Código fonte da Plataforma Voadora

```
#include <main_mpu6050.h>
#include <stdlib.h>
#include <math.h>
#pin_select IC1=PIN_B14
#pin_select IC3=PIN_B14

// Ganho do controlador de distancia
float KP_distancia_Z=3500;
float KD_distancia_Z=8000;
float KI_distancia_Z=2;

// Ganho do controlador de Pitch / Roll
float KP=72; //Porpocional
float KD=4050; //Derivativo
float KI=3.2;  //Integral
float KPB=00; //Porpocional
float KDB=00; //Derivativo
float KIB=0;  //Integral

//Valores de offset do giroscopio
float GYRO_XOUT_OFFSET = +109.38;
float GYRO_YOUT_OFFSET = -59.76;
float GYRO_ZOUT_OFFSET = -114.98;

//Valores de offset do acelerometro
int ACELL_XOUT_OFFSET=+139;
int ACELL_YOUT_OFFSET=-108;
int ACELL_ZOUT_OFFSET=0;

//Variaveis
int media_acelaracao=13550;// 13800 estabilizaçao
```

```
int ref_acelaracao=13550;
float d_Z_filter[20]={0.1,0.1,0.1,0.1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int distancia_Z_i=0;
int distancia_Z_anterior=0;
int count_distancia_Z_filter=0;
float CMD_distancia_Z=0;
float CMD_compass_angulo=0;
float erro_compass_angulo=0;
float media_distancia_Z=0;
float64 soma_acelaracao=0;

int count_media_aceleracao=0;
int aux_acelaracao=0;
float med_pressao=0;
float soma_pressao=0;
float CMD_pressao=0;
float pressao=0;
float erro_pressao=0;
float integral_pressao=0;
float derivativo_pressao=0;
float erro_anterior_pressao=0;
float KI_erro_pressao=5;
float KD_erro_pressao=4000;
float KP_erro_pressao=2000;
int in=0;

float distancia_z;
signed int ACCEL_XOUT = 0;
signed int ACCEL_YOUT = 0;
signed int ACCEL_ZOUT = 0;
float GYRO_XRATE = 0;
float GYRO_YRATE = 0;
```

```
float GYRO_ZRATE = 0;
int GYRO_XRATERAW = 0;
int GYRO_YRATERAW = 0;
int GYRO_ZRATERAW = 0;

unsigned int8 GYRO_XOUT_L ;
unsigned int8 GYRO_XOUT_H ;
unsigned int8 GYRO_YOUT_L ;
unsigned int8 GYRO_YOUT_H ;
unsigned int8 GYRO_ZOUT_L ;
unsigned int8 GYRO_ZOUT_H ;
float GYRO_XOUT = 0;
float GYRO_YOUT = 0;
float GYRO_ZOUT = 0;

unsigned int8  ACCEL_XOUT_L ;
unsigned int8  ACCEL_XOUT_H ;
unsigned int8  ACCEL_YOUT_L ;
unsigned int8  ACCEL_YOUT_H ;
unsigned int8  ACCEL_ZOUT_L ;
unsigned int8  ACCEL_ZOUT_H ;

signed long GYRO_XOUT_OFFSET_1000SUM = 0;
signed long GYRO_YOUT_OFFSET_1000SUM = 0;
signed long GYRO_ZOUT_OFFSET_1000SUM = 0;

float XANGLE = 0;
float YANGLE = 0;

float GYRO_XANGLE = 0;
float GYRO_YANGLE = 0;
float GYRO_ZANGLE = 0;
```

```
long GYRO_XANGLERAW = 0;
long GYRO_YANGLERAW = 0;
long GYRO_ZANGLERAW = 0;
float ACCEL_XANGLE = 0;
float ACCEL_YANGLE = 0;
float ACCEL_ZANGLE = 0;
int count = 0;


int8 xMSB;
int8 xLSB;
int8 yMSB;
int8 yLSB;
int8 zMSB;
int8 zLSB;
signed int16 rate_x;
signed int16 rate_y;
signed int16 rate_z;
signed int32 dc_offset_x=0;
signed int32 dc_offset_y=0;
signed int32 dc_offset_z=0;
signed int16 prev_rate_x=0;
signed int16 prev_rate_y=0;
signed int16 prev_rate_z=0;
signed int16 gyrogx=0;
signed int16 gyrogy=0;
signed int16 gyrogz=0;

float noise=0;
float xG=0;
float yG=0;
float zG=0;
```

```
float compass_angulo=0,compass_angulo_anterior=0;

//dados de voo
int comando_altitude=0;
int comando_angulo_x=0;
int comando_angulo_y=0;
int comando_angulo_r=0;
int v_motor_xf_1=0;
int v_motor_yt_2=0;
int v_motor_xf_3=0;
int v_motor_yf_4=0;
unsigned velocidade_M1;
unsigned velocidade_M2;
unsigned velocidade_M3;
unsigned velocidade_M4;
unsigned long IC1_valor1, IC1_valor2;
int count_IC1=0;

//float KP,KD,KI,KPB,KDB,KIB;
#define timeConstant .1
int8 i,accel_data[6],giro_data[6],j,k;
signed int16 x=0,y=0,z=0;
signed int16 x0=0,y0=0,z0=0;
signed int16 x1=0,y1=0,z1=0;
signed int16 x2=0,y2=0,z2=0;
signed int16 x3=0,y3=0,z3=0;
signed int16 x4=0,y4=0,z4=0;
signed int16 x5=0,y5=0,z5=0;
signed int16 x6=0,y6=0,z6=0;
signed int16 x7=0,y7=0,z7=0;
```

```
signed int16 x8=0,y8=0,z8=0;
signed int16 x9=0,y9=0,z9=0;
//unsigned= int off_x=6,off_y=9,off_z=232;
float roll=0;
float pitch=0;
float roll_radio=0;
float pitch_radio=0;
float acel_roll=0;
float acel_pitch=0;
int count_acel=0;

float anterior_roll=0;
float anterior_pitch=0;
int32
K_2_velocidade_M1=0,K_2_velocidade_M2=0,K_2_velocidade_M3=0,K_2_velocidade_
M4=0;
int
K_1_velocidade_M1=0,K_1_velocidade_M2=0,K_1_velocidade_M3=0,K_1_velocidade_
M4=0;
int erro_roll=0,K_1_erro_roll=0,erro_pitch=0,K_1_erro_pitch=0;
float integral_erro_roll=0,integral_erro_pitch=0,integral_erro_compass_angulo=0;
float
derivativo_erro_roll=0,derivativo_erro_pitch=0,derivativo_erro_compass_angulo=0;
float
KI_erro_pitch=0,KI_erro_roll=0,KP_erro_pitch=0,KP_erro_roll=0,KD_erro_pitch=0,KD_
erro_roll=0;
float erro_anterior_distancia_Z=0,erro_distancia_Z=0;
float integral_distancia_Z=0,derivativo_distancia_Z=0;
float KI_erro_distancia_Z=0,KP_erro_distancia_Z=0,KD_erro_distancia_Z=0;
unsigned int aceleracao=0;
int contg=0;
int count_rf=150,count_bosula=0;
```

```
//int count=0;

// MPU6050  atribuição nomes aos registos i2c
#define MPU6050_ADDRESS 0b11010000 // Address with end write bit
#define MPU6050_RA_XG_OFFS_TC 0x00 //[7] PWR_MODE, [6:1] XG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_YG_OFFS_TC 0x01 //[7] PWR_MODE, [6:1] YG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_ZG_OFFS_TC 0x02 //[7] PWR_MODE, [6:1] ZG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_X_FINE_GAIN 0x03 //[7:0] X_FINE_GAIN
#define MPU6050_RA_Y_FINE_GAIN 0x04 //[7:0] Y_FINE_GAIN
#define MPU6050_RA_Z_FINE_GAIN 0x05 //[7:0] Z_FINE_GAIN
#define MPU6050_RA_XA_OFFS_H 0x06 //[15:0] XA_OFFS
#define MPU6050_RA_XA_OFFS_L_TC 0x07
#define MPU6050_RA_YA_OFFS_H 0x08 //[15:0] YA_OFFS
#define MPU6050_RA_YA_OFFS_L_TC 0x09
#define MPU6050_RA_ZA_OFFS_H 0x0A //[15:0] ZA_OFFS
#define MPU6050_RA_ZA_OFFS_L_TC 0x0B
#define MPU6050_RA_XG_OFFS_USRH 0x13 //[15:0] XG_OFFS_USR
#define MPU6050_RA_XG_OFFS_USRL 0x14
#define MPU6050_RA_YG_OFFS_USRH 0x15 //[15:0] YG_OFFS_USR
#define MPU6050_RA_YG_OFFS_USRL 0x16
#define MPU6050_RA_ZG_OFFS_USRH 0x17 //[15:0] ZG_OFFS_USR
#define MPU6050_RA_ZG_OFFS_USRL 0x18
#define MPU6050_RA_SMPLRT_DIV 0x19
#define MPU6050_RA_CONFIG 0x1A
#define MPU6050_RA_GYRO_CONFIG 0x1B
#define MPU6050_RA_ACCEL_CONFIG 0x1C
#define MPU6050_RA_FF_THR 0x1D
#define MPU6050_RA_FF_DUR 0x1E
#define MPU6050_RA_MOT_THR 0x1F
```

```
#define MPU6050_RA_MOT_DUR 0x20
#define MPU6050_RA_ZRMOT_THR 0x21
#define MPU6050_RA_ZRMOT_DUR 0x22
#define MPU6050_RA_FIFO_EN 0x23
#define MPU6050_RA_I2C_MST_CTRL 0x24
#define MPU6050_RA_I2C_SLV0_ADDR 0x25
#define MPU6050_RA_I2C_SLV0_REG 0x26
#define MPU6050_RA_I2C_SLV0_CTRL 0x27
#define MPU6050_RA_I2C_SLV1_ADDR 0x28
#define MPU6050_RA_I2C_SLV1_REG 0x29
#define MPU6050_RA_I2C_SLV1_CTRL 0x2A
#define MPU6050_RA_I2C_SLV2_ADDR 0x2B
#define MPU6050_RA_I2C_SLV2_REG 0x2C
#define MPU6050_RA_I2C_SLV2_CTRL 0x2D
#define MPU6050_RA_I2C_SLV3_ADDR 0x2E
#define MPU6050_RA_I2C_SLV3_REG 0x2F
#define MPU6050_RA_I2C_SLV3_CTRL 0x30
#define MPU6050_RA_I2C_SLV4_ADDR 0x31
#define MPU6050_RA_I2C_SLV4_REG 0x32
#define MPU6050_RA_I2C_SLV4_DO 0x33
#define MPU6050_RA_I2C_SLV4_CTRL 0x34
#define MPU6050_RA_I2C_SLV4_DI 0x35
#define MPU6050_RA_I2C_MST_STATUS 0x36
#define MPU6050_RA_INT_PIN_CFG 0x37
#define MPU6050_RA_INT_ENABLE 0x38
#define MPU6050_RA_DMP_INT_STATUS 0x39
#define MPU6050_RA_INT_STATUS 0x3A
#define MPU6050_RA_ACCEL_XOUT_H 0x3B
#define MPU6050_RA_ACCEL_XOUT_L 0x3C
#define MPU6050_RA_ACCEL_YOUT_H 0x3D
#define MPU6050_RA_ACCEL_YOUT_L 0x3E
#define MPU6050_RA_ACCEL_ZOUT_H 0x3F
```

```
#define MPU6050_RA_ACCEL_ZOUT_L 0x40
#define MPU6050_RA_TEMP_OUT_H 0x41
#define MPU6050_RA_TEMP_OUT_L 0x42
#define MPU6050_RA_GYRO_XOUT_H 0x43
#define MPU6050_RA_GYRO_XOUT_L 0x44
#define MPU6050_RA_GYRO_YOUT_H 0x45
#define MPU6050_RA_GYRO_YOUT_L 0x46
#define MPU6050_RA_GYRO_ZOUT_H 0x47
#define MPU6050_RA_GYRO_ZOUT_L 0x48
#define MPU6050_RA_EXT_SENS_DATA_00 0x49
#define MPU6050_RA_EXT_SENS_DATA_01 0x4A
#define MPU6050_RA_EXT_SENS_DATA_02 0x4B
#define MPU6050_RA_EXT_SENS_DATA_03 0x4C
#define MPU6050_RA_EXT_SENS_DATA_04 0x4D
#define MPU6050_RA_EXT_SENS_DATA_05 0x4E
#define MPU6050_RA_EXT_SENS_DATA_06 0x4F
#define MPU6050_RA_EXT_SENS_DATA_07 0x50
#define MPU6050_RA_EXT_SENS_DATA_08 0x51
#define MPU6050_RA_EXT_SENS_DATA_09 0x52
#define MPU6050_RA_EXT_SENS_DATA_10 0x53
#define MPU6050_RA_EXT_SENS_DATA_11 0x54
#define MPU6050_RA_EXT_SENS_DATA_12 0x55
#define MPU6050_RA_EXT_SENS_DATA_13 0x56
#define MPU6050_RA_EXT_SENS_DATA_14 0x57
#define MPU6050_RA_EXT_SENS_DATA_15 0x58
#define MPU6050_RA_EXT_SENS_DATA_16 0x59
#define MPU6050_RA_EXT_SENS_DATA_17 0x5A
#define MPU6050_RA_EXT_SENS_DATA_18 0x5B
#define MPU6050_RA_EXT_SENS_DATA_19 0x5C
#define MPU6050_RA_EXT_SENS_DATA_20 0x5D
#define MPU6050_RA_EXT_SENS_DATA_21 0x5E
#define MPU6050_RA_EXT_SENS_DATA_22 0x5F
```

```
#define MPU6050_RA_EXT_SENS_DATA_23 0x60
#define MPU6050_RA_MOT_DETECT_STATUS 0x61
#define MPU6050_RA_I2C_SLV0_DO 0x63
#define MPU6050_RA_I2C_SLV1_DO 0x64
#define MPU6050_RA_I2C_SLV2_DO 0x65
#define MPU6050_RA_I2C_SLV3_DO 0x66
#define MPU6050_RA_I2C_MST_DELAY_CTRL 0x67
#define MPU6050_RA_SIGNAL_PATH_RESET 0x68
#define MPU6050_RA_MOT_DETECT_CTRL 0x69
#define MPU6050_RA_USER_CTRL 0x6A
#define MPU6050_RA_PWR_MGMT_1 0x6B
#define MPU6050_RA_PWR_MGMT_2 0x6C
#define MPU6050_RA_BANK_SEL 0x6D
#define MPU6050_RA_MEM_START_ADDR 0x6E
#define MPU6050_RA_MEM_R_W 0x6F
#define MPU6050_RA_DMP_CFG_1 0x70
#define MPU6050_RA_DMP_CFG_2 0x71
#define MPU6050_RA_FIFO_COUNTH 0x72
#define MPU6050_RA_FIFO_COUNTL 0x73
#define MPU6050_RA_FIFO_R_W 0x74
#define MPU6050_RA_WHO_AM_I 0x75
#define gyro_xsensitivity 131 //66.5 Dead on at last check
#define gyro_ysensitivity 131 //72.7 Dead on at last check
#define gyro_zsensitivity 131
#define a 0.01
#define dt 0.004


// SI4432 atribuição nomes aos registos i2c
#define SI4432_PWRSTATE_READY 01 // Si4432 ready mode define
#define SI4432_PWRSTATE_TX 0x09 // Si4432 Tx mode define
#define SI4432_PWRSTATE_RX 05 // Si4432 Rx mode define
```

```
#define SI4432_PACKET_SENT_INTERRUPT 04 // Si4432 packet sent interrupt define
#define SI4432_Rx_packet_received_interrupt 0x02 // Si4432 packet received interrupt define
#define nIRQ PIN_B6 // MCU input port
#define nSEL PIN_B7 // MCU output port
#define SDN PIN_B5// MCU output port
unsigned char rx_buf[15]= {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
unsigned char tx_test_data[10] = {0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x6d};
//FIM RF


void spi_rw(unsigned char addr, unsigned char data)
{
output_bit(nSEL,0);
delay_us(1);
spi_write2(addr);
spi_write2(data);
delay_us(1);
output_bit(nSEL,1);
}


void init_RF(void)
{
output_bit(SDN,1);
delay_ms(200); // RF module reset
output_bit(SDN,0);
delay_ms(300); // Delay 200 ms
```

```
spi_rw(0x03,0x00); // clear interrupt factor of Si4432
spi_rw(0x04,0x00);
spi_rw(0x06|0x80, 0x00); // disable the interrupt of Si4432
spi_rw(0x07|0x80, SI4432_PWRSTATE_READY); // to Ready mode
spi_rw(0x09|0x80, 0x7f); // load Capacitance = 12PF
spi_rw(0x0a|0x80, 0x05); // output clock
spi_rw(0x0b|0x80, 0xea); // GPIO 0 = digital output
spi_rw(0x0c|0x80, 0xea); //GPIO 1 = digital output
spi_rw(0x0d|0x80, 0xf4); // /GPIO 2 = Rx data
// The settings below are obtained from the Excel calculation table from silicon labs
spi_rw(0x70|0x80, 0x2c);
spi_rw(0x1d|0x80, 0x40); // enable AFC
// 1.2K bps setting
spi_rw(0x1c|0x06, 0x1B);
spi_rw(0x20|0x80, 0x3F);
spi_rw(0x21|0x80, 0x02); //
spi_rw(0x22|0x80, 0x0C);//
spi_rw(0x23|0x80, 0x4A); //
spi_rw(0x24|0x80, 0x04); //
spi_rw(0x25|0x80, 0x58); //
spi_rw(0x2a|0x80, 0x14);
spi_rw(0x6e|0x80, 0x10);
spi_rw(0x6f|0x80, 0x62);
//spi_rw(0x6e|0x80, 0x41);
//spi_rw(0x6f|0x80, 0x89);
spi_rw(0x70|0x80, 0x04);
//1.2K bps setting end
spi_rw(0x30|0x80, 0x8c); // PH + FiFo, MSB , CRC enabled
spi_rw(0x32|0x80, 0xff); // Header= Byte0, 1, 2, 3
spi_rw(0x33|0x80, 0x42); // Sync = byte 3,2
spi_rw(0x34|0x80, 16); // Tx Preamble = 16 nibble
spi_rw(0x35|0x80, 0x20); // Detected preamble = 4 nibble
```

```
spi_rw(0x36|0x80, 0x2d); // Sync word = 0x2dd4
spi_rw(0x37|0x80, 0xd4);
spi_rw(0x38|0x80, 0x00);
spi_rw(0x39|0x80, 0x00);
spi_rw(0x3a|0x80, 's'); // Tx header = swwx"
spi_rw(0x3b|0x80, 'w');
spi_rw(0x3c|0x80, 'w');
spi_rw(0x3d|0x80, 'x');
spi_rw(0x3e|0x80, 10); // payload = 10 byte
spi_rw(0x3f|0x80, 's'); // header checked = "swwx"
spi_rw(0x40|0x80, 'w');
spi_rw(0x41|0x80, 'w');
spi_rw(0x42|0x80, 'x');
spi_rw(0x43|0x80, 0xff); // all bit need be checked
spi_rw(0x44|0x80, 0xff); //
spi_rw(0x45|0x80, 0xff); //
spi_rw(0x46|0x80, 0xff); //
spi_rw(0x6d|0x80, 0x07); // max power output
spi_rw(0x79|0x80, 0x0); // no hopping
spi_rw(0x7a|0x80, 0x0); // no hopping
spi_rw(0x71|0x80, 0x2B); // RF mode = FSK , FiFo
spi_rw(0x72|0x80, 0x28); // Frequency deviation = 30KHz
spi_rw(0x73|0x80, 0x0); // No freq offset
spi_rw(0x74|0x80, 0x0); // No freq offset
spi_rw(0x75|0x80, 0x53); // freq = 433.5MHz
spi_rw(0x76|0x80, 0x57); //
spi_rw(0x77|0x80, 0x80);
}


void Modo_RX_RF(void)
{
```

```
spi_rw(0x07|0x80, SI4432_PWRSTATE_READY); // enter Ready mode
delay_ms(1); // stablize the OSC; not needed if OSC is on
spi_rw(0x0e|0x80, 0x02); // Antenna switch to Rx mode //TX0_RX1; // antenna switch = Rx mode
spi_rw(0x08|0x80, 0x03); //clear Tx/Rx fifo
spi_rw(0x08|0x80, 0x00); //clear Tx/Rx fifo
spi_rw(0x07|0x80,SI4432_PWRSTATE_RX ); // enter Rx mode
spi_rw(0x05|0x80, SI4432_Rx_packet_received_interrupt); // interrupt for packet received
spi_rw(0x03,0x00); //clear all interrupt factor
spi_rw(0x04,0x00); //clear all interrupt factor
}

void Ler_Dados_RF(void)
{
if(input(nirq)==1)
{
//printf("Sem dados\n\r");
return;
}
unsigned char i, j, chksum;
spi_rw(0x03,0x00); // clear interrupt factor
//read the Interrupt Status1 register
spi_rw(0x04,0x00); // clear interrupt factor

output_bit(nSEL,0);
delay_us(2);
spi_write2(0x7f); // read data from the Si4432 FiFo
for(i = 0;i<10;i++)
{
rx_buf[i] = spi_read2(0x7F);
}
```

```
output_bit(nSEL,1);
spi_rw(0x07|0x80, SI4432_PWRSTATE_READY);
// Exit Rx mode after all the data read from the FiFo
chksum = 0;
for(i=0;i<9;i++) // calculate the checksum for the received data
chksum += rx_buf[i];
if(( 0x41  == 0x41 ))
{
//printf("dados oK\n\r");
//printf("dados rx0=%x rx1=%x rx2=%x\n\r",rx_buf[0],rx_buf[1],rx_buf[2]);
}
else
{
//printf("dados Erro\n\r");
//printf("dados rx%c\n\r",rx_buf[0]);
}

modo_rx_rf();
}

void LDByteWriteI2C( int8 deviceAddress,int8 address,int8 val_pass)
{
  i2c_start();             // start transmission to device
  i2c_write(0b11010000);  // send device address
  i2c_write(address);       // send register address
  i2c_write(val_pass);           // send value to write
  i2c_stop();              // end transmission
  delay_ms(1);
}

int8 LDByteReadI2C(unsigned char deviceAddress, unsigned char address)
{
```

```
  unsigned int8 dados_read;
  i2c_start();              // start transmission to device
  i2c_write(0b11010000);    // This is where you have to _write_ the register number you want
  i2c_write(address);       // register to read
  i2c_start();              // restart - the bus is now set to _read_
  i2c_write(0b11010001); // now turn the bus round
  dados_read= i2c_read(0);
  i2c_stop();               // This will update x, y, and z with new values
  return dados_read;
}

// Inicialização do Acelerometro
void acel_init()
{
    restart_init:

     int8 Data = 0x00;
     unsigned char Failed = 0;
       LDByteWriteI2C(MPU6050_ADDRESS,  MPU6050_RA_PWR_MGMT_1, 0b10000000);
       delay_ms(100);
       LDByteWriteI2C(MPU6050_ADDRESS,  MPU6050_RA_PWR_MGMT_1, 0b00000000);

  //Sets sample rate to 1000/1+1 = 500Hz
  LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_SMPLRT_DIV, 0x01);
  //Disable FSync, 48Hz DLPF
  LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_CONFIG, 0x05);
  //Disable gyro self tests, scale of 500 degrees/s
  LDByteWriteI2C(MPU6050_ADDRESS,       MPU6050_RA_GYRO_CONFIG, 0b00001000);
```

```
        //Disable accel self tests, scale of +-4g, no DHPF
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_ACCEL_CONFIG,
0b00001000);
        //Freefall threshold of <|0mg|
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_FF_THR, 0x00);
        //Freefall duration limit of 0
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_FF_DUR, 0x00);
        //Motion threshold of >0mg
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_MOT_THR, 0x00);
        //Motion duration of >0s
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_MOT_DUR, 0x00);
        //Zero motion threshold
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_ZRMOT_THR, 0x00);
        //Zero motion duration threshold
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_ZRMOT_DUR, 0x00);
        //Disable sensor output to FIFO buffer
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_FIFO_EN, 0x00);

        //AUX I2C setup
        //Sets AUX I2C to single master control, plus other config
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_MST_CTRL,
0x00);
        //Setup AUX I2C slaves
         LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_ADDR,
0x00);
         LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_REG,
0x00);
         LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_CTRL,
0x00);
         LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_ADDR,
0x00);
```

```
        LDByteWriteI2C(MPU6050_ADDRESS,         MPU6050_RA_I2C_SLV1_REG,
0x00);
        LDByteWriteI2C(MPU6050_ADDRESS,       MPU6050_RA_I2C_SLV1_CTRL,
0x00);
        LDByteWriteI2C(MPU6050_ADDRESS,      MPU6050_RA_I2C_SLV2_ADDR,
0x00);
        LDByteWriteI2C(MPU6050_ADDRESS,         MPU6050_RA_I2C_SLV2_REG,
0x00);
        LDByteWriteI2C(MPU6050_ADDRESS,       MPU6050_RA_I2C_SLV2_CTRL,
0x00);
        LDByteWriteI2C(MPU6050_ADDRESS,      MPU6050_RA_I2C_SLV3_ADDR,
0x00);
        LDByteWriteI2C(MPU6050_ADDRESS,         MPU6050_RA_I2C_SLV3_REG,
0x00);
        LDByteWriteI2C(MPU6050_ADDRESS,       MPU6050_RA_I2C_SLV3_CTRL,
0x00);
        LDByteWriteI2C(MPU6050_ADDRESS,      MPU6050_RA_I2C_SLV4_ADDR,
0x00);
        LDByteWriteI2C(MPU6050_ADDRESS,         MPU6050_RA_I2C_SLV4_REG,
0x00);
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_DO, 0x00);
        LDByteWriteI2C(MPU6050_ADDRESS,       MPU6050_RA_I2C_SLV4_CTRL,
0x00);
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_DI, 0x00);

        //MPU6050_RA_I2C_MST_STATUS //Read-only
        //Setup INT pin and AUX I2C pass through
       LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_INT_PIN_CFG, 0x02);
       //Enable data ready interrupt
       LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_INT_ENABLE, 0x00);

       //Slave out, dont care
```

```
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_DO, 0x00);
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_DO, 0x00);
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_DO, 0x00);
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_DO, 0x00);
        //More slave config
        LDByteWriteI2C(MPU6050_ADDRESS,
MPU6050_RA_I2C_MST_DELAY_CTRL, 0x00);
        //Reset sensor signal paths
        LDByteWriteI2C(MPU6050_ADDRESS,
MPU6050_RA_SIGNAL_PATH_RESET, 0x00);
        //Motion detection control
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_MOT_DETECT_CTRL,
0x00);
        //Disables FIFO, AUX I2C, FIFO and I2C reset bits to 0
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_USER_CTRL, 0x00);
        //Sets clock source to gyro reference w/ PLL
        LDByteWriteI2C(MPU6050_ADDRESS,          MPU6050_RA_PWR_MGMT_1,
0b00000010);
        //Controls frequency of wakeups in accel low power mode plus the sensor standby
modes
        LDByteWriteI2C(MPU6050_ADDRESS,          MPU6050_RA_PWR_MGMT_2,
0x00);
         //MPU6050_RA_BANK_SEL        //Not in datasheet
         //MPU6050_RA_MEM_START_ADDR      //Not in datasheet
         //MPU6050_RA_MEM_R_W        //Not in datasheet
         //MPU6050_RA_DMP_CFG_1        //Not in datasheet
         //MPU6050_RA_DMP_CFG_2        //Not in datasheet
         //MPU6050_RA_FIFO_COUNTH      //Read-only
         //MPU6050_RA_FIFO_COUNTL      //Read-only
        //Data transfer to and from the FIFO buffer
        LDByteWriteI2C(MPU6050_ADDRESS, MPU6050_RA_FIFO_R_W, 0x00);
         //MPU6050_RA_WHO_AM_I         //Read-only, I2C address
```

```
printf("\nMPU6050 Setup Complete");
delay_ms(1000);

// Teste de leitura dos registos
Data=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_SMPLRT_DIV);
if(Data != 0x01) { printf("\nRegister check 1 failed, value should be 0x01, was
0x%x\n\r", Data); Failed = 1; }

if (Failed == 0)
{
printf("\nRegister value check passed\n\r");

}
else {
printf("Register value check failed\n\r");
i2c_start();
i2c_stop();
goto restart_init;
}
delay_ms(100);
test:

int x = 0;
GYRO_XOUT_OFFSET_1000SUM = 0;
GYRO_YOUT_OFFSET_1000SUM = 0;
GYRO_ZOUT_OFFSET_1000SUM = 0;
GYRO_XOUT= 0;
GYRO_YOUT = 0;
GYRO_ZOUT = 0;
goto pass_gyro_cal;
```

```
for(x = 1; x<=1000; ++x)
{
/*
    GYRO_XOUT_H=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_GYRO_XOUT_H);
    GYRO_XOUT_L=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_GYRO_XOUT_L);
   GYRO_YOUT_H=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_GYRO_YOUT_H);
    GYRO_YOUT_L=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_GYRO_YOUT_L);
    GYRO_ZOUT_H=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_GYRO_ZOUT_H);
   GYRO_ZOUT_L=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_GYRO_ZOUT_L);
 */
     GYRO_XOUT_H=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_XOUT_H);
    GYRO_XOUT_L=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_XOUT_L);
   GYRO_YOUT_H=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_YOUT_H);
   GYRO_YOUT_L=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_YOUT_L);
   GYRO_ZOUT_H=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_ZOUT_H);
  GYRO_ZOUT_L=LDByteReadI2C(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_ZOUT_L);

    GYRO_XOUT_OFFSET_1000SUM += (signed
int)make16(GYRO_XOUT_H,GYRO_XOUT_L);
```

```
                GYRO_YOUT_OFFSET_1000SUM += (signed int)make16(GYRO_YOUT_H,GYRO_YOUT_L);
                GYRO_ZOUT_OFFSET_1000SUM += (signed int)make16(GYRO_ZOUT_H,GYRO_ZOUT_L);

        //    printf("\nGyro X offset sum: %ld \n\r", GYRO_XOUT_OFFSET_1000SUM);
        //   printf("\nGyro Y offset sum: %Ld \n\r", GYRO_YOUT_OFFSET_1000SUM);
           // printf("\nGyro Z offset sum: %Ld \n\r", GYRO_ZOUT_OFFSET_1000SUM);
                delay_ms(5);
            }
            GYRO_XOUT_OFFSET = (float)GYRO_XOUT_OFFSET_1000SUM/1000;
            GYRO_YOUT_OFFSET =(float)GYRO_YOUT_OFFSET_1000SUM/1000;
            GYRO_ZOUT_OFFSET = (float)GYRO_ZOUT_OFFSET_1000SUM/1000;
        pass_gyro_cal:
            printf("\nGyro X offset sum: %ld Gyro X offset: %f\n\r", GYRO_XOUT_OFFSET_1000SUM, GYRO_XOUT_OFFSET);
            printf("\nGyro Y offset sum: %ld Gyro Y offset: %f\n\r", GYRO_YOUT_OFFSET_1000SUM, GYRO_YOUT_OFFSET);
            printf("\nGyro Z offset sum: %ld Gyro Z offset: %f\n\r", GYRO_ZOUT_OFFSET_1000SUM, GYRO_ZOUT_OFFSET);
         delay_ms(100);
        }

        //Gets raw accelerometer data, performs no processing
        void Accel_Angle()
        {
         //                ACCEL_XOUT_H=LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_ACCEL_XOUT_H);
        //                 ACCEL_XOUT_L=LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_ACCEL_XOUT_L);
        //                 ACCEL_YOUT_H=LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_ACCEL_YOUT_H);
```

```
//                    ACCEL_YOUT_L=  LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_ACCEL_YOUT_L);
//                    ACCEL_ZOUT_H=LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_ACCEL_ZOUT_H);
//                    ACCEL_ZOUT_H=LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_ACCEL_ZOUT_L);
//                     GYRO_XOUT_H=LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_GYRO_XOUT_H);
//                     GYRO_XOUT_L=LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_GYRO_XOUT_L);
//                    GYRO_YOUT_H=LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_GYRO_YOUT_H);
//                     GYRO_YOUT_L=LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_GYRO_YOUT_L);
//                     GYRO_ZOUT_H=LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_GYRO_ZOUT_H);
//                     GYRO_ZOUT_L=LDByteReadI2C(MPU6050_ADDRESS, MPU6050_RA_GYRO_ZOUT_L);

//goto as;
  i2c_start();              // start transmission to device
  i2c_write(0b11010000);    // This is where you have to _write_ the register number you want
  i2c_write(MPU6050_RA_ACCEL_XOUT_H);         // register to read
  i2c_start();              // restart - the bus is now set to _read_
  i2c_write(0b11010001); // now turn the bus round
  ACCEL_XOUT_H= i2c_read();
  ACCEL_XOUT_L= i2c_read();
  ACCEL_YOUT_H= i2c_read();
  ACCEL_YOUT_L= i2c_read();
  ACCEL_ZOUT_H= i2c_read();
  ACCEL_ZOUT_L= i2c_read();
```

```
        i2c_read(); //temperatura bit alto
        i2c_read(); //temperatura bit baixo
  GYRO_XOUT_H= i2c_read();
  GYRO_XOUT_L= i2c_read();
  GYRO_YOUT_H= i2c_read();
  GYRO_YOUT_L= i2c_read();
  GYRO_ZOUT_H= i2c_read();
  GYRO_ZOUT_L= i2c_read(0);

  i2c_stop();          // This will update x, y, and z with new values
  //return dados_read;
```

as:

```
    ACCEL_xOUT = (int)make16(ACCEL_XOUT_H,ACCEL_XOUT_L)-
ACELL_XOUT_OFFSET;
    ACCEL_yOUT = (int)make16(ACCEL_yOUT_H,ACCEL_yOUT_L)-
ACELL_YOUT_OFFSET;
    ACCEL_zOUT = (int)make16(ACCEL_zOUT_H,ACCEL_zOUT_L);

     ACCEL_XANGLE = 57.29578*atan2((float)ACCEL_YOUT,
(float)sqrt(pow(ACCEL_ZOUT,2)+pow(ACCEL_XOUT,2)));
     ACCEL_YANGLE = 57.29578*atan2((float)-
ACCEL_XOUT,(float)sqrt(pow(ACCEL_ZOUT,2)+pow(ACCEL_YOUT,2)));

     GYRO_XOUT = (int) make16(GYRO_XOUT_H,GYRO_XOUT_L)-
GYRO_XOUT_OFFSET;
     GYRO_YOUT = (int)make16(GYRO_YOUT_H,GYRO_YOUT_L) -
GYRO_YOUT_OFFSET;
     GYRO_ZOUT = (int)make16(GYRO_ZOUT_H,GYRO_ZOUT_L)-
GYRO_ZOUT_OFFSET;
```

```
      GYRO_XRATE = (float)GYRO_XOUT/gyro_xsensitivity;
      GYRO_YRATE = (float)GYRO_YOUT/gyro_ysensitivity;
      GYRO_ZRATE = (float)GYRO_ZOUT/gyro_zsensitivity;

      // Integracao da velocidade angular do giroscopio
      GYRO_XANGLE = GYRO_XRATE*dt+XANGLE;
      GYRO_YANGLE = GYRO_YRATE*dt+YANGLE;
      GYRO_ZANGLE = GYRO_ZANGLE-GYRO_ZRATE*dt;
      compass_angulo_anterior=compass_angulo;
      compass_angulo =(GYRO_ZANGLE);
//       printf("\GYRO_XANGLE: %f \n\r", GYRO_XANGLE);
//       printf("\GYRO_YANGLE: %f \n\r", GYRO_YANGLE);
//       printf("\XANGLE: %f \n\r", XANGLE);
//       printf("\YANGLE: %f \n\r", YANGLE);
// delay_ms(1000);
}


// Leitura do canal 1 Inpute Capture
#INT_IC1
//80Cm leitura de Inpute Capture 324560
//distância = 0.8*leituraIC/324560 =leituraIC/405700
void  ic1_isr(void)
{
disable_interrupts(INTR_GLOBAL);
IC1_valor1=get_capture32(1 );
enable_interrupts(INTR_GLOBAL);
}

// Leitura do canal 2 Inpute Capture
#INT_IC3
void  ic3_isr(void)
```

```
{
aux_acelaracao=1; // teste sem controlo de altura
if(aux_acelaracao==1)
{
goto semcontroloaltura;
}

disable_interrupts(INTR_GLOBAL);
distancia_Z_anterior=distancia_Z;
IC1_valor2=get_capture32(3);
erro_anterior_distancia_Z=erro_distancia_Z;
//printf("ultrasonico=%Lu\n\r",IC1_valor2-IC1_valor1);
if(IC1_valor1>IC1_valor2)
{
distancia_z=(((4294967295-IC1_valor1)+IC1_valor2)/(float)405700);
}
else
distancia_z=(((IC1_valor2-
IC1_valor1)/(float)405700));//*cos(YANGLE/(180/Pi))*cos(XANGLE/(180/Pi));

distancia_z=((distancia_z/2)*cos(YANGLE/(180/Pi)))+((distancia_z/2)*cos(XANG
LE/(180/Pi)));
//if((distancia_Z>(media_distancia_Z+.1))|(distancia_Z<(media_distancia_Z-.1)))
//d_Z_filter[count_distancia_Z_filter]=media_distancia_z;
//else
d_Z_filter[count_distancia_Z_filter]=distancia_z;
if((distancia_Z>(distancia_Z_anterior+0.3))|(distancia_Z<(distancia_Z_anterior-
0.3)))
{
distancia_Z_i=0;
distancia_z=((d_Z_filter[0]+d_Z_filter[1]+d_Z_filter[2]+d_Z_filter[3]+d_Z_filter[4]
)/5);
```

```
}
//printf("distancia_z=%4f\n\r",distancia_z);
//if((distancia_Z>(media_distancia_Z+.1))|(distancia_Z<(media_distancia_Z-.1)))
//{
//distancia_z=media_distancia_z;
//distancia_Z_i=0;
//}
//else
//distancia_Z_i=1;
count_distancia_Z_filter++;
if(count_distancia_Z_filter>=5)
count_distancia_Z_filter=0;
//printf("distancia_z=%4f\n\r",distancia_z);
erro_distancia_Z=CMD_distancia_Z-distancia_z;

if(erro_distancia_Z>0.3)
erro_distancia_Z=0.3;
if(erro_distancia_Z<-0.3)
erro_distancia_Z=-0.3;

integral_distancia_Z=(integral_distancia_Z+erro_distancia_Z);
derivativo_distancia_Z=(erro_distancia_Z-erro_anterior_distancia_Z);
//if(distancia_Z_i==0)
//derivativo_distancia_Z=0;
KI_erro_distancia_Z=(integral_distancia_Z)*KI_distancia_Z;
KD_erro_distancia_Z=(derivativo_distancia_Z)*KD_distancia_Z;
KP_erro_distancia_Z=(erro_distancia_Z)*KP_distancia_Z;
//goto xs;

if(KP_erro_distancia_Z>1500)
KP_erro_distancia_Z=1500;
if(KP_erro_distancia_Z<-1500)
```

```
KP_erro_distancia_Z=-1500;

if(KD_erro_distancia_Z>1500)
KD_erro_distancia_Z=1500;
if(KD_erro_distancia_Z<-1500)
KD_erro_distancia_Z=-1500;

if(KI_erro_distancia_Z>1500)
{
KI_erro_distancia_Z=1500;
integral_distancia_Z=1500/(KI_distancia_Z);
}
if(KI_erro_distancia_Z<-1500)
{
KI_erro_distancia_Z=-1500;
integral_distancia_Z=-1500/(KI_distancia_Z);
}
xs:
//printf("KI%f\n\r",KI_erro_distancia_Z);
//printf("KP%f\n\r",KP_erro_distancia_Z);
aceleracao=media_acelaracao+(int)(KP_erro_distancia_Z+KD_erro_distancia_Z+KI
_erro_distancia_Z);
if(aceleracao>ref_acelaracao+1500)
aceleracao=ref_acelaracao+1500;
if(aceleracao<ref_acelaracao-1500)
aceleracao=ref_acelaracao-1500;
if(bit_test (rx_buf[9],0)==0)
{
aux_acelaracao==0;
count_media_aceleracao=0;
media_acelaracao=ref_acelaracao;
goto semcontroloaltura;
```

```
}
if((count_media_aceleracao<202)&(distancia_z>0.1)&bit_test (rx_buf[9],0)==1)
{
soma_acelaracao=soma_acelaracao+aceleracao;
count_media_aceleracao++;
}

if((count_media_aceleracao>=202)&(aux_acelaracao==0)&bit_test
(rx_buf[9],0)==1)
{
media_acelaracao=(int)(soma_acelaracao/200);
integral_distancia_Z=0;
KI_erro_distancia_Z=0;

// desabilita média
aux_acelaracao=1;
aceleracao=media_acelaracao;
CMD_pressao=pressao;
}

semcontroloaltura:
enable_interrupts(INTR_GLOBAL);
}

//Interrupção a cada 4 segundos
#INT_TIMER4
void  timer4_isr(void)
{
{
output_bit(Pin_A1,0);
disable_interrupts(INT_TIMER4);
```

```
//Controlo de altura por BMP180
if((aux_acelaracao==1)&(bit_test (rx_buf[9],0)==1))
//aceleracao=media_acelaracao+(rx_buf[0]-128)*3;
//goto teste_ana;
aceleracao=media_acelaracao;
if(rx_buf[0]>168)
KP=KP+0.01;
if(rx_buf[0]<88)
KP=KP-0.01;

}
teste_ana:

if (count_rf++>200)
{
count_rf=200;
}

if (count_IC1++>20) // Controlo medidor de distancia
{
count_IC1=0;
output_bit( PIN_B15, 1);
delay_us(15);
output_bit( PIN_B15, 0);
//goto ws;

// Estabilização inicial em função do ultra-sónico
if(CMD_distancia_Z<2&aux_acelaracao==0)
{
if(rx_buf[0]>144&rx_buf[0]<160)
CMD_distancia_Z=CMD_distancia_Z+.001;
if(rx_buf[0]>160&rx_buf[0]<176)
```

```
CMD_distancia_Z=CMD_distancia_Z+.002;
if(rx_buf[0]>176&rx_buf[0]<192)
CMD_distancia_Z=CMD_distancia_Z+.003;
if(rx_buf[0]>192&rx_buf[0]<208)
CMD_distancia_Z=CMD_distancia_Z+.004;
if(rx_buf[0]>208)
CMD_distancia_Z=CMD_distancia_Z+.005;
}
if(CMD_distancia_Z>0.005&aux_acelaracao==0)
{
if(rx_buf[0]>96&rx_buf[0]<112)
CMD_distancia_Z=CMD_distancia_Z-.001;
if(rx_buf[0]>80&rx_buf[0]<96)
CMD_distancia_Z=CMD_distancia_Z-.002;
if(rx_buf[0]>64&rx_buf[0]<80)
CMD_distancia_Z=CMD_distancia_Z-.002;
if(rx_buf[0]>48&rx_buf[0]<64)
CMD_distancia_Z=CMD_distancia_Z-.004;
if(rx_buf[0]<48)
CMD_distancia_Z=CMD_distancia_Z-.005;
}

// Ângulo de rotação segundo o eixo do Z
if(rx_buf[1]>144&rx_buf[1]<160)
CMD_compass_angulo=CMD_compass_angulo+.02;
if(rx_buf[1]>160&rx_buf[1]<176)
CMD_compass_angulo=CMD_compass_angulo+.04;
if(rx_buf[1]>176&rx_buf[1]<192)
CMD_compass_angulo=CMD_compass_angulo+.06;
if(rx_buf[1]>192&rx_buf[1]<208)
CMD_compass_angulo=CMD_compass_angulo+.08;
if(rx_buf[1]>208)
```

```
CMD_compass_angulo=CMD_compass_angulo+.1;

if(rx_buf[1]>96&rx_buf[1]<112)
CMD_compass_angulo=CMD_compass_angulo-.02;
if(rx_buf[1]>80&rx_buf[1]<96)
CMD_compass_angulo=CMD_compass_angulo-.04;
if(rx_buf[1]>64&rx_buf[1]<80)
CMD_compass_angulo=CMD_compass_angulo-.06;
if(rx_buf[1]>48&rx_buf[1]<64)
CMD_compass_angulo=CMD_compass_angulo-.08;
if(rx_buf[1]<48)
CMD_compass_angulo=CMD_compass_angulo-.1;
}

salta_compass:

if(input(nirq)==0)
{
ler_dados_rf();
count_rf=0;
}
derivativo_erro_pitch=pitch;
derivativo_erro_roll=roll;

output_bit(Pin_A1,1);
Accel_Angle();
output_bit(Pin_A1,0);

// Filtro complementar, fusão dos ângulos acelerometro e gisroscopio
XANGLE=ACCEL_XANGLE*.001+GYRO_XANGLE*0.999;
YANGLE=ACCEL_YANGLE*.001+GYRO_YANGLE*0.999;
```

```
// Leitura dos ângulos de X e Y enviados pelo comando
pitch_radio=(float)((int)rx_buf[2]-130)/15+(float)((int)rx_buf[4]-130)/15;
roll_radio=-(float)((int)rx_buf[2]-130)/15+(float)((int)rx_buf[4]-130)/15;


roll=-XANGLE-roll_radio;
pitch=YANGLE-pitch_radio;

// Controlador PID para os angulos segundo X, Y, Z
integral_erro_roll=(integral_erro_roll+roll);
integral_erro_pitch=(integral_erro_pitch+pitch);
derivativo_erro_pitch=(pitch-derivativo_erro_pitch);
derivativo_erro_roll=(roll-derivativo_erro_roll);
KI_erro_pitch=integral_erro_pitch*KI*.1;
KI_erro_roll=integral_erro_roll*KI*.1;
KD_erro_pitch=derivativo_erro_pitch*KD;
KD_erro_roll=derivativo_erro_roll*KD;
KP_erro_pitch=pitch*KP;
KP_erro_roll=roll*KP;

erro_compass_angulo=compass_angulo-CMD_compass_angulo;
integral_erro_compass_angulo=integral_erro_compass_angulo+erro_compass_angul
o;

// Limitação do sinal de saida dos controladores PID
if(KD_erro_roll>2000)
KD_erro_roll=2000;
if(KD_erro_roll<-2000)
KD_erro_roll=-2000;

if(KD_erro_pitch>2000)
```

```
KD_erro_pitch=2000;
if(KD_erro_pitch<-2000)
KD_erro_pitch=-2000;

if(KP_erro_roll>2000)
KP_erro_roll=2000;
if(KP_erro_roll<-2000)
KP_erro_roll=-2000;

if(KP_erro_pitch>2000)
KP_erro_pitch=2000;
if(KP_erro_pitch<-2000)
KP_erro_pitch=-2000;

if(KI_erro_roll>1250)
{
KI_erro_roll=1250;
integral_erro_roll=1250*10/KI;
}
if(KI_erro_roll<-1250)
{
KI_erro_roll=-1250;
integral_erro_roll=-1250*10/KI;
}

if(KI_erro_pitch>1250)
{
integral_erro_pitch=1250*10/KI;
KI_erro_pitch=1250;
}
if(KI_erro_pitch<-1250)
{
```

```
KI_erro_pitch=-1250;
integral_erro_pitch=-1250*10/KI;
}

// Calculo do sinal de atuação do motor 1
if((int32)(aceleracao-
((KP_erro_pitch)+(KI_erro_pitch)+(KD_erro_pitch))+(KPB*erro_compass_angulo+KDB
*derivativo_erro_compass_angulo+KIB*integral_erro_compass_angulo))<16600)
velocidade_M1=(int)(aceleracao-
((KP_erro_pitch)+(KI_erro_pitch)+(KD_erro_pitch))+(KPB*erro_compass_angulo+KDB
*derivativo_erro_compass_angulo+KIB*integral_erro_compass_angulo));
else
velocidade_M1=16600;

// Calculo do sinal de atuação do motor 2
if((int32)(aceleracao+((KP_erro_roll)+(KI_erro_roll)+(KD_erro_roll))-
(KPB*erro_compass_angulo*derivativo_erro_compass_angulo+KIB*integral_erro_compa
ss_angulo))<16600)
velocidade_M2=(int)(aceleracao+((KP_erro_roll)+(KI_erro_roll)  +(KD_erro_roll))-
(erro_compass_angulo+KDB*derivativo_erro_compass_angulo+KIB*integral_erro_comp
ass_angulo));
else
velocidade_M2=16600;

// Calculo do sinal de atuação do motor 3
if((int32)(aceleracao+((KP_erro_pitch)+(KI_erro_pitch)
+(KD_erro_pitch))+(KPB*erro_compass_angulo*derivativo_erro_compass_angulo+KIB*
integral_erro_compass_angulo))<16600)
velocidade_M3=(int)(aceleracao+((KP_erro_pitch)+(KI_erro_pitch)
+(KD_erro_pitch))+(KPB*erro_compass_angulo+KDB*derivativo_erro_compass_angulo
+KIB*integral_erro_compass_angulo));
else
```

```
velocidade_M3=16600;

// Calculo do sinal de atuação do motor 4
if((int32)(aceleracao-((KP_erro_roll)+(KI_erro_roll)+(KD_erro_roll))-
(KPB*erro_compass_angulo+KDB*derivativo_erro_compass_angulo+KIB*integral_erro_
compass_angulo))<16600)
velocidade_M4=(int)(aceleracao-((KP_erro_roll)+(KI_erro_roll)+(KD_erro_roll))-
(KPB*erro_compass_angulo+KDB*derivativo_erro_compass_angulo+KIB*integral_erro_
compass_angulo));
else
velocidade_M4=16600;


// Envio de dados por RS232 para debug
if(count++>250)
{
//printf("Pressao=%f\n\r",pressao);
//printf("media_distancia_z=%f\n\r",media_distancia_z);
//printf("CMD_pressao=%f\n\r",erro_pressao);
//printf("rG=%f;pG=%f;\n\r",xG,yG);

//printf("M1=%Lu;M2=%Lu;M3=%Lu;M4=%Lu\n\r",velocidade_M1,velocidade_M
2,velocidade_M3,velocidade_M4);

printf("ACCEL_XANGLE=%f;YANGLE=%f\n\r",ACCEL_XANGLE,ACCEL_YA
NGLE);
printf("GYRO_XANGLE=%f;GYRO_YANGLE=%f\n\r",GYRO_XANGLE,GYRO
_YANGLE);
//printf("distancia_z=%4f\n\r",distancia_z);
//printf("XANGLE=%f;YANGLE=%f;compass_angulo=%f\n\r",XANGLE,YANGL
E,compass_angulo);
printf("media_acelaracao=%Ld",media_acelaracao);
```

```
        printf("KP=%f\n\r",KP);
        //printf("Pressao=%f\n\r",BMP085Pressure(1) );
        ///printf("KP_ROLL=%f;KD_ROLL=%f;KI_ROLL=%f\n\r",KP_erro_roll,KD_erro
_roll,KI_erro_roll);
        ///printf("KP_PITCH=%f;KD_PITCH=%f;KI_PITCH=%f\n\r",KP_erro_pitch,KD_e
rro_pitch,KI_erro_pitch);
        //printf("magx=%f;mag_y=%f;mag_z=%f\n\r",mag_x,mag_y,mag_z);
        //printf("pitch_radio=%f;roll_radio=%f\n\r",pitch_radio,roll_radio);
        //printf("RX0T=%d\n\r",((signed int)((acelz/500))));
        //printf("roll=%f;pitch=%f;bossula=%f\n\r",roll,pitch,compass_angulo);
        //printf("rx1=%u;=%u\n\r",rx_buf[1],rx_buf[0]);
        //printf("rx3=%u;=%u\n\r",rx_buf[3],rx_buf[2]);
        //printf("rx5=%u;=%u\n\r",rx_buf[5],rx_buf[4]);
        //printf("roll_radio=%f;pitch_radio=%f\n\r",pitch_radio,roll_radio);

        count=0;
        }



        //Desliga os motores caso se perca o sinal rádio
        if(count_rf<150)
        {
        set_pwm_duty(1,velocidade_M1); //motor (1) x frente
        set_pwm_duty(2,velocidade_M2) ; //motor (2) y tras
        set_pwm_duty(3,velocidade_M3) ; //motor (3) x tras
        set_pwm_duty(4,velocidade_M4) ; //motor (4) y frente
        }
        else
        {
          CMD_distancia_Z=0;
          integral_erro_compass_angulo=0;
```

```
  integral_distancia_Z=0;
  integral_erro_roll=0;
  integral_erro_pitch=0;
  CMD_compass_angulo=0;
  Ws:
  GYRO_ZANGLE=0;
  set_pwm_duty(1,8500) ; //motor (1) x frente
  set_pwm_duty(2,8500) ; //motor (2) y tras
  set_pwm_duty(3,8500) ; //motor (3) x tras
  set_pwm_duty(4,8500) ; //motor (4) y frente
}

//output_bit(Pin_g0,0);
enable_interrupts(INT_TIMER4);

}

void main()
{
//Configuração do Hardware
  setup_capture(1, CAPTURE_RE | CAPTURE_32_BIT| INTERRUPT_EVERY_CAPTURE | CAPTURE_SYSTEM_CLOCK | CAPTURE_SYNCHRONIZE | CAPTURE_TRIG_SYNC_NONE);
  setup_capture(3, CAPTURE_FE | CAPTURE_32_BIT| INTERRUPT_EVERY_CAPTURE | CAPTURE_SYSTEM_CLOCK | CAPTURE_SYNCHRONIZE | CAPTURE_TRIG_SYNC_NONE);
  //setup_timer1(TMR_INTERNAL | TMR_DIV_BY_8, 43749 );
  setup_timer1(TMR_INTERNAL | TMR_DIV_BY_8, 17499);//PWM 500Hz
  setup_timer4(TMR_INTERNAL | TMR_DIV_BY_8, 17499); // Interrupt 250 HZ

  enable_interrupts(INT_IC1);
  enable_interrupts(INT_IC3);
```

```
  setup_adc_ports(sAN0, VSS_VDD);
  setup_adc(ADC_OFF | ADC_TAD_MUL_0);
ad:output_bit( PIN_B2, 1);
  delay_ms(100);
  output_bit( PIN_B2, 0);
  delay_ms(100);

//Configuração canais Inpucapture
  setup_compare(1, COMPARE_PWM_EDGE | COMPARE_TIMER1 | COMPARE_SYNCHRONIZE | COMPARE_TRIG_SYNC_TIMER1);
  set_pwm_duty(1, 8000);
  setup_compare(2, COMPARE_PWM_EDGE | COMPARE_TIMER1 | COMPARE_SYNCHRONIZE | COMPARE_TRIG_SYNC_TIMER1);
  set_pwm_duty(2, 8000);
  setup_compare(3, COMPARE_PWM_EDGE | COMPARE_TIMER1 | COMPARE_SYNCHRONIZE | COMPARE_TRIG_SYNC_TIMER1);
  set_pwm_duty(3, 8000);
  setup_compare(4, COMPARE_PWM_EDGE | COMPARE_TIMER1 | COMPARE_SYNCHRONIZE | COMPARE_TRIG_SYNC_TIMER1);
  set_pwm_duty(4, 8000);

init_rf();
//spi_write2(00);
modo_rx_rf();

  set_adc_channel(0);
s:
  i2c_stop();
  i2c_start();
  i2c_write(0b11010000);
  i2c_write(MPU6050_RA_WHO_AM_I);
  i2c_start();
```

```
   i2c_write(0b11010001);
   x1=i2c_read(0);
   i2c_stop();
   Printf( "test=%x\n\r", x1);
   delay_ms(100);

  delay_ms(2000);

  Acel_Init();
  delay_ms(1000);

// Abilitação de interrupção a cada 4ms
  enable_interrupts(INT_TIMER4);
  enable_interrupts(INTR_GLOBAL);
   test:
   while(TRUE)

   output_bit( PIN_B0, 0);
   output_bit( PIN_B0, 1);

}
}
```

Código fonte do comando rádio

```
#include <16F876A.h>
#DEVICE ADC=10
#use delay(crystal=8000000)
#use
rs232(baud=19200,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=PORT1)
#USE SPI (MASTER, SPI1, BAUD=100000, MODE=0, BITS=8, STREAM=SPI_1)

//RF
#define SI4432_PWRSTATE_READY 01 // Si4432 ready mode define
#define SI4432_PWRSTATE_TX 0x09 // Si4432 Tx mode define
#define SI4432_PWRSTATE_RX 05 // Si4432 Rx mode define
#define SI4432_PACKET_SENT_INTERRUPT 04 // Si4432 packet sent interrupt
define
#define SI4432_Rx_packet_received_interrupt 0x02 // Si4432 packet received
interrupt define
//#define TX1_RX0 spi(0x0e|0x80, 0x01) // Antenna switch to tx mode
//#define TX0_RX1 spi_wr(0x0e|0x80, 0x02) // Antenna switch to Rx mode
//#define TX0_RX0 spi_wr(0x0e|0x80, 0x00) // Antenna is not in Tx/Rx mode
#define nIRQ PIN_C1 // MCU input port
#define nSEL PIN_C2 // MCU output port
#define SDN PIN_C0 // MCU output port
inqt itstatus1;
int itstatus2;
unsigned char rx_buf[15];
unsigned char tx_test_data[10] =
{0x10,0x20,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x6d};
//FIM RF

void spi_rw(unsigned char addr, unsigned char data)
{
output_bit(nSEL,0);
```

```
delay_us(10);
spi_write(addr);
spi_write(data);
delay_us(10);
output_bit(nSEL,1);
}

void init_RF(void)
{
output_bit(SDN,1);
delay_ms(20); // RF module reset
output_bit(SDN,0);
delay_ms(300); // Delay 200 ms
spi_rw(0x03,0x00); // clear interrupt factor of Si4432
spi_rw(0x04,0x00);
spi_rw(0x06|0x80, 0x00); // disable the interrupt of Si4432
spi_rw(0x07|0x80, SI4432_PWRSTATE_READY); // to Ready mode
spi_rw(0x09|0x80, 0x7f); // load Capacitance = 12PF
spi_rw(0x0a|0x80, 0x05); // output clock
spi_rw(0x0b|0x80, 0xea); // GPIO 0 = digital output
spi_rw(0x0c|0x80, 0xea); //GPIO 1 = digital output
spi_rw(0x0d|0x80, 0xf4); // /GPIO 2 = Rx data
// The settings below are obtained from the Excel calculation table from silicon labs
spi_rw(0x70|0x80, 0x2c);
spi_rw(0x1d|0x80, 0x40); // enable AFC
// 1.2K bps setting
spi_rw(0x1c|0x06, 0x1B);
spi_rw(0x20|0x80, 0x3F);
spi_rw(0x21|0x80, 0x02); //
spi_rw(0x22|0x80, 0x0C);//
spi_rw(0x23|0x80, 0x4A); //
spi_rw(0x24|0x80, 0x04); //
```

```
spi_rw(0x25|0x80, 0x58); //
spi_rw(0x2a|0x80, 0x14);
spi_rw(0x6e|0x80, 0x10);
spi_rw(0x6f|0x80, 0x62);
//spi_rw(0x6e|0x80, 0x41);
//spi_rw(0x6f|0x80, 0x89);
spi_rw(0x70|0x80, 0x04);
//1.2K bps setting end
spi_rw(0x30|0x80, 0x8c); // PH + FiFo, MSB , CRC enabled
spi_rw(0x32|0x80, 0xff); // Header= Byte0, 1, 2, 3
spi_rw(0x33|0x80, 0x42); // Sync = byte 3,2
spi_rw(0x34|0x80, 16); // Tx Preamble = 16 nibble
spi_rw(0x35|0x80, 0x20); // Detected preamble = 4 nibble
spi_rw(0x36|0x80, 0x2d); // Sync word = 0x2dd4
spi_rw(0x37|0x80, 0xd4);
spi_rw(0x38|0x80, 0x00);
spi_rw(0x39|0x80, 0x00);
spi_rw(0x3a|0x80, 's'); // Tx header = swwx"
spi_rw(0x3b|0x80, 'w');
spi_rw(0x3c|0x80, 'w');
spi_rw(0x3d|0x80, 'x');
spi_rw(0x3e|0x80, 10); // payload = 10 byte
spi_rw(0x3f|0x80, 's'); // header checked = ”swwx"
spi_rw(0x40|0x80, 'w');
spi_rw(0x41|0x80, 'w');
spi_rw(0x42|0x80, 'x');
spi_rw(0x43|0x80, 0xff); // all bit need be checked
spi_rw(0x44|0x80, 0xff); //
spi_rw(0x45|0x80, 0xff); //
spi_rw(0x46|0x80, 0xff); //
spi_rw(0x6d|0x80, 0x07); // max power output
spi_rw(0x79|0x80, 0x0); // no hopping
```

```
spi_rw(0x7a|0x80, 0x0); // no hopping
spi_rw(0x71|0x80, 0x2B); // RF mode = FSK , FiFo
spi_rw(0x72|0x80, 0x28); // Frequency deviation = 30KHz
spi_rw(0x73|0x80, 0x0); // No freq offset
spi_rw(0x74|0x80, 0x0); // No freq offset
spi_rw(0x75|0x80, 0x53); // freq = 433.5MHz
spi_rw(0x76|0x80, 0x57); //
spi_rw(0x77|0x80, 0x80);
}

void Modo_RX_RF(void)
{
unsigned char i, chksum;
spi_rw(0x07|0x80, SI4432_PWRSTATE_READY); // enter Ready mode
delay_ms(5); // stablize the OSC; not needed if OSC is on
spi_rw(0x0e|0x80, 0x02); // Antenna switch to Rx mode //TX0_RX1; // antenna switch = Rx mode
spi_rw(0x08|0x80, 0x03); //clear Tx/Rx fifo
spi_rw(0x08|0x80, 0x00); //clear Tx/Rx fifo
spi_rw(0x07|0x80,SI4432_PWRSTATE_RX ); // enter Rx mode
spi_rw(0x05|0x80, SI4432_Rx_packet_received_interrupt); // interrupt for packet received
spi_rw(0x03,0x00); //clear all interrupt factor
spi_rw(0x04,0x00); //clear all interrupt factor
}

void Ler_Dados_RF(void)
{
unsigned char i, j, chksum;
spi_rw(0x03,0x00); // clear interrupt factor
//read the Interrupt Status1 register
spi_rw(0x04,0x00); // clear interrupt factor
```

```
output_bit(nSEL,0);
spi_write(0x7f); // read data from the Si4432 FiFo
for(i = 0;i<10;i++)
{
rx_buf[i] = spi_read(0x00);
}
output_bit(nSEL,1);
spi_rw(0x07|0x80, SI4432_PWRSTATE_READY);
// Exit Rx mode after all the data read from the FiFo
chksum = 0;
for(i=0;i<9;i++) // calculate the checksum for the received data
chksum += rx_buf[i];
if(( chksum == rx_buf[9] )&&( rx_buf[0] == 0x41 ))
{
; // data verified OK
}
else
{
; // Data Verified error, then restart to Rx
}

}

void Escrever_Dados_RF(void)
{
unsigned char i;

spi_rw(0x07|0x80, SI4432_PWRSTATE_READY); // enter Ready mode
spi_rw(0x0e|0x80, 0x01);   //TX1_RX0; //Antenna switch = Tx mode
delay_ms(5); // stablize the OSC; not needed if OSC is on
spi_rw(0x08|0x80, 0x03); // clear Tx/Rx fifo
spi_rw(0x08|0x80, 0x00); // clear Tx/Rx fifo
```

```
spi_rw(0x34|0x80, 16); // preamble tx = 16 nibble
spi_rw(0x3e|0x80, 10); // payload = 10 bytes
for (i = 0; i<10; i++)
{
spi_rw(0x7f|0x80, tx_test_data[i]); //load payload into the fifo
}
spi_rw(0x05|0x80, SI4432_PACKET_SENT_INTERRUPT); // interrupt after packet is sent
spi_rw(0x03,0x00); // clear interrupt factor
spi_rw(0x04,0x00);
spi_rw(0x07|0x80, SI4432_PWRSTATE_TX); // enter Tx mode
}

void main()
{
as:
printf("NUNO");
output_bit(pin_b0,0);
output_bit(pin_b1,0);
delay_ms(50);
output_bit(pin_b0,1);
output_bit(pin_b1,1);
delay_ms(50);
//goto as;

aw:
//output_bit(pin_b5,1);
delay_ms(100);
output_bit(pin_b5,0);
delay_ms(100);
//goto aw;
//trisb.b5=1;
```

```
int16 leituta_adc;
setup_adc_ports(ALL_ANALOG);
setup_adc(ADC_CLOCK_INTERNAL);

test:
   delay_ms(300);
   init_RF();
   delay_ms(200);
   //escrever_dados_rf();
   while(TRUE)
   {
  //  set_adc_channel(3); alterar para este
  // Controlo do angulo de rotação~
    set_adc_channel(0);
  delay_ms(7);
  leituta_adc=read_adc();
 // printf("ADC=%Lu\n\r",leituta_adc);
 if(leituta_adc>181)
 {
output_bit(pin_b0,0);
 }
 else
 {
output_bit(pin_b0,1);
 }

  set_adc_channel(4);
  delay_ms(7);
  leituta_adc=read_adc()>>2;
  tx_test_data[1]=  make8(leituta_adc,0);
  set_adc_channel(3);
  delay_ms(7);
```

```
   // Controlo da acelaração
 leituta_adc=read_adc()>>2;
 tx_test_data[0]=  make8(leituta_adc,0);
 set_adc_channel(1);
 delay_ms(7);
 // Controlo da roll
 leituta_adc=read_adc()>>2;
 tx_test_data[3]=  make8(leituta_adc,1);
 tx_test_data[2]=  make8(leituta_adc,0);
 set_adc_channel(2);
 delay_ms(7);
 // Controlo da pitch
 leituta_adc=read_adc()>>2;
 tx_test_data[5]=  make8(leituta_adc,1);
 tx_test_data[4]=  make8(leituta_adc,0);
//tx_test_data[1]=10;
//tx_test_data[0]=10;
if(input(PIN_B2)==0)
{
bit_set(tx_test_data[9],0);
}
else
{
bit_clear(tx_test_data[9],0);
}

if(input(PIN_B3)==0)
{
bit_set(tx_test_data[9],1);
}
else
{
```

```
bit_clear(tx_test_data[9],1);
}

if(input(PIN_B4)==0)
{
bit_set(tx_test_data[9],2);
}
else
{
bit_clear(tx_test_data[9],2);
}

Escrever_dados_RF();
a1:if(input(nIRQ)==1)
{
delay_ms(10);
goto a1;
}
}
}
```