Rotor AZ-EL con motores de etapas, basado en Arduino Uno y escudo CNC

Por: Miguel Dario XE2UD tomando varios proyectos similares como base para mi proyecto personal

*Controlando un rotor de dos ejes basado originalmente en el proyecto SatNOGS (y otros más...) como un sistema **independiente** usando GPredict como "tracker automático" y Arduino UNO con motores NEMA*

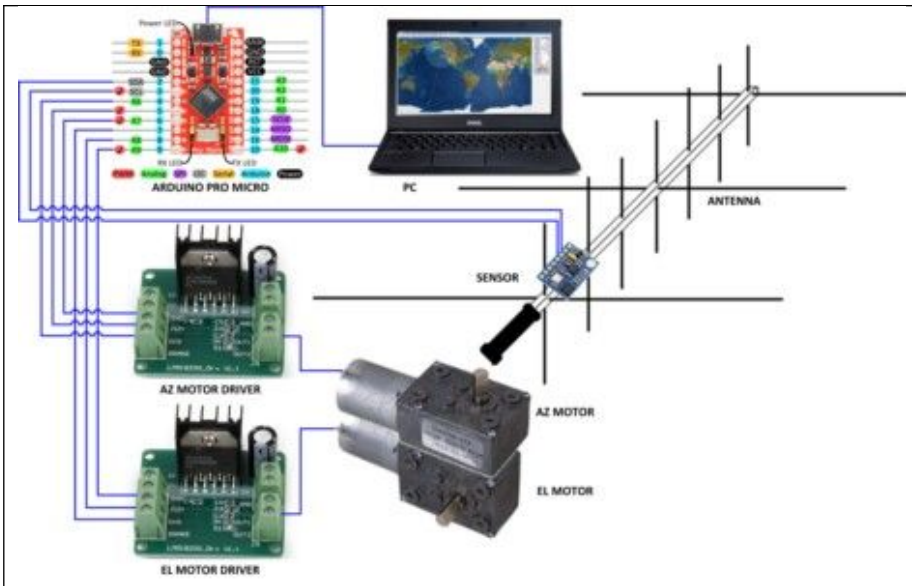
El rotor del proyecto **SatNOGS** está diseñado primordialmente como un **componente** de una estación terrestre conectada a su red de monitoreo satelital global.

Sin embargo en muchas circunstancias es preferible tener una estación independiente de la red **SatNOGS**. Y esto puede lograrse usando diversos equipos comerciales


O también usando un sistema hecho en casa que (re)utilice componentes disponibles en su localidad o reciclados de otros proyectos, destacando entre ellos el proyecto escolar **SARCNET**. Que tiene como objetivo dotar de estaciones satelitales de radioaficionados a las escuelas de Australia (*financiado con proyectos de aficionados*).

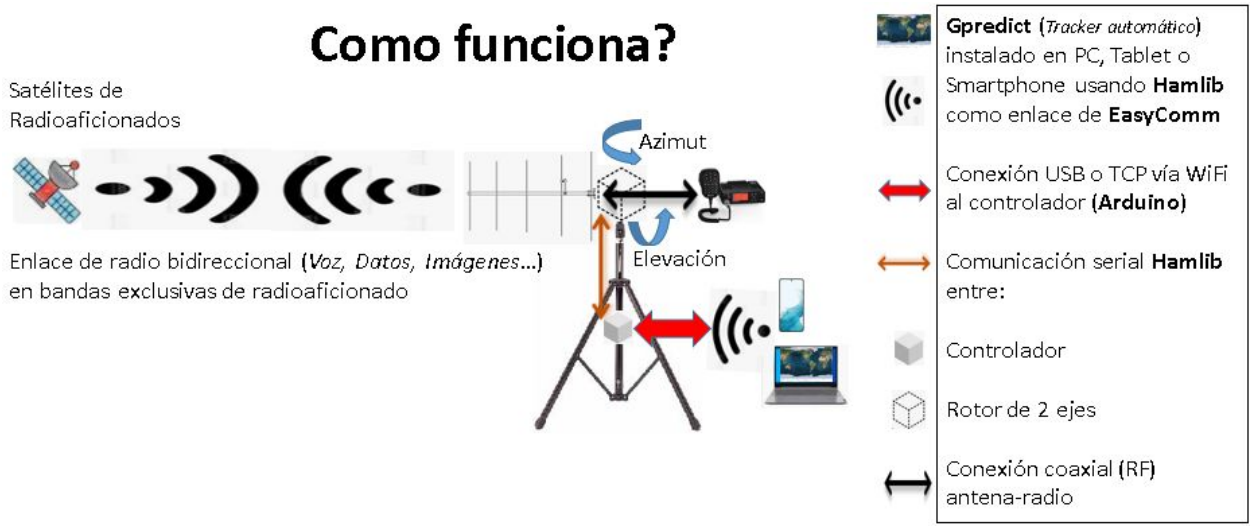
El concepto básico de este proyecto es muy simple: dotar de un rotor AZ-EL completamente automático para antenas direccionales y establecer contactos con los satélites de radioaficionados e inclusive con la Estación Espacial Internacional (ISS) permitiendo dedicar toda nuestra atención exclusivamente al uso del radio.

La siguiente información es para conectar tu **controlador de rotor** DIRECTAMENTE a una PC, pero también puede usarse con otros dispositivos como Raspberry Pi's, tabletas Android e incluso smartphones como "trackers" de satélites, y en cuyo caso deberán conservar el protocolo RS485 de comunicaciones como parte del sistema.



Esquema simplificado de conexiones entre el equipo "Tracker" y los motores del Rotor AZ-EL

En el mismo equipo de tracking se encuentran tambien alojadas las librerias de **Hamlib** que conectan el CPU con **Gpredict** (internamente) y el **microprocesador de control** *Arduino en este caso* por medio de TCP/IP y **Arduino** controla los motores usando **AMSAT EasyComm**) a traves de comunicación serial



Investigando sobre el tema de control de otros proyectos, y debido a que los componentes son muy específicos, partes difíciles de conseguir, o porque los ejemplos de software son muy vagos sin detalles o incompletos, decidí seguir **en lo general** el diseño del proyecto **SatNOGS** pero con algunos cambios importantes…

1. No utilizo el ***controlador*** de rotor original, sino que decidí fabricar mi propio controlador de rotor AZ-EL y antena con los mismos componentes básicos de electrónica y mecánicos que en el proyecto **SarcNet** por su sencillez y bajo costo, pero reutilizando componentes que tenía guardados (todos tenemos)
2. Al no formar parte de la Red de SATNOGS y mantener mi equipo completamente **independiente**, modifique el firmware para evitar las conexiones remotas y tener la libertad de usar mi antena donde y cuando yo lo desee (*POTA, SOTA, Field days, etc*.) y que se adecuara a mi rotor (originalmente con

1 motor NEMA para Azimut y 1 Servo control para Elevación, sin embargo ahora utilizare 2 motores NEMA, 1 para cada eje de rotacion)

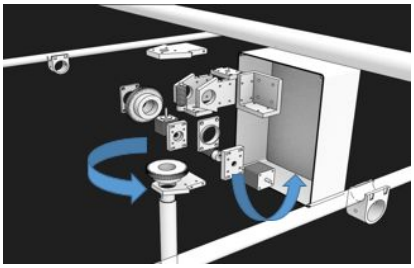
3. Debido a las modificaciones del punto anterior tuve el problema de no poder acceder a algunas librerías de C++ que no estaban en los repositorios públicos, pero por fortuna encontré una derivación basada en **PlatformIO** en GitLab, que emula perfectamente la IDE oficial de Arduino (y otras…) y que contiene todo el código fuente con las modificaciones iniciales para lograr el objetivo del punto 2 (*Usar Arduino y motores NEMA sin ser parte de la red SATNOGS*)
4. El punto anterior nos abre la posibilidad de utilizar otros microprocesadores como ATMEL, ESP32 e incluso Raspberry Pi's como plataformas de control en otros proyectos dándonos una gran flexibilidad y control sobre aplicaciones.

Necesitaremos para este proyecto software y hardware ligeramente modificados, siendo los principales cambios la remoción del protocolo RS485 y los watchdogs (*o sistemas de monitoreo de red*) del nuevo firmware para nuestro placa Arduino UNO.

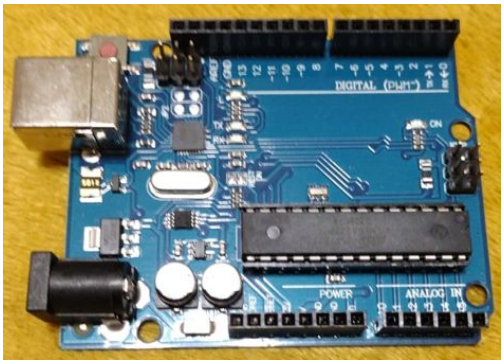
Componentes del proyecto:

Hardware

- 1 Rotor AZ-EL (hardware) similar en concepto al de SatNOGS V3.1 que utilice **2 motores de pasos tipo NEMA** como los de impresoras 3D (aunque también pudieran usarse servomotores o motores DC con moto reductor y **modificaciones adicionales a la programación**)

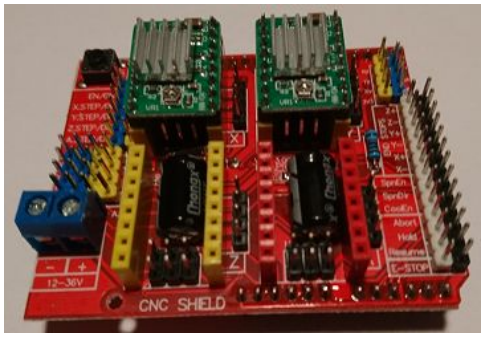


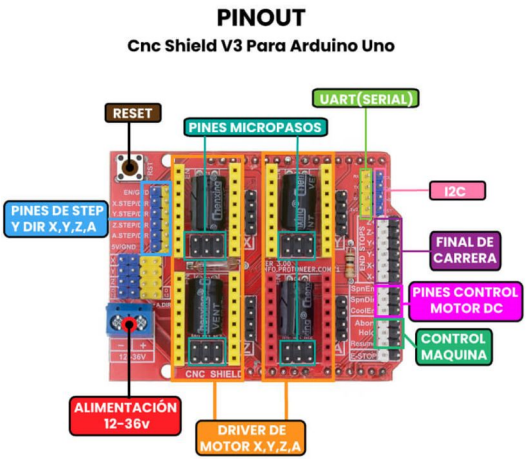
- 1 placa Arduino Uno R3 (o compatible) de bajo costo

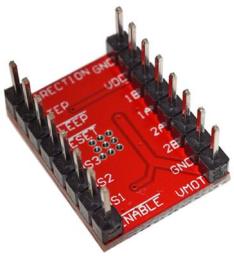


- 1 Cable USB largo (3 mts) para conectar el Arduino con la PC

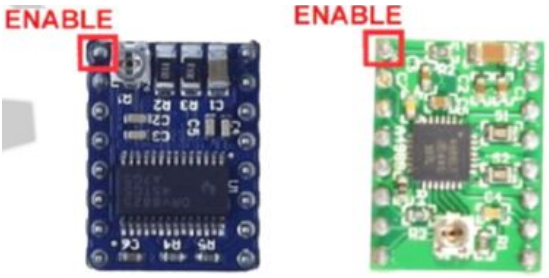
- 1 Escudo de control CNC (Shield V3) con drivers tipo **A4988, DRV8825** o similares para separar la placa de control de los motores (se pueden conectar directo, pero el amperaje puede causar problemas en la placa de control)







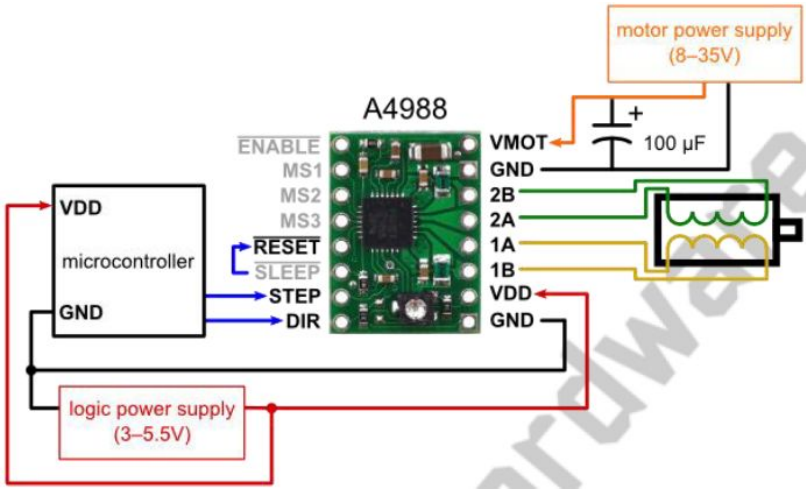
*Cuando se instalen los drivers, asegúrense de conectarlos orientados correctamente. De forma que el PIN de ENABLE (**EN**) coincida con el PIN en la tablilla del CNC (**arriba a la izquierda**). Noten que el pequeño potenciómetro de calibración de corriente se encuentra “abajo” de los **drivers A4988**, mientras que en los **drivers DRV8825** se encuentra “arriba”, además asegúrense de no confundir las tablillas de drivers.

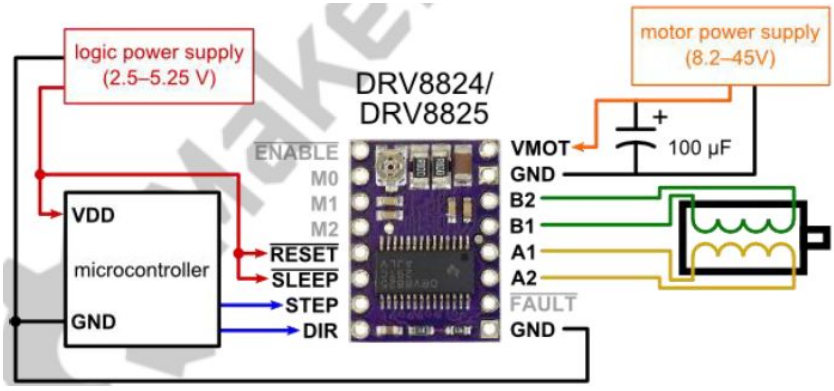


DRV8825 A4988

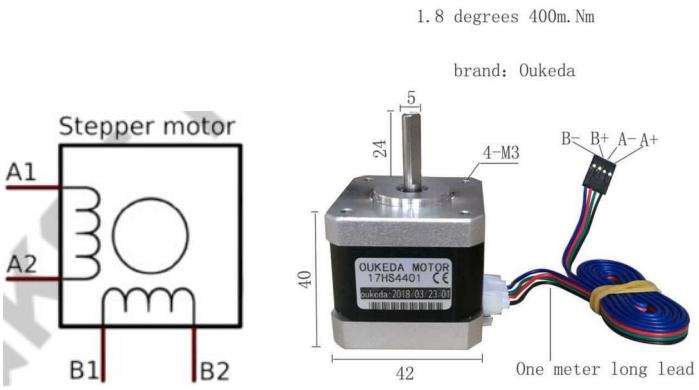
NOTA: ANTES de conectar los motores hay que hacer una calibración en los drivers para limitar el flujo de corriente máxima que los motores requieren (hay numerosos tutoriales en internet para esta actividad), consulten las tablas de especificaciones de los motores NEMA que vayan a utilizar.

Esta es la manera correcta de conectar los drivers con los motores de paso y la PLACA del controlador (según el driver que utilicen):





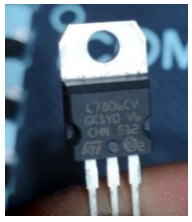
- 2 Motores Stepper (paso a paso) tipo NEMA 17 o 23 con torque suficiente para manejar el peso de la(s) antena(s) seleccionada(s), o agregar un moto reductor mecánico al motor (*solo cambia el valor inicial de una variable en el software*)



- 2 micros switches de tope (*de botón, de Carrera o sensores de proximidad*)

- Fuente de poder o batería de 12v con amperaje suficiente para alimentar al controlador y a los motores (*el escudo CNC acepta entre 12 y 36 volts*), si vas a usar un radio móvil puedes conectarlo a la misma fuente ya que usamos baja potencia en esta modalidad (*si usas portátil obviamente es independiente*), hagan sus cálculos de consumo para el tiempo que vayan a operar.

- Reguladores de voltaje si se van a usar Servo motores de 6 volts (**7806**) o placas de 5 volts (**7805**) o dispositivos, displays LCD/OLED, etc. Que no soporten 12 volts directamente. *También se pueden usar reguladores de ajuste variable*.



Software

Para los propósitos de este proyecto usaremos software de Windows, pero el mismo resultado se puede lograr con Linux o Mac OS (*todas las utilerías, programas, plugins y scripts usados son Freeware, o están bajo licencia de operación de software libre*)

Se requiere de:

1. Una copia de **Arduino IDE**, o instalar en nuestro caso una copia del plugin **PlatformIO** (hay muchas guías en Internet para usar esta IDE en **Visual Studio Code** de Microsoft, y que es uno de sus pocos productos gratuitos)
2. **GPredict** para Windows (mi mejor opción, aunque hay varios de donde escoger)
3. **Hamlib** para Windows (puede ser usada por muchos otros programas de radio)
4. Cualquier editor der texto para crear o modificar scripts (yo uso **Notepad++**)
5. Pueden encontrar el código fuente del firmware para Arduino y sus librerías y dependencias para **PlatformIO** en la siguiente derivación del repositorio de GitLAB:

   https://gitlab.com/Quartapound/satnogs-rotator-firmware

Pasos a seguir…

Instalación del Hardware

NO usaremos el controlador propietario SatNOGS PCB. En su lugar este proyecto usara una placa Arduino UNO y una placa “escudo” CNC (Shield V3).

El ensamblado requiere de muy pocas habilidades, típicamente Arduino no requiere ensamblado, la placa “escudo” requiere dos drivers A4988 o similares de acuerdo a las fotos anteriores, los drivers encajan en los cabezales hembra de la placa escudo, pero observen que existen unos “jumpers” debajo de los drivers que son usados para el control de “pasos o etapas” de los motores. Los motores se conectan directo a los cabezales macho indicados en la placa.

El pin out de los 4 cabezales disponibles es como sigue iniciando desde arriba (4 pines verticales a la izquierda de las tablillas de los drivers A4988, marcadas como **X**, **Y** y **Z**)

Pin 1 - A

Pin 2 - A+

Pin 3 - B

Pin 4 - B+

Esta configuración sigue un estándar, de manera que la gran mayoría de los motores NEMA podrán conectarse directamente en los cabezales sin cambios, sin embargo existen excepciones y puede haber casos donde los pines estén cableados diferente y los motores se moverán en la dirección opuesta.

Esta configuración es muy básica pero se debe tener cuidado de no doblar los pines, ya que es muy fácil hacerlo si no pone cuidado o presión en exceso cuando las cosas no están correctamente alineadas. Nada es perfecto !

Los “jumpers” que se encuentran abajo de los ***pololus*** (*así conocen también a los drivers no sé porque*), necesitan ser colocados en las posiciones **M0** y **M1** y dejar libre la posición **M2** ya que solo usaremos 2 ejes de rotación en este proyecto.

Al final tendremos:

- Driver/Motor de Azimut conectado al puerto **Y** en la placa CNC
- Driver/Motor de Elevación conectado al puerto **X** en la placa CNC
- El switch “tope” para la Elevación conectado a la posición **X** en la placa CNC (+ o -)
- El switch “tope” para el Azimut conectado a la posición **Y** en la placa CNC (+ o -)

Configuración del Software

Modificaciones (*Aqui es donde se pone interesante*)

Al día de hoy no existe una derivación **oficial** que cubra nuestros requerimientos del firmware en los repositorios del proyecto. Esto podría cambiar en el futuro, pero por el momento se requieren unos pequeños cambios para lograr los resultados deseados.

Básicamente necesitamos modificar dos elementos del firmware, **rotator_pins.h** y **easycomm.h** que son librerías requeridas por el programa principal (que por cierto en el plugin de **PlatformIO** se llama “**main.cpp**” a diferencia del IDE oficial de Arduino que utiliza el mismo nombre del proyecto con la extensión **.ino** (esto me causó cierta confusión al inicio ya que yo no soy un programador experto).

Estos cambios son necesarios para evitar conflictos con las comunicaciones entre las placas y la PC (*o cualquier otro dispositivo usado como “tracker”*), y también para reasignar los pines para que coincidan con nuestra placa CNC.

Modificaremos primero **rotator_pins.h**

Noten que se usara el uso de comentarios (//) para “remover” líneas de código y cambiaremos las designaciones de los pines para que coincidan con la placa CNC.

```
/*!
* @file rotator_pins.h
*
* Este es un archivo de encabezado para el mapeo de los pines.
* Version para mi rotor de 2 motores NEMA XE2UD v1.1
*
* Bajo Licencia GPLv3
*
*/

#ifndef ROTATOR_PINS_H_
#define ROTATOR_PINS_H_
```

```
//#define M1IN1 10 ///< Motor 1 pin PWM
#define M1IN1 2 ///< Motor 1 pin PWM
#define M1IN2 5  ///< Motor 1 pin PWM
#define M1SF  7  ///< Motor 1 entrada digital, bandera de estatus para Drivers de motores CD
#define M1FB  A1 ///< Motor 1 entrada analoga, retroalimentacion de corriente/carga para Drivers de motores CD

#define M2IN1 3 ///< Motor 2 pin PWM
#define M2IN2 6  ///< Motor 2 pin PWM
#define M2SF  7 ///< Motor 2 entrada digital, bandera de estatus para Drivers de motores CD
#define M2FB  A0 ///< Motor 2 entrada analoga, retroalimentacion de corriente/carga para Drivers de motores CD

#define MOTOR_EN 8 ///< Salida Digital, para activar los motores

#define SW1 10 ///< Entrada Digital, para leer el estatus de detencion del motor 1
#define SW2 9 ///< Entrada Digital, para leer el estatus de detencion del motor 2

#define RS485_DIR 2 ///< Salida Digital, para definir la direccion de la comunicacion RS485

#define SDA_PIN 3 ///< I2C pin de datos
#define SCL_PIN 4 ///< I2C pin de reloj

#define PIN12 12 ///< Pin I/O de proposito General
#define PIN13 13 ///< Pin I/O de proposito General
#define A2    A2 ///< I/O de proposito General & pin analogo
#define A3    A3 ///< I/O de proposito General & pin analogo
#endif /* ROTATOR_PINS_H_ */
```

La siguiente edición de código removerá los elementos que hacen referencia tanto al protocolo **RS485** como al “**watchdog**” dentro del programa **easycomm.h** de comunicaciones, lo cual permitirá a la PC a “hablar” directamente con Arduino y en consecuencia no será necesario usar una Raspberry Pi como “host” intermedio del sistema en el proyecto original.

A continuación el código modificado. Noten nuevamente el uso de los comentarios (//) para “remover” las líneas de código no deseadas.

```
/*!
* @file easycomm.h
*
* Este es un driver para el protocolo easycomm 3 como es referido en las librerias Hamlib
*
* Bajo Licencia GPLv3
*
*/
```

```
#ifndef LIBRARIES_EASYCOMM_H_
#define LIBRARIES_EASYCOMM_H_

#include <Arduino.h>
#include <WString.h>
#include <avr/wdt.h>
//#include "rs485.h"
#include "rotator_pins.h"
#include "globals.h"

#define RS485_TX_TIME 9     ///< Retardo de "t"ms para escribir en la implementacion de RS485
#define BUFFER_SIZE   256   ///< Fija el tamano del buffer serial
#define BAUDRATE      19200 ///< Fija el Baudrate del protocolo easycomm 3

//rs485 rs485(RS485_DIR, RS485_TX_TIME);

/*****************************************************************************/
/*!
    @brief    Clasificacion de funciones para la implementacion de easycomm 3
*/
/*****************************************************************************/
class easycomm {
public:

    /*****************************************************************************/
    /*!
        @brief    Inicializa el bus de RS485
    */
    /*****************************************************************************/
    void easycomm_init() {
       // rs485.begin(BAUDRATE);
        Serial.begin(9600);
    }

    /*****************************************************************************/
    /*!
        @brief    Obtiene los comandos desde RS485 y responde al cliente
    */
    /*****************************************************************************/
    void easycomm_proc() {
        char buffer[BUFFER_SIZE];
        char incomingByte;
        char *Data = buffer;
        char *rawData;
        static uint16_t BufferCnt = 0;
```

```
char data[100];
String str1, str2, str3, str4, str5, str6;

// Read from serial
while (Serial.available() > 0) {
    incomingByte = Serial.read();

    // Lee nuevos datos, '\n' significa nuevo paquete
    if (incomingByte == '\n' || incomingByte == '\r') {
        buffer[BufferCnt] = 0;
        if (buffer[0] == 'A' && buffer[1] == 'Z') {
            if (buffer[2] == ' ' && buffer[3] == 'E' &&
                buffer[4] == 'L') {
                // Envia la posicion actual absoluta en grados
                str1 = String("AZ");
                str2 = String(control_az.input, 1);
                str3 = String(" EL");
                str4 = String(control_el.input, 1);
                str5 = String("\n");
                Serial.print(str1 + str2 + str3 + str4 + str5);
            } else {
                // Lee la posicion absoluta en grados para azimut
                rotator.control_mode = position;
                rawData = strtok_r(Data, " ", &Data);
                strncpy(data, rawData + 2, 10);
                if (isNumber(data)) {
                    control_az.setpoint = atof(data);
                }
                // Lee la posicion absoluta en grados para elevacion
                rawData = strtok_r(Data, " ", &Data);
                if (rawData[0] == 'E' && rawData[1] == 'L') {
                    strncpy(data, rawData + 2, 10);
                    if (isNumber(data)) {
                        control_el.setpoint = atof(data);
                    }
                }
            }
        } else if (buffer[0] == 'E' && buffer[1] == 'L') {
                // Lee la posicion absoluta en grados para elevacion
                rotator.control_mode = position;
                rawData = strtok_r(Data, " ", &Data);
                if (rawData[0] == 'E' && rawData[1] == 'L') {
                    strncpy(data, rawData + 2, 10);
                    if (isNumber(data)) {
                        control_el.setpoint = atof(data);
```

```
            }
        }
    } else if (buffer[0] == 'V' && buffer[1] == 'U') {
        // Incremento en la velocidad de Elevacion en mgrados/s
        rotator.control_mode = speed;
        strncpy(data, Data + 2, 10);
        if (isNumber(data)) {
            // Convierte a grados/s
            control_el.setpoint_speed = atof(data) / 1000;
        }
    } else if (buffer[0] == 'V' && buffer[1] == 'D') {
        // Decremento en la velocidad de Elevacion en mgrados/s
        rotator.control_mode = speed;
        strncpy(data, Data + 2, 10);
        if (isNumber(data)) {
            // Convierte a grados/s
            control_el.setpoint_speed = - atof(data) / 1000;
        }
    } else if (buffer[0] == 'V' && buffer[1] == 'L') {
        // Incremento en la velocidad de Azimut en mgrados/s
        rotator.control_mode = speed;
        strncpy(data, Data + 2, 10);
        if (isNumber(data)) {
            // Convierte a grados/s
            control_az.setpoint_speed = atof(data) / 1000;
        }
    } else if (buffer[0] == 'V' && buffer[1] == 'R') {
        // Decremento en la velocidad de Azimut en mgrados/s
        rotator.control_mode = speed;
        strncpy(data, Data + 2, 10);
        if (isNumber(data)) {
            // Convierte a grados/s
            control_az.setpoint_speed = - atof(data) / 1000;
        }
    } else if (buffer[0] == 'S' && buffer[1] == 'A' &&
               buffer[2] == ' ' && buffer[3] == 'S' &&
               buffer[4] == 'E') {
        // Detiene el movimiento
        rotator.control_mode = position;
        str1 = String("AZ");
        str2 = String(control_az.input, 1);
        str3 = String(" EL");
        str4 = String(control_el.input, 1);
        str5 = String("\n");
        Serial.print(str1 + str2 + str3 + str4 + str5);
```

```
    control_az.setpoint = control_az.input;
    control_el.setpoint = control_el.input;
} else if (buffer[0] == 'R' && buffer[1] == 'E' &&
           buffer[2] == 'S' && buffer[3] == 'E' &&
           buffer[4] == 'T') {
    // Resetea el rotor, y se dirige a la posicion home
    str1 = String("AZ");
    str2 = String(control_az.input, 1);
    str3 = String(" EL");
    str4 = String(control_el.input, 1);
    str5 = String("\n");
    Serial.print(str1 + str2 + str3 + str4 + str5);
    rotator.homing_flag = false;
} else if (buffer[0] == 'P' && buffer[1] == 'A' &&
           buffer[2] == 'R' && buffer[3] == 'K' ) {
    // Estaciona el rotor
    rotator.control_mode = position;
    str1 = String("AZ");
    str2 = String(control_az.input, 1);
    str3 = String(" EL");
    str4 = String(control_el.input, 1);
    str5 = String("\n");
    Serial.print(str1 + str2 + str3 + str4 + str5);
    control_az.setpoint = rotator.park_az;
    control_el.setpoint = rotator.park_el;
} else if (buffer[0] == 'V' && buffer[1] == 'E') {
    // Lee la version del controlador de rotor
    str1 = String("VE");
    str2 = String("XE2UD-v1.1");
    str3 = String("\n");
    Serial.print(str1 + str2 + str3);
} else if (buffer[0] == 'I' && buffer[1] == 'P' &&
           buffer[2] == '0') {
    // Lee la temperatura interior
    str1 = String("IP0,");
    str2 = String(rotator.inside_temperature, DEC);
    str3 = String("\n");
    Serial.print(str1 + str2 + str3);
} else if (buffer[0] == 'I' && buffer[1] == 'P' &&
           buffer[2] == '1') {
    // Lee el estatus de la parada final, Azimut
    str1 = String("IP1,");
    str2 = String(rotator.switch_az, DEC);
    str3 = String("\n");
    Serial.print(str1 + str2 + str3);
```

```
} else if (buffer[0] == 'I' && buffer[1] == 'P' &&
           buffer[2] == '2') {
    // Lee el estatus de la parada final, Elevacion
    str1 = String("IP2,");
    str2 = String(rotator.switch_el, DEC);
    str3 = String("\n");
    Serial.print(str1 + str2 + str3);
} else if (buffer[0] == 'I' && buffer[1] == 'P' &&
           buffer[2] == '3') {
    // Lee la posicion actual del Azimut en grados
    str1 = String("IP3,");
    str2 = String(control_az.input, 2);
    str3 = String("\n");
    Serial.print(str1 + str2 + str3);
} else if (buffer[0] == 'I' && buffer[1] == 'P' &&
           buffer[2] == '4') {
    // Lee la posicion actual de Elevacion en grados
    str1 = String("IP4,");
    str2 = String(control_el.input, 2);
    str3 = String("\n");
    Serial.print(str1 + str2 + str3);
} else if (buffer[0] == 'I' && buffer[1] == 'P' &&
           buffer[2] == '5') {
    // Lee la carga del Azimut, en un rango de 0-1023
    str1 = String("IP5,");
    str2 = String(control_az.load, DEC);
    str3 = String("\n");
    Serial.print(str1 + str2 + str3);
} else if (buffer[0] == 'I' && buffer[1] == 'P' &&
           buffer[2] == '6') {
    // Lee la carga de la Elevacion, en un rango de 0-1023
    str1 = String("IP6,");
    str2 = String(control_el.load, DEC);
    str3 = String("\n");
    Serial.print(str1 + str2 + str3);
} else if (buffer[0] == 'I' && buffer[1] == 'P' &&
           buffer[2] == '7') {
    // Lee la velocidad del Azimuth en grados/s
    str1 = String("IP7,");
    str2 = String(control_az.speed, 2);
    str3 = String("\n");
    Serial.print(str1 + str2 + str3);
} else if (buffer[0] == 'I' && buffer[1] == 'P' &&
           buffer[2] == '8') {
    // Lee la velocidad de Elevacion en grados/s
```

```
            str1 = String("IP8,");
            str2 = String(control_el.speed, 2);
            str3 = String("\n");
            Serial.print(str1 + str2 + str3);
        } else if (buffer[0] == 'G' && buffer[1] == 'S') {
            // Lee el estatus del rotor
            str1 = String("GS");
            str2 = String(rotator.rotator_status, DEC);
            str3 = String("\n");
            Serial.print(str1 + str2 + str3);
        } else if (buffer[0] == 'G' && buffer[1] == 'E') {
            // Lee el error del rotor
            str1 = String("GE");
            str2 = String(rotator.rotator_error, DEC);
            str3 = String("\n");
            Serial.print(str1 + str2 + str3);
        } else if(buffer[0] == 'C' && buffer[1] == 'R') {
            // Lee la configuracion del rotor
            if (buffer[3] == '1') {
                // Lee la ganancia Kp del Azimut
                str1 = String("1,");
                str2 = String(control_az.p, 2);
                str3 = String("\n");
                Serial.print(str1 + str2 + str3);
            } else if (buffer[3] == '2') {
                // Lee la ganancia Ki del Azimut
                str1 = String("2,");
                 str2 = String(control_az.i, 2);
                 str3 = String("\n");
                 Serial.print(str1 + str2 + str3);
            } else if (buffer[3] == '3') {
                // Lee la ganancia Kd del Azimut
                str1 = String("3,");
                str2 = String(control_az.d, 2);
                str3 = String("\n");
                Serial.print(str1 + str2 + str3);
            } else if (buffer[3] == '4') {
                // Lee la ganancia Kp del Azimut
                str1 = String("4,");
                 str2 = String(control_el.p, 2);
                 str3 = String("\n");
                 Serial.print(str1 + str2 + str3);
            } else if (buffer[3] == '5') {
                // Lee la ganancia Ki de la Elevacion
                str1 = String("5,");
```

```
                str2 = String(control_el.i, 2);
                str3 = String("\n");
                Serial.print(str1 + str2 + str3);
            } else if (buffer[3] == '6') {
                // Lee la ganancia Kd de Elevacion
                str1 = String("6,");
                str2 = String(control_el.d, 2);
                str3 = String("\n");
                Serial.print(str1 + str2 + str3);
            } else if (buffer[3] == '7') {
                // Lee la posicion de estacionamiento del Azimuth
                str1 = String("7,");
                str2 = String(rotator.park_az, 2);
                str3 = String("\n");
                Serial.print(str1 + str2 + str3);
            } else if (buffer[3] == '8') {
                // Lee la posicion de estacionamiento de la Elevacion
                str1 = String("8,");
                str2 = String(rotator.park_el, 2);
                str3 = String("\n");
                Serial.print(str1 + str2 + str3);
            } else if (buffer[3] == '9') {
                // Lee el modo de control
                str1 = String("9,");
                str2 = String(rotator.control_mode);
                str3 = String("\n");
                Serial.print(str1 + str2 + str3);
            }
        } else if (buffer[0] == 'C' && buffer[1] == 'W') {
            // Fija Configuracion
            if (buffer[2] == '1') {
                // Fija la ganancia Kp del Azimut
                rawData = strtok_r(Data, ",", &Data);
                strncpy(data, rawData + 4, 10);
                if (isNumber(data)) {
                    control_az.p = atof(data);
                }
            } else if (buffer[2] == '2') {
                // Fija la ganancia Ki del Azimut
                rawData = strtok_r(Data, ",", &Data);
                strncpy(data, rawData + 4, 10);
                if (isNumber(data)) {
                    control_az.i = atof(data);
                }
            } else if (buffer[2] == '3') {
```

```
                // Fija la ganancia Kd del Azimut
                rawData = strtok_r(Data, ",", &Data);
                strncpy(data, rawData + 4, 10);
                if (isNumber(data)) {
                    control_az.d = atof(data);
                }
            } else if (buffer[2] == '4') {
                // Fija la ganancia Kp de la Elevacion
                rawData = strtok_r(Data, ",", &Data);
                strncpy(data, rawData + 4, 10);
                if (isNumber(data)) {
                    control_el.p = atof(data);
                }
            } else if (buffer[2] == '5') {
                // Fija la ganancia Ki de la Elevacion
                rawData = strtok_r(Data, ",", &Data);
                strncpy(data, rawData + 4, 10);
                if (isNumber(data)) {
                    control_el.i = atof(data);
                }
            } else if (buffer[2] == '6') {
                // Fija la ganancia Kd de la Elevacion
                rawData = strtok_r(Data, ",", &Data);
                strncpy(data, rawData + 4, 10);
                if (isNumber(data)) {
                    control_el.d = atof(data);
                }
            }  else if (buffer[2] == '7') {
                // Fija la posicion de estacionamiento del Azimuth
                rawData = strtok_r(Data, ",", &Data);
                strncpy(data, rawData + 4, 10);
                if (isNumber(data)) {
                    rotator.park_az = atof(data);
                }
            } else if (buffer[2] == '8') {
                // Fija la posicion de estacionamiento de Elevacion
                rawData = strtok_r(Data, ",", &Data);
                strncpy(data, rawData + 4, 10);
                if (isNumber(data)) {
                    rotator.park_el = atof(data);
                }
            }
        } else if (buffer[0] == 'R' && buffer[1] == 'S'
                && buffer[2] == 'T') {
            // Comando especial para probar la rutina timer del watchdog
```

```
                    while(1)
                        ;
                } else if (buffer[0] == 'R' && buffer[1] == 'B') {
                    // Comando especial para rebootear el uC
                    wdt_enable(WDTO_2S);
                    while(1);
                }
                // Reset del buffer y limpia buffer serial
                BufferCnt = 0;
                Serial.flush();
            } else {
                // Llena el buffer con datos de entrada
                buffer[BufferCnt] = incomingByte;
                BufferCnt++;
            }
        }
    }

private:
    bool isNumber(char *input) {
        for (uint16_t i = 0; input[i] != '\0'; i++) {
            if (isalpha(input[i]))
                return false;
        }
        return true;
    }
};

#endif /* LIBRARIES_EASYCOMM_H_ */
```

NOTA: Todos los demás códigos de programación, encabezados y librerías necesarias quedan igual que en el proyecto original por lo que solo falta ejecutar la rutina **DEBUG, COMPILAR** (*si no existen errores*) y cargar el código **ejecutable** en la placa de **Arduino.**

El siguiente código corresponde al programa principal "**main.cpp**" que **NO REQUIERE** ser modificado a menos que ustedes deseen hacer cambios MAYORES al comportamiento de sus rotores y es mostrado únicamente para efectos de comprensión del funcionamiento del firmware de control.

```
//  Programa principal de control para rotor XE2UD v1.1 (2 motores NEMA)
#define SAMPLE_TIME         0.1   ///< Control loop en segundos
#define RATIO               54    ///< Relacion de engrane del motoreductor en el proyecto Satnogs V3.1
#define MICROSTEP           8     ///< VALOR de Microstep
#define MIN_PULSE_WIDTH     20    ///< En microsegundos para AccelStepper
#define MAX_SPEED           3200  ///< En pasos/s, considera el valor de microstep, default 3200
#define MAX_ACCELERATION    1600  ///< En pasos/s^2, considera el valor de microstep, default 1600
```

```
#define SPR                1600L ///< Pasos por Revolucion, considera el valor de microstep
#define MIN_M1_ANGLE       0     ///< Minimo angulo del azimuth
#define MAX_M1_ANGLE       360   ///< Maximo angulo del azimuth
#define MIN_M2_ANGLE       0     ///< Minimo angulo de elevacion
#define MAX_M2_ANGLE       360   ///< Maximo angulo de elevation
#define DEFAULT_HOME_STATE LOW   ///< Cambia a LOW de acuerdo al sensor Home
#define HOME_DELAY         400   ///< Tiempo para Desaceleracion de homing en milisegundos // original 12000

#include <AccelStepper.h>
#include <Wire.h>
#include "globals.h"
#include "easycomm.h"
#include "rotator_pins.h"
//#include "rs485.h"        //Deshabilita el protocolo RS485
#include "endstop.h"
//#include "watchdog.h"     //Deshabilita el watchdog

uint32_t t_run = 0; // run time del uC
easycomm comm; // implementa Easycomm version 3
AccelStepper stepper_az(1, M1IN1, M1IN2);
AccelStepper stepper_el(1, M2IN1, M2IN2);
endstop switch_az(SW1, DEFAULT_HOME_STATE), switch_el(SW2, DEFAULT_HOME_STATE);
//wdt_timer wdt;

enum _rotator_error homing(int32_t seek_az, int32_t seek_el);
int32_t deg2step(float deg);
float step2deg(int32_t step);

void setup() {
    // Switch de Homing
    switch_az.init();
    switch_el.init();

    // Comunicacion Serial
    comm.easycomm_init();
    Serial.println("Rotor XE2UD listo...");

    // Setup del motor paso a paso
    stepper_az.setEnablePin(MOTOR_EN);
    stepper_az.setPinsInverted(false, false, true);
    stepper_az.enableOutputs();
    stepper_az.setMaxSpeed(MAX_SPEED);
    stepper_az.setAcceleration(MAX_ACCELERATION);
    stepper_az.setMinPulseWidth(MIN_PULSE_WIDTH);
    stepper_el.setPinsInverted(false, false, true);
```

```
    stepper_el.enableOutputs();
    stepper_el.setMaxSpeed(MAX_SPEED);
    stepper_el.setAcceleration(MAX_ACCELERATION);
    stepper_el.setMinPulseWidth(MIN_PULSE_WIDTH);

    // Inicializa WDT (tiempo del watchdog)
    // wdt.watchdog_init();
}

void loop() {
    // Actualiza el WDT
    // wdt.watchdog_reset();

    // Lee el estado de paro desde los switches tope
    rotator.switch_az = switch_az.get_state();
    rotator.switch_el = switch_el.get_state();

    // Ejecuta la implementacion del protocolo Easycomm
    comm.easycomm_proc();

    // Lee la posicion de ambos ejes
    control_az.input = step2deg(stepper_az.currentPosition());
    control_el.input = step2deg(stepper_el.currentPosition());

    // Checa el estado del rotor
    if (rotator.rotator_status != error) {
        if (rotator.homing_flag == false) {
            // Checa la bandera de la posicion Homing
            rotator.control_mode = position;
            // Homing
            rotator.rotator_error = homing(deg2step(-MAX_M1_ANGLE),
                                           deg2step(-MAX_M2_ANGLE));
            if (rotator.rotator_error == no_error) {
                // Sin error
                rotator.rotator_status = idle;
                rotator.homing_flag = true;
            } else {
                // Error
                rotator.rotator_status = error;
                rotator.rotator_error = homing_error;
            }
        } else {
            // Control Loop
            stepper_az.moveTo(deg2step(control_az.setpoint));
            stepper_el.moveTo(deg2step(control_el.setpoint));
```

```
            rotator.rotator_status = pointing;
            // Mueve los motores de Azimut y Elevacion
            stepper_az.run();
            stepper_el.run();
            // Rotor en reposo
            if (stepper_az.distanceToGo() == 0 && stepper_el.distanceToGo() == 0) {
                rotator.rotator_status = idle;
            }
        }
    } else {
        // Manejo de Error, para los motores y desactiva los drivers de motor
        stepper_az.stop();
        stepper_az.disableOutputs();
        stepper_el.stop();
        stepper_el.disableOutputs();
        if (rotator.rotator_error != homing_error) {
            // Restablece error de acuerdo al valor del estado de error
            rotator.rotator_error = no_error;
            rotator.rotator_status = idle;
        }
    }
}

/*************************************************************/
/*!
    @brief    Mueve ambos ejes hasta encontrar la posicion Home
    @param    seek_az
              Etapas para encontrar la posicion Home de Azimut
    @param    seek_el
              Etapas para encontrar la posicion Home de Elevacion
    @return   _rotator_error
*/
/*************************************************************/
enum _rotator_error homing(int32_t seek_az, int32_t seek_el) {
    bool isHome_az = false;
    bool isHome_el = false;

    // Mueve los motores a la posicion de "busqueda"
    stepper_az.moveTo(seek_az);
    stepper_el.moveTo(seek_el);

    // Loop de Homing
    while (isHome_az == false || isHome_el == false) {
       // Actualiza WDT
       // wdt.watchdog_reset();
```

```
        if (switch_az.get_state() == true && !isHome_az) {
            // Encuentra el Home de azimuth
            stepper_az.moveTo(stepper_az.currentPosition());
            isHome_az = true;
        }
        if (switch_el.get_state() == true && !isHome_el) {
            // Encuentra el Home de elevacion
            stepper_el.moveTo(stepper_el.currentPosition());
            isHome_el = true;
        }
        // Checa si el rotor sale de limites o existe error mecanico
        if ((stepper_az.distanceToGo() == 0 && !isHome_az) ||
            (stepper_el.distanceToGo() == 0 && !isHome_el)){
            return homing_error;
        }
        // Mueve motores a la posicion de "busqueda"
        stepper_az.run();
        stepper_el.run();
    }
    // Retardo para Desacelerar y Homing hasta completar los movimientos
    uint32_t time = millis();
    while (millis() - time < HOME_DELAY) {
        // wdt.watchdog_reset();
        stepper_az.run();
        stepper_el.run();
    }
    // Fija la posicion Home y resetea todas la variables criticas de control
    stepper_az.setCurrentPosition(0);
    stepper_el.setCurrentPosition(0);
    control_az.setpoint = 0;
    control_el.setpoint = 0;

    return no_error;
}

/*****************************************************************/
/*!
    @brief    Convierte GRADOS a PASOS de acuerdo a los pasos/revolucion
              la relacion del reductor y el valor de Microstep
    @param    deg
              Grados en formato flotante
    @return   Etapas para el driver del motor, int32_t
*/
/*****************************************************************/
int32_t deg2step(float deg) {
```

```
    return (RATIO * SPR * deg / 360);
}


/*****************************************************************/
/*!
    @brief    Convierte PASOS a GRADOS de acuerdo a los pasos/revolucion
              la relacion del reductor y el valor de Microstep
    @param    step
              Etapas en formato int32_t
    @return   Grados en formato flotante
*/
/*****************************************************************/
float step2deg(int32_t step) {
    return (360.00 * step / (SPR * RATIO));
}
```

Si se realiza la verificación del código, se compila y envía este programa a la placa Arduino, deberá recibir algo similar a lo mostrado a continuación y lo más importante es que al finalizar diga lo siguiente…

```
*  Executing task:

. . . .(Varios mensajes del avance)

avrdude done.  Thank you.

=== [SUCCESS] Took 15.74 seconds
```

Software adicional necesario

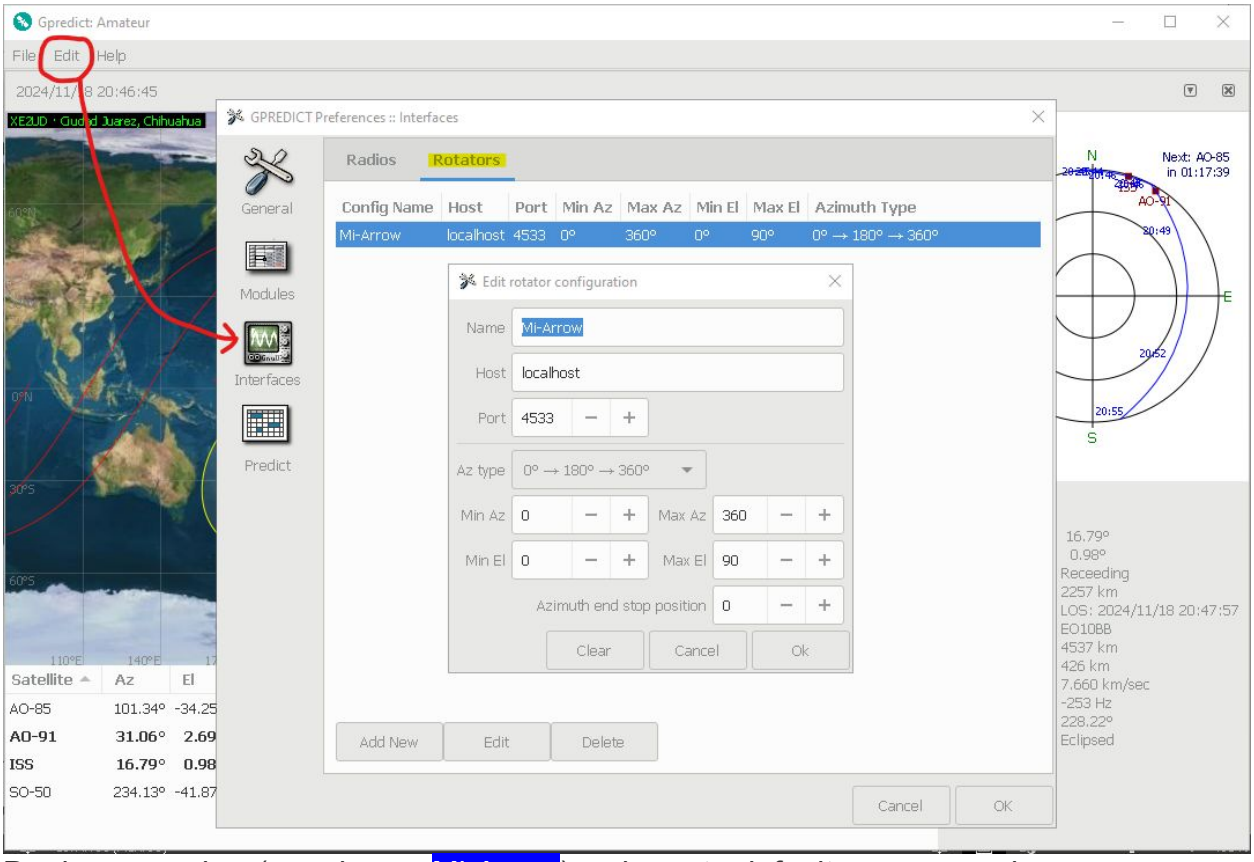
Las piezas finales de nuestro proyecto son un par de programas muy interesantes.

Primero debemos instalar nuestro programa principal de predicción (“Tracker”) **GPredict**. No entraremos en detalle acerca de la configuración de GPredict pero asumo que ustedes han seguido la guía de instalación y han explorado todas sus opciones y configuraciones a su preferencia para que tengan una idea completa de las capacidades de este excelente programa y como funciona.

En segundo lugar **Hamlib**, que es una pieza indispensable para controlar radios, rotores y otros dispositivos de nuestro hobby y que permite conectar otros programas con nuestros radios utilizando el protocolo estándar **EasyComm**. **Este programa no muestra ninguna pantalla** y solo aparecen cuando algún otro programa le pide **bajo demanda** que haga “algo” como configurar parámetros, etc…

Instalar GPredict

Bajen e instalen la versión correcta de GPredict (*en mi caso la versión de Windows solo requiere expandir el archivo ZIP en una nueva carpeta*), una vez instalado y arrancado el ejecutable **gpredict.exe** **debemos agregar un rotor**. Vayan a Edit > Preferences > Interfaces > Rotators > Add New



Denle un nombre (en mi caso Mi-Arrow) y el puerto default para comunicarse con **Hamlib en la misma PC** es localhost 4533 (*Si se conectan vía un host intermedio por red o WiFi ajustar en su caso poniendo el nombre del host o la IP correcta*)

Instalación de Hamlib

Bajen el software de Hamlib (en nuestro caso la versión de Windows) y sigan el proceso de instalación. Esto podría requerir privilegios de Administrador en Windows.

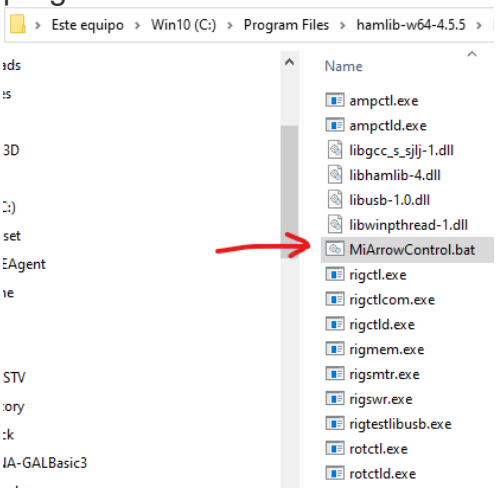
Y **FINALMENTE**…

Usen el “Administrador de Dispositivos” para encontrar en cual puerto **COM** se encuentra nuestro Arduino, para propósitos de demostración asumiremos que está conectado al puerto COM6

Usando un editor de textos (como Notepad++) crearemos un archivo **.BAT** con las siguientes instrucciones y que ustedes deberán ajustar dependiendo de su propio puerto COM detectado.

Salven este archivo como "**MiArrowControl.bat**" o algo similar (ejemplo: con la extensión **.bat** en la misma carpeta que Hamlib "**rotctld**") generalmente encontrada en **C:\Program Files\hamlib-w64-4.5.5\bin** para que al ejecutar este archivo podamos inicializar las utilerías de Hamlib y si lo desean (**adicional y opcionalmente**) arrancar el programa de "tracking" de nuestra predilección en un solo paso.

En mi siguiente ejemplo inicializo **Hamlib** y después arranco **automáticamente** el programa **GPredict** como se muestra a continuación…

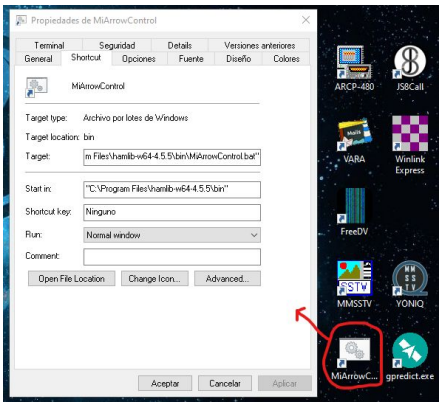


"C:\Program Files\hamlib-w64-4.5.5\bin\MiArrowControl.bat"

rotctld -m 202 -r COM6 -s 9600 -T 127.0.0.1 -t 4533 -C timeout=500 -C retry=0 -vvvvvvvv > pause

D:\MEGAsync\Install\Hardware\Radios\Docs\SATs\gpredict-win32-2.3.37\gpredict.exe

(También pueden crear una liga a este archivo desde su Desktop para iniciar cómodamente con un par de clicks el programa **Hamlib** y también el de **rastreo GPredict** u otro... muy conveniente)

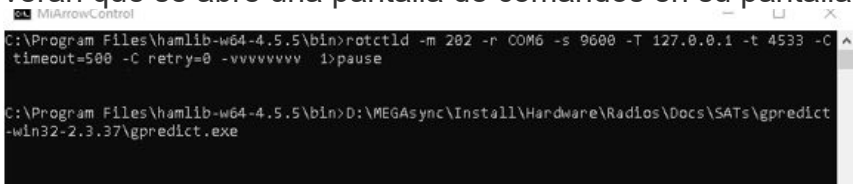


Esto es todo para las configuraciones, ahora probemos el sistema…

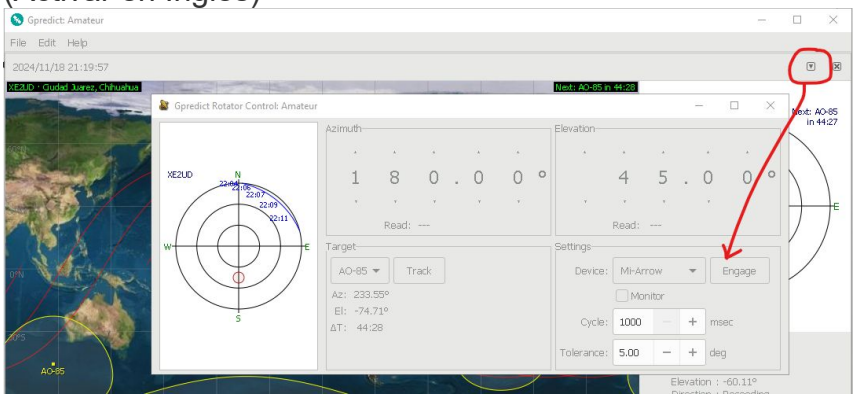
Pruebas

Debemos hacer algunas cosas para que nuestro sistema funcione.

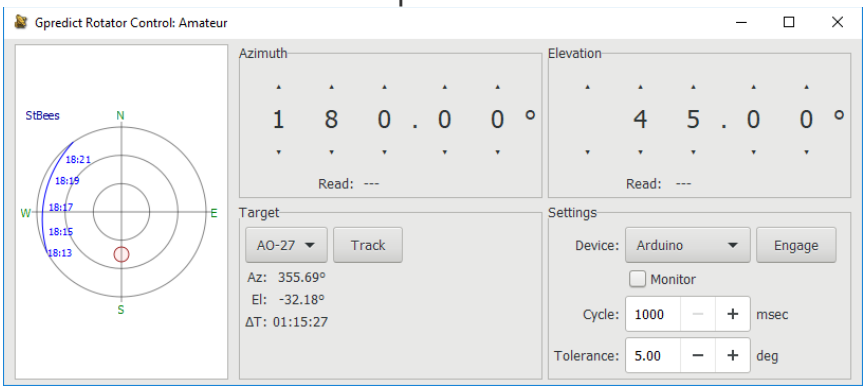
1. Iniciar el programa **Nombre.bat** (con el nombre que le hayan puesto) y verán que se abre una pantalla de comandos en su pantalla



   Esta es la pantalla "terminal" de **Hamlib** en donde aparecerán los comandos de comunicación y/o mensajes en tiempo real.
   Inmediatamente después de aparecer la pantalla de comandos se abrirá nuestro programa de rastreo si ustedes lo definieron opcionalmente en el archivo **Nombre.bat** (en mi caso **GPredict**)
2. En GPredict abrir el módulo de "Antenna" y oprimir el botón de "Engage" (Activar en Ingles)



3. En la pantalla del archivo batch (el que abrimos en el punto 1) aparecerán una serie de comandos que son enviados y recibidos a la placa Arduino
4. Chequen que tanto el Azimut y la Elevación de su rotor funcionen correctamente manejando manualmente el rotor o seleccionando un pase en curso de alguno de los satélites que hayan configurado para rastrear desde su localidad definida previamente usando el botón de "**Track**".



Puede suceder que esto no funcione a la primera vez. Para solucionarlo haga un reinicio o “restart” tanto de Hamlib como de GPredict.

Trucos y sugerencias:

Debido a la gran cantidad de partes que pueden ser utilizadas en estos proyectos, los siguientes trucos pueden auxiliarlos si detectan algún problema en caso de ser necesario algún ajuste.

Los Motores giran en sentido contrario – Revisar la asignación de pines

Solo se mueve la mitad (o parte) de la distancia – Ajustar la relación o “ratio” de rotación del motor correspondiente en el programa principal sobre todo si están usando moto reductores mecánicos para aumentar la capacidad de torque en su proyecto

Los switches de límite no están funcionando – Cambiar el siguiente código en el programa principal

```
define DEFAULT_HOME_STATE LOW ///< Cambiar a LOW dependiendo del sensor HOME utilizado
```

Operación

Operar nuestra estación de rastreo satelital es tan simple como seleccionar un satélite desde **GPredict** y seleccionar el botón de “**Track**”. El rotor debe comenzar a seguir automáticamente la trayectoria del satélite u objeto definido para nuestra localidad seleccionada en el módulo correspondiente.

Referencias principales:

Proyecto SatNOGS https://network.satnogs.org/

Proyectos SARCNET (School Amateur Radio Club Network)
https://www.sarcnet.org/default.html

GPredict https://oz9aec.dk/gpredict/

Hamlib para Windows https://sourceforge.net/projects/hamlib/

Es muy importante **pero no indispensable** que vean el siguiente video acerca de cómo instalar y utilizar el plugin **PlatformIO** dentro del entorno de programación del IDE dentro de **Visual Studio Code** de Microsoft (*en Ingles muy claro y conciso, presentado por un verdadero experto en programación y dispositivos robóticos*)

https://www.youtube.com/watch?v=JmvMvIphMnY

En otro artículo por separado les presentare la construcción de mi antena doble banda VHF/UHF de polarización cruzada para trabajar satélites de radioaficionados, basada en la popular Yagi Arrow II de 10 elementos pero con modificaciones, construida prácticamente con materiales de desecho y reciclados y que me ha dado excelentes resultados. Y en donde les presentare fotos de mi estación satelital portátil automática completa y en pleno funcionamiento.

Hasta una próxima oportunidad en cualquier banda y en cualquier modo…

73 de Miguel Dario XE2UD (ex XE1UD)