////////////////////////////////////////////////////////////////////////////////////

// БИБЛИОТЕКИ //

////////////////////////////////////////////////////////////////////////////////////

//// ////

// БИБЛИОТЕКИ ШАГОВЫХ ДВИГАТЕЛЕЙ //

//// ////

// БИБЛИОТЕКА НЕ ПОДДЕРЖИВАЕТ МАССИВЫ ДВИГАТЕЛЕЙ, ЕСЛИ ИЗМЕНИТСЯ КОЛИЧЕСТВО

// ДВИГАТЕЛЕЙ, ИЗМЕНЕНИЯ БУДУТ И В "НАСТРОКАХ", И В "РАБОТЕ", И В "СВЯЗИ С ПК"

#include <AccelStepper.h>

//// ////

// БИБЛИОТЕКИ ЭНКОДЕРОВ //

//// ////

#include <ACE128.h>

#include <ACE128map87654321.h> // Карта пинов 87654321

////////////////////////////////////////////////////////////////////////////////////

// ПЕРЕМЕННЫЕ //

////////////////////////////////////////////////////////////////////////////////////

//// ////

// ПЕРЕМЕННЫЕ КНОПОК //

//// ////

#define BUTTDELAY 50 // Время между двумя изменениями

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО КНОПОК!

#define BUTTNUM 3 // Количество кнопок

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО КНОПОК!

const byte buttPin[BUTTNUM] = {14, 15, 16}; // Пины кнопок {14, 15, 16};

boolean buttPos[BUTTNUM] = {false, false, false}; // Положение кнопки

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО КНОПОК!

boolean buttRev[BUTTNUM] = {true, true, true}; // Реверс кнопки

unsigned long buttTime[BUTTNUM] = {0, 0, 0}; // Момент времени изменения положения кнопки

//// ////

// ПЕРЕМЕННЫЕ ПРЕДОХРАНИТЕЛЕЙ //

//// ////

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО ПРЕДОХРАНИТЕЛЕЙ!

#define FUSENUM 1

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО ПРЕДОХРАНИТЕЛЕЙ!

const byte fuseIn[] = {17}; // Пины предохранителей

bool fuseLog[] = {false}; // Переменная для определения изменений в положении предохранителей

//// ////

// ПЕРЕМЕННЫЕ ПОТЕНЦИОМЕТРОВ //

//// ////

#define POTERROR 20 // Шум в значении потенциометра

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО ПОТЕНЦИОМЕТРОВ!

#define POTNUM 1

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО ПОТЕНЦИОМЕТРОВ!

const byte potPin[POTNUM] = {A0}; // Пины потенциометров

int potPos[POTNUM] = {0}; // Положение потенциометра

int oldPotPos[POTNUM] = {0}; // Предыдущее положение потенциометра

int potType[POTNUM] = {0}; // Тип потенциометра

int potBorder[POTNUM][2] = {{0, 1000}};

// Границы значений потенциометра

double truePotPos[POTNUM] = {0}; // Положение потенциометра по шкале

double oldTruePotPos[POTNUM] = {0}; // Старое положение потенциометра по шкале

int truePotPosMap[POTNUM] = {0}; // Положение потенциометра по шкале

int oldTruePotPosMap[POTNUM] = {0}; // Старое положение потенциометра по шкале

//// ////

// ПЕРЕМЕННЫЕ РАЗЪЕМОВ 2 - new //

//// ////

//#define razSetup

#define razERROR 20 // Шум в значении потенциометра

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО РАЗЪЕМОВ 2!

#define razNUM 3

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО РАЗЪЕМОВ 2!

const byte razPin[razNUM] = {1, 2, 3}; // Пины РАЗЪЕМОВ 2 {0, 1, 2}

int razPos[razNUM] = {0, 0, 0}; // Положение потенциометра

int oldrazPos[razNUM] = {0, 0, 0}; // Предыдущее положение потенциометра

int razType[razNUM] = {0, 0, 0}; // Тип потенциометра

int razBorder[razNUM][2] = {{0, 1000}, {0, 1000}, {0, 1000}};

int razAnswer[razNUM] = {480, 420, 460};

#define razERRORAnswer 15

// Границы значений потенциометра

double truerazPos[razNUM] = {0, 0, 0}; // Положение потенциометра по шкале

double oldTruerazPos[razNUM] = {0, 0, 0}; // Старое положение потенциометра по шкале

int truerazAnswer[razNUM] = {0, 0, 0}; // Положение потенциометра по шкале

int oldTruerazAnswer[razNUM] = {2, 2, 2}; // Старое положение потенциометра по шкале

unsigned long razTime[razNUM] = {0, 0, 0}; // Момент времени изменения положения кнопки

#define razDELAY 1000 // Время между двумя изменениями

#define razDELAYsetup 20 // Время между двумя изменениями

unsigned long traz = 0;

/////////////////////////////////// 2020 raz end

//// ////

// ПЕРЕМЕННЫЕ РАЗЪЕМОВ ШР //

//// ////

#define CONNDELAY0 50 // Время между двумя изменениями

#define CONNDELAY1 4000 // Время между двумя изменениями (для закручивающихся)

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО РАЗЪЕМОВ!

#define CONNNUM 2 // Количество разъемов

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО РАЗЪЕМОВ!

int connType[CONNNUM] = {0, 1}; // Тип разъема

int connDelay[CONNNUM] = {0, 0}; // Время между двумя изменениями

const byte connPin[CONNNUM] = {18, 19}; // Пины разъемов

boolean connPos[CONNNUM] = {false, false}; // Положение разъемов

unsigned long connTime[CONNNUM] = {0, 0}; // Момент времени изменения положения разъемов

/////////////////////////////////////////LED//tut/////////////////////////////////////

#include <FastLED.h>

#define LED\_NUM 3 // Количество светодиодов

#define LED\_PIN 13 // Пин светодиодов

boolean ledPos[LED\_NUM]; // Положение /\* определено в setup(){} \*/

int i2, n2;

int led\_privod[] = {1, 2, 0};

String led\_privod\_str[] = {"on1", "on2", "on0"};

String led\_privod\_str2[] = {"off1", "off2", "off0"};

int im;

CRGB led[LED\_NUM]; // Определение

///////////////////////ПЕРЕМЕННЫЕ МОТОРОВ//////////////////////////

#include <MCP3017AccelStepper.h>

long j2;

//#define STEPPER\_SETUP // Раскомментировать для настройки

//#define STEPPER\_DEBUG //Раскомментировать для поиска ошибок

#define STEPPER\_NUM 2 // Количество шаговых двигателей

MCP3017AccelStepper stepper[STEPPER\_NUM] = { // Определение двигателей

MCP3017AccelStepper (AccelStepper::HALF4WIRE, 22, 24, 26, 28),

MCP3017AccelStepper (AccelStepper::HALF4WIRE, 46, 48, 50, 52)

};

const byte stepperOptronPin[STEPPER\_NUM] = {30, 44}; // Пины оптронов

String stepperSymbol = "s"; // Обозначение шаговых двигателей

String stepperSymbolB = "r";

#define STEPPER\_ACCELERATION 8000 // Ускорение //-

#define STEPPER\_SPEED 800 // Скорость //-

#define STEPPER\_ACCELERATION\_yst 400 // Ускорение //-

#define STEPPER\_SPEED\_yst 200 // Скорость //-

boolean stepperBuzy[STEPPER\_NUM] = {false, false}; // Статус работы

/////////////

boolean stepperZeroSettingLock[STEPPER\_NUM] = {false, false}; // Блокировка движения в конце loop(){}

boolean stepperZeroSettingStepA[STEPPER\_NUM] = {false, false}; // Логические шаги установки нуля

boolean stepperZeroSettingStepB[STEPPER\_NUM] = {false, false};

boolean stepperZeroSettingStepC[STEPPER\_NUM] = {false, false};

boolean stepperZeroSettingStepD[STEPPER\_NUM] = {false, false};

boolean stepperZeroSettingStepE[STEPPER\_NUM] = {false, false};

unsigned long stepperZeroSettingTimeC[STEPPER\_NUM] = {0, 0}; // Измерение времени в логическом шаге

unsigned long stepperZeroSettingTimeD[STEPPER\_NUM] = {0, 0};

#define STEPPER\_ZERO\_SETTING\_SPEED\_B 100 // Скорость, выставляемая в логическом шаге

#define STEPPER\_ZERO\_SETTING\_DEAD\_TIME\_C 50 // Время ожидания перед шагом

#define STEPPER\_ZERO\_SETTING\_DEAD\_TIME\_D 50

long stepperZeroPos[STEPPER\_NUM] = {0, 0}; // Положение нуля

/////////////

#ifndef STEPPER\_SETUP

boolean stepperZeroSettingStepF[STEPPER\_NUM] = {false, false};

// Дополнительные логические шаги установки нуля

boolean stepperMoveStepA[STEPPER\_NUM] = {false, false}; // Логические шаги сдвига

boolean stepperMoveStepB[STEPPER\_NUM] = {false, false};

long stepperTrueZeroPos[STEPPER\_NUM] = {3 , 3}; // Положение нуля шкалы

long stepperBorder[STEPPER\_NUM][2] = {{0, 204}, {0, 145}}; // Границы шкалы

long stepperPos[STEPPER\_NUM] = {0, 0}; // Текущее положение

#endif

//////////////

#ifdef STEPPER\_DEBUG

long stepperDebugPos[STEPPER\_NUM] = {0, 0};

#endif

//// ////

// ПЕРЕМЕННЫЕ ЭНКОДЕРОВ //

//// ////

//#define ENCSETUP // Раскомментировать для настройки

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО ЭНКОДЕРОВ!

#define ENCNUM 1 // Количество энкодеров

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО ЭНКОДЕРОВ!

const byte encoderAddress[ENCNUM]= {32}; // Адреса энкодеров

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО ЭНКОДЕРОВ!

ACE128 enc0((uint8\_t)encoderAddress[0], (uint8\_t\*)encoderMap\_87654321); // Адрес и карта энкодера

uint8\_t rawPos[ENCNUM] = {0}; // Текущее положение энкодера

uint8\_t oldRawPos[ENCNUM] = {127}; // Предыдущее положение энкодера

#ifndef ENCSETUP

int encType[ENCNUM] = {3}; // Тип энкодера

int rawBorder[ENCNUM][2] = {{82, 82}}; // Границы положений энкодера

int encBorder[ENCNUM][2] = {{0, 5}}; // Границы положений энкодера по шкале

double encPoint[ENCNUM] = {0}; //

double encPos[ENCNUM] = {0}; // Положение энкодера по шкале

double oldEncPos[ENCNUM] = {0}; // Предыдущее положение энкодера по шкале

double rawNorm[ENCNUM] = {0};

#endif

//// ////

// ОБЩИЕ ПЕРЕМЕННЫЕ //

//// ////

//#define DEBUG // Раскомментировать для отладки

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО ЭЛЕМЕНТОВ!

#define ELEMNUM 40

String str = "";

int n = 0;

int i;

double j = 0;

double m = 0;

unsigned long t = 0;

boolean checkElem = false;

/////////////////////////////////////////////////////////////////////////////////////////////

void setup() {

////////////////////////////////////////////////////////////////////////////////////

//////// ////////

//// ////

// НАСТРОЙКИ //

//// ////

//////// ////////

////////////////////////////////////////////////////////////////////////////////////

Serial.begin(115200); // Инициализация порта для работы с ПК

Serial.println("kpa33");

Serial.println("complite");

#ifdef DEBUG

// Serial.println("DEBUG DEFINED");

#endif

///////////////////////////////////////////////////////////////////

//// ////

// НАСТРОЙКИ КНОПОК //

//// ////

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

pinMode(buttPin[i], INPUT); // Настройка пина на вход

digitalWrite(buttPin[i], HIGH); // Подтяжка пина к питанию

}

for(i = 0; i < BUTTNUM; i++){ // Опрос кнопок

if(buttRev[i] == false){

if(digitalRead(buttPin[i]) == HIGH){

Serial.print("b");

Serial.print(i);

Serial.print(" off");

Serial.println("z");

}

else if(digitalRead(buttPin[i]) == LOW){

Serial.print("b");

Serial.print(i);

Serial.print(" on");

Serial.println("z");

buttPos[i] = true;

}

}

else{

if(digitalRead(buttPin[i]) == HIGH){

Serial.print("b");

Serial.print(i);

Serial.print(" on");

Serial.println("z");

buttPos[i] = true;

}

else if(digitalRead(buttPin[i]) == LOW){

Serial.print("b");

Serial.print(i);

Serial.print(" off");

Serial.println("z");

}

}

n++;

}

//// ////

// НАСТРОЙКИ ПРЕДОХРАНИТЕЛЕЙ //

//// ////

for (int i = 0; i < FUSENUM; i++) {

pinMode(fuseIn[i], INPUT); // Настройка пина на вход

digitalWrite(fuseIn[i], HIGH); // Подтяжка пина к питанию

}

for (int i = 0; i < FUSENUM; i++) { // Опрос предохранителей

if (digitalRead(fuseIn[i]) == HIGH) {

Serial.print("f");

Serial.print(i);

Serial.print(" off");

Serial.println("z");

}

else if (digitalRead(fuseIn[i]) == LOW) {

Serial.print("f");

Serial.print(i);

Serial.print(" on");

Serial.println("z");

fuseLog[i] = true;

}

}

//// ////

// НАСТРОЙКИ ПОТЕНЦИОМЕТРОВ //

//// ////

for (i = 0; i < POTNUM; i++) { // Опрос потенциометров

potPos[i] = analogRead(potPin[i]);

oldPotPos[i] = potPos[i];

truePotPos[i] = potPos[i];

if (truePotPos[i] > potBorder[i][1]) {

truePotPos[i] = potBorder[i][1];

}

else if (truePotPos[i] < potBorder[i][0]) {

truePotPos[i] = potBorder[i][0];

}

truePotPos[i] = (truePotPos[i] - potBorder[i][0]) / (potBorder[i][1] - potBorder[i][0]) \* 1000;

truePotPos[i] = (int)round(truePotPos[i]) / POTERROR \* POTERROR;

oldTruePotPos[i] = truePotPos[i];

truePotPosMap[i] = map(truePotPos[i], 0, 1000, 0, 1000);

double vf = truePotPos[i] / 1000;

oldTruePotPosMap[i] = truePotPosMap[i];

Serial.print("p0");

Serial.print(" ");

if (vf!=1){

Serial.print(vf);

}else{

Serial.print("1.00");

}

Serial.println("z");

}

////////////////////////////////// 2020 raz

//////////////НАСТРОЙКИ РАЗЪЕМОВ 2///////////////////

for (i = 0; i < razNUM; i++) { // Опрос потенциометров

razPos[i] = analogRead(razPin[i]);

oldrazPos[i] = razPos[i];

truerazPos[i] = razPos[i];

if (truerazPos[i] > razBorder[i][1]) {

truerazPos[i] = razBorder[i][1];

}

else if (truerazPos[i] < razBorder[i][0]) {

truerazPos[i] = razBorder[i][0];

}

truerazPos[i] = (truerazPos[i] - razBorder[i][0]) / (razBorder[i][1] - razBorder[i][0]) \* 1000;

truerazPos[i] = (int)round(truerazPos[i]) / razERROR \* razERROR;

oldTruerazPos[i] = truerazPos[i];

#ifndef razSetup

if (razAnswer[i] >= truerazPos[i] - razERRORAnswer && razAnswer[i] <= truerazPos[i] + razERRORAnswer)

{

//if (razAnswer[i] == truerazPos[i]) {

Serial.print("c");

Serial.print(i + 2);

Serial.print(" ");

Serial.println("onz");

truerazAnswer[i] = 1;

} else {

Serial.print("c");

Serial.print(i + 2);

Serial.print(" ");

Serial.println("offz");

}

oldTruerazAnswer[i] = truerazAnswer[i];

#else

Serial.print("c");

Serial.print(i + 2);

Serial.print(" ");

Serial.println(truerazPos[i]);

#endif

}

//////////////////////////////////////2020 raz end

//// ////

// НАСТРОЙКИ РАЗЪЕМОВ ШР //

//// ////

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

pinMode(connPin[i], INPUT); // Настройка пина на вход

digitalWrite(connPin[i], HIGH); // Подтяжка пина к питанию

if(connType[i] == 0){

connDelay[i] = CONNDELAY0;

}

else if(connType[i] == 1){

connDelay[i] = CONNDELAY1;

}

}

for(i = 0; i < CONNNUM; i++){ // Опрос разъемов

if(digitalRead(connPin[i]) == HIGH){

Serial.print("c");

Serial.print(i);

Serial.print(" off");

Serial.println("z");

}

else if(digitalRead(connPin[i]) == LOW){

Serial.print("c");

Serial.print(i);

Serial.print(" on");

Serial.println("z");

connPos[i] = true;

}

n++;

}

//// ////

// НАСТРОЙКИ СВЕТОДИОДОВ tyt //

//// ////

FastLED.addLeds<WS2811, LED\_PIN, RGB>(led, LED\_NUM); // Определение пина и количества

FastLED.setBrightness(255); // Задание яркости

for (i = 0; i < LED\_NUM; i++) { // Выключение светодиодов

led[i] = CRGB(0, 0, 0);

ledPos[i] = false;

n2++;

}

FastLED.show();

///////////////////НАСТРОЙКИ МОТОРОВ////////////////////

#ifdef STEPPER\_SETUP

Serial.println("STEPPER SETUP DEFINED");

#endif

#ifdef STEPPER\_DEBUG

Serial.println("STEPPER DEBUG DEFINED");

#endif

#ifndef STEPPER\_SETUP

for (i = 0; i < STEPPER\_NUM; i++) { // Включение установки нуля

stepperZeroSettingStepA[i] = true;

}

#endif

for (i = 0; i < STEPPER\_NUM; i++) { // Настройка двигателей

stepper[i].setMaxSpeed(STEPPER\_SPEED);

stepper[i].setSpeed(STEPPER\_SPEED);

stepper[i].setAcceleration(STEPPER\_ACCELERATION);

}

//// ////

// НАСТРОЙКА ЭНКОДЕРОВ //

//// ////

#ifdef ENCODERSETUP

Serial.println("ENCODERSETUP DEFINED");

#endif

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО ЭНКОДЕРОВ!

enc0.begin(); // Инициализация работы энкодера

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО ЭНКОДЕРОВ!

rawPos[0] = enc0.rawPos(); // Опрос энкодера

#ifdef ENCSETUP

for(i = 0; i < ENCNUM; i++){ // Вывод текущего положения энкодера

oldRawPos[i] = rawPos[i];

Serial.print("e");

Serial.print(i);

Serial.print(" ");

Serial.println(rawPos[i]);

}

#endif

#ifndef ENCSETUP

for(i = 0; i < ENCNUM; i++){ // Вывод в порт ПК текущего состояния энкодера

oldRawPos[i] = rawPos[i];

if(rawPos[i] == 255){

#ifdef DEBUG

Serial.print("e");

Serial.print(i);

Serial.print(" ");

Serial.println("ERROR");

#endif

encPos[i] = encBorder[i][0];

}

else{

if (rawBorder[i][0] < rawBorder[i][1]){

rawNorm[i] = rawBorder[i][1] - rawBorder[i][0];

encPoint[i] = (rawBorder[i][0] + 128 - rawBorder[i][1] - 1) / 2.0;

if(rawBorder[i][0] < encPoint[i]){

encPoint[i] += rawBorder[i][1];

}

else{

encPoint[i] -= rawBorder[i][0];

}

if(encPoint[i] > 0){

if((rawPos[i] >= rawBorder[i][0]) && (rawPos[i] <= rawBorder[i][1])){

encPos[i] = (rawPos[i] - rawBorder[i][0]) / rawNorm[i];

}

else if((rawPos[i] < encPoint[i]) && (rawPos[i] >= rawBorder[i][1])){

encPos[i] = 1.00;

}

else{

encPos[i] = 0.00;

}

}

else{

if((rawPos[i] >= rawBorder[i][0]) && (rawPos[i] <= rawBorder[i][1])){

encPos[i] = (rawPos[i] - rawBorder[i][0]) / rawNorm[i];

}

else if((rawPos[i] > -encPoint[i]) && (rawPos[i] <= rawBorder[i][0])){

encPos[i] = 0.00;

}

else{

encPos[i] = 1.00;

}

}

}

else{

rawNorm[i] = 128 - rawBorder[i][0] + rawBorder[i][1] + 1;

encPoint[i] = (rawBorder[i][0] - rawBorder[i][1] - 1) / 2.0;

encPoint[i] += rawBorder[i][1];

if(rawPos[i] >= rawBorder[i][0]){

encPos[i] = (rawPos[i] - rawBorder[i][0]) / rawNorm[i];

}

else if(rawPos[i] <= rawBorder[i][1]){

encPos[i] = (rawPos[i] + 128 - rawBorder[i][0]) / rawNorm[i];

}

else if(rawPos[i] < encPoint[i]){

encPos[i] = 1.00;

}

else{

encPos[i] = 0.00;

}

}

if(encType[i] == 0){

encPos[i] = round(encPos[i] \* (encBorder[i][1] - encBorder[i][0]));

}

if(encType[i] == 3){

encPos[i] = round((1 - encPos[i]) \* (encBorder[i][1] - encBorder[i][0]));

if(encPos[i] == encBorder[i][1]){

encPos[i] = encBorder[i][0];

}

}

}

oldEncPos[i] = encPos[i];

Serial.print("e");

Serial.print(i);

Serial.print(" ");

Serial.print(encPos[i]);

Serial.println("z");

n++;

}

#endif

} // КОНЕЦ НАСТРОЕК

void loop() {

////////////////////////////////////////////////////////////////////////////////////

// РАБОТА //

////////////////////////////////////////////////////////////////////////////////////

if(checkElem == false)// Проверка окончания всех настроек

/\*if (n == ELEMNUM)\*/{

//Serial.print("SETUP DONE");

Serial.println("stop");

checkElem = true;

}

//// ////

// РАБОТА КНОПОК //

//// ////

for(int i = 0; i < BUTTNUM; i++){ // Опрос кнопок

if(buttRev[i] == false){

if((digitalRead(buttPin[i]) == HIGH) && (buttPos[i] == true)){

t = millis();

if((t - buttTime[i]) > BUTTDELAY){

Serial.print("b");

Serial.print(i);

Serial.print(" off");

Serial.println("z");

buttTime[i] = t;

buttPos[i] = false;

}

}

else if((digitalRead(buttPin[i]) == LOW) && (buttPos[i] == false)){

t = millis();

if((t - buttTime[i]) > BUTTDELAY){

Serial.print("b");

Serial.print(i);

Serial.print(" on");

Serial.println("z");

buttTime[i] = t;

buttPos[i] = true;

}

}

}

else{

if((digitalRead(buttPin[i]) == HIGH) && (buttPos[i] == false)){

t = millis();

if((t - buttTime[i]) > BUTTDELAY){

Serial.print("b");

Serial.print(i);

Serial.print(" on");

Serial.println("z");

buttTime[i] = t;

buttPos[i] = true;

}

}

else if((digitalRead(buttPin[i]) == LOW) && (buttPos[i] == true)){

t = millis();

if((t - buttTime[i]) > BUTTDELAY){

Serial.print("b");

Serial.print(i);

Serial.print(" off");

Serial.println("z");

buttTime[i] = t;

buttPos[i] = false;

}

}

}

}

//// ////

// РАБОТА ПРЕДОХРАНИТЕЛЕЙ //

//// ////

for (int i = 0; i < FUSENUM; i++) { // Опрос предохранителей

if (digitalRead(fuseIn[i]) == LOW && fuseLog[i] == false) {

delay(50);

Serial.print("f");

Serial.print(i);

Serial.print(" on");

Serial.println("z");

fuseLog[i] = true;

}

else if (digitalRead(fuseIn[i]) == HIGH && fuseLog[i] == true) {

delay(50);

Serial.print("f");

Serial.print(i);

Serial.print(" off");

Serial.println("z");

fuseLog[i] = false;

}

}

//// ////

// РАБОТА ПОТЕНЦИОМЕНТРОВ //

for (i = 0; i < POTNUM; i++) { // Опрос потенциометров

potPos[i] = analogRead(potPin[i]);

if (abs(potPos[i] - oldPotPos[i]) > POTERROR) {

oldPotPos[i] = potPos[i];

truePotPos[i] = potPos[i];

if (truePotPos[i] > potBorder[i][1]) {

truePotPos[i] = potBorder[i][1];

}

else if (truePotPos[i] < potBorder[i][0]) {

truePotPos[i] = potBorder[i][0];

}

truePotPos[i] = (truePotPos[i] - potBorder[i][0]) / (potBorder[i][1] - potBorder[i][0]) \* 1000;

truePotPos[i] = (int)round(truePotPos[i]) / POTERROR \* POTERROR;

if (potType[i] == 0) {

if (oldTruePotPos[i] != truePotPos[i]) {

oldTruePotPos[i] = truePotPos[i];

truePotPosMap[i] = map(truePotPos[i], 0, 1000, 0, 1000);

double vf2 = truePotPos[i] / 1000;

oldTruePotPosMap[i] = truePotPosMap[i];

Serial.print("p0");

Serial.print(" ");

if (vf2!=1){

Serial.print(vf2);

}else{

Serial.print("1.00");

}

Serial.println("z");

}

}

}

}

//////////////////////////////////////2020 raz

for (i = 0; i < razNUM; i++) { // Опрос потенциометров

traz = millis();

if ((traz - razTime[i]) > razDELAY) {

razPos[i] = analogRead(razPin[i]);

if (abs(razPos[i] - oldrazPos[i]) > razERROR) {

oldrazPos[i] = razPos[i];

truerazPos[i] = razPos[i];

if (truerazPos[i] > razBorder[i][1]) {

truerazPos[i] = razBorder[i][1];

}

else if (truerazPos[i] < razBorder[i][0]) {

truerazPos[i] = razBorder[i][0];

}

truerazPos[i] = (truerazPos[i] - razBorder[i][0]) / (razBorder[i][1] - razBorder[i][0]) \* 1000;

truerazPos[i] = (int)round(truerazPos[i]) / razERROR \* razERROR;

if (razType[i] == 0) {

if (oldTruerazPos[i] != truerazPos[i]) {

oldTruerazPos[i] = truerazPos[i];

//if (razAnswer[i] == truerazPos[i]) {

#ifndef razSetup

if (razAnswer[i] >= truerazPos[i] - razERRORAnswer && razAnswer[i] <= truerazPos[i] + razERRORAnswer)

{

truerazAnswer[i] = 1;

} else {

truerazAnswer[i] = 0;

}

if (truerazAnswer[i] != oldTruerazAnswer[i]) {

Serial.print("c");

Serial.print(i + 2);

Serial.print(" ");

if (truerazAnswer[i] == 1) {

Serial.println("onz");

} else {

Serial.println("offz");

}

oldTruerazAnswer[i] = truerazAnswer[i];

/\*

Serial.print("p");

Serial.print(i);

Serial.print(" ");

Serial.print(truerazPos[i]);

Serial.println("z");

\*/

}

#else

Serial.print("c");

Serial.print(i + 2);

Serial.print(" ");

Serial.println(truerazPos[i]);

#endif

}

}

}

razTime[i] = traz;

}

}

//////////////////////////////////////2020 raz end

//// ////

// РАБОТА РАЗЪЕМОВ ШР //

//// ////

for(int i = 0; i < CONNNUM; i++){ // Опрос разъемов

if((digitalRead(connPin[i]) == HIGH) && (connPos[i] == true)){

t = millis();

if((t - connTime[i]) > connDelay[i]){

Serial.print("c");

Serial.print(i);

Serial.print(" off");

Serial.println("z");

connTime[i] = t;

connPos[i] = false;

}

}

else if((digitalRead(connPin[i]) == LOW) && (connPos[i] == false)){

t = millis();

if((t - connTime[i]) > connDelay[i]){

Serial.print("c");

Serial.print(i);

Serial.print(" on");

Serial.println("z");

connTime[i] = t;

connPos[i] = true;

}

}

}

///////////////////////РАБОТА МОТОРОВ////////////////////////////////////////

for (i = 0; i < STEPPER\_NUM; i++) {

// УСТАНОВКА НУЛЯ ШАГОВЫХ ДВИГАТЕЛЕЙ //

if (stepperZeroSettingStepA[i] == true) {

if (digitalRead(stepperOptronPin[i]) == LOW) {

stepper[i].setMaxSpeed(100);

stepper[i].setSpeed(50);

stepper[i].setAcceleration(8000);

stepper[i].move(-800);

stepperZeroSettingStepA[i] = false;

stepperZeroSettingStepB[i] = true;

stepperZeroSettingLock[i] = true;

#ifdef STEPPER\_DEBUG

Serial.println("a-b");

#endif

}

else {

stepperZeroSettingStepA[i] = false;

stepperZeroSettingStepC[i] = true;

#ifdef STEPPER\_DEBUG

Serial.println("a-c");

#endif

}

stepperBuzy[i] = true;

}

if (stepperZeroSettingStepB[i] == true) {

if (digitalRead(stepperOptronPin[i]) == LOW) {

stepper[i].run();

}

else {

stepper[i].setMaxSpeed(STEPPER\_ZERO\_SETTING\_SPEED\_B);

stepper[i].moveTo(stepper[i].currentPosition());

stepperZeroSettingStepB[i] = false;

stepperZeroSettingStepC[i] = true;

stepperZeroSettingLock[i] = false;

stepperZeroSettingTimeC[i] = millis();

#ifdef STEPPER\_DEBUG

Serial.println("b-c");

#endif

}

}

if (stepperZeroSettingStepC[i] == true && stepper[i].distanceToGo() == 0

&& (millis() - stepperZeroSettingTimeC[i]) > STEPPER\_ZERO\_SETTING\_DEAD\_TIME\_C) {

stepper[i].move(1);

stepperZeroSettingStepC[i] = false;

stepperZeroSettingStepD[i] = true;

stepperZeroSettingTimeD[i] = millis();

#ifdef STEPPER\_DEBUG

Serial.println("c-d");

#endif

}

if (stepperZeroSettingStepD[i] == true && stepper[i].distanceToGo() == 0

&& (millis() - stepperZeroSettingTimeD[i]) > STEPPER\_ZERO\_SETTING\_DEAD\_TIME\_D) {

if (digitalRead(stepperOptronPin[i]) == LOW) {

stepper[i].move(-1);

stepperZeroSettingStepD[i] = false;

stepperZeroSettingStepE[i] = true;

#ifdef STEPPER\_DEBUG

Serial.println("d-e");

#endif

}

else {

stepperZeroSettingStepD[i] = false;

stepperZeroSettingStepC[i] = true;

#ifdef STEPPER\_DEBUG

Serial.println("d-c");

#endif

}

}

if (stepperZeroSettingStepE[i] == true && stepper[i].distanceToGo() == 0) {

stepper[i].setMaxSpeed(STEPPER\_SPEED);

stepper[i].setSpeed(STEPPER\_SPEED);

stepper[i].setAcceleration(STEPPER\_ACCELERATION);

stepperZeroSettingStepE[i] = false;

stepperZeroPos[i] = stepper[i].currentPosition();

#ifdef STEPPER\_SETUP

stepperBuzy[i] = false;

Serial.print(i);

Serial.print("\*=");

Serial.print(stepperZeroPos[i]);

Serial.print(" ");

Serial.print(i);

Serial.println("-=0");

#endif

#ifndef STEPPER\_SETUP

stepperZeroSettingStepF[i] = true;

stepper[i].move(stepperTrueZeroPos[i]);

#ifdef STEPPER\_DEBUG

Serial.println("d-f");

#endif

#endif

}

#ifndef STEPPER\_SETUP

if (stepperZeroSettingStepF[i] == true & stepper[i].distanceToGo() == 0) {

stepperZeroSettingStepF[i] = false;

stepperBuzy[i] = false;

// Serial.print(arduinoSymbol);

// Serial.print(stepperSymbol);

// if(i < 10){

// Serial.print(0);

// }

// Serial.print(i);

// Serial.print(0);

// Serial.println(endSymbol);

if (i == (STEPPER\_NUM - 1))

{

Serial.println("stop");

}

}

// СДВИГ СТРЕЛКИ ШАГОВЫХ ДВИГАТЕЛЕЙ //

if (stepperMoveStepA[i] == true && stepperBuzy[i] == false) {

stepper[i].moveTo(stepperZeroPos[i] + stepperTrueZeroPos[i] + stepperPos[i]);

stepperMoveStepA[i] = false;

// stepperMoveStepB[i] = true;

}

/\* if(stepperMoveStepB[i] == true && stepper[i].distanceToGo() == 0){

stepperMoveStepB[i] = false;

Serial.print(arduinoSymbol);

Serial.print(stepperSymbol);

if(i < 10){

Serial.print(0);

}

Serial.print(i);

Serial.print(stepperPos[i]);

Serial.println(endSymbol);

}\*/

#endif

}

/////////////////////////////////

//// ////

// РАБОТА ЭНКОДЕРОВ //

//// ////

// ИЗМЕНИТЬ, ЕСЛИ ИЗМЕНИЛОСЬ КОЛИЧЕСТВО ЭНКОДЕРОВ!

rawPos[0] = enc0.rawPos(); // Опрос энкодера

#ifdef ENCSETUP

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

if (rawPos[i] != oldRawPos[i]){ // Вывод текущего положения энкодера при наличии изменений

oldRawPos[i] = rawPos[i];

Serial.print("e");

Serial.print(i);

Serial.print(" ");

Serial.println(rawPos[i]);

}

}

#endif

#ifndef ENCSETUP

for(i = 0; i < ENCNUM; i++){ // Вывод в порт ПК текущего состояния энкодера

if (rawPos[i] != oldRawPos[i]){

oldRawPos[i] = rawPos[i];

if(rawPos[i] == 255){

#ifdef DEBUG

Serial.print("e");

Serial.print(i);

Serial.print(" ");

Serial.println("ERROR");

#endif

}

else{

if (rawBorder[i][0] < rawBorder[i][1]){

if(encPoint[i] > 0){

if((rawPos[i] >= rawBorder[i][0]) && (rawPos[i] <= rawBorder[i][1])){

encPos[i] = (rawPos[i] - rawBorder[i][0]) / rawNorm[i];

}

else if((rawPos[i] < encPoint[i]) && (rawPos[i] >= rawBorder[i][1])){

encPos[i] = 1.00;

}

else{

encPos[i] = 0.00;

}

}

else{

if((rawPos[i] >= rawBorder[i][0]) && (rawPos[i] <= rawBorder[i][1])){

encPos[i] = (rawPos[i] - rawBorder[i][0]) / rawNorm[i];

}

else if((rawPos[i] > -encPoint[i]) && (rawPos[i] <= rawBorder[i][0])){

encPos[i] = 0.00;

}

else{

encPos[i] = 1.00;

}

}

}

else{

if(rawPos[i] >= rawBorder[i][0]){

encPos[i] = (rawPos[i] - rawBorder[i][0]) / rawNorm[i];

}

else if(rawPos[i] <= rawBorder[i][1]){

encPos[i] = (rawPos[i] + 128 - rawBorder[i][0]) / rawNorm[i];

}

else if(rawPos[i] < encPoint[i]){

encPos[i] = 1.00;

}

else{

encPos[i] = 0.00;

}

}

if(encType[i] == 0){

encPos[i] = round(encPos[i] \* (encBorder[i][1] - encBorder[i][0]));

}

if(encType[i] == 3){

encPos[i] = round((1 - encPos[i]) \* (encBorder[i][1] - encBorder[i][0]));

if(encPos[i] == encBorder[i][1]){

encPos[i] = encBorder[i][0];

}

}

if(encPos[i] != oldEncPos[i]){

oldEncPos[i] = encPos[i];

Serial.print("e");

Serial.print(i);

Serial.print(" ");

Serial.print(encPos[i]);

Serial.println("z");

}

}

}

}

#endif

////////////////////////////////////////////////////////////////////////////////////

//////// ////////

//// ////

// СВЯЗЬ С ПК //

//// ////

//////// ////////

////////////////////////////////////////////////////////////////////////////////////

if (Serial.available() > 0) { // Проверка, что есть данные от ПК

str = Serial.readStringUntil('\n');

#ifdef DEBUG

/\* Serial.print("USER: ");

Serial.println(str); \*/

#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//// ////

// СВЯЗЬ С ПК ДЛЯ КНОПОК //

//// ////

if (str.substring(0, 4).equals("butt")){// Принудительный опрос кнопки

i = str.substring(4, 6).toInt();

if(i < BUTTNUM){

if(buttPos[i] == false){

Serial.print("b");

Serial.print(i);

Serial.print(" off");

Serial.println("z");

}

else if(buttPos[i] == true){

Serial.print("b");

Serial.print(i);

Serial.print(" on");

Serial.println("z");

}

}

}

//// ////

// СВЯЗЬ С ПК ДЛЯ ПРЕДОХРАНИТЕЛЕЙ //

//// ////

else if (str.substring(0, 4).equals("fuse")) { // Принудительный опрос предохранителя

i = str.substring(4, 5).toInt();

if (digitalRead(fuseIn[i]) == HIGH) {

Serial.print("f");

Serial.print(i);

Serial.println(" off");

}

else if (digitalRead(fuseIn[i]) == LOW) {

Serial.print("f");

Serial.print(i);

Serial.println(" on");

}

}

//// ////

// СВЯЗЬ С ПК ДЛЯ РАЗЪЕМОВ ШР //

//// ////

if (str.substring(0, 4).equals("conn")){// Принудительный опрос разъема

i = str.substring(4, 6).toInt();

if(i < CONNNUM){

if(connPos[i] == false){

Serial.print("c");

Serial.print(i);

Serial.print(" off");

Serial.println("z");

}

else if(connPos[i] == true){

Serial.print("c");

Serial.print(i);

Serial.print(" on");

Serial.println("z");

}

}

}

//// ////

// СВЯЗЬ С ПК ДЛЯ СВЕТОДИОДОВ //

//// ////

if (str.substring(0, 2).equals("on")) { // Включить светодиод

i2 = str.substring(2, 4).toInt();

str = led\_privod\_str[i2];

i = str.substring(2, 4).toInt();

// i = led\_privod[i];

if (i < LED\_NUM) {

led[i] = CRGB(0, 255, 0);

ledPos[i] = true;

FastLED.show();

}

}

else if (str.substring(0, 3).equals("off")) { // Выключить светодиод

i2 = str.substring(3, 5).toInt();

str = led\_privod\_str2[i2];

i = str.substring(3, 5).toInt();

// i = led\_privod[i];

if (i < LED\_NUM) {

led[i] = CRGB(0, 0, 0);

ledPos[i] = false;

FastLED.show();

}

}

////////////////////СВЯЗЬ МОТОРОВ////////////////////

#ifdef STEPPER\_SETUP

if (str.substring(1, 2).equals("+") && stepperBuzy[i] == false) {

// Принудительная установка нуля

stepperZeroSettingStepA[str.substring(0, 1).toInt()] = true;

}

else if (str.substring(1, 2).equals("\*")) { // Принудительный абсолютный сдвиг стрелки

stepper[str.substring(0, 1).toInt()].moveTo(str.substring(2, 13).toInt());

#ifdef STEPPER\_DEBUG

Serial.println(digitalRead(stepperOptronPin[str.substring(0, 1).toInt()]));

#endif

}

else if (str.substring(1, 2).equals("-")) { // Принудительный относительный сдвиг стрелки

i = str.substring(0, 1).toInt();

stepper[i].moveTo(str.substring(2, 13).toInt() + stepperZeroPos[i]);

}

#endif

if (str.substring(0, 1).equals("s")) {

//// ////

// СВЯЗЬ ДЛЯ ДВИГАТЕЛЯ //

//// ////

#ifndef STEPPER\_SETUP

if (str.substring(1, 2).equals(stepperSymbol) && str.substring(4, 5).equals("+")) {

// Принудительная установка нуля

stepperZeroSettingStepA[str.substring(2, 4).toInt()] = true;

}

else if (str.substring(0, 1).equals("s")) {

// Принудительный относительный сдвиг стрелки

i = str.substring(1, 2).toInt();

char myChar[11];

str.substring(6, 10).toCharArray(myChar, 11);

j2 = atol(myChar);

//j2 = j2 \* 4;

Serial.println(j2);

if (j2 >= stepperBorder[i][0] && j2 <= stepperBorder[i][1]) {

stepperPos[i] = j2;

stepperMoveStepA[i] = true;

#ifdef STEPPER\_DEBUG

Serial.println(digitalRead(stepperOptronPin[i]));

#endif

}

}

else if (str.substring(1, 2).equals(stepperSymbolB)) {

// Принудительный относительный сдвиг стрелки

i = str.substring(2, 4).toInt();

if (i != 0) {

char myChar[11];

str.substring(4, 15).toCharArray(myChar, 11);

j2 = atol(myChar);

/\*

if(j >= stepperBorder[i][0] && j <= stepperBorder[i][1]){

stepperPos[i] += j;

stepperMoveStepA[i] = true;

}

\*/

if (j2 >= 0 && j2 <= 800) {

if (j2 == 800) {

j2 = 0;

}

long myI;

myI = stepperPos[i] / 800;

if (abs(j2 + 800 \* myI - stepperPos[i]) < 400) {

stepperPos[i] = j2 + 800 \* myI;

}

else if (abs(j2 + 800 \* (myI + 1) - stepperPos[i]) < 400) {

stepperPos[i] = j2 + 800 \* (myI + 1);

}

else if (abs(j2 + 800 \* (myI - 1) - stepperPos[i]) < 400) {

stepperPos[i] = j2 + 800 \* (myI - 1);

}

else if (abs(j2 + 800 \* (myI + 2) - stepperPos[i]) < 400) {

stepperPos[i] = j2 + 800 \* (myI + 2);

}

else if (abs(j2 + 800 \* (myI - 2) - stepperPos[i]) < 400) {

stepperPos[i] = j2 + 800 \* (myI - 2);

}

stepperMoveStepA[i] = true;

}

}

}

#endif

}

//// ////

// СВЯЗЬ С ПК ДЛЯ ЭНКОДЕРОВ //

//// ////

else if(str.substring(0, 3).equals("enc")){ // Запрос состояния энкодера от ПК и вывод в порт ПК

i = str.substring(3, 4).toInt();

if(i < ENCNUM){

Serial.print("e");

Serial.print(i);

Serial.print(" ");

#ifdef ENCSETUP

// Serial.println(rawPos[i]);

#endif

#ifndef ENCSETUP

Serial.print(encPos[i]);

Serial.println("z");

#endif

}

}

} // КОНЕЦ СВЯЗИ С ПК

for (i = 0; i < STEPPER\_NUM; i++) {

if (stepperZeroSettingLock[i] == false) {

stepper[i].run();

}

#ifdef STEPPER\_DEBUG

if (stepper[i].currentPosition() != stepperDebugPos[i]) {

stepperDebugPos[i] = stepper[i].currentPosition();

Serial.println(stepperDebugPos[i]);

}

#endif

}

} // КОНЕЦ РАБОТЫ