#include <MsTimer2.h>

int MoPin = 5; // vibrator connected to digital pin 5

int plane\_PosPin = 1, plane\_NegPin = 2; //无人机正负极接数字口1和2

int car\_PosPin = 3, car\_NegPin = 4; //无人车输出电正负极接数字口3和4

#define maxCapacity 40000

enum status {RESET=0,

BULK,

ABSORPTION,

FLOAT};

struct e\_quanlity {

int cur\_quanlity;

int Remain\_time;

int Remain\_frq;

} //储存充电信息，当前电量，还需充电时间，以及预测无人机电池还能使用次数

struct battery\_health\_data {

int SOH\_C;

int SOH\_R;

int SOH\_P;

int SOH\_Rd;

};//无人机充好点以后，会发送一个上述数据结构，给小车提供一个无人机电池的健康状况以供分析

int currentSum; //实时电池电量(mAh)

int chargerV;//充电器电压

int batteryV;//电池电压

int floatpoint;

int bulkpoint;

int state = 0;

int ticker = 0

e\_quanlity UAV\_Ele\_quanlity;

void setup() {

pinMode(MoPin, OUTPUT);

pinMode(plane\_PosPin, INPUT);

pinMode(plane\_NegPin, INPUT);

digitalWrite(car\_PosPin, HIGH);

digitalWrite(car\_NegPin, LOW);

pinMode(car\_PosPin, OUTPUT);

pinMode(car\_NegPin, OUTPUT);

Serial.begin(9600);

pinMode(13,OUTPUT);

MsTimer2::set(3600000,flash);

Serial.begin(9600);

}

void pin\_ShockResponse(f) {

if (f >= 1000) {

return 0; //

} else {

return f \* 3; //降低振动频率

}

}

void vibrator() {

int vib\_f = 20; //设置初始震动频率

//当频率从50hz逐渐变低到接近1hz时，停止工作，此时无人机已经调整到正确的充电位置

while (vib\_f != 0) {

digitalWrite(MoPin, HIGH);

delay(vib\_f);

digitalWrite(MoPin, LOW);

delay(vib\_f);

vib\_f = pin\_ShockResponse(vib\_f);

}

break;

}

void charge\_callback(){

ticker = 1;

}

int charge\_state() {

int pos, neg; //小车正负极放电信号

// while(a!=1&&b!=1){

// DistanceMode(b); //获取距离并开启放电模式（）

get\_carPin(pos, neg); //获取小车放电的信号（）

if (pos != 1 || neg != 0) {

vabrator(); //振动函数使得无人机归位

return 0;

}else if{

return 1;

}

}

void charge() {

startCharger();

currentSum = get\_QualityCharge();//获取无人机当前电量

while (currentSum <= maxCapacity \* 0.96) {

while(currentSum<=maxCapacity\*0.96)

{ charging();//调用充电模块

delay 200;

for(int i=0,i<1000,i++){

int senValue = analogRead(A0);//读取传感器数据

float volt = senValue \* 5;//传感器数值转换为电压值

volt = volt / 1023;

float current = volt / 185;//转换为电流值

current = current / 1000;

average\_current += current;//得出1000ms内总电流

delay(1); //1 ms延迟为了程序稳定

}

average\_current = average\_current /1000;//1000ms内平均电流

currentSum+=average\_current;//对电流积分

delay(1);

UAV\_Ele\_quanlity->cur\_quanlity = currentSum / maxCapacity; //当前电量%

UAV\_Ele\_quanlity->Remain\_time = (maxCapacity - currentSum) / average\_current; //剩余充电时间s

}

cutPower();

}

int state\_check(){//充电状态检测

while(startCharger())

{

if(batteryV < bulkpoint && state = 0){

state=BULK;

return BULK;

}//检测快充状态

else if(batteryV > bulkpoint && batteryV < chargerV){

state=ABSORPTION;

MsTimer2::start();//恒流计时

return ABSORPTION;

}//检测恒流状态

else if(ticker=1&batteryV < bulkpoint){

state=FLOAT;

return FLOAT;

}//检测涓流状态

else(currentSum >= maxCapacity\*0.96){

state=RESET;

return RESET;

}//检测是否复位

}

}

void charging()//充电模式选择

{ int constantcur;//设定的恒定电流

int current;//电流当前设定值

while(1){

state=state\_check();//状态检测

switch(state){

case BULK://进入快充

fastcharging()

case ABSORPTION://恒流模式

charger(constantcur);

case FLOAT://涓流，充电电流逐渐减小

while(1){

current=constantcur;

charger(current);

delay(10);

current-=0.01;

if(currentSum>=maxCapacity\*0.96){

break;

}

default:state=RESET;//复位

}

}

}

int data\_c(){

//当无人机电池健康状况出现一定程度的下滑，开始实时记录电池的健康状况参数指标，用于预测电池寿命

if (data\_judge() == 1 || frequent != 0) {

frequent++;

}

if (frequent == 0) {

data\_buf[0][0] = data\_buf[0][1];

data\_buf[1][0] = data\_buf[1][1];

data\_buf[2][0] = data\_buf[2][1];

data\_buf[3][0] = data\_buf[3][1];

data\_buf[0][1] = bat\_data->SOH\_C;

data\_buf[1][1] = bat\_data->SOH\_R;

data\_buf[2][1] = bat\_data->SOH\_P;

data\_buf[3][1] = bat\_data->SOH\_Rd;

return 0;

}

else if (frequent < 10) {

data\_buf[0][frequent] = (R\_end - R) / (R\_end - R0);

data\_buf[1][frequent] = C - C0;

data\_buf[2][frequent] = 1 - P / P0;

data\_buf[3][frequent] = (Rd\_end - Rd) / (Rd\_end - Rd0);

return 1;

}

else if (frequent > 10) {

int i, j;

for (i = 0; i < 9; ++i) {

data\_buf[0][i] = data\_buf[0][i + 1];

data\_buf[1][i] = data\_buf[1][i + 1];

data\_buf[2][i] = data\_buf[2][i + 1];

data\_buf[3][i] = data\_buf[3][i + 1];

}

data\_buf[0][1] = (R\_end - bat\_data->) / (R\_end - R0);

data\_buf[1][1] = C - C0;

data\_buf[2][1] = 1 - P / P0;

data\_buf[3][1] = (Rd\_end - Rd) / (Rd\_end - Rd0);

return 1;

}

};

int data\_judge() {

//当电池健康状况出现较大的衰减时（此处根据无人机的性能，当健康状况达不到上一次的95%时，开始记录电池状况

//，即每一次的电阻，电容等健康参数指标）

int a = 0;

int b = 0;

int c = 0;

int d = 0;

while (data\_buf[0][1] < data\_buf[0][0] \* 0.95) {

a = 1;

}

while (data\_buf[1][1] < data\_buf[1][1] \* 0.95) {

b = 1;

}

while (data\_buf[2][1] < data\_buf[2][1] \* 0.95) {

c = 1;

}

while (data\_buf[2][1] < data\_buf[2][1] \* 0.95) {

d = 1;

}

if (a == 1 && b == 1 && c == 1 && d == 1) {

return 1;

}

else {

return 0;

}

}

int frq\_forcast(int \*x, int \*y) {

int x0 = 0;

int y0 = 0;

int i;

for (i = 0; i < 10; ++i) {

x0 += x[i];

y0 += log(y[i]);

}

x0 = x0 / 10;

y0 = y0 / 10;

int a\_top = 0;

int a\_bottom = 0;

int a, b;

for (i = 0; i < 10; ++i) {

a\_top += (x[i] - x0) \* log(y[i]);

a\_bottom += (x[i] - x0) \* (x[i] - x0);

}

a = a\_top / a\_bottom;

b = y0 - a \* x0;//通过最小二乘法算出目标函数

i = 0;

while (exp(a \* i + b) >= 0.2) {

i++;

}

return i;当电池健康状况达不到百分之二十时，认为无人机需要更换电池，返回值为还可以巡检飞行的次数

}

void loop() {

while (1) {

if (charge\_state() == 1) {

break;

}

else {

vibrator();

}

}//无人机充电口正负极若没有正确连接时，通过简谐振动将无人机调整至正常位置，如果连接正常，退出循环

charge();//充电，并实时记录电量与剩余充电时间

battery\_health\_data bat\_data = hal\_read\_sample();//从无人机获取电池健康信息

if(data\_c() == 1){

int x[] = {0,1,2,3,4,5,6,7,8,9};

int i = 0;

UAV\_Ele\_quanlity->Remain\_frq = 0;

for(i = 0; i < 4; ++i){

int j = frq\_forcast(x, data\_buf[i]);

if (j < UAV\_Ele\_quanlity->Remain\_frq){

UAV\_Ele\_quanlity->Remain\_frq = j;

}

}

}//通过电池的数据分析预测还能飞行巡检的次数，储存在UAV\_Ele\_quanlity->Remain\_frq中，

//以便于更换无人机电池，防止无人机因为电池老化导致续航过短，出现无人机损坏

}