```
//*********************************************************
//****** boomerX6.ino   *********************************
//*********************************************************
#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

BluetoothSerial SerialBT;

#include <Wire.h>
#include <Adafruit_MPU6050.h>

// Подключаем внутренние программные файлы.
#include "defin.h"  //Переменные и константы
#include "MadgwickAHRS.h" //Фильтр Маджвика
#include "core_motorstep.h"  //Описание моторов
#include "core_sonar.h" // Настрока сонара
#include "core_servo_hand.h" //Обмен командами
#include "core_irq_robot.h"  //Работа с прерываниями
#include "CalcSpeed.h" //Пересчет шагов
#include "move_case.h" //Обмен командами со смартфоном

//=====================================
//+++++++++++++++++++++++++++++++++++++
//=====================================
bool error_Flag = false;


//IBusBM IBus; // IBus object
//HardwareSerial FSkySerial(2); //rx in 16gpio

Adafruit_MPU6050 mpu;
sensors_event_t a, g, temper;

#include "storage.h"

uint32_t  t2, t0;
//Задаем GPIO для Wire
const int _SDA = 21;
const int _SCL = 22;
void setup()
{
  servo_hand_setup();
  sonar_setup();
  setup_motor_system();
  //Выключаем мотор перемещения

  //Wire.setPins(_SDA,_SCL); //можно использовать в 5й версии
  Wire.begin(_SDA, _SCL);
  Serial.begin(115200);

  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);//MPU6050_RANGE_1000_DEG
  mpu.setFilterBandwidth(MPU6050_BAND_260_HZ);
  delay(1000);
  Calc_CompensatorZ(5000);
  arming = false;
  timer_setup();
  Serial.print("start2");
  Serial.print("start3=");
  Serial.println(portTICK_PERIOD_MS);//configMAX_PRIORITIES);

  mpu.getEvent(&a, &g, &temper);
```

```
  commandRIGHT_WEEL_SPEED = 0;
  commandLEFT_WEEL_SPEED = 0;
  OldTime =  micros();
  Pereschet = millis() + 3000;
  SerialBT.begin("ESP32BALROBOT"); //Bluetooth device name
}

uint32_t Dtime = 5000;
uint32_t start_period = 0;
double dta = 0;

void loop()
{
  //Получаем от приемника два значения
  // 1. Командный угол поворота newROT_Angel_RAD - в радианах
  // 2. Командную скорость newWheelSpeed - в шагах на секунду
  BT_input();
  PrintDebug();

  micros_1 = micros();
  if ((micros_1 - OldTime) < 5000) return;

  Dtime = micros_1 - OldTime;
  Dtime = constrain(Dtime, 4000, 10000);
  Dt = double(Dtime) * 0.000001;
  OldTime = micros_1;

  mpu.getEvent(&a, &g, &temper);
  //Фильтр Маджвика
  MadgwickAHRSupdateIMU(Dt, g.gyro.x - CompensatorX , g.gyro.y - CompensatorY,
  g.gyro.z - CompensatorZ, a.acceleration.x, a.acceleration.y, a.acceleration.z);
  quat[0] = q0; quat[1] = q1; quat[2] = q2; quat[3] = q3;
  if ((millis() < Pereschet) && (!avtostart)) return;
  else avtostart = true;

  // преобразуем кватернион в углы Эйлера
  quat2Euler(&quat[0], &imu[0]);
  if (fabs(imu[1]) > 0.75) arming = false;
  else if ((fabs(imu[1]) < 0.55) && (!arming)){ arming = true; }

  opros_sys_EN_motors(); //Отключаем моторы в случае необходимости

  if (STEPER_EN_level == MOTORS_ON)
  {
    Error_ang = imu[1] - base_ang; //Ошибка угла наклона в радианах
    dError_ang = (Error_ang - OldError_ang) / Dt; //Скорость изменения ошибки угла
    наклона
    Error_speed = Sbo - CommandSpeed; //Ошибка скорости в шагах
    dError_speed = (Error_speed - OldError_speed) / Dt; //Скорость изменения ошибки
    скорости
    iError_speed += Error_speed * Dt * Ki_s; //Накопленная - интегральная ошибка
    скорости

    iError_speed = constrain(iError_speed, -2 * MAX_WEEL_SPEED, 2 * MAX_WEEL_SPEED);

    New_Speed = Error_ang * Kp_a + dError_ang * Kd_a + Error_speed * Kp_s +
    dError_speed * Kd_s + iError_speed;
    OldError_ang = Error_ang;
    OldError_speed = Error_speed;
    commandRIGHT_WEEL_SPEED = constrain(New_Speed - Turn, commandRIGHT_WEEL_SPEED -
    Dt * ACCELERATION_WEEL, commandRIGHT_WEEL_SPEED + Dt * ACCELERATION_WEEL);
    commandLEFT_WEEL_SPEED = constrain(New_Speed + Turn, commandLEFT_WEEL_SPEED - Dt
    * ACCELERATION_WEEL, commandLEFT_WEEL_SPEED + Dt * ACCELERATION_WEEL);
    OldSpeed = (commandLEFT_WEEL_SPEED + commandRIGHT_WEEL_SPEED) * 0.5 ;
  }
  else
  {
    New_Speed = OldSpeed = 0;
    commandRIGHT_WEEL_SPEED = 0;
    commandLEFT_WEEL_SPEED = 0;
  }
  CalcSpeedAndPosition(Dt);
}
```

```
140
141 void PrintDebug()
142 {
143   static uint32_t printTime = 0;
144   if (printTime < millis())
145   {
146     Serial.println("=");
147     Serial.print("X=");
148     Serial.println(imu[1]); //Serial.print("  acc=");
      Serial.print(a.acceleration.x);  Serial.print("  gyro=");
      Serial.println(g.gyro.x);
149     Serial.print("Y=");
150     Serial.println(imu[0]); //Serial.print("  acc=");
      Serial.print(a.acceleration.y);  Serial.print("  gyro=");
      Serial.println(g.gyro.y);
151     Serial.print("Z=");
152     Serial.println(imu[2]); //Serial.print("  acc=");
      Serial.print(a.acceleration.z);  Serial.print("  gyro=");
      Serial.println(g.gyro.z);
153     // Serial.print("flag_sonar=");Serial.println(flag_sonar);
154     Serial.print("arming=");
155     Serial.println(arming);
156     Serial.print("dError_ang="); Serial.println(dError_ang);
157     printTime = millis() + 200;
158   }
159 }
160
161 //*******************************************************
162 //****** CalcSpeed.h   ********************************
163 //*******************************************************
164
165 //*****************************************************************************
166 //*****************************************************************************
167 //********** функции обработки сосотояний моторов для режимов ******************
168 //*****************************************************************************
169 //*****************************************************************************
170 void opros_sys_EN_motors()
171 {
172   static bool wait_MOTORS_ON = false;
173   static uint32_t safe_start_timer = 0;
174   static bool realSTEPER_EN_level = MOTORS_OFF;
175   if ((arming == true) && (STEPER_EN_level != MOTORS_ON))
176   {
177     if (!wait_MOTORS_ON)
178     {
179       realSTEPER_EN_level = MOTORS_ON;
180       digitalWrite(GPIO_MOTORS_EN, MOTORS_ON);
181       safe_start_timer = 1000 + millis(); // даем секунду на безопасное включение
182       wait_MOTORS_ON = true;
183     }
184     else
185     {
186       if (safe_start_timer < millis())
187       {
188         STEPER_EN_level = MOTORS_ON;
189         wait_MOTORS_ON = false;
190       }
191     }
192   }
193   else
194   { if ((arming == false) && (realSTEPER_EN_level == MOTORS_ON))
195     {
196       realSTEPER_EN_level = MOTORS_OFF;
197       STEPER_EN_level = MOTORS_OFF;
198       digitalWrite(GPIO_MOTORS_EN, STEPER_EN_level);
199
200       wait_MOTORS_ON = false;
201     }
202   }
203 }
204 void CalcSpeedAndPosition(double Dt)
205 {
206   //Получаем от приемника два значения
```

```
207     // 1. Командный угол поворота newROT_Angel_RAD - в радианах
208     // 2. Командную скорость newWheelSpeed - в шагах на секунду
209
210
211     if (STEPER_EN_level == MOTORS_ON)
212     {
213       RIGHT_WEEL_SPEED = constrain(commandRIGHT_WEEL_SPEED, -MAX_WEEL_SPEED,
        MAX_WEEL_SPEED);
214       LEFT_WEEL_SPEED = constrain(commandLEFT_WEEL_SPEED, -MAX_WEEL_SPEED,
        MAX_WEEL_SPEED);
215     }
216     else
217     {
218       RIGHT_WEEL_SPEED = 0;
219       LEFT_WEEL_SPEED = 0;
220     }
221     reaLsPeedL = 0;
222     temp = fabs(LEFT_WEEL_SPEED);
223     if (temp >= MIN_WEEL_SPEED) {
224       LONG_STEP_WEEL_LEFT = uint32_t(100000.0 / temp);
225       LONG_STEP_WEEL_LEFT = constrain(LONG_STEP_WEEL_LEFT, 3, MaxLONGStep);
226       reaLsPeedL = 100000.0 / double(LONG_STEP_WEEL_LEFT);
227       if (LEFT_WEEL_SPEED < 0) reaLsPeedL = -reaLsPeedL;
228     } else  LONG_STEP_WEEL_LEFT = infinity_large_number;
229     if (LEFT_WEEL_SPEED < 0.0) dir_STEP_WEEL_LEFT = DIR_LEFT_BACKWARD;
230     else dir_STEP_WEEL_LEFT = DIR_LEFT_FORWARD;
231     {
232
233       static bool faza0 = false;
234       if (DIR_tec_WEEL_LEFT_ != dir_STEP_WEEL_LEFT)
235       {
236         if (!faza0)
237         {
238           reaLsPeedL = 0;
239
240           LONG_STEP_WEEL_LEFT = infinity_large_number; // Висим на шаге максимально
          долго не начиная следующий
241           faza0 = true;
242         }
243         else
244         {
245           DIR_tec_WEEL_LEFT_ = dir_STEP_WEEL_LEFT;
246           digitalWrite(GPIO_LEFT_WEEL_DIR, DIR_tec_WEEL_LEFT_);
247           reaLsPeedL = 0;
248           faza0 = false;
249         }
250       }
251       else faza0 = false;
252
253       counter_do_step_WEEL_LEFT_ = LONG_STEP_WEEL_LEFT;
254     }
255
256     reaLsPeedR = 0;
257     temp = fabs(RIGHT_WEEL_SPEED);
258     if (temp >= MIN_WEEL_SPEED) {
259       LONG_STEP_WEEL_RIGHT = uint32_t(100000.0 / temp);
260       LONG_STEP_WEEL_RIGHT = constrain(LONG_STEP_WEEL_RIGHT, 3, MaxLONGStep);
261
262       if (RIGHT_WEEL_SPEED < 0) reaLsPeedR -= 100000.0 / double(LONG_STEP_WEEL_RIGHT);
263       else reaLsPeedR += 100000.0 / double(LONG_STEP_WEEL_RIGHT);
264
265     } else LONG_STEP_WEEL_RIGHT = infinity_large_number;
266     if (RIGHT_WEEL_SPEED < 0.0) dir_STEP_WEEL_RIGHT = DIR_RIGHT_BACKWARD;
267     else dir_STEP_WEEL_RIGHT = DIR_RIGHT_FORWARD;
268     {
269       static bool faza0 = false;
270       if (DIR_tec_WEEL_RIGHT_ != dir_STEP_WEEL_RIGHT)
271       {
272         if (!faza0)
273         {
274           reaLsPeedR = 0;
275
276           LONG_STEP_WEEL_RIGHT = infinity_large_number; // Висим на шаге максимально
```

```
          долго не начиная следующий
277           faza0 = true;
278         }
279         else
280         {
281           DIR_tec_WEEL_RIGHT_ = dir_STEP_WEEL_RIGHT;
282           digitalWrite(GPIO_RIGHT_WEEL_DIR, DIR_tec_WEEL_RIGHT_);
283           faza0 = false;
284         }
285       }
286       else faza0 = false;
287       counter_do_step_WEEL_RIGHT_ = LONG_STEP_WEEL_RIGHT;
288     }
289
290
291     newSpeedflag_ = true;
292     //Sbo = (RIGHT_WEEL_SPEED + LEFT_WEEL_SPEED) / 2.0;
293     //reaLsPeed *= 0.5;
294     Sbo = (reaLsPeedL + reaLsPeedR) * 0.5;
295   }
296
297   //*************************************************************************
298   //**************Расчет для прямолинейного движения ************************
299   //*************************************************************************
300
301   //*******************************************************
302   //****** core_irq_robot.h   ********************************
303   //*******************************************************
304   //Создаем указатель на таймер
305   //генерации шагов для моторов робота
306   hw_timer_t * Timer = NULL;
307
308   // Объявляем переменные, хранящие текущее
309   // состояние уровня сигнала на GPIO шагов для моторов
310   //=======================================
311   // Флаг получения новой скорости
312   volatile bool newSpeedflag_ = false;
313   //+++++++++++++++++++++++++++++++++++++++
314   //Маска выключения высокого состояния шагов всех моторов
315   //Маски высокого состояния шагов для каждого мотора
316   //Маски подъема сигналов на GPIO шагов моторов
317   const uint32_t  STEP_mask_on_WEEL_LEFT_ = uint32_t(1) << GPIO_LEFT_WEEL_STEP;
318   const uint32_t  STEP_mask_on_WEEL_RIGHT_ = uint32_t(1) << GPIO_RIGHT_WEEL_STEP;
319   const uint32_t off_step = 0xFFFFFFFF ^  ( STEP_mask_on_WEEL_LEFT_ |
      STEP_mask_on_WEEL_RIGHT_);
320   //
321   ////Количество шагов, которые нужно сделать
322   //int32_t  STEPS_counter_WEEL_LEFT_ = 0;
323   //int32_t  STEPS_counter_WEEL_RIGHT_ = 0;
324
325   //================================================
326   //================================================
327   uint32_t sost;
328   //Скорость для расчета длительности следующего шага
329   //Направление шагов
330
331   bool DIR_tec_WEEL_LEFT_ = DIR_LEFT_FORWARD;
332   bool DIR_tec_WEEL_RIGHT_ = DIR_RIGHT_FORWARD;
333
334   // Новое время - длительность шага в 10микСек
335   uint32_t newcounter_do_step_WEEL_LEFT_ = 100;
336   uint32_t newcounter_do_step_WEEL_RIGHT_ = 100;
337
338   //Установленное время - длительность шага в 10микСек
339   uint32_t  counter_do_step_WEEL_LEFT_ = 100;
340   uint32_t  counter_do_step_WEEL_RIGHT_ = 100;
341
342   //Пройденное время в 10х мк.сек
343
344   //Прошедшее время текущего шага в 10микСек
345   uint32_t counter_WEEL_LEFT_ = 0;
346   uint32_t counter_WEEL_RIGHT_ = 0;
347   bool old_sost = false;
```

```
348
349
350   //+++++++++++++++++++++++++++++++++++++++//
351   //==== ==//
352   const uint32_t half_step_ = 2; //пусть будет на 2 такте шага
353
354   uint32_t LONG_STEP_WEEL_LEFT_;
355   uint32_t LONG_STEP_WEEL_LEFT;
356   bool dir_STEP_WEEL_LEFT_;
357   bool dir_STEP_WEEL_LEFT;
358
359   bool True_speed_WEEL_LEFT_;
360   bool True_speed_WEEL_LEFT;
361
362   uint32_t LONG_STEP_WEEL_RIGHT_;
363   uint32_t LONG_STEP_WEEL_RIGHT;
364   bool dir_STEP_WEEL_RIGHT_;
365   bool dir_STEP_WEEL_RIGHT;
366
367   bool True_speed_WEEL_RIGHT_;
368   bool True_speed_WEEL_RIGHT;
369
370   bool True_speed_WEEL_RIGHT_Old = false;
371   bool True_speed_WEEL_LEFT_Old = false;
372
373
374   //+++++++++++++++++++++++++++++++++++++++//
375   void IRAM_ATTR core_timer() {
376     //==========================================
377     //++++++++++ обработка шагов ++++++++++++++++
378     //==========================================
379     //Опустить шаг для всех моторов
380     // Заменяем множество опускающих команд на
381     sost = 0;
382
383     if (STEPER_EN_level == MOTORS_ON)
384     {
385       if (counter_WEEL_RIGHT_ > counter_do_step_WEEL_RIGHT_)//Если шаг закончен
386       {
387         sost |= STEP_mask_on_WEEL_RIGHT_; //Поднять шаг -  пока только флаг
388         //Счетчик длительности шага - обнуляем для начала нового шага
389         counter_WEEL_RIGHT_ = 0;
390         //Изменяем положение каретки
391       }
392       counter_WEEL_RIGHT_++; //Счетчик длительности шага
393
394       if (counter_WEEL_LEFT_ > counter_do_step_WEEL_LEFT_)//Если шаг закончен
395       {
396         sost |= STEP_mask_on_WEEL_LEFT_; //Поднять шаг -  пока только флаг
397         //Счетчик длительности шага - обнуляем для начала нового шага
398         counter_WEEL_LEFT_ = 0;
399       }
400       counter_WEEL_LEFT_++; //Счетчик длительности шага
401     }
402     //Если шаг нужно делать
403     //Опускаем сигналы шагов (пока только в переменной)
404     //Поднимаем шаги для моторов, которым это нужно
405     //Записываем значение с измененными битами шагов в регистр GPIO (0-31)
406     if (sost != 0)
407     {
408       old_sost = true;
409       REG_WRITE(GPIO_OUT_REG, (REG_READ(GPIO_OUT_REG) | sost));
410     }
411     // Если шаги делать не нужно, но нужно опустить
412     // поднятые при прошлом вызове импульсы шагов
413     else if (old_sost) {
414       old_sost = false;
415       // Опускаем все значения на шагов на 0
416       REG_WRITE(GPIO_OUT_REG, (REG_READ(GPIO_OUT_REG) & off_step));
417     }
418     //*****************************************************************************
419
420     if (sonarAVTO)
```

```
421     {
422       static uint32_t start_time = 0;
423       static uint32_t stop_time = 0;
424       static bool startFlag = false;
425       static bool stopFlag = false;
426       static uint32_t sonar_delay_time = 0;
427       static uint32_t duration = 0;
428       static uint32_t time_ = 0;
429       static uint32_t flag_sonar = FLAG_SONAR_START1;
430       if (flag_sonar == FLAG_SONAR_START1)
431       {
432         startFlag = false;
433         stopFlag = false;
434         // Переводим лимит из сантиметров в относительные величины (микросек.)
435         duration = (Limit * 588) / 100;
436         //Генерируем импульс
437         digitalWrite(SONAR_TRIG, HIGH);
438         flag_sonar = FLAG_SONAR_START2;
439 
440       }
441       else if (flag_sonar == FLAG_SONAR_START2)
442       {
443         digitalWrite(SONAR_TRIG, LOW);
444         //Ждем отражения импульса.
445         // Пересчитываем время в расстояние (по скорости звуку).
446         time_ = 0;
447         stop_time = duration;
448         flag_sonar = FLAG_SONAR_REGread;
449       }
450       else if (flag_sonar == FLAG_SONAR_REGread)
451       {
452         if ((time_ < duration) && (stopFlag == false))
453         {
454 
455           registr_in = REG_READ(GPIO_IN_REG);
456 
457           if (startFlag == false)
458           {
459             if ((registr_in & long_mm_GPIO) != 0) {
460               start_time = time_;
461               startFlag = true;
462             }
463           }
464           else
465           {
466             if ((registr_in & long_mm_GPIO) == 0)
467             {
468               stop_time = time_ - start_time;
469               stopFlag = true;
470             }
471           }
472           time_ ++;
473         }
474         else
475         {
476           flag_sonar = FLAG_SONAR_STOP;
477           dist_mm  = (stop_time * 1000) / 588;
478           sonar_delay_time = 0;
479         }
480       }
481       else if (flag_sonar == FLAG_SONAR_STOP)
482       {
483         if (sonar_delay_time < 8000) //80 миллисекунд пауза
484           sonar_delay_time++;
485         else
486           flag_sonar = FLAG_SONAR_START1;
487       }
488     }
489 
490     static bool startSig = false;
491     if (handACTIV)
492     {
493       if (LongSig < 2000) //частота 50Гц
```

```
      {
        LongSig++;
        if ((LongSig > handLS) && startSig)
        {
          startSig = false;
          REG_WRITE(GPIO_OUT_REG, (REG_READ(GPIO_OUT_REG) & hand_maskSigOff));
          //     digitalWrite(2,LOW);
        }
      }
      else
      {
        REG_WRITE(GPIO_OUT_REG, (REG_READ(GPIO_OUT_REG) | hand_maskSigOn));
        //   digitalWrite(2,HIGH);
        startSig = true;
        LongSig = 0;
      }
    }
    else if (startSig) {
      digitalWrite(2, LOW);
      startSig = false;
    }
  }


  //++++++++++++++++++++++++++++++++++++++++++++++++//
  //++++++++++++++++++++++++++++++++++++++++++++++++//
  //Готовим состояния масок
  // Создание таймеров, прикрепление к ним
  // функций обработчиков прерываний
  //++++++++++++++++++++++++++++++++++++++++++++++++//
  void timer_setup()
  {
    digitalWrite(GPIO_LEFT_WEEL_STEP, false);
    digitalWrite(GPIO_LEFT_WEEL_DIR, DIR_LEFT_FORWARD);
    DIR_tec_WEEL_LEFT_ = DIR_LEFT_FORWARD;

    digitalWrite(GPIO_RIGHT_WEEL_STEP, false);
    digitalWrite(GPIO_RIGHT_WEEL_DIR, DIR_RIGHT_FORWARD);
    DIR_tec_WEEL_RIGHT_ = DIR_RIGHT_FORWARD;


    // Создаем таймер
    Timer = timerBegin(2, 80, true);
    // Прикрепляем функциию обработчики
    timerAttachInterrupt(Timer, &core_timer, true);
    //Прерывание повторяется каждые 10 микросекунд
    timerAlarmWrite(Timer, 10, true);
    //Старт таймера, теперь прерывания будут генерироваться
    timerAlarmEnable(Timer);
  }


  //******************************************************
  //****** core_motorstep.h  *********************************
  //******************************************************

  #include "motors_gpio.h"
  //Позиция текущего положения коробки
  // 0 - минимальная
  //
  //=======================================
  //=======================================
  bool STEPER_EN_level = false;
  //int timer_stop; // для отключения двигателей при простое
  // Функция инициализации управления моторами.//
  //===============================
  void setup_motor_system()
  {
    STEPER_EN_level = MOTORS_OFF;
    pinMode(GPIO_LEFT_WEEL_STEP, OUTPUT);
    pinMode(GPIO_LEFT_WEEL_DIR, OUTPUT);
    pinMode(GPIO_RIGHT_WEEL_STEP, OUTPUT);
    pinMode(GPIO_RIGHT_WEEL_DIR, OUTPUT);
```

```
  pinMode(GPIO_MOTORS_EN, OUTPUT);

  digitalWrite(GPIO_LEFT_WEEL_STEP, false);
  digitalWrite(GPIO_LEFT_WEEL_DIR, DIR_LEFT_FORWARD);

  digitalWrite(GPIO_RIGHT_WEEL_STEP, false);
  digitalWrite(GPIO_RIGHT_WEEL_DIR, DIR_RIGHT_FORWARD);

  digitalWrite(GPIO_MOTORS_EN, STEPER_EN_level);
}

//*******************************************************
//****** core_servo_hand.h  ********************************
//*******************************************************

// Published values for SG90 servos; adjust if needed
int minUs_div10 = 80; //800
int maxUs_div10 = 250; //2500
int one_handGPIO = 23;
bool one_handattach = false;
#define hand_GPIO 2
const uint32_t hand_maskSigOn = 1 << hand_GPIO;
const uint32_t hand_maskSigOff = 0xFFFFFFFF ^ hand_maskSigOn;
bool handACTIV = false;
uint32_t handANGLE = 0;
uint32_t handLS = 0;
//static
uint32_t LongSig = 0;
void write_newHandrot(uint ang)
{
  ang = constrain(ang, 0, 180);
  handLS = map(ang, 0, 180, 80, 250);
  handANGLE = ang;
}
void servo_hand_setup()
{
  pinMode(hand_GPIO, OUTPUT);

  handACTIV = false;
  write_newHandrot(180);
}


void servo_hand_bottom(double test_angle)
{
  if (test_angle < - 0.7) //Если вошли в критичный режим
  {
     write_newHandrot(200);
  }
  else if (test_angle >  0.7) //Если вошли в критичный режим
  {
     write_newHandrot(0);
  }
}

void servo_hand_up()
{
  static uint32_t timess = 0;
  write_newHandrot(90);
}

//*******************************************************
//****** core_sonar.h  ********************************
//*******************************************************
//Создаем указатель на таймер
//генерации шагов для моторов робота
hw_timer_t * sonar_timer_t = NULL;

#define FLAG_SONAR_START1 1
#define FLAG_SONAR_START2 2
#define FLAG_SONAR_REGread 3
#define FLAG_SONAR_STOP 4

```

```
//Номер GPIO сонара.
#define SONAR_GPIO 17
#define SONAR_TRIG  27

bool sonarAVTO=false;
const uint32_t long_mm_GPIO = 1 << SONAR_GPIO;



volatile uint32_t dist_mm;


//= Измерение расстояния ============================
uint32_t registr_in;

uint32_t Limit = 55;



void sonar_setup()
{
  //=Режим пинов/портов ================================
  pinMode(SONAR_TRIG, OUTPUT);
  pinMode(SONAR_GPIO, INPUT);
  // Создаем таймер

  // Прикрепляем функциию обработчики

  //Прерывание повторяется каждые 10 микросекунд

  //Старт таймера, теперь прерывания будут генерироваться
sonarAVTO=true; //Включаем
}

//*****************************************************
//****** defin.h  *********************************
//*****************************************************

const int32_t dSmeshenie = 20;

//uint16_t motorONoffCommand = 500; //Мотор включен // если Меньше 100, то выключен.

uint32_t safe_start_timer = 0; // Таймер включения моторов.

double  CompensatorZ = 0;
double  CompensatorX = 0;
double  CompensatorY = 0;

int32_t newWheelSpeed; //скорость ведущего колеса
int32_t newSpeedLEFTwheel; //скорость левого колеса
int32_t newSpeedRIGHTwheel; //скорость правого колеса

double Turn=0;
int32_t newWEEL_SPEED_DOLI = 0;


const double DTIME = 0.005; //Типовое время в секундах между опросами
double _1_d_DTIME = 1.0 / DTIME; //Типовое перевернутое время в секундах между
опросами
//const int32_t MiniROT = (maxSPEED_ROT_LEFT_ * 5) / 1000;
const uint32_t MaxLONGStep = 100000 / 400;

double temp = 0;
//int FLAGGG = 0;

const double MAX_WEEL_SPEED = 15000;//28000;//25000; //Шагов в сек
const double MIN_WEEL_SPEED = 400;//400; //Шагов в сек
double ACCELERATION_WEEL = 37000;//500 //Шагов в сек
double commandRIGHT_WEEL_SPEED = 0;
double commandLEFT_WEEL_SPEED = 0;
double WEEL_SPEED = 0;
double LEFT_WEEL_SPEED = 0;
double RIGHT_WEEL_SPEED = 0;
```

```
712
713   //Колебание колеса для уменьшения трения при поворотах на месте
714   //const int32_t maxfluctuation = 3000;
715   //
716   //volatile int32_t WEEL_POSITION_RIGHT = 0;
717   //volatile int32_t WEEL_POSITION_LEFT  = 0;
718
719   //++++++++++++++++++++++++++++++++++++
720   uint32_t Dtime_to_off_power = 30000;
721   uint32_t millis_in_time;
722   const uint32_t dScansensor = 50;
723   uint32_t ControlDistance = 150;
724   uint32_t Pereschet;
725   bool avtostart=false;
726   bool arming = false;
727   bool Varming = false;
728
729   uint32_t t_period = 5000;
730   uint32_t micros_;
731   int i = 0, j = 0; //Счетчики
732   float imu[3];
733   float quat[4];
734   float e[3];
735   double Dt = 0;
736
737   double reaLsPeedL=0;
738   double reaLsPeedR=0;
739   double Sbo=0;
740   const uint32_t infinity_large_number=1000000; //Некоторое большое число
741
742
743   double New_Speed = 0;//новая скорость
744   double OldSpeed = 0; //Скорость от прошлой итерации
745   double  Error_ang = 0; //Ошибка угла
746   double  OldError_ang = 0; //Ошибка угла
747   double  base_ang = 0; //Базовый угол
748   double  dError_ang = 0; //Скорость изменения ошибки
749
750   double  Error_speed = 0; //Ошибка скорости в шагах
751   double  dError_speed; //Скорость изменения ошибки скорости
752   double  OldError_speed; //Ошибка скорости в шагах
753   double  iError_speed;//Накопленная - интегральная ошибка скорости
754
755   double  Kp_a = 200000; //Коэффициент при ошибке угла
756   double  Kd_a = 20000;//0.1; //Коэффициент при скорости изменения ошибки угла
757   double  Kp_s = 4;//0.1;//1; //Коэффициент при ошибке скорости движения
758   double  Kd_s = 0.001; //Коэффициент при скорости изменения ошибки скорости движения
759   double  Ki_s = 4;//10; //Коэффициент при интегральной ошибке скорости
760
761   uint32_t micros_1 = 0;
762   uint32_t OldTime = 0;
763
764   double CommandSpeed = 0;
765   double OldCommandSpeed = 0;
766
767   //*****************************************************
768   //****** motors_gpio.h  ********************************
769   //*****************************************************
770   #define GPIO_LEFT_WEEL_STEP 16// //Пин шагового шага
771   #define GPIO_LEFT_WEEL_DIR 19// //Пин направления левого шага
772
773   #define GPIO_RIGHT_WEEL_STEP 15// //Пин шагового шага
774   #define GPIO_RIGHT_WEEL_DIR 13// ////Пин направления правого шага
775
776   #define GPIO_MOTORS_EN 5 //Пин включения моторов
777
778   // Константы направления вращения приводов
779
780   bool DIR_LEFT_FORWARD = true;
781   bool DIR_LEFT_BACKWARD = !DIR_LEFT_FORWARD;
782
783   bool DIR_RIGHT_FORWARD = false;
784   bool DIR_RIGHT_BACKWARD = !DIR_RIGHT_FORWARD;
```

```
785
786   const bool MOTORS_ON = false;
787   const bool MOTORS_OFF = !MOTORS_ON;
788
789   //******************************************************
790   //****** move_case.h  ********************************
791   //******************************************************
792   uint32_t MOVE_PERIOD = 300;
793   uint32_t   time_start_move = 0;
794   const int32_t  MOVSPEED0 = (MAX_WEEL_SPEED / 100) * 3;
795   const int32_t  MOVSPEED1 = (MAX_WEEL_SPEED / 100) * 5;
796   const int32_t  MOVSPEED2 = (MAX_WEEL_SPEED / 100) * 10;
797   const int32_t  MOVSPEED3 = (MAX_WEEL_SPEED / 100) * 15;
798   const int32_t  MOVSPEED4 = (MAX_WEEL_SPEED / 100) * 20;
799   const int32_t  MOVSPEED5 = (MAX_WEEL_SPEED / 100) * 25;
800   const int32_t  MOVSPEED6 = (MAX_WEEL_SPEED / 100) * 30;
801   const int32_t  MOVSPEED7 = (MAX_WEEL_SPEED / 100) * 35;
802   const int32_t  MOVSPEED8 = (MAX_WEEL_SPEED / 100) * 40;
803   const int32_t  MOVSPEED9 = (MAX_WEEL_SPEED / 100) * 45;
804   const int32_t  MOVSPEED10 = (MAX_WEEL_SPEED / 100) * 50;
805   int32_t ASPEED = MOVSPEED2;
806   int32_t DeltaSpeed_100sec = 10000;
807   uint32_t Timing = 0;
808
809   bool BT_input()
810   {
811     static bool local_shot_dist = false;
812     static char bt_input;
813
814     if (SerialBT.available())
815     {
816       bt_input = (char)SerialBT.read();
817     }
818     else
819     {
820       if (time_start_move < millis())
821       {
822         // CommandSpeed = 0; //Обнуляем скорость управляемого движения.
823         if (CommandSpeed > 0) CommandSpeed = constrain(CommandSpeed -
          DeltaSpeed_100sec, 0, CommandSpeed); //Обнуляем скорость управляемого движения.
824         if (CommandSpeed < 0) CommandSpeed = constrain(CommandSpeed +
          DeltaSpeed_100sec, CommandSpeed, 0); //Обнуляем скорость управляемого движения.
825         Turn = 0;
826       }
827       return false;
828     }
829     time_start_move = millis() + MOVE_PERIOD;
830
831     if ((dist_mm < 450) && ((bt_input == 'F') || (bt_input == 'G') || (bt_input ==
      'I')))
832     {
833       bt_input = 'L';
834       CommandSpeed = -ASPEED;
835       Turn = -ASPEED;
836       bt_input = 'e';
837       //Timing = millis() + 500;
838       //local_shot_dist = true; //
839     }
840     else
841     {
842       // if (Timing < millis()) local_shot_dist = false; //
843     }
844
845     //====================
846
847     switch (bt_input)
848     {
849       case 'S': // вперед
850         //      CommandSpeed = 0; //Обнуляем скорость управляемого движения.
851         if (CommandSpeed > 0) CommandSpeed = constrain(CommandSpeed -
          DeltaSpeed_100sec, 0, CommandSpeed); //Обнуляем скорость управляемого движения.
852         if (CommandSpeed < 0) CommandSpeed = constrain(CommandSpeed +
          DeltaSpeed_100sec, CommandSpeed, 0); //Обнуляем скорость управляемого движения.
```

```
853         if (local_shot_dist)
854         {
855           Turn = -ASPEED / 2;
856         }
857         else
858         {
859           Turn = 0;
860         }
861         break;
862       case 'F': // вперед
863 
864         if (!local_shot_dist)
865         {
866           Turn = 0;
867           CommandSpeed = constrain(CommandSpeed + DeltaSpeed_100sec, CommandSpeed,
            ASPEED);
868         }
869         else
870         {
871           Turn = -ASPEED * 2;
872           //Обнуляем скорость управляемого движения.
873           if (CommandSpeed > 0) CommandSpeed = constrain(CommandSpeed -
            DeltaSpeed_100sec, 0, CommandSpeed);
874           if (CommandSpeed < 0) CommandSpeed = constrain(CommandSpeed +
            DeltaSpeed_100sec, CommandSpeed, 0);
875         }
876         break;
877       case 'B': //назад
878         CommandSpeed = constrain(CommandSpeed - DeltaSpeed_100sec, -ASPEED,
          CommandSpeed);
879         Turn = 0;
880         break;
881       case 'L':  //Влево на месте
882         Turn = -ASPEED / 2;
883         //Обнуляем скорость управляемого движения.
884         if (CommandSpeed > 0) CommandSpeed = constrain(CommandSpeed -
          DeltaSpeed_100sec, 0, CommandSpeed);
885         if (CommandSpeed < 0) CommandSpeed = constrain(CommandSpeed +
          DeltaSpeed_100sec, CommandSpeed, 0);
886 
887         break;
888       case 'R':  //Вправо на месте
889         Turn = ASPEED / 2;
890         //Обнуляем скорость управляемого движения.
891         if (CommandSpeed > 0) CommandSpeed = constrain(CommandSpeed -
          DeltaSpeed_100sec, 0, CommandSpeed);
892         if (CommandSpeed < 0) CommandSpeed = constrain(CommandSpeed +
          DeltaSpeed_100sec, CommandSpeed, 0);
893         break;
894       case 'G':  // Влево вперед
895         //CommandSpeed = ASPEED; //
896 
897         CommandSpeed = constrain(CommandSpeed + DeltaSpeed_100sec, CommandSpeed,
          ASPEED);
898         Turn = -ASPEED / 2;
899         break;
900       case 'I':  //Вправо вперед
901         //CommandSpeed = ASPEED; //
902         CommandSpeed = constrain(CommandSpeed + DeltaSpeed_100sec, CommandSpeed,
          ASPEED);
903         Turn = ASPEED / 2;
904         break;
905       case 'H':  //Вправо
906         CommandSpeed = constrain(CommandSpeed - DeltaSpeed_100sec, -ASPEED,
          CommandSpeed);
907         Turn = ASPEED / 2;
908         break;
909       case 'J':  //Влево
910         CommandSpeed = constrain(CommandSpeed - DeltaSpeed_100sec, -ASPEED,
          CommandSpeed);
911         Turn = -ASPEED / 2;
912         break;
913       case '0':
```

```
914         ASPEED = MOVSPEED0;
915         break;
916       // Скорость 10%
917       case '1':
918         ASPEED = MOVSPEED1;
919         break;
920       // Скорость 20%
921       case '2':
922         ASPEED = MOVSPEED2;
923         break;
924       // Скорость 30%
925       case '3':
926         ASPEED = MOVSPEED3;
927         break;
928       // Скорость 40%
929       case '4':
930         ASPEED = MOVSPEED4;
931         break;
932       // Скорость 50%
933       case '5':
934         ASPEED = MOVSPEED5;
935         break;
936       // Скорость 60%
937       case '6':
938         ASPEED = MOVSPEED6;
939         break;
940       // Скорость 70%
941       case '7':
942         ASPEED = MOVSPEED7;
943         break;
944       case '8':
945         // Скорость 80%
946         ASPEED = MOVSPEED8;
947         break;
948       // Скорость 90%
949       case '9':
950         ASPEED = MOVSPEED9;
951         break;
952       // Скорость 100%
953       case 'q':
954         ASPEED = MOVSPEED10;
955         break;
956       case 'V':
957         if (avtostart) {
958           handACTIV = true;
959           servo_hand_bottom(imu[1]);
960         }
961         break;
962       case 'v':
963         if (avtostart) {
964           handACTIV = true;
965           servo_hand_up();
966         }
967         break;
968       case 'X':
969         break;
970       case 'x':
971         break;
972       default:
973         break;
974     }
975     return true;
976   }
977
978   //=============================================
979   //=============================================
980   //=============================================
981   //*****************************************************
982   //****** storage.h   ***********************************
983   //*****************************************************
984   //=============================================
985   //=============================================
986   //=============================================
```

```
987
988   #include <Preferences.h>
989   Preferences preferences;
990   //============================================
991   //============================================
992   bool write_to_flash()
993   {
994     //Очищаем хранилище
995     preferences.begin("robo_steps", false);
996     preferences.clear();
997     preferences.putFloat("CompensatorZ", CompensatorZ);
998     preferences.putFloat("CompensatorX", CompensatorX);
999     preferences.putFloat("CompensatorY", CompensatorY);
1000    preferences.end();
1001    return true;
1002  }
1003  //============================================
1004  //============================================
1005  //============================================
1006  void Calc_CompensatorZ(float mill_sec)
1007  {
1008    uint32_t ms = mill_sec;
1009    double i = 0;
1010    CompensatorZ = 0;
1011    CompensatorX = 0;
1012    CompensatorY = 0;
1013    uint32_t  timer = millis();
1014    uint32_t endtime = millis() + ms;
1015    while (endtime > timer) {
1016      timer = millis();
1017      mpu.getEvent(&a, &g, &temper);
1018
1019      CompensatorZ += g.gyro.z;
1020      CompensatorX += g.gyro.x;
1021      CompensatorY += g.gyro.y;
1022      delay(2);
1023      i++;
1024    }
1025    CompensatorZ /= i;
1026    CompensatorX /= i;
1027    CompensatorY /= i;
1028  //  write_to_flash();
1029  }
1030  //============================================
1031  //============================================
1032  //============================================
1033  bool read_from_flash()
1034  {
1035    //Открываем хранилище на чтение
1036    preferences.begin("robo_steps", false);
1037    CompensatorZ= preferences.getDouble("CompensatorZ",0);
1038    CompensatorX= preferences.getDouble("CompensatorX",0);
1039    CompensatorY= preferences.getDouble("CompensatorY",0);
1040    preferences.end();
1041    return true;
1042  }
1043  //============================================
1044  //============================================
1045  //============================================
1046
```