:
        void clear_odom();//清除累计的位置姿态信息
        void update_velocity();//更新下发的实时控制速度
    private:
        MotorController* motor[MOTOR_COUNT];//电机控制器接口
        Encoder*         encoder[MOTOR_COUNT];//编码器接口
        PID*             pid[MOTOR_COUNT];
        float            input[MOTOR_COUNT];//PID间隔时间内期望的encoder增加或减少的
个数
        float            feedback[MOTOR_COUNT];//PID间隔时间内反馈的encoder增加或减少
的个数

        Transport*       trans;//通讯接口
        Dataframe*       frame;//通讯数据包接口
        Model*           model;//机器人模型接口
        bool             do_kinmatics_flag; //进行运动控制的标记

        Odom             odom;//机器人位置姿态实时速度信息

        unsigned long    last_velocity_command_time;//上次下发速度的时间点
};

#endif
```

3.2robot.cpp关键代码

初始化

```
void Robot::init(){
    Data_holder::get()->load_parameter();//加载EPPROM中的配置机器人信息
```

```
#if DEBUG_ENABLE
    Board::get()->usart_debug_init();//调试串口初始化 printf通过串口3输出
#endif

    printf("RobotParameters: %d %d %d %d %d %d %d %d %d %d %d\r\n",
        Data_holder::get()->parameter.wheel_diameter, Data_holder::get()-
>parameter.wheel_track,  Data_holder::get()->parameter.encoder_resolution,
        Data_holder::get()->parameter.do_pid_interval, Data_holder::get()-
>parameter.kp, Data_holder::get()->parameter.ki, Data_holder::get()->parameter.kd,
Data_holder::get()->parameter.ko,
        Data_holder::get()->parameter.cmd_last_time, Data_holder::get()-
>parameter.max_v_liner_x, Data_holder::get()->parameter.max_v_liner_y,
Data_holder::get()->parameter.max_v_angular_z);

    printf("init_motor\r\n");
    init_motor();

    printf("init_trans\r\n");
    init_trans();

    printf("pibot startup\r\n");
}
```

电机及编码器相关初始化

```
void Robot::init_motor(){

#if MOTOR_COUNT>0//根据配置的个数定义编码器、电机控制器及PID的具体实现，MOTOR_COUNT在
具体的机器人模型中定义
    #if MOTOR_CONTROLLER == COMMON_CONTROLLER//根据使用的电机控制器选择电机控制器的实
现 这里CommonMotorController与AFSMotorController都MotorController接口类的实现
    static CommonMotorController motor1(MOTOR_1_PWM_PIN, MOTOR_1_DIR_A_PIN,
MOTOR_1_DIR_B_PIN, MAX_PWM_VALUE);
    #elif MOTOR_CONTROLLER == AF_SHIELD_CONTROLLER
    static AFSMotorController motor1(MOTOR_1_PORT_NUM, MAX_PWM_VALUE);
    #endif

        //EncoderImp是Encoder接口类的实现
    static EncoderImp encoder1(MOTOR_1_ENCODER_A_PIN, MOTOR_1_ENCODER_B_PIN);
    //PID计算接口，给定参数为期望的数据地址、反馈的数据地址以及各个PID参数值
    static PID pid1(&input[0], &feedback[0], float(Data_holder::get()-
>parameter.kp)/Data_holder::get()->parameter.ko,
                                            float(Data_holder::get()-
>parameter.ki)/Data_holder::get()->parameter.ko,
                                            float(Data_holder::get()-
>parameter.kd)/Data_holder::get()->parameter.ko , MAX_PWM_VALUE);
#endif

#if MOTOR_COUNT>1
    #if MOTOR_CONTROLLER == COMMON_CONTROLLER
```

```
    static CommonMotorController motor2(MOTOR_2_PWM_PIN, MOTOR_2_DIR_A_PIN,
MOTOR_2_DIR_B_PIN, MAX_PWM_VALUE);
    #elif MOTOR_CONTROLLER == AF_SHIELD_CONTROLLER
    static AFSMotorController motor2(MOTOR_2_PORT_NUM, MAX_PWM_VALUE);
    #endif
    static EncoderImp encoder2(MOTOR_2_ENCODER_A_PIN, MOTOR_2_ENCODER_B_PIN);
    static PID pid2(&input[1], &feedback[1], float(Data_holder::get()-
>parameter.kp)/Data_holder::get()->parameter.ko,
                                            float(Data_holder::get()-
>parameter.ki)/Data_holder::get()->parameter.ko,
                                            float(Data_holder::get()-
>parameter.kd)/Data_holder::get()->parameter.ko , MAX_PWM_VALUE);
#endif

#if MOTOR_COUNT>2
    #if MOTOR_CONTROLLER == COMMON_CONTROLLER
    static CommonMotorController motor3(MOTOR_3_PWM_PIN, MOTOR_3_DIR_A_PIN,
MOTOR_3_DIR_B_PIN, MAX_PWM_VALUE);
    #elif MOTOR_CONTROLLER == AF_SHIELD_CONTROLLER
    static AFSMotorController motor3(MOTOR_3_PORT_NUM, MAX_PWM_VALUE);
    #endif

    static EncoderImp encoder3(MOTOR_3_ENCODER_A_PIN, MOTOR_3_ENCODER_B_PIN);
    static PID pid3(&input[2], &feedback[2], float(Data_holder::get()-
>parameter.kp)/Data_holder::get()->parameter.ko,
                                            float(Data_holder::get()-
>parameter.ki)/Data_holder::get()->parameter.ko,
                                            float(Data_holder::get()-
>parameter.kd)/Data_holder::get()->parameter.ko , MAX_PWM_VALUE);
#endif

#if MOTOR_COUNT>0
    motor[0] = &motor1;//接口指向具体实现
    encoder[0] = &encoder1;
    pid[0] = &pid1;
#endif

#if MOTOR_COUNT>1
    motor[1] = &motor2;
    encoder[1] = &encoder2;
    pid[1] = &pid2;
#endif

#if MOTOR_COUNT>2
    motor[2] = &motor3;
    encoder[2] = &encoder3;
    pid[2] = &pid3;
#endif

#if ROBOT_MODEL == ROBOT_MODEL_DIFF//根据配置的机器人模型选择解算类的实现，这个
Differential与Omni3是运动解算接口Model的实现
    static Differential diff(Data_holder::get()->parameter.wheel_diameter*0.0005,
Data_holder::get()->parameter.wheel_track*0.0005);
    model = &diff;
```

```
#endif

#if ROBOT_MODEL == ROBOT_OMNI_3
    static Omni3 omni3(Data_holder::get()->parameter.wheel_diameter*0.0005,
Data_holder::get()->parameter.wheel_track*0.0005);
    model = &omni3;
#endif

    //初始化电机驱动器
    for (int i=0;i<MOTOR_COUNT;i++){
        motor[i]->init();
    }

    do_kinmatics_flag = false;

    memset(&odom, 0 , sizeof(odom));
    memset(&input, 0 , sizeof(input));
    memset(&feedback, 0 , sizeof(feedback));

    last_velocity_command_time = 0;
}
```

通讯相关初始化

```
void Robot::init_trans(){
    static USART_transport _trans(MASTER_USART, 115200);//使用串口作为通讯接口
    static Simple_dataframe    _frame(&_trans);//使用Simple_dataframe协议数据包实现
数据打包解包
    trans = &_trans;
    frame = &_frame;

    trans->init();
    frame->init();

    //注册相关消息的通知， 收到该消息this->update回调会被调用
    frame->register_notify(ID_SET_ROBOT_PARAMTER, this);
    frame->register_notify(ID_CLEAR_ODOM, this);
    frame->register_notify(ID_SET_VELOCITY, this);
}
```

上位命令处理

```
void Robot::check_command(){
    unsigned char ch=0;
    if (trans->read(ch)){//从通讯口中读取数据
        //printf("%02x ", ch);
        if (frame->data_recv(ch)){//使用数据包接收和解析数据
            //printf("\r\n");
            frame->data_parse();
```

```
        }
    }
}
```

运动处理

```
void Robot::do_kinmatics(){
    if (!do_kinmatics_flag){//该标记收到上位的命令会置true， 超时停止会置false
        for(int i=0;i<MOTOR_COUNT;i++){
            pid[i]->clear();//停止后清除pid变量值
            encoder[i]->get_increment_count_for_dopid();//读取掉停止时编码器的变化
至，放置手动转动电机导致下次启动pid时的异常
        }
        return;
    }

    static unsigned long last_millis=0;
    //根据配置PID间隔时间进行pid运算
    if (Board::get()->get_tick_count()-last_millis>=Data_holder::get()-
>parameter.do_pid_interval){
        last_millis = Board::get()->get_tick_count();
        //得到PID间隔时间反馈编码器的值
        for(int i=0;i<MOTOR_COUNT;i++){
            feedback[i] = encoder[i]->get_increment_count_for_dopid();
        }
#ifdef PID_DEBUG_OUTPUT
        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
        //判断超时，则无需继续PID运算
        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{
            //计算得到输出PWM  input[i]在update回调通知中给定
            for(int i=0;i<MOTOR_COUNT;i++){
                output[i] = pid[i]->compute(Data_holder::get()-
```

```
>parameter.do_pid_interval*0.001);
            }
        }

#ifdef PID_DEBUG_OUTPUT
        printf("output=%ld %ld %ld\r\n\r\n", output[0], output[1], output[2]);
#endif
        //控制各个电机
        for(int i=0;i<MOTOR_COUNT;i++){
            motor[i]->control(output[i]);
        }

        //超时判断
        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;
            }
        }
    }
}
```

计算里程位置姿态

```
void Robot::calc_odom(){

    static unsigned long last_millis=0;
    //每间CALC_ODOM_INTERVAL隔时间计算
    if (Board::get()->get_tick_count()-last_millis>=CALC_ODOM_INTERVAL){
        last_millis = Board::get()->get_tick_count();
#ifdef 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();
        }

        printf("total_count=%ld %ld\r\n", total_count[0], total_count[1]);
#endif
        float dis[MOTOR_COUNT] = {0};
        //得到间隔时间内个轮子行径的距离(m)
        for(int i=0;i<MOTOR_COUNT;i++){
            dis[i] = encoder[i]-
>get_increment_count_for_odom()*__PI*Data_holder::get()-
>parameter.wheel_diameter*0.001/Data_holder::get()->parameter.encoder_resolution;
#ifdef ODOM_DEBUG_OUTPUT
            printf(" %ld ", long(dis[i]*1000000));
#endif
        }

        //通过使用的模型接口到当前里程信息
        model->get_odom(&odom, dis, CALC_ODOM_INTERVAL);
```

```
#ifdef ODOM_DEBUG_OUTPUT
        printf(" x=%ld y=%ld yaw=%ld", long(odom.x*1000), long(odom.y*1000),
long(odom.z*1000));
        printf("\r\n");
#endif
        //更新至数据存储中
        Data_holder::get()->odom.v_liner_x = odom.vel_x*100;
        Data_holder::get()->odom.v_liner_y = odom.vel_y*100;
        Data_holder::get()->odom.v_angular_z = odom.vel_z*100;
        Data_holder::get()->odom.x = odom.x*100;
        Data_holder::get()->odom.y = odom.y*100;
        Data_holder::get()->odom.yaw = odom.z*100;
    }
}
```