**drone2.ino**

#include <Servo.h>

//+++++++++++++++++++++++++++++++++++++++++++++

//variable de recepcion de datos y cariables de control de elevacion

//+++++++++++++++++++++++++++++++++++++++++++++

int option;

int led = 13;

int contador=0;

int contador2=85;

int contador3=90;

int inicio=0;

int centinela=0;

Servo miServo1;

Servo miServo2;

Servo miServo3;

Servo miServo4;

//+++++++++++++++++++++++++++++++++++++++++++++

void inicio(contador2)

{

miServo2.write(contador2);

delay(500);

contador2=contador2-20;

delay(500);

miServo2.write(contador2);

Serial.println("izq");

contador2=contador2-15;

miServo2.write(contador2);

delay(500);

contador2=contador2+15;

miServo2.write(contador2);

Serial.println("der");

}

void setup()

{

inicio();

//asignacion de comunicacion de datos entre arduino y antena bt

Serial.begin(9600);

//Asignacion de pines del servomotor

miServo1.attach(3);

miServo2.attach(5);

miServo3.attach(2);

miServo4.attach(9);

//Posicionamiento inicial

miServo1.write(contador); //Altura

miServo2.write(contador2); //giro en el eje

miServo3.write(contador3); //giro en el eje

contador2=contador2+20;

}

//++++++++++++++++++++++++++++++++++++++++++++

void desbloquear(angulo)

{

if (centinela==0)

{

miServo1.write(angulo);

delay(3000);

miServo1.write(0);

delay(2000);

centinela=1;

}

}

//+++++++++++++++++++++++++++++++++++++++++++++

void estacionar(angulo2)

{

miServo1.write(angulo2);

contador=0;

}

//+++++++++++++++++++++++++++++++++++++++++++++

void elevar(contador)

{

contador=contador+30;

if (contador>90)//limitantre de elevacion

{

contador=90;

}

miServo1.write(contador);

Serial.println(contador);

}

//+++++++++++++++++++++++++++++++++++++++++++++

void descender(contador)

{

contador=contador-10;

if (contador<0)

{

contador=0;

}

miServo1.write(contador);

Serial.println(contador);

}

//+++++++++++++++++++++++++++++++++++++++++++++

void giroizquierda()

{

contador3=contador3+70;

miServo3.write(contador3);

delay(500);

contador3=contador3-70;

miServo3.write(contador3);

Serial.println("adelante");

}

//+++++++++++++++++++++++++++++++++++++++++++++

void giroderecha(contador3)

{

contador3=contador3-70;

miServo3.write(contador3);

delay(500);

contador3=contador3+70;

miServo3.write(contador3);

Serial.println("atras");

}

//+++++++++++++++++++++++++++++++++++++++++++++

void seteoinicializquierda()

{

contador2=contador2+20;

miServo2.write(contador2);

delay(300);

contador2=contador2-20;

miServo2.write(contador2);

Serial.println("izq");

}

//+++++++++++++++++++++++++++++++++++++++++++++

void seteoinicialderecha()

{

contador2=contador2-15;

miServo2.write(contador2);

delay(200);

contador2=contador2+15;

miServo2.write(contador2);

Serial.println("der");

}

//+++++++++++++++++++++++++++++++++++++++++++++

void loop()

{

//si existe datos disponibles los leemos

if (Serial.available()>0)

{

//leemos la opcion enviada

option=Serial.read();

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

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

if(option=='5')

{

desbloquear(180);

}

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

if(option=='6')

{

estacionar(0);

}

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

if(option=='1')

{

elevar();

}

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

if(option=='2')

{

descender();

}

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

if(option=='X')

{

seteoinicializquierda();

}

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

if(option=='X')

{

seteoinicialderecha();

}

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

if(option=='3')

{

giroizquierda();

}

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

if(option=='4')

{

giroderecha();

}

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

}

}

//+++++++++++++FIN+++++++++++++++++++++++++++++

**servo.h**

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// Architecture specific include

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#if defined(ARDUINO\_ARCH\_AVR)

#include "avr/ServoTimers.h"

#elif defined(ARDUINO\_ARCH\_SAM)

#include "sam/ServoTimers.h"

#elif defined(ARDUINO\_ARCH\_SAMD)

#include "samd/ServoTimers.h"

#else

#error "This library only supports boards with an AVR, SAM or SAMD processor."

#endif

#define Servo\_VERSION 2 // software version of this library

#define MIN\_PULSE\_WIDTH 544 // the shortest pulse sent to a servo

#define MAX\_PULSE\_WIDTH 2400 // the longest pulse sent to a servo

#define DEFAULT\_PULSE\_WIDTH 1500 // default pulse width when servo is attached

#define REFRESH\_INTERVAL 20000 // minumim time to refresh servos in microseconds

#define SERVOS\_PER\_TIMER 12 // the maximum number of servos controlled by one timer

#define MAX\_SERVOS (\_Nbr\_16timers \* SERVOS\_PER\_TIMER)

#define INVALID\_SERVO 255 // flag indicating an invalid servo index

typedef struct {

uint8\_t nbr :6 ; // a pin number from 0 to 63

uint8\_t isActive :1 ; // true if this channel is enabled, pin not pulsed if false

} ServoPin\_t ;

typedef struct {

ServoPin\_t Pin;

volatile unsigned int ticks;

} servo\_t;

class Servo

{

public:

Servo();

uint8\_t attach(int pin); // attach the given pin to the next free channel, sets pinMode, returns channel number or 0 if failure

uint8\_t attach(int pin, int min, int max); // as above but also sets min and max values for writes.

void detach();

void write(int value); // if value is < 200 its treated as an angle, otherwise as pulse width in microseconds

void writeMicroseconds(int value); // Write pulse width in microseconds

int read(); // returns current pulse width as an angle between 0 and 180 degrees

int readMicroseconds(); // returns current pulse width in microseconds for this servo (was read\_us() in first release)

bool attached(); // return true if this servo is attached, otherwise false

private:

uint8\_t servoIndex; // index into the channel data for this servo

int8\_t min; // minimum is this value times 4 added to MIN\_PULSE\_WIDTH

int8\_t max; // maximum is this value times 4 added to MAX\_PULSE\_WIDTH

};

#endif

**Timer.c**

// Say which 16 bit timers can be used and in what order

#if defined(\_\_AVR\_ATmega1280\_\_) || defined(\_\_AVR\_ATmega2560\_\_)

#define \_useTimer5

#define \_useTimer1

#define \_useTimer3

#define \_useTimer4

typedef enum { \_timer5, \_timer1, \_timer3, \_timer4, \_Nbr\_16timers } timer16\_Sequence\_t;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#elif defined(\_\_AVR\_ATmega32U4\_\_)

#define \_useTimer1

typedef enum { \_timer1, \_Nbr\_16timers } timer16\_Sequence\_t;

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#elif defined(\_\_AVR\_AT90USB646\_\_) || defined(\_\_AVR\_AT90USB1286\_\_)

#define \_useTimer3

#define \_useTimer1

typedef enum { \_timer3, \_timer1, \_Nbr\_16timers } timer16\_Sequence\_t;

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#elif defined(\_\_AVR\_ATmega128\_\_) ||defined(\_\_AVR\_ATmega1281\_\_)||defined(\_\_AVR\_ATmega2561\_\_)

#define \_useTimer3

#define \_useTimer1

typedef enum { \_timer3, \_timer1, \_Nbr\_16timers } timer16\_Sequence\_t;

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#else // everything else

#define \_useTimer1

typedef enum { \_timer1, \_Nbr\_16timers } timer16\_Sequence\_t;

#endif

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++