Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Diseño e implementación de un sistema motorizado para telescopio basado en microcontrolador

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Eduardo Ortega Serrano
**Tutor**: Alberto José Pérez Jiménez
2019-2020

# Agradecimientos

Quisiera comenzar esta memoria expresando mi reconocimiento, a todas aquellas personas que, de una forma u otra, han aportado algo a la realización de este TFG.

En primer lugar, a los profesores, que durante estos años tanto desde la Politécnica de Valencia, como desde Wroclaw (Polonia) han estado invirtiendo su tiempo y esfuerzo en la tarea de formarme como ingeniero y como persona y han aportado unos conocimientos y competencias avanzados, para hacer de mi un profesional y que en este momento me dan la posibilidad de desarrollar este trabajo final de grado espero, con éxito

Me gustaría poner en estima sobre todo al tutor, que me ha acompañado durante todo el transcurso del TFG, ya que sin sus acertados consejos y sus incondicionales ayudas, difícilmente hubiera puesto fin a este emocionante y laborioso proyecto, de manera acertada

Para finalizar, por supuesto agradecer a mi familia por el apoyo que me han dado en todo momento, pero en especial a mi madre que casi termina dominando estos términos, por sus correcciones, a mi hermano mayor por sus ideas gráficas y en general, a todos por la paciencia que han tenido conmigo en consideración a mi ilusión y esfuerzo, que siempre han potenciado positivamente.

# Resumen

En este Trabajo Fin de Grado se expone, una aplicación de ordenador, con la cual se puede dirigir y manejar, el posicionamiento de un telescopio y el control de este con un movimiento continuo, que nos permitiría, el seguimiento de diferentes astros a través de una placa Arduino Mega.

El programa se comunica con el Arduino mediante el Serial y le dice las acciones que debe realizar. La aplicación tiene dos métodos diferentes que son:

- El **método Goto** sirve para poner el Telescopio en las posiciones AR y DEC deseadas.
- El **método Follow** sirve para poner una velocidad y un tiempo para hacer un seguimiento.

En el Arduino encontramos una placa CNC la cual controla dos Drivers DVR8825 para los motores de las posiciones DEC y AR.

El programa también nos permite cambiar la velocidad de posicionamiento del telescopio y optimiza el seguimiento del objeto a estudiar con una mayor precisión, además de estas prestaciones y características, tenemos la opción de poder manejar o dirigir la posición de dicho telescopio de forma manual.

**Palabras clave:** Arduino, Serial, Goto, Follow, aplicación, AR, Dec, DVR8825 y telescopio.

# Abstract

In this thesis, I expose a computer application, in which we can handle and direct the position of a telescope, it also allows us to control the telescope with a continuous movement to follow different stars, we can do this through an Arduino Mega.

The program communicates with Arduino through Serial and it commands the actions which Arduino must do. The applicaction has two differents methods and these are:

- The **Goto method** where we can move the telescope and change our positions on the AR axis and the DEC axis in order to reach the desired position.
- The **Follow method** is used to trace stars, planets and other space elements. In this method you can select the time and the speed for the tracing.

In the Arduino we find a CNC board, which controls two DBR8825 Drivers for the movement of the motors on the AR axis and the DEC axis.

The program also allows us to change the positioning speed of the telescope and optimize the tracking of the subject matter with greater precision. In addition to these features and characteristics, we have the option of being able to manually manage or direct the position of the telescope.

**Keywords:** Telescope, Arduino Mega, stars, computer application, Follow, Goto,AR, DEC , DVR8825.

# Tabla de contenidos

Índice de figuras ..... 9

Índice de tablas ..... 11

1 INTRODUCCIÓN ..... 12

1.1 Motivación ..... 12

1.2 Objetivos ..... 13

1.3 Estructura de la memoria ..... 14

1.4 Metodología ..... 15

2 Conceptos Básicos y Estado del Arte ..... 16

2.1 Estudio del Universo(Astronomía) ..... 16

2.1.1 Telescopio y monturas. ..... 17

2.1.2 Velocidades de seguimiento/translación. ..... 21

2.2 Estado del arte ..... 22

2.2.1 Microcontroladores más avanzados ..... 22

2.2.2 Avances de los Telescopios ..... 24

3 Arduino Y Componentes físicos ..... 26

3.1 Arduino ..... 26

3.1.1 Placa electrónica ..... 26

3.1.2 Entorno de desarrollo ARDUINO. ..... 34

3.2 Motores paso a paso. ..... 37

3.3 CNC y Drivers ..... 40

3.3.1 DRV8825 Stepper Motor Driver Carrier ..... 40

3.3.2 AZ-Delivery CNC Shield V3 Pinout ..... 44

4 NetBeans y SceneBuilder ..... 47

4.1 Netbeans ..... 47

4.1.1 JavaFX ..... 49

4.2 JavaFX Scene Builder ..... 49

5 Desarrollo del proyecto y resultado final ..... 51

5.1 Desarrollo ..... 51

5.2 Resultado final ..... 63

5.2.1 Diagramas de flujo métodos ARDUINO. ..... 68

6 Conclusiones ..... 71

6.1 Líneas futuras ..... 71

6.2 Evaluación Particular ..... 72

7 Bibliografía ..... 74

8 Anexos ..... 77

# ÍNDICE DE FIGURAS

Figura 1 Telescopio y sus elementos. Fuente [5] ..... 17
Figura 2 Esquema de espejos de un telescopio reflector. Fuente [5]. ..... 18
Figura 3 Esquema de espejos de un telescopio refractor. Fuente [5]. ..... 18
Figura 4 Esquema de espejos de un telescopio catadióptrico. Fuente [6]. ..... 19
Figura 5 Esquema movimientos montura. Fuente [8] ..... 19
Figura 6 Esquema movimientos montura. Fuente [9] ..... 20
Figura 7 Montura Skywatcher EQ5. Fuente edición propia ..... 21
Figura 8 Microchip SAMV71RT.Fuente[13] ..... 22
Figura 9 Microchip SAMRH71.Fuente[13] ..... 23
Figura 10.Nebulosa Carina Fuente [18] ..... 24
Figura 11 Telescopio James Webb. Fuente [19] ..... 25
Figura 12 Logo ARDUINO. Fuente [21] ..... 26
Figura 13 Arquitectura Harvard. Fuente [22] ..... 27
Figura 14 Microchip Atmel AVR. Fuente [23] ..... 28
Figura 15 Placa Duemilanove. Fuente [24] ..... 28
Figura 16 Placa Arduino UNO. Fuente [25] ..... 29
Figura 17 Placa MEGA. Fuente [26] ..... 30
Figura 18 Placa DUE. Fuente [27] ..... 30
Figura 19 Placa NANO. Fuente [28] ..... 31
Figura 20 Placa BLUETOOTH. Fuente [29] ..... 32
Figura 21 Placa UNO WIFI. Fuente [30] ..... 33
Figura 22 Entorno de desarrollo Arduino. Edición propia ..... 34
Figura 23 Botón verificar. Edición propia ..... 35
Figura 24.Botón subir. Edición propia ..... 35
Figura 25 Botón Nuevo. Edición propia ..... 35
Figura 26 Botón Abrir. Edición propia ..... 35
Figura 27 Botón Salvar. Edición propia ..... 36
Figura 28 Botón Monitor Serial. Edición propia ..... 36
Figura 29 Ventana del monitor Serie. Edición propia ..... 36
Figura 30 Interior de un motor paso a paso. Fuente [32] ..... 38
Figura 31 Esquema del interior de un motor paso a paso. Fuente [33] ..... 38
Figura 32 Esquema de excitación de los polos. Fuente [33] ..... 39
Figura 33 Estructura del puente H. Fuente [34] ..... 40
Figura 34 Driver DRV8825. Fuente [35] ..... 41
Figura 35 Esquema de conexiones del DRV8825. Fuente [36] ..... 42

Figura 36 Grafica de las señales haciendo microstepping. Fuente [37] ..... 43
Figura 37 Código para realizar un paso de motor. Edición propia ..... 44
Figura 38 Estructura del CNC y conexiones con los dispositivos. Fuente [38] ..... 45
Figura 39 Posiciones correspondientes de los Jumpers. Fuente [38] ..... 46
Figura 40 Código para establecer ENABLE LOW. Edición propia ..... 46
Figura 41 Entorno de desarrollo NetBeans. Edición propia ..... 48
Figura 42 Esquema de proyectos. Edición propia. ..... 48
Figura 43 Entorno de edición grafica Scene Builder. Edición propia ..... 50
Figura 44 Función Make Controller. Edición propia ..... 50
Figura 45 Conexión y comunicación Serie. Fuente [39] ..... 52
Figura 46 Código para mover motor con una velocidad y una dirección. Edición propia ..... 52
Figura 47 Para el programa y espera una entrada para continuar. Edición propia ..... 53
Figura 48 Esquema de actuación de una interrupción. Fuente [40] ..... 53
Figura 49 Primera modelo de interfaz gráfica de la configuración. Edición propia ..... 55
Figura 50 Segundo modelo de la interfaz gráfica de configuración. Edición propia ..... 56
Figura 51 Formula para obtener el delay. Edición propia ..... 57
Figura 52 Código para obtener el delay. Edición propia ..... 57
Figura 53 Primer modelo de interfaz gráfica de la operativa. Edición propia ..... 58
Figura 54 Segundo modelo de la interfaz gráfica de la operativa. Edición propia ..... 59
Figura 55 Circuito completo . Edición propia ..... 60
Figura 56 Ventana de configuración. Edición propia ..... 63
Figura 57 Pseudocódigo método GET. Edición propia. ..... 64
Figura 58 Ventana de alerta, Arduino mal puesto. Edición propia ..... 64
Figura 59 Estado inicial ventana operativa. Edición propia ..... 65
Figura 60 Botón de configuración. Edición propia. ..... 65
Figura 61 Ventana operativa con MODO MANUAL activo. Edición propia ..... 66
Figura 62 Ventana GOTO. Edición propia ..... 67
Figura 63 Ventana método FOLLOW. Edición propia. ..... 68
Figura 64 diagrama flujo de elección de modo. Edición propia. ..... 68
Figura 65 Diagrama de flujo del método GOTO. Edición propia ..... 69
Figura 66 Diagrama de flujo del método FOLLOW. EDICION PROPIA. ..... 69
Figura 67 Diagrama de flujo del método Manual. Edición propia. ..... 70

# ÍNDICE DE TABLAS

Tabla 1 Características Arduino Uno. Fuente [25] ..... 29
Tabla 2 Características Arduino MEGA. Fuente [26] ..... 30
Tabla 3 Características Arduino DUE. Fuente [27] ..... 31
Tabla 4 Características Arduino NANO. Fuente [28] ..... 31
Tabla 5 Características Arduino BT. Fuente [29] ..... 32
Tabla 6 Características Arduino UNO WIFI. Fuente [30] ..... 33
Tabla 7 Estados para poner las diferentes resoluciones. Fuente [37] ..... 43
Tabla 8 Tabla de Delay por segundo para dar una vuelta dependiendo de la resolución. Edición propia ..... 56

# 1 INTRODUCCIÓN

Durante el desarrollo de este trabajo se podrá ver el proceso de como trabajar en la plataforma de Arduino mediante una Placa Arduino Mega 2650 para conseguir el movimiento de dos Motores bipolares de paso a paso y lo haremos mediante una comunicación con el puerto serial de un ordenado. Trabajaremos con dos tipos de entornos y lenguajes de programación. Mediante los cuales podremos intercambiar información entre el Arduino y la aplicación para mover los motores que tengamos en la montura a una determinada velocidad, con una cantidad concreta de número pasos o un tiempo elegido por el usuario. Los motores estarán conectados a través de unos engranajes a los ejes de un telescopio (ecuatorial) y tendrán unos drivers para controlar la potencia y los pasos que estos darán.

## 1.1 MOTIVACIÓN

El telescopio es un instrumento científico cuya invención se le atribuye a Hans Lippershey, el primero que lo uso para la astronomía fue Galileo Galilei en 1609, sirve para la observación del Universo y ha sido fundamental para muchos descubrimientos que hoy en día conocemos sobre el Universo en el que estamos. Desde entonces se han producido una gran cantidad de avances tecnológicos por ello considere interesante el poder trabajar y desarrollar una aplicación para este tipo de instrumento, llevandome a aceptar este proyecto en detrimento de otros de un ámbito parecido o similar.

También al poder unirlo con una de las placas más populares y versátiles que hay actualmente en el mercado que es Arduino que permite una variada posibilidad de modelos y diferentes formas trabajo, además de un entorno sencillo e intuitivo.

Gracias a esto puedo aprovechar para desarrollar mis conocimientos de hardware al tener que realizar el montaje de las placas y los drivers, y el software tras implementar una aplicación del ordenador y todo el uso de la plataforma de Arduino para que tengan comunicación y se puedan mover los motores.

Como adyacente también veo una gran motivación en el hecho de poder aprender más sobre el estudio de nuestro universo y conocerlo más profundamente. Y ser capaz de usar los conocimientos adquiridos en el transcurso del grado universitario para realizar una aplicación.

## 1.2 OBJETIVOS

El objetivo principal que se va a desarrollar, como ya viene dado por el título es el diseño y la implementación de un sistema motorizado para un telescopio. Precisando más se pueden distinguir dos objetivos diferentes:

- **Posicionamiento**, proporcionar la capacidad de realizar un movimiento de los ejes del motor para llevarlos a una posición determinada por unos grados.
- **Seguimiento**, ofrecer capacidad de poder mover el telescopio en el eje AR con una velocidad fija y sin aceleración, con un tiempo proporcionado por el usuario.

Una vez especificados los objetivos podemos introducir el objetivo secundario que es el desarrollo y la implementación de una aplicación del ordenador, para obtener un uso del programa Arduino, que mueve los motores, de una mayor una simplicidad y que sea más intuitivo a la hora de usarlo. De esta manera se logra que tenga un mayor rango de usuarios ya sean expertos en este campo o no.

Este objetivo se desarrolla en los siguientes subobjetivos:

- Una configuración para que el usuario pueda meter los datos de sus motores y sus pares de una forma sencilla sin realizar muchos cálculos.
- Facultad para establecer unos grados en los ejes.
- Seleccionar la velocidad a la que se mueve el telescopio.
- Señalizar un tiempo de movimiento del telescopio.
- Un botón con la capacidad de parar el telescopio en cualquier momento.
- Proporcionar la facilidad para poder cambiar la posición del telescopio manualmente a través de la aplicación.

## 1.3 ESTRUCTURA DE LA MEMORIA

La memoria del TFG constara de los agradecimientos, un resumen y abstract al comienzo, un índice de figuras y 8 apartados.

El primer apartado es la introducción donde se explicará porque esta elección de TFG además que se introducirá al lector en algunos aspectos básicos y los objetivos a realizar.

En el Segundo apartado se verán unos conceptos básicos explicando temas sobre el telescopio y como se mueve en los ejes. Incluyendo también el estado del arte.

En el tercer apartado se nos explica las diferentes placas de Arduino y porque usamos la Mega 2650, se podrá ver la plataforma y el lenguaje en el que trabajamos. Estudiaremos los diferentes instrumentos físicos conectados

El Cuarto apartado nos muestra NetBeans que es nuestro programa donde desarrollaremos la aplicación a nivel de software y nos apoyaremos de Scene Builder para el aspecto físico de esta.

El quinto apartado nos enseña cómo se ha analizado los diferentes problemas que se han dado en el proyecto y como se han propuesto y solucionado estos. Podremos ver también el resultado final del trabajo.

El sexto apartado tiene las conclusiones donde vienen unas posibles vías futuras de aplicación de este TFG y la relación que ha tenido con los estudios cursados.

El séptimo apartado nos muestra toda la bibliografía usada durante el proyecto.

Para finalizar el octavo apartado es el anexo donde se encontrarán los códigos que se han trabajado e implementado para el Trabajo fin de Grado.

## 1.4 METODOLOGÍA

La metodología empleada durante la realización del proyecto ha sido la siguiente:

a) Instalación del entorno de trabajo Arduino y librerías de trabajo TimerOne para la ejecución y comprobación del código para mover los motores.
b) Montaje y diseño del diagrama del Arduino con la placa CNC y los drivers de los motores y conectando los motores.
c) Se estudia el comportamiento de los motores y sus drivers con las diferentes velocidades posibles y se establece la más precisa.
d) Calcular las velocidades de los diferentes astros a mirar y realización de las fórmulas para que los motores muevan el telescopio a esas velocidades.
e) Instalación de NetBeans y Scene Builder y realización de las diferentes ventanas que tendrá nuestra aplicación.
f) Instalación de las librerías PanamaHitek y Parse para la comunicación entre el entorno Arduino y el programa en NetBeans y poder hacer comprobaciones desde el.
g) Observación del funcionamiento de los motores a través de la aplicación y realización de métodos secundarios para un uso más intuitivo y simple del usuario además de más detallado y con obtención de más información.

# 2 CONCEPTOS BÁSICOS Y ESTADO DEL ARTE

En este apartado se van a desplegar unos conceptos básicos sobre el estudio del universo y el telescopio para una mejor compresión del desarrollo del TFG. Asimismo, se hablará sobre el estado del arte de los microcontroladores y los telescopios en sus avances más novedosos.

## 2.1 ESTUDIO DEL UNIVERSO(ASTRONOMÍA)

La **astronomía** es la ciencia que se encarga de estudiar los cuerpos celestes de nuestro universo, por lo que estudia sus movimientos y los fenómenos ligados a ellos. La astronomía está formada por 4 ramas [1]:

- **Astronomía de posición.** Sitúa en la esfera celeste la posición de los astros estableciendo ángulos con relación a unos planos fundamentales, empleando diferentes sistemas de coordenadas astronómicas. Asimismo, explica el movimiento de los astros en la bóveda y otros fenómenos como los eclipses o el recorrido de los planetas alrededor del sol. Son tareas fundamentales de esta rama dar el valor de la hora y la determinación de las coordenadas geográficas.
- **Mecánicas celestes**. Observa el movimiento de los astros bajo los efectos gravitatorios ejercidos por otro cuerpo celeste masivo.
- **Astrofísica**. Analiza la composición, estructura y evolución de los astros y comprender los fenómenos que suceden en el espacio.
- **Cosmología**. Se centra en los orígenes, estructura y evolución del universo. Intenta entender la naturaleza a gran escala del mundo material que nos rodea, su evolución y su destino, acatando el método de las ciencias naturales.

La astronomía ha estado en una continua evolución desde la antigüedad, pero hasta mediado del siglo XVII no comenzó a desarrollarse el campo de la metodología científica, donde uno de los factores claves fue la invención del telescopio que permitió la observación del cielo de una forma más detallado, gracias a esto se han conseguido muchos descubrimientos de nuestro universo.

### 2.1.1 Telescopio y monturas.

Es un instrumento óptico que nos permite capturar luz (radiación electromagnética) de tal forma que sea factible estudiar objetos que son de un brillo muy bajo o que se encuentra a grandes distancias. La cantidad de luz que si puede capturar dependerá fundamentalmente del diámetro que tenga el objetivo o apertura del telescopio. Cuanto mayor diámetro tengamos mayor será la cantidad de luz que nos entrará proporcionándonos una imagen más clara y detallada. Están constituidos esencialmente de dos partes: el **tubo** que está formado por el **objetivo** (orientado al cielo) y el **ocular** y la **montura** que sirve para sostener el tubo y nos hace posible el movimiento del telescopio de una forma suave y precisa para orientarlo. En la Figura 1 se puede observar los diferentes elementos que compononen el telescopio.


*Figura 1 Telescopio y sus elementos. Fuente [5]*

Se pueden encontrar tres grupos de telescopios diferentes que poseen diseños ópticos particulares:

- **Reflectores** (Figura 2). Se componen por un espejo cóncavo localizado en extremo inferior del tubo, que canaliza la luz y la orienta hacia un espejo secundario plano que es de un tamaño más pequeño y se encuentra en la boca del tubo, donde rebota la luz 45º y la manda hacia el ocular.

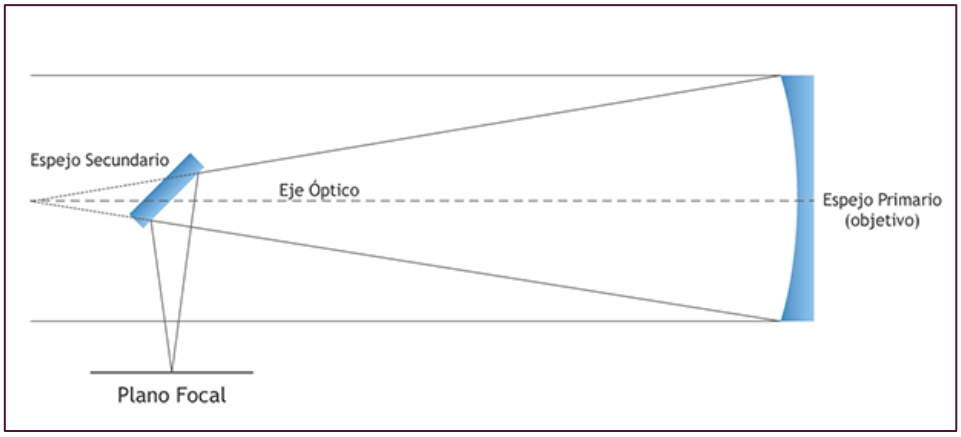


*Figura 2 Esquema de espejos de un telescopio reflector. Fuente [5].*

- **Refractores** (Figura 3). Están compuestos por una agrupación de lentes que recogen la luz concentrándola en el foco donde tendremos el ocular.

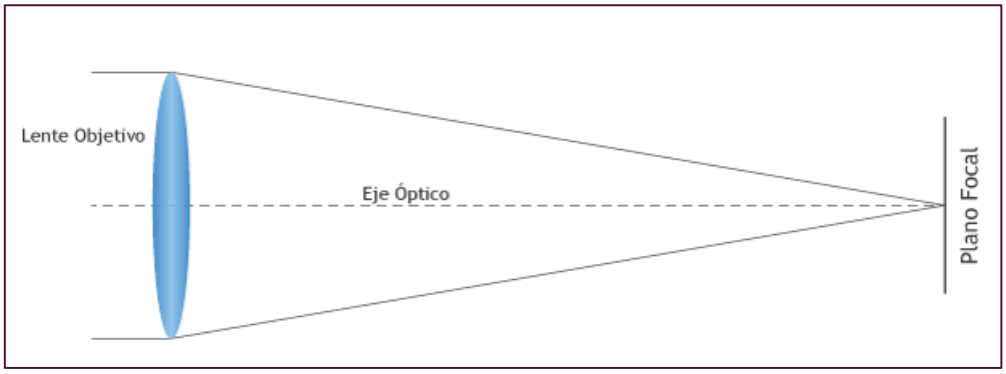


*Figura 3 Esquema de espejos de un telescopio refractor. Fuente [5].*

- **Catadióptricos** (Figura 4). Es la combinación de los dos grupos anteriores, usa tanto lentes como espejos. Podemos observar que el objetivo es un espejo cóncavo, pero en la apertura tiene una lente correctora que mantiene un espejo secundario, aquí se capta la luz para orientarla a un hueco en centro del espejo principal al final del tubo. En la Figura 4 se puede ver la distribución tanto de las lentes como de los espejos.

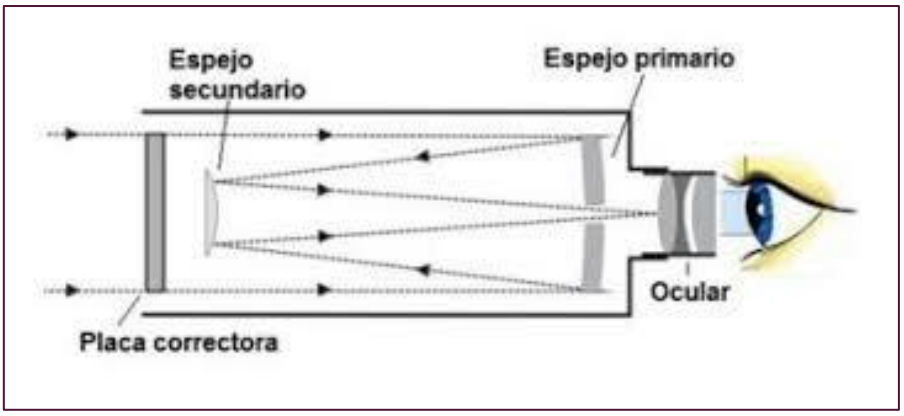


*Figura 4 Esquema de espejos de un telescopio catadióptrico. Fuente [6].*

En todos los casos el objetivo (ya sea lente o espejo) concentrará la luz capturada en el plano focal del instrumento, lugar donde se ubican los oculares (piezas ópticas intercambiables que tienen como función proveer de aumento al telescopio).

Como ya hemos dicho la montura es otra de las partes fundamentales del telescopio ya que no solo se encarga de sostener el telescopio si no que aporta un movimiento de los objetos observados compensando el movimiento de rotación de la tierra. Existen varios tipos de monturas [8] desde las más simples e intuitivas a otras complejas. Hay dos tipos fundamentales:

- **Monturas altacimutales o azimutales**. Son las monturas más simples e intuitivas. El movimiento es ejercido sobre los ejes de azimut(horizontal) y de altitud(vertical). Normalmente son monturas robustas y que se pueden computarizar fácilmente. Podemos observar sus movimientos en la Figura 5.

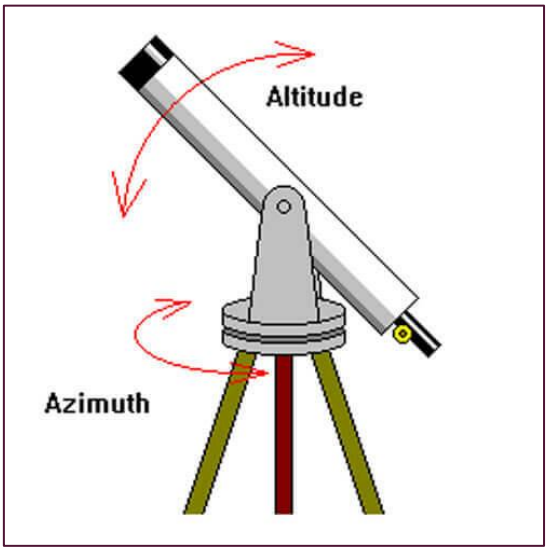


*Figura 5 Esquema movimientos montura. Fuente [8]*

- **Monturas ecuatoriales**. En esta montura se trabaja con dos ejes:
    - **AR (Ascensión Recta).** Se dispone con el paralelo al eje de rotación terrestre (eje polar).
    - **DEC (Declinación).** Compensa el movimiento de rotación de nuestro planeta.

Este tipo de monturas utilizan una proyección de los polos norte y sur en el cielo, el polo norte celeste (cercano a la estrella polar) y el polo sur celeste. Son difíciles de manejar y son poco intuitivas. Cuando haces un seguimiento se realiza moviendo solamente el eje polar. En la Figura 6 se pueden observar los movimientos en los ejes del telescopio.


*Figura 6 Esquema movimientos montura. Fuente [9]*

Este es el tipo de montura con el que trabajamos en nuestro proyecto ya que pueden ser motorizadas. Nuestro modelo es Skywatcher EQ5

#### 2.1.1.1 modelo Skywatcher EQ5

Ofrece una base sólida para la mayoría de telescopio que se necesiten. Puede aguantar una carga máxima de 9 kg, puede estar motorizada, no tiene un método de alineación. Tiene disponible un alojamiento para buscar la estrella polar. En la Figura 7 podemos ver una imagen de esta montura.

*Figura 7. Montura Skywatcher EQ5. Fuente edición propia*

### 2.1.2 Velocidades de seguimiento/translación.

Si vamos a observar el espacio una de las cosas que debemos conocer es que dependiendo del cuerpo celeste que vayamos a estudiar vamos a necesitar una velocidad u otra. Podemos diferenciar tres tiempos de rotación diferentes que son sideral, lunar y solar.

- **Velocidad o tiempo sideral**, es una graduación de tiempo que tiene como base la velocidad de rotación terrestre en relación con las estrellas fijas. Un día sideral es de aproximadamente 23 horas 56 minutos y 4,09 segundos por lo que nuestra velocidad sideral será de 15,041 arco segundos por segundo.
- **Velocidad o tiempo lunar**, es una proporción de tiempo con fundamento en la velocidad de rotación de la tierra respecto a la luna. La velocidad es de 14,685 arco segundos.
- **Velocidad o tiempo solar**, es la graduación de tiempo que tenemos en nuestro día a día y es la base de nuestro horario. Tiene como base la velocidad de rotación de la tierra respecto al sol. Un día solar es de 24 horas y nuestra velocidad sideral es de 15 arco segundos.

## 2.2 ESTADO DEL ARTE

En este capítulo se realiza el estado del arte de dos elementos fundamentales implicados en el desarrollo del proyecto. Estos son los microcontroladores y los avances que hay en los telescopios.

### 2.2.1 Microcontroladores más avanzados.

Actualmente, cada vez más dispositivos hacen uso de microcontroladores (MCU), y su desarrollo ha crecido de manera exponencial. Desde su primera aparición ha cambiado nuestra forma de vida si apenas ser conscientes y nos ha facilitado muchas tareas complejas y costosas. Podemos encontrar muchos grupos de investigación sobre los microcontroladores ya que hoy en día nos encontramos rodeados de estos de tal forma que no nos percatamos que se encuentran presentes.

Una de las mejores características de los microcontroladores es su bajo coste esta es una de las causas por la que su uso es tan habitual y se están desarrollando para aplicaciones más específicas y difíciles como las espaciales. Hace poco Microchip anuncio los primeros MCU basados en Arm para el sector espacial ofreciendo diferentes niveles de prestaciones frente a la radiación [12]. EL MCU **SAMV71Q21RT (Figura 8)** que es tolerante a la radicación y el **SAMRH71**(Figura 9) resistente a la radiación, poseen anexo el sistema chip (System on Chip, SoC) Arm CortexM7.

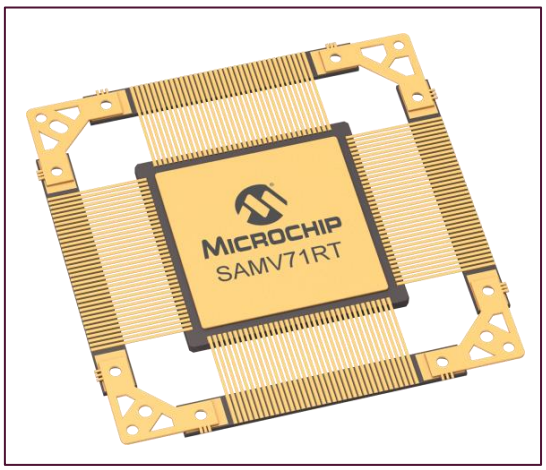


*Figura 8 Microchip SAMV71RT.Fuente[13]*

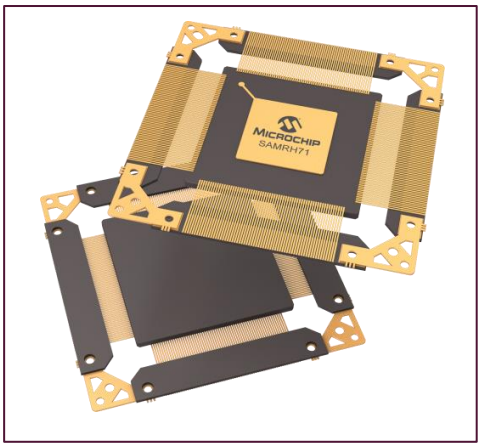


*Figura 9 Microchip SAMRH71.Fuente[13]*

El SAMV71Q21RT y el SAMRH71 dan la posibilidad a los desarrolladores de software para que comiencen antes su trabajo sin tener que recurrir a componentes de grado espacial reduciendo significadamente el tiempo y el coste de desarrollo.

Las prestaciones del SAMV71Q21RT se diseñan para usarlas en unas nuevas aplicaciones espaciales como constelaciones de satélites de orbita baja (Low Earth Orbit, LEO) y en la robótica. Mientras el SAMRH71 esta adaptado a subsistemas mucho más complicados como los giroscopios y equipos de rastreo de estrellas, está diseñado para aplicaciones en el espacio lejano. Ambos son inmunes al bloqueo por evento único hasta 62 MeV.cm^2 /mg, tiene altas prestaciones y bajos consumo proporcionando una vida operativa para aplicaciones aeroespaciales con un tiempo duradero.

Muchas de las aplicaciones de los microcontroladores actuales o que se prevén lo sean un futuro cercano son:

1. Sistemas de comunicación: En los teléfonos móviles, fax, grandes automatismos como centrales...
2. Electrodomésticos: hornos, microondas, equipos de música, mandos a distancia
3. Industria informática: lo podemos encontrar en los periféricos como ratones, teclados, escáner...
4. Automoción para los ABS, piloto automático, seguridad, climatización…
5. Sistemas de supervisión, vigilancia y alarmas.
6. En aplicaciones espaciales.
7. Automatización de robots, que tengan más movimientos y capacidad de control.
8. Electromedicina, para equipos de diagnóstico y tratamiento, más potente y fiable.

### 2.2.2 Avances de los Telescopios

En la actualidad el universo continúa siendo uno de los enigmas más grandes que nos quedan por descubrir. Por ello a lo largo del tiempo uno de los instrumentos con el que hemos obtenidos más hallazgos en este ámbito es el telescopio. El telescopio espacial más potente en funcionamiento es el **Hubble** que fue lanzado en 1990 y ha cambiado todos los conocimientos fundamentales que sabíamos del universo. Como menciona la NASA en [16] "Hubble's launch and deployment in April 1990 marked the most significant advance in astronomy since Galileo's telescope". En la Figura 10 se muestra una imagen tomada por el Hubble de la nebulosa de Carina.

*Figura 10.Nebulosa Carina Fuente [18]*

Con la llegada de las nuevas tecnologías y de todos los avances tecnológicos que se han desarrollado en la tierra desde que el Hubble fuera enviado al espacio ha permitido a la NASA mejorar la tecnología que hasta ese momento utilizaban. Por lo que les ha sido posible construir un telescopio espacial nuevo con tecnología de vanguardia que es el **James Webb** (Figura 11) que dispone de un espejo principal constituido por 18 segmentos con un área de 25 m^2, que es un área cinco veces superior a la del Hubble. Además, de contar con reflectores para proteger el telescopio frente a la radiación, este aún se encuentra en la tierra, la NASA tiene previsto su lanzamiento al espacio en 2021.

*Figura 11 Telescopio James Webb. Fuente [19]*

La astronomía es una de las pocas ciencias en la que actualmente un aficionado puede realizar un descubrimiento, esto hace necesario el desarrollo de telescopios para un uso más común. Por ello se requiere de mejoras en los elementos básicos como las monturas o en el tubo del telescopio que simplifiquen y agilicen su utilización.

Las monturas motorizadas y automatizadas son unos de los desarrollos más importantes dentro de este ámbito ya que nos permite una mayor precisión y comodidad al usar un telescopio. Para poder conseguir, por ejemplo, un seguimiento de una estrella o la observación de una coordenada en cielo nocturno durante varias horas. Por ello los microcontroladores desempeña un papel muy importante, permitiendo una automatización del telescopio de una manera más sencilla y con un coste muy inferior. Permitiendo a mucha más gente la posibilidad de disfrutar estudiando el cielo nocturno de una forma más nítida y precisa.

Esto ha permitido a un aficionado el poder conseguir un hallazgo como sucedió en Santa Cruz de Tenerife, 6 mayo (EFE) donde el astrónomo aficionado josé J. Chambó y el administrador del Observatorio del Teide, Miquel Serra-Ricart descubrieron que el núcleo del cometa C/2019 Y4 había comenzado a fragmentarse.[20]

# 3 ARDUINO Y COMPONENTES FÍSICOS

Este apartado se centra en explicar el hardware. Mas en concreto conoceremos los tipos de microcontroladores que vamos a usar o podríamos, la plataforma con la que le daremos las ordenes, los diferentes elementos físicos que necesitamos para un funcionamiento más preciso, los motores de los cuales dispondremos para mover el telescopio.

## 3.1 ARDUINO


*Figura 12. Logo ARDUINO. Fuente [21]*

**Arduino** (Figura 12) surgió en el Instituto de Diseño Interactivo de Ivrea(Italia) en el año 2005 y es una plataforma de desarrollo establecida en una placa electrónica de hardware libre que tiene integrado un microcontrolador reprogramable y unos pines hembra, permitiendo establecer conexiones entre el microcontrolador y diferentes periféricos de entrada y salida de una manera sencilla, para desarrollar esta comunicación, la plataforma también está constituida por un entorno de desarrollo (IDE) para la programación.

### 3.1.1 Placa electrónica

Una placa electrónica es una **PCB ("Printed Circuit Board") o placa de Circuito Impreso**. Es un soporte físico, donde se instalan componentes electrónicos que se interconectan entre ellos. Tiene una superficie de un material no conductor, que está constituida por caminos, buses o pistas de un material conductor, para poder conectar los componentes electrónicos y que se puedan enviar información. Podemos destacar el microcontrolador que es el componente que programamos, para que mande las ordenes que deben realizar los diferentes dispositivos.

Cuando hablamos de los PCB de "Arduino" debemos saber que hay muchos modelos de placas, cada uno tiene un propósito diferente y unas características apropiadas para ese uso. Pero normalmente todas tienen en común que son de la misma familia, ya que usan microcontroladores de la misma gama que es AVR marca Atmel. Lo que significa que comparten la mayoría de sus características de software, como arquitectura, librerías y documentación.

#### *3.1.1.1* Microcontrolador

Es una estructura de dimensiones reducidas, elaborada de un material conductor, normalmente silicio, que contiene un circuito integrado programable y donde podemos encontrar una **unidad central de procesamiento (CPU)**, **unidades de memoria (RAM y ROM)**, **puertos de entrada y salida y periféricos**. Todas estas partes se encuentran interconectadas en su interior y en conjunto se le conoce como microcomputadora. Se puede afirmar que un microcontrolador es como una microcomputadora pero que esta encapsulada en un circuito integrado.

El **microcontrolador (MCU, µC o UC)** necesita de un programa para poder realizar acciones específicas, normalmente se guarda en la memoria ROM. Es un sistema cerrado ya que todas sus partes, se encuentran en su interior, en el exterior solo están las líneas que gobiernan los periféricos.

Actualmente la arquitectura interna que más usan los microcontroladores es **Harvard**, aunque algunos aun dispongan de una arquitectura von Neumann. La Arquitectura Harvard se caracteriza por disponer de dos memorias independientes, donde una contiene las instrucciones y la otra los datos como se puede determinar en la Figura 13.

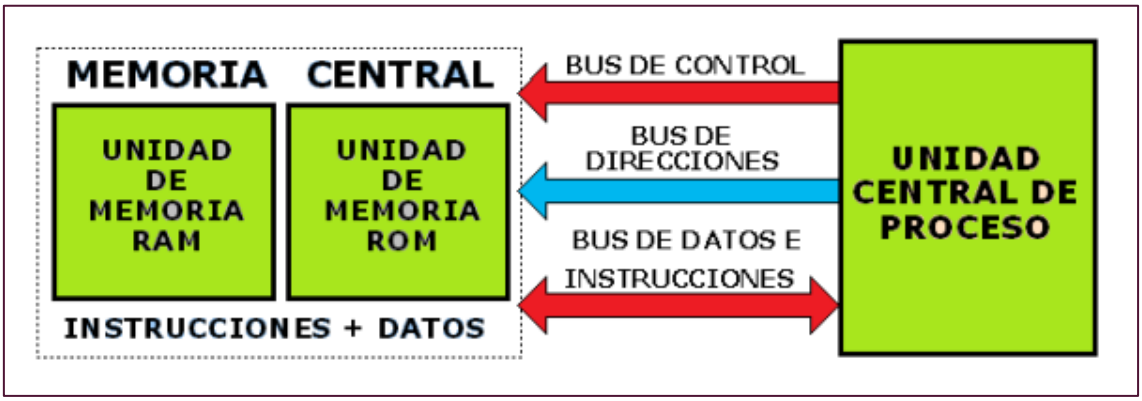


*Figura 13. Arquitectura Harvard. Fuente [22]*

Normalmente las placas Arduino llevan un microcontrolador AVR marca Atmel (Figura 14), En general, son procesadores 8-bit RISC con arquitectura Harvard.


*Figura 14 Microchip Atmel AVR. Fuente [23]*

#### *3.1.1.2 Modelos* de *placas*

Arduino consta de muchos tipos de placas electrónicas diferentes en este punto se van a explicar los principales modelos, con la placa que hemos trabajado y algunas de las variantes con las que podríamos a ver realizado el proyecto.

- **"Duemilanove"** (Figura 15). Se podría decir que es la primera versión de la placa básica de Arduino. Es capaz de seleccionar una fuente de alimentación idónea, USB o fuente externa de manera automática eliminando así la necesidad de utilizar un jumper a modo de conmutador para seleccionar, tal y como sucedía en las versiones de las placas precedentes.

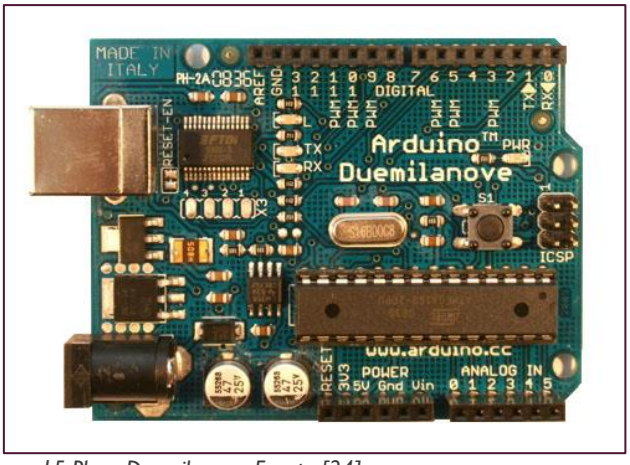


*Figura 15 Placa Duemilanove. Fuente [24]*

- **"UNO"** (Figura 16). Es la evolución de la Duemilanove. Posiblemente sea la placa más popular que hay en el mercado. Basada en el microcontrolador ATmega328. Cuenta con 14 entradas/salidas digitales, de las cuales 6 pueden usarse como salidas PWM (Modulación por de pulsos) y otras 6 entradas analógicas. Se conecta al ordenador mediante un cable USB estándar. Se encuentran en ellas elementos como un resonador cerámico de 16 MHZ, un conector para la alimentación, una cabecera ICSP y un botón de reseteado.

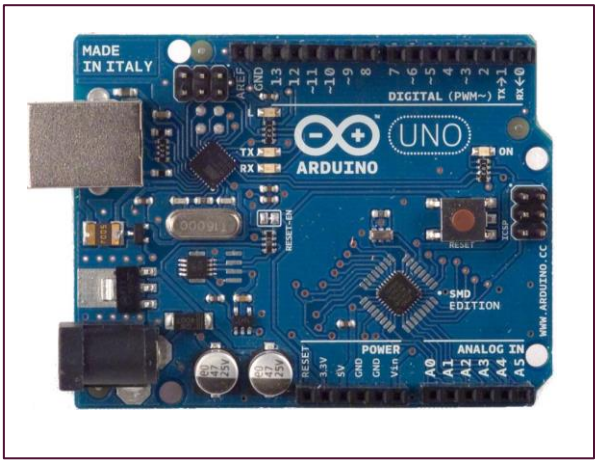


*Figura 16. Placa Arduino UNO. Fuente [25]*

Sus características principales son (Tabla 1):

| | |
|---|---|
| Microcontrolador | ATmega328 |
| Voltaje de operación | 5V |
| Voltaje de entrada recomendado | 7-12V |
| Pines digitales E/S | 14 (6 disponen de salida PWM) |
| Pines de entrada analógica | 6 |
| Consumo por pin E/S | 40mA |
| Consumo del pin 3.3V | 50mA |
| Memoria flash | 32kB (0.5kB empleados por el bootloader) |
| SRAM | 2kB |
| EEPROM | 1kB |
| Frecuencia de reloj | 16MHz |

*Tabla 1 . Características Arduino Uno. Fuente [25]*

- **"MEGA"** (Figura 17). Es probablemente la placa con mayores prestaciones dentro de la familia Arduino. Su microprocesador es el ATmega2560 cuenta con 54 pines digitales de entrada/salida y 14 se pueden utilizar como salidas PWM, 16 entradas analógicas, 4 UARTs (puertos serie de hardware), un oscilador de 16 MHZ, un conector de alimentación, un conector ICSP y un botón de reseteo. Es una de las placa más grande y potente de Arduino. Tiene una memoria con capacidad superior a la UNO. Es compatible con la mayoría de los shields diseñados para Arduino UNO y Duemilanove y con la mayoría de las librerías.

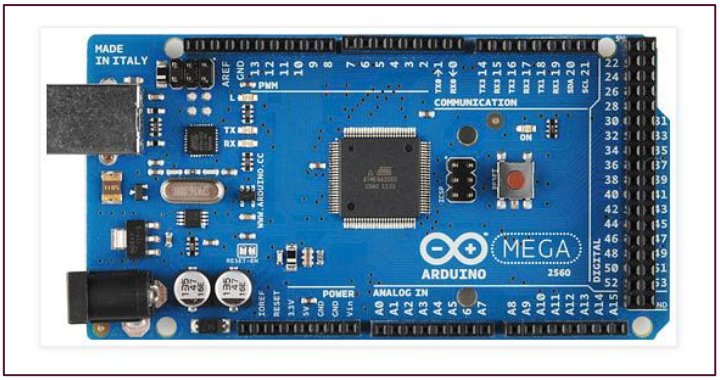


*Figura 17. Placa MEGA. Fuente [26]*

Las principales características son (Tabla 2):

| | |
|---|---|
| Microcontrolador | ATmega2560 |
| Voltaje de operación | 5V |
| Voltaje de entrada recomendado | 7-12V |
| Voltaje de entrada (límites) | 6-20V |
| Consumo por pin E/S | 40mA |
| Consumo del pin 3.3V | 50mA |
| EEPROM | 4kB |
| SRAM | 8kB |
| Memoria flash | 256kB (8kB empleados por el bootloader) |
| Frecuencia de reloj 16MHz | 16MHz |
| Pines digitales E/S | 54 (14 disponen de salida PWM) |
| Pines de entrada analógica | 16 |

*Tabla 2 Características Arduino MEGA. Fuente [26]*

- **"DUE"** (Figura 18). Una de las características más importante de esta placa es que no comparte familia de microcontrolador ya que es un Atmel SAM3X8E ARM Cortex-M3 de 32 bits. Este chip aporta una potencia de cálculo bastante superior a las demás placas. Y la otra es la capacidad que tiene de SRAM ya que también es superior a las demás e incorpora un controlador DMA para el acceso directo a la memoria intensificando el acceso a memoria que hace la CPU.

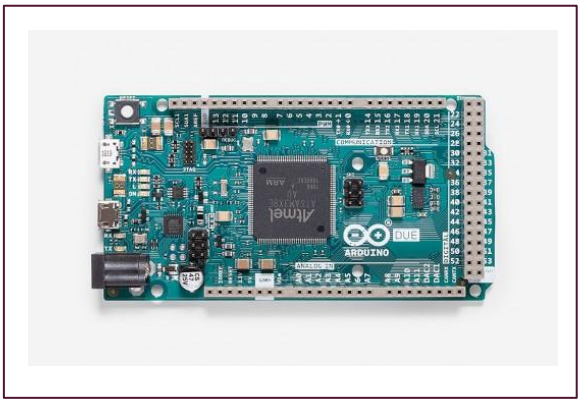


*Figura 18 Placa DUE. Fuente [27]*

Las principales características son (Tabla 3):

| Microcontrolador | Atmel SAM3X8E ARM Cortex-M3 |
|---|---|
| Voltaje de operación | 3.3V |
| Voltaje de entrada recomendado | 7-12V |
| Voltaje de entrada (límites) | 6-20V |
| Consumo por pin E/S | 130-800mA |
| SRAM | 96KB (dos bancos: 64 KB y 32KB) |
| Memoria flash | 512KB |
| Frecuencia de reloj 16MHz | 84MHz |
| Pines digitales E/S | 54 (12 disponen de salida PWM) |
| Pines de entrada analógica | 12 |

*Tabla 3 Características Arduino DUE. Fuente [27]*

- **"Nano"** (Figura 19). Su principal característica es que su tamaño ya que es muy reducido, esta versión fue diseñada y producida por la compañía Gravitech. Está pensada para aplicaciones de un bajo coste y donde importe mucho el tamaño. Tiene un microcontrolador ATmega328, se comporta como una placa UNO, tiene 22 pines digitales y 8 analógicos. Necesita de un cable mini USB y no posee conector de alimentación externa.

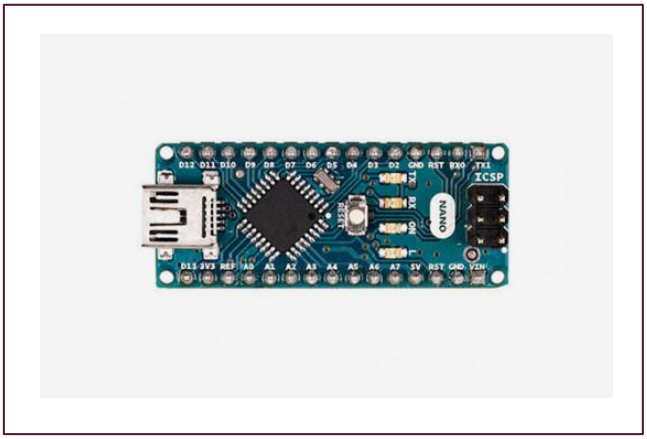


*Figura 19 Placa NANO. Fuente [28]*

Las principales características son (Tabla 4):

| Microcontrolador | ATmega328 |
|---|---|
| Voltaje de operación | 5V |
| Voltaje de entrada recomendado | 7-12V |
| Consumo por pin E/S | 40mA |
| SRAM | 2kB |
| Memoria flash | 32kB (2kB empleados por el bootloader) |
| Frecuencia de reloj 16MHz | 16MHz |
| Pines digitales E/S | 22 (6 disponen de salida PWM) |
| Pines de entrada analógica | 8 |

*Tabla 4 Características Arduino NANO. Fuente [28]*

- **“Bluetooth (BT)”** (Figura 20). Tiene unas propiedades muy parecidas a la placa UNO, pero con una ventaja al tener incorporado un módulo Bluetooth permitiendo una comunicación inalámbrica sin necesidad de comprar un shield independiente. El módulo incorporado es Bluegiga WT11. Tiene un microcontrolador ATmega328.

  Esta fue una de las opciones que barajamos para el desarrollo hardware y con las que se puede trabajar ya que nos aportaría una conexión inalámbrica. Pero decidimos usar la placa MEGA por las prestaciones ofrecidas y que podíamos instalar un módulo bluetooth manualmente.

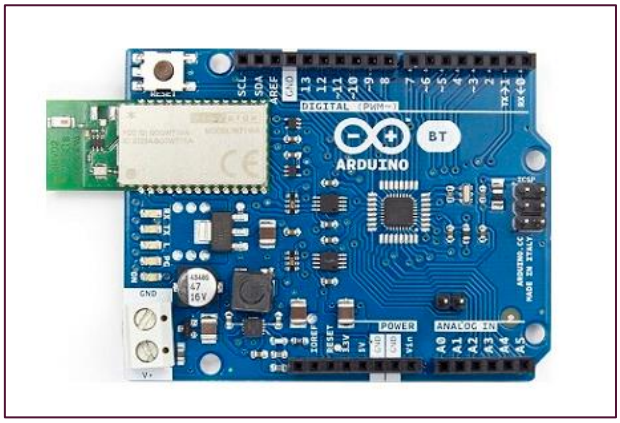


*Figura 20. Placa BLUETOOTH. Fuente [29]*

Las principales características son (Tabla 5):

| | |
|---|---|
| Microcontrolador | ATmega328 |
| Voltaje de operación | 5V |
| Voltaje de entrada recomendado | 2.5-12V |
| Módulo Bluetooth | Bluegiga WT11 |
| Consumo por pin E/S | 40mA |
| EEPROM | 1kB |
| SRAM | 2KB |
| Memoria flash | 32KB (2kB empleados por el bootloader) |
| Frecuencia de reloj 16MHz | 16MHz |
| Pines digitales E/S | 14 (6 disponen de salida PWM) |
| Pines de entrada analógica | 6 |

*Tabla 5 Características Arduino BT. Fuente [29]*

- “**UNO WIFI**” (Figura 21). Esta placa es funcionalmente igual a las placas Arduino UNO, pero lleva un módulo Wifi integrado, esto nos aporta una conexión a internet, contiene un sensor IMU (Unidad de medición Inercial, que en inglés es Ineartial Measurement Unir). Tiene un microcontrolador ATmega4809 de 8 bits, incorpora 6KB de SRAM y 256 bytes de EEPROM. El módulo SoC de comunicaciones inalámbricas que tiene incorporado lleva una pila de protocolo TCP/IP para acceder a redes Wifi y pudiendo usarse como un

punto de acceso (AP). La conexión está protegida por un acelerador de criptoprocesador ATECC608A.

Esta es otra de las placas con la que se podría realizar el proyecto. Nos aportaría una conexión inalámbrica sencilla y segura desde cualquier dispositivo.

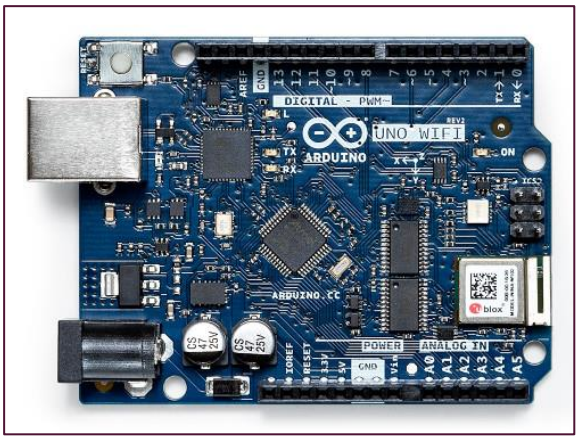


*Figura 21Placa UNO WIFI. Fuente [30]*

Las principales características son (Tabla 6):

| | |
|---|---|
| Microcontrolador | ATmega4809 de 8 bits |
| Voltaje de operación | 5V |
| Voltaje de entrada recomendado | 7-12V |
| Voltaje de entrada (límites) | 6-20V |
| Consumo por pin E/S | 20mA |
| Consumo del pin 3.3V | 50mA |
| EEPROM | 256 bytes (256 KB en ATmega4809) |
| SRAM | 6,144 bytes (6 KB en ATmega4809) |
| Memoria flash | 48 KB |
| Frecuencia de reloj 16MHz | 16MHz |
| Pines digitales E/S | 14 (5 disponen de salida PWM) |
| Pines de entrada analógica | 6 |
| IMU | LSM6DS3T3 |

*Tabla 6 Características Arduino UNO WIFI. Fuente [30]*

En el proyecto se decidió el uso de la placa **MEGA** con el procesador **ATmega2560** para el desarrollo del trabajo. Porque nos aporta de gran cantidad de pines de entrada y salida. Es una de las placas más potentes y con una de las memorias de mayor capacidad. Y esto nos ofrece la posibilidad poner en ella una mayor carga de código y permite usar más dispositivos para un trabajo más preciso y correcto. Uno de los problemas es enviarle el código y que se conecte al ordenador, para que funcione debes instalarte unos drivers específicos de la placa, otro problema como ya se ha descrito anteriormente en alguna de las placas explicadas, es que la placa MEGA debe estar conectada al ordenador siempre a través del serial, a no ser que se le instale un driver de bluetooth o un dispositivo similar que consiga una comunicación inalámbrica.

### 3.1.2 Entorno de desarrollo ARDUINO.

EL entorno de desarrollo Arduino (IDE) (Figura 22) se encarga de gestionar la conectividad entre el hardware de Arduino y el PC para establecer la comunicación entre ellos. Esta conexión se consigue cargando programas en la placa. La IDE está compuesta por:

- Un **editor de texto**. Lugar en el que se escribe el código del programa.

- Una **barra de herramientas**. Para acceder a una serie de menús y botones con acceso directo a las funciones principales del Arduino.

- Un **área de mensajes**. Se nos muestra en todo momento, el proceso de la ejecución, los errores del código, problemas de comunicación, finalización de carga, etc.

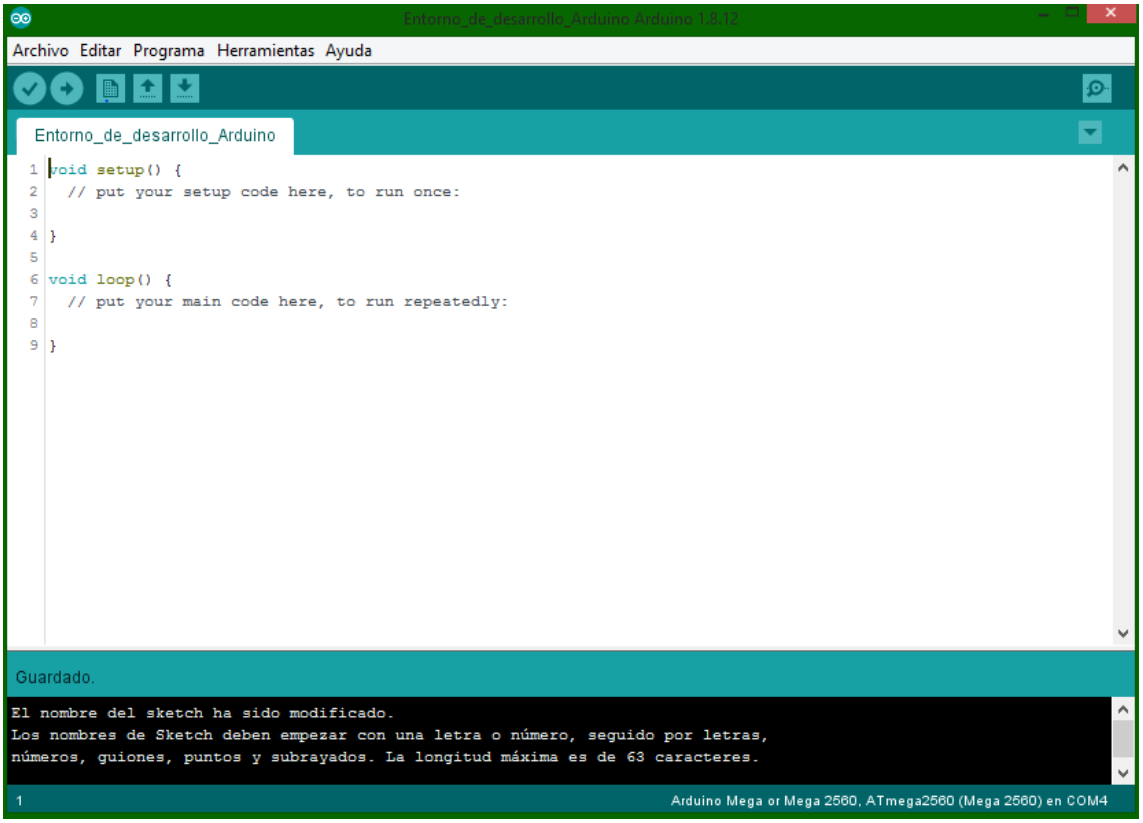


*Figura 22.Entorno de desarrollo Arduino. Edición propia*

En la IDE de Arduino, escribiremos el código que en la plataforma se llama sketch. Los sketches son guardados con la extensión **".ino".** El área de mensaje nos dice el estado de nuestro código cuando realizamos diferentes acciones no solo cuando compilamos, sino que mientras guardamos o exportamos también muestra los errores.

El lenguaje que utiliza el entorno está basado en C, pero también es posible usar comandos estándar de C++. Normalmente el editor de texto está dividido en tres partes: funciones, valores o variables y la estructura del código donde siempre podremos encontrar dos métodos setup () y loop ().

- **Setup**. Se ejecuta al comienzo del programa y normalmente se utiliza para sincronizar el serial a través del método Serial.begin(Un valor) y para establecer el funcionamiento de cada pin que esté conectado.
- **Loop**. En este método introducimos la parte del código que queremos que se esté realizando continuamente.

Como se muestra en la Figura 22 tenemos unos botones para realizar acciones rápidamente, estos botones son:

- **Verificar** (Figura 23). Compila el sketch y busca errores que tengamos dentro. Si se encuentra algún error nos saldrá en el área de mensaje y nos dice en que línea se encuentra el error y de que tipo.

*Figura 23 Botón verificar. Edición propia*

- **Subir** (Figura 24). Compila el sketch y lo carga en la placa que tengas conectada.

*Figura 24.Botón subir. Edición propia*

- **Nuevo** (Figura 25). Crea un sketch nuevo.

*Figura 25. Botón Nuevo. Edición propia*

- **Abrir** (Figura 26). Presenta un menú con los Sketch con los que has trabajado y también los sketches de ejemplo de la propia plataforma.

*Figura 26 Botón Abrir. Edición propia*

- **Salvar** (Figura 27). Guarda el código que tienes abierto.

*Figura 27 Botón Salvar. Edición propia*

- **Monitor serial** (Figura 28). Cuando pinchamos sobre este icono Arduino se nos abre una nueva ventana (Figura 29) que muestra la comunicación entre el **puerto serial** del Arduino y el PC. En esta pestaña recibimos la información que viene desde la placa y tenemos una barra de escritura en la parte superior desde donde se pueden enviar datos, por ejemplo, si el código tuviera diferentes modos dependiendo del lugar o el momento, o para enviar el tipo de paso que debe dar un motor... El **serial** nos permite la opción de tener autoscroll y mostrar las marcas temporales. También hay un despegable dándonos la opción de enviar algunos caracteres con el mensaje de salto de línea, retorno de carro, ambos o ninguno. Dentro del serial disponemos de la opción de cambiar la velocidad de **baudios (Baudatre)**, son las unidades de señal que se trasmiten por segundo. Debe estar sincronizada con la del código para que funcione correctamente, sino la información del serial será ilegible o poco acorde a los resultados esperados. La parte superior de la ventana muestra en que puerto está conectada la placa.

*Figura 28 Botón Monitor Serial. Edición propia*

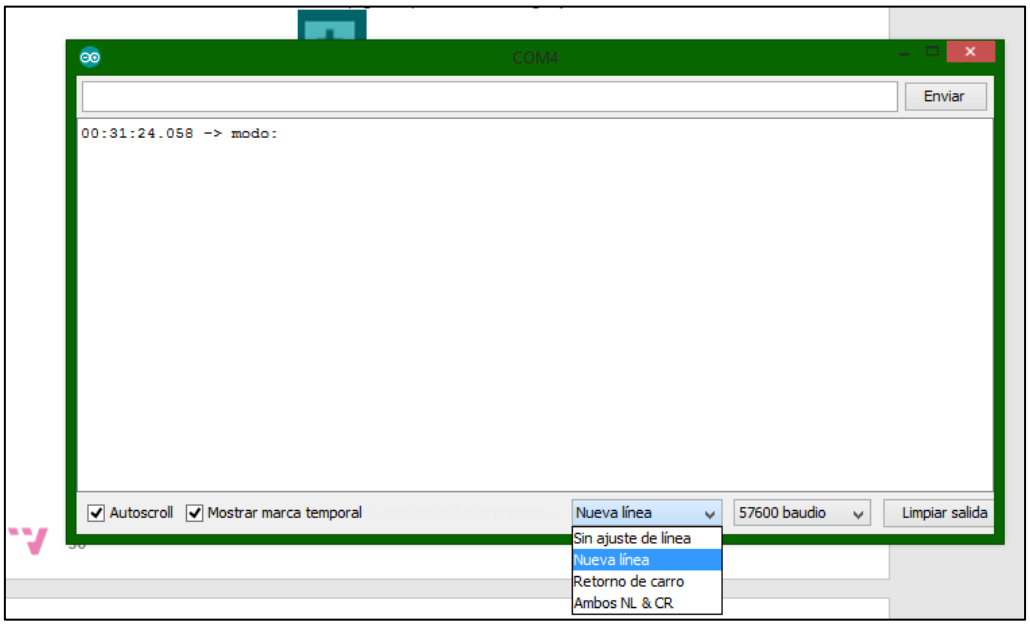


*Figura 29 Ventana del monitor Serie. Edición propia*

La barra de herramientas de la plataforma nos permite la opción de acceder a muchos menús como la configuración de la página o las preferencias de escritura. Pero se debe destacar la existencia de las librerías, ya que nos aportan muchas propiedades adicionales, como métodos para la utilización más correcta de los componentes hardware. Para que estas funcionen deberemos declararlas en el código mediante la sentencia tipo **"#include"**. Antes debemos añadir la librería a través de la opción de programa en la barra de herramientas. El único problema es que estas librerías también, ocupan un espacio dentro de la memoria del Arduino al usarlas.

## 3.2 MOTORES PASO A PASO.

En este punto pasamos a explicar el comportamiento y características de los motores que se han usado durante el proyecto.

Los **motores paso a paso**, **motor de pasos o stepper´s** en inglés es un dispositivo electromecánico que transforma los impulsos eléctricos en desplazamientos angulares discretos, lo que significa que tiene la habilidad de moverse un numero de grados dependiendo de las entradas de control que se le otorguen, estas pueden ser un paso, medio paso y hasta un treintaidosavo paso. La magnitud o resolución de los pasos que ejecuta dependerá de sus características constructiva pueden ser 1º hasta 15, o más, según el modelo.

Usualmente las bobinas del motor se encuentran en el estator y el rotor, dependiendo del caso, es un imán permanente o un motor de reluctancia variable, es un bloque de material magnéticamente blando.

Deben manejarse externamente por el controlador del motor, habitualmente se diseñan con el propósito de hacerlos rotar hacia un sentido u otro. La mayoría de estos motores pueden ser manejados de tal forma que se paren en posiciones específicas, esto es gracias al torque de detención o par de detención.

Las características comunes de los motores paso a paso son:

a) **Voltaje**. Los motores de pasos funcionan mediante una tensión eléctrica de trabajo. Este valor podemos encontrarlo en su carcasa o en su hoja de datos.

b) **Resistencia eléctrica**. Determina la corriente que consume el motor, y su valor se verá reflejado en la curva de torque del motor y su velocidad máxima.

c) **Resolución**. Como se ha mencionado antes, es uno de los factores más importantes, define la cantidad de grados por paso que rota el eje. Se puede cambiar dependiendo la aplicación deseada, por ejemplo, para una mayor precisión se harían pasos más pequeños.

Dentro de los motores paso a paso encontramos 3 tipos: de reluctancia variable, de rotor de imán permanente y los híbridos. En nuestro proyecto hemos trabajado con motores del modelo hibrido, que se apoya por la combinación de bobinas y un imán permanente para crear el circuito magnético. Situando el imán en el rotor y las bobinas en los polos del estator. El estator normalmente es dentado. Se pueden observar de forma más precisa la constitución del motor en la Figura 30.

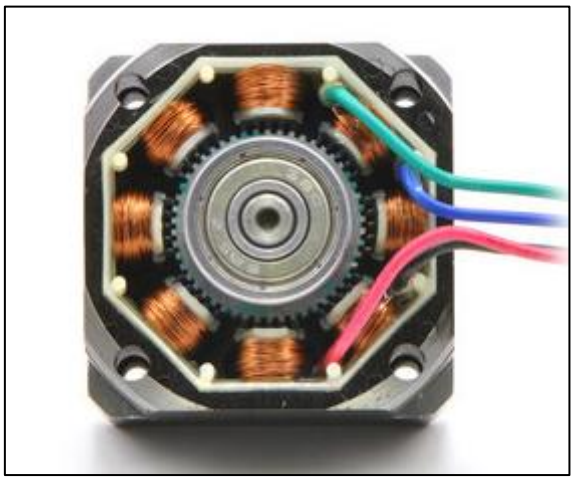

*Figura 30 Interior de un motor paso a paso. Fuente [32]*

Generalmente el estator es de ocho polos que están envueltos en unas bobinas que sirven para habilitar el paso del flujo por los distintos polos, Figura 31. Las bobinas se alimentan mediante dos fases de forma que en cada una de estas fases se alimentaran cuatro bobinas de manera alterna. Por ejemplo, en una fase se alimentan los polos 1,3, 5 y 7 y en el otro el 2, 4, 6 y 8. Alimentando de esta forma los polos sucesivos de forma opuesta para que dos bobinas dejen pasar el flujo en un sentido y las otras dos en el contrario.

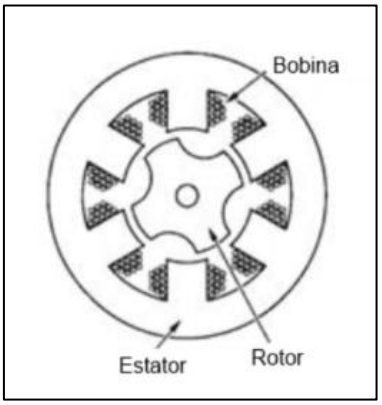


*Figura 31 Esquema del interior de un motor paso a paso. Fuente [33]*

En este tipo de motor tendremos dos modelos diferentes(unipolar, bipolar) y esto depende de la manera que se excitan los polos. Se pueden ver los dos modelos en la Figura 32

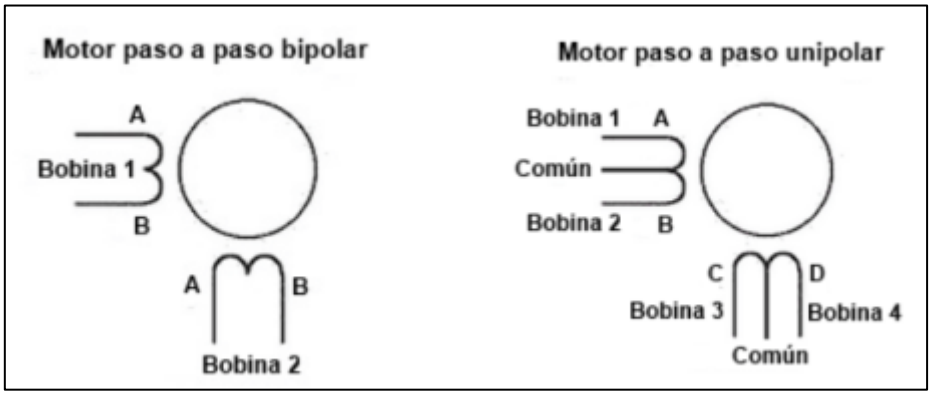


*Figura 32 Esquema de excitación de los polos. Fuente [33]*

- **Unipolar**: Son relativamente fáciles de controlar, gracias a que poseen devanados duplicados. Para realizar la secuencia de transmisión se tienen que poner en común los puntos de conexión (Común) y se conectan al positivo de la fuente de alimentación continúa pudiéndose cerrar el circuito por cualquier terminal de las fases. Normalmente estos motores tienen 5 o 6 cables de salida dependiendo de su conexión interna.

- **Bipolar**: Normalmente tienen más dificultad para ser controlados debido a que requieren del cambio de dirección de flujo de corriente mediante las bobinas y con una secuencia apropiada Su movimiento se realiza como hemos explicado anteriormente en el ejemplo. Normalmente tienen 4 cables de salida.

En cualquiera de los modelos anteriores tenemos la posibilidad de alimentar más de una fase simultáneamente, consiguiendo crear fuerzas que nos aportan unas posiciones del eje alternativas. Por ejemplo, para conseguir medio paso se alimentarían dos fases consecutivas a la vez dejando el eje entres estas dos fases. Incluso se podría variar la corriente de una de las fases para la obtención de otra orientación. Consiguiendo así un número mayor de pasos por vuelta del motor y alcanzar mayor precisión a la hora de colocar el eje en una posición requerida.

Para controlar un motor paso a paso mediante el uso de Arduino, se tienen entre otras las siguientes opciones: a través de una librería de la propia plataforma, drivers o la utilización de módulos shield para la placa que usemos.

En la ejecución de este proyecto se ha utilizado para el control del paso, un **Driver DRV8825** que dispone de 3 modos, que se activan a través del código Arduino dándonos diferentes amplitudes del paso (1/2,1/4,1/8,1/16,1/32), dependiendo de los modos activados y este lo colocamos sobre un módulo shield que es el **AZ-Delivery CNC Shield V3 Pinout**. Donde se pueden integrar hasta cuatro drivers para el uso simultaneo de cuatro motores.

## 3.3 CNC y Drivers

Aquí se van a desarrollar los dos elementos hardware que conectan la placa y el motor. A través de estos vamos a poder controlar los motores de una manera más precisa, ordenada y sencilla que si conectáramos los motores directamente en la placa.

Se han usado específicamente la AZ-Delivery CNC Shield V3 Pinout como modulo shield y el driver DRV8825.

### 3.3.1 DRV8825 Stepper Motor Driver Carrier

Antes de explicar nuestro modelo, se necesita una breve introducción sobre que son los drivers.

Un **driver o controlador** para motores es un circuito que nos permite tener un control de los motores de corriente continua de una forma muy simple Estos controladores nos dan la opción de manipular las intensidades y voltajes que suministramos al motor para poder manejar la velocidad a la que gira. Además, nos limita la corriente que circula (chopping) en el circuito por lo que es un buen método para proteger la electrónica de los motores.

En los motores paso a paso bipolares son más complejos por ellos sus drivers también lo son como el **DRV8825** o **A4988**. En este caso la corriente se puede activar en una u otra dirección. El driver tiene el poder sobre la dirección, para cambiar la polaridad del campo magnético producido en el interior del motor. El puente H es el circuito más conocido para realizar esta acción.

El **puente H** sirve para girar el motor en ambos sentidos, avance y retroceso, nos permite también controlar variables como la velocidad y torque. En la Figura 33 se puede observar la estructura de un puente H. Esta construida por cuatro interruptores (mecánicos o transistores) dos en cada barra lateral y en la barra central se encuentra la salida para el motor.

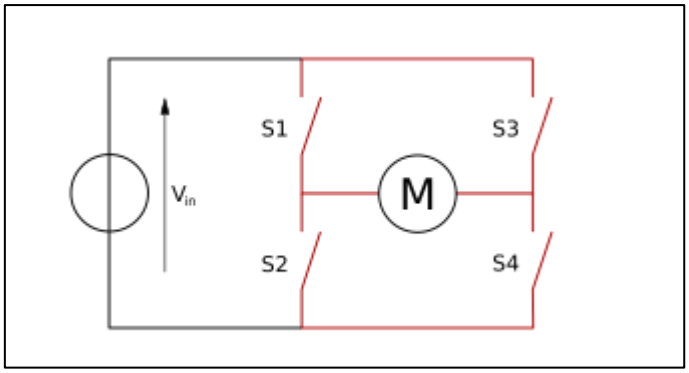


*Figura 33. Estructura del puente H. Fuente [34]*

Hay varios modelos de drivers en el mercado, pero como ya se ha comentado anteriormente hemos trabajado con el DRV8825 que es la versión mejorada del A4988. En la Figura 34 se puede observar este Driver. Solo requiere de dos salidas digitales del microcontrolador para poder accionar adecuadamente el motor, de esta manera ya se pueden manejar el paso y el sentido del motor. Lo que significa que otorga la capacidad de girar el motor paso a paso(stepping).

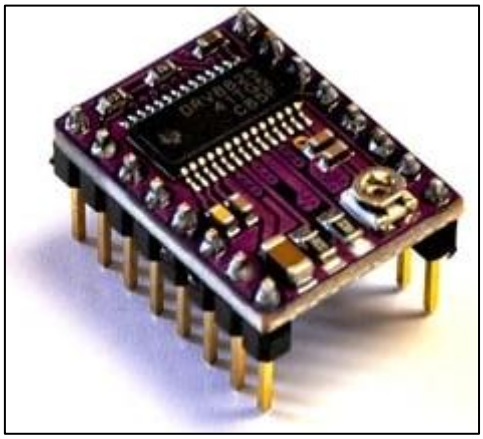

*Figura 34 Driver DRV8825. Fuente [35]*

Algunas de sus características básicas son:

- Una interfaz de control de paso y dirección simple.
- **6 resoluciones** de paso diferente: full-step, half-step, 1/4-step, 1/8-step, 1/16-step y 1/32-step.
- Se puede ajustar el control de corriente permitiendo configurar la salida de corriente máxima a través de un potenciómetro.
- La tensión de **alimentación máxima es 45 Voltios**. Requiere de una alimentación de 8.2-45 voltios conectados con Vmot.
- Puede interactuar con dispositivos de 3.3 Voltios y 5 Voltios.
- Tiene un apagado por sobrecalentamiento por superar la corriente y se bloque por la llegada un voltaje bajo.
- Regulador incorporado (no es necesario un suministro de voltaje lógico)
- PCB es de cobre y tiene 4 capas para una mejor disipación del calor (normalmente tiene viene con un disipador el cual hay que colocarle)

El esquema de conexiones para el uso del Driver se ve reflejado en la Figura 35, además te muestra cual es la acción que va a realizar cada uno de los pines. Es un esquema sencillo, pero tiene muchos pines las acciones de estos pines son:

- VMOT: Es la conexión a la alimentación. Máximo 45 V
- GND: Se conecta a la tierra del motor
- SLP: Sleep. Se conecta a 5v
- RST: Reset. Se conecta a 5v
- GND: a tierra (logic)
- STP: Se conecta a un pin del Arduino. Marca el estado del paso.
- DIR: Se conecta a un pin del Arduino. Es la dirección.
- A1, A2, B1, B2: Se conectan a las entradas del stepper (motor)
- M, M1, M2: Se conectan al arduino, pero y funcionan para activar los modos de microstepping.
- Enable.

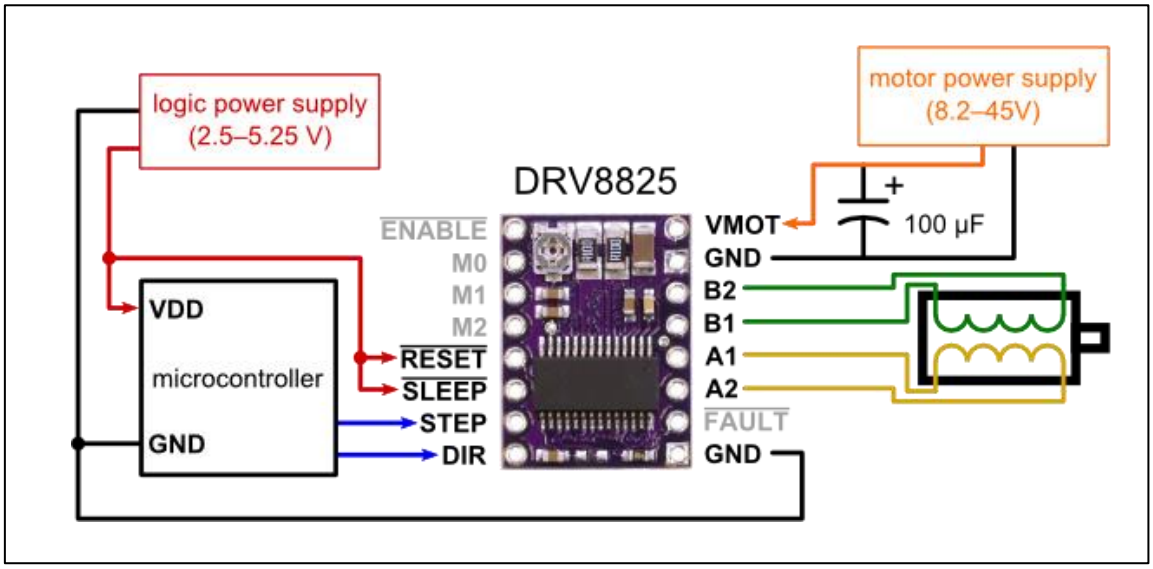


*Figura 35. Esquema de conexiones del DRV8825. Fuente [36]*

El controlador nos da la capacidad de realizar **microstepping o micro pasos** esto lo conseguimos gracias a los pines M0, M1 y M2. El microstepping es una técnica, para conseguir pasos inferiores a un paso completo del motor. Es decir que divide el giro que realiza el motor en más porciones para obtener un avance más lento o de una forma más precisa. Para ello varia la corriente aplicada en las diferentes bobinas, emulando valores analógicos con las distintas señales digitales de las que se dispone. Si consiguiéramos señales analógicas senoidales perfectas y desfasadas 90º entre si obtendremos la rotación deseada.

Pero no se puede conseguir dicha señal analógica, ya que usamos señales digitales. Por ello se intenta reproducir la señal analógica deseada, a través de saltos más pequeños de la señal eléctrica, obteniendo así pasos la mitad de grandes o más pequeños. En la Figura 36 se pueden observar algunas de estas señales.

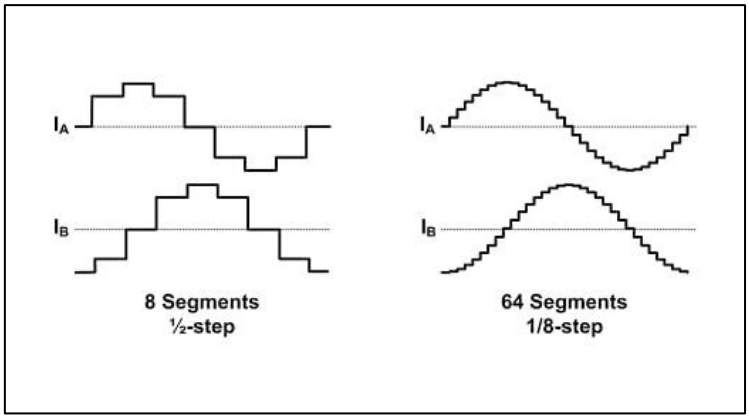


*Figura 36 Grafica de las señales haciendo microstepping. Fuente [37]*

Estos diferentes microstepping lo conseguiremos al cambiar el valor de los pines M0, M1 y M2. Normalmente estos pines vienen de serie conectados a tierra o GND mediante resistencias pull-down, por lo que si no tenemos conectado nada o no le damos un valor siempre serán 0 o LOW. Este valor se cambia forzándolos mediante una línea de código, poniéndolos a HIGH o 1. Por ejemplo, para poner M1 a HIGH se debe crear una variable apuntando al pin M1 si este fuera el 9 seria "int M1 = 9;" y después habría que poner en la parte de setup "digitalWrite (M1, HIGH);". En la siguiente Tabla 7 se pueden observar los diferentes casos de micro pasos que obtendríamos dándole valores con diferentes combinaciones a los Modos.

| **MODO 0** | **MODO 1** | **MODO 2** | **RESOLUCION** |
|---|---|---|---|
| LOW | LOW | LOW | Full Step |
| HIGH | LOW | LOW | Half Step |
| LOW | HIGH | LOW | 1/4-Step |
| HIGH | HIGH | LOW | 1/8-Step |
| LOW | LOW | HIGH | 1/16-Step |
| HIGH | LOW | HIGH | 1/32-Step |
| LOW | HIGH | HIGH | 1/32-Step |
| HIGH | HIGH | HIGH | 1/32-Step |

*Tabla 7. Estados para poner las diferentes resoluciones. Fuente [37]*

Los pasos que el motor va a realizar se controlan con el pin STEP, por cada pulso que le llega se corresponde con un micro paso y se moverá en la dirección que tengamos puesta en el pin DIR. Estas entradas están puestas en LOW por defecto a través de una resistencia pull-down de 100kilo ohmios. En la Figura 37 se puede observar cómo declarar en el código de la plataforma Arduino un paso y el sentido en el que gira el motor.

```
void Paso() {
  digitalWrite(Dir, HIGH);
  digitalWrite(Steps , HIGH);
  digitalWrite(Steps , LOW);
```

*Figura 37 Código para realizar un paso de motor. Edición propia*

El chip tiene 3 entradas diferentes para controlar su estado: nRESET, nSLEEP y nENABLE. El pin RESET si se activa reinicia la lógica interna y la tabla de pasos para llevarla a su posición inicial, el pin STEP es ignorado mientras este activo. nENABLE es para controlar las salidas del driver y habilitar las operaciones del indexador. nSLEEP pone el dispositivo en un estado de baja potencia. Se paran todos los relojes internos, se deshabilita el puente h. En este estado todas las entradas son ignoradas. Hay que tener en cuenta que nSLEEP su estado natural es LOW gracias a una resistencia pull-down de 1Mega ohmio y nRESET and nENABLE ambos están Low con una resistencia pull-down de 100kilo ohmios. Lo que significa que nRESET y nENABLE podrían prevenir que el driver funcione si se deja conectado. Deben ponerse directamente conectados a un voltaje de trabajo entre 2,2 y 5,25 voltios.

### 3.3.2 AZ-Delivery CNC Shield V3 Pinout

Este **módulo shield** está diseñado para poder soportar 4 motores porque originalmente se creó para controlar Impresoras 3D y otras máquinas de control numérico como fresadoras, de corte o grabado laser, … por eso aparecen las denominaciones **de X-Axis, Y-Axis y Z-Axis**. El objetivo de esta placa es facilitar el control de una maquina **CNC (Computador de Control Numérico)** y tiene la capacidad de funcionar con drivers del modelo A4988 o DRV8825, que como hemos dicho en el anterior apartado pueden hacer microstepping de hasta 1/32. En la Figura 38 se puede ver la estructura del módulo, las conexiones con los motores los jumpers con los que se trabaja y donde se conecta la alimentación.

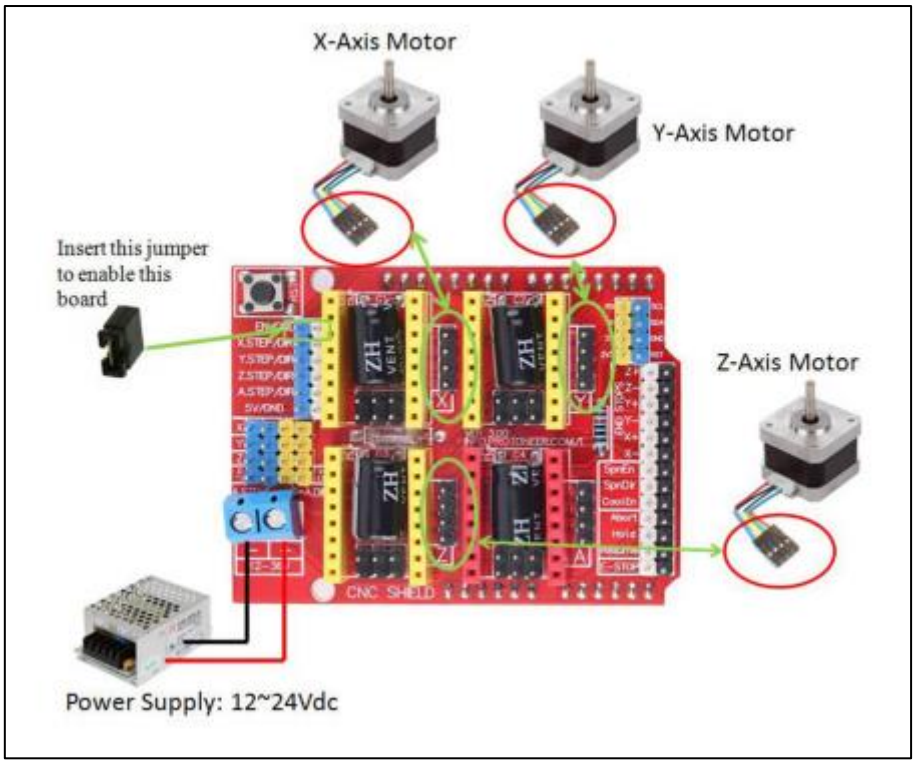


*Figura 38 Estructura del CNC y conexiones con los dispositivos. Fuente [38]*

Sus características principales son:

- Es compatible con GRBL 0.9.
- Permite manejar 3 motores paso a paso de forma independiente (X, Y, Z) y 1 motor duplicado adicional (A).
- 2 finales de carrera por cada eje (6 totales).
- Un pin ENABLE refrigerante.
- Habilitador y dirección Spindle.
- Jumpers para cambiar el microstepping requerido (con DRV8825 puede llegar a 1/32).
- Diseño compacto
- Los motores pueden conectarse a 4-pin o pueden soldarse.
- Voltaje de potencia de 12-36 Voltios DC.

Para configurar los modos de los micro pasos cada eje cuenta con tres jumpers que dependiendo de cuales estén insertados o no darán una configuración. En la Figura 39 se pueden ver donde se encuentran los jumpers en cada uno de los ejes y en la tabla 7 como es la configuración, cuando son el estado es HIGH es que esta insertado y LOW en el caso que no haya ningún jumper insertado en esa posición.

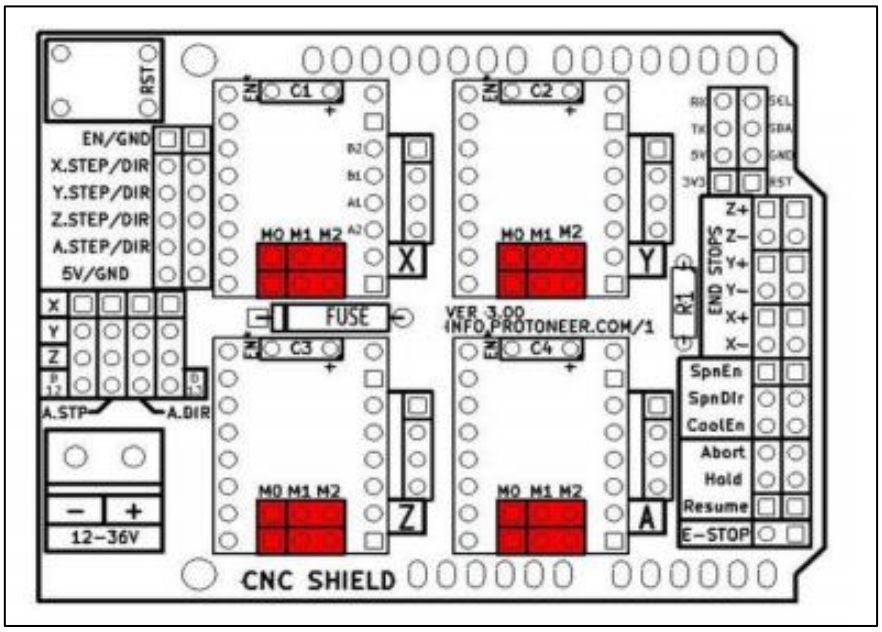


*Figura 39. Posiciones correspondientes de los Jumpers. Fuente [38]*

Para funcionen los motores correctamente se debe insertar un jumper en la posición del ENABLE o se debe declarar en la plataforma Arduino con el estado LOW, para ello se debe declarar en las variables apuntando nuestra variable al pin 8 que es donde se encuentra conectado el ENABLE seria a través de la línea "int ENABLE = 8" y para ponerlo LOW lo haríamos en la parte del setup explicada anteriormente mediante las líneas de código mostradas en la Figura 40.

```
pinMode(ENABLE, OUTPUT);
digitalWrite(ENABLE, LOW);
```

*Figura 40 Código para establecer ENABLE LOW. Edición propia*

# 4 NETBEANS Y SCENEBUILDER

El desarrollo del programa de ordenador que se ha realizado para controlar los motores de una forma más intuitiva y sencilla, hemos utilizado la plataforma Netbeans y así conseguir una mejor interfaz, nos hemos apoyado en Scene Builder.

## 4.1 NETBEANS

Es un entorno de **desarrollo integrado (IDE) libre**. Principalmente está adaptado para trabajar con el lenguaje de programación Java. **NetBeans IDE** es un producto de uso gratuito y libre sin restricciones. Es un proyecto de código abierto que tiene una gran base de usuarios y una comunidad en constante crecimiento. En la plataforma se desarrollan las aplicaciones a través de un conjunto de componentes software que se llaman módulos. Los módulos son archivos Java que incluye clases de Java escritas para poder comunicarse con APIs de Netbeans y un archivo Manifest file para identificarlo como un módulo. Se pueden extender las aplicaciones creadas por módulos si se les añade nuevos.

Es un **framework** que simplifica el desarrollo de aplicaciones Java. La plataforma trabaja a nivel de proyectos. Los proyectos incluyen todos los recursos necesarios para la construcción de un programa. En la Figura 42 se observa cómo se ven los proyectos.

La Figura 41 se puede observar el entorno de la IDE de Netbeans, la cual está compuesta por:

- Un **editor de texto**. Es la zona donde se escribe el código del programa.
- Un **navegador** del programa con el que estamos trabajando. Nos permite acceder rápidamente a un método del programa.
- Un **área de texto**. En esta área la plataforma se comunica con nosotros. Cuando compilamos nos transmite donde tenemos un fallo y cual es. Nos muestra la versión que usamos y el directorio del archivo que estamos ejecutando para la construcción del programa. Además, a través de esta área nos podemos comunicar con el programa si así lo requiriera.
- Una ventana donde se pueden observar los proyectos con los que trabajas, sus módulos interiores, los ficheros y los servicios.

- En la parte superior tenemos dos barras de herramientas donde se puede usar botones para la realización de acciones rápidas como ejecutar el proyecto, construirlo, abrir uno nuevo, etc.

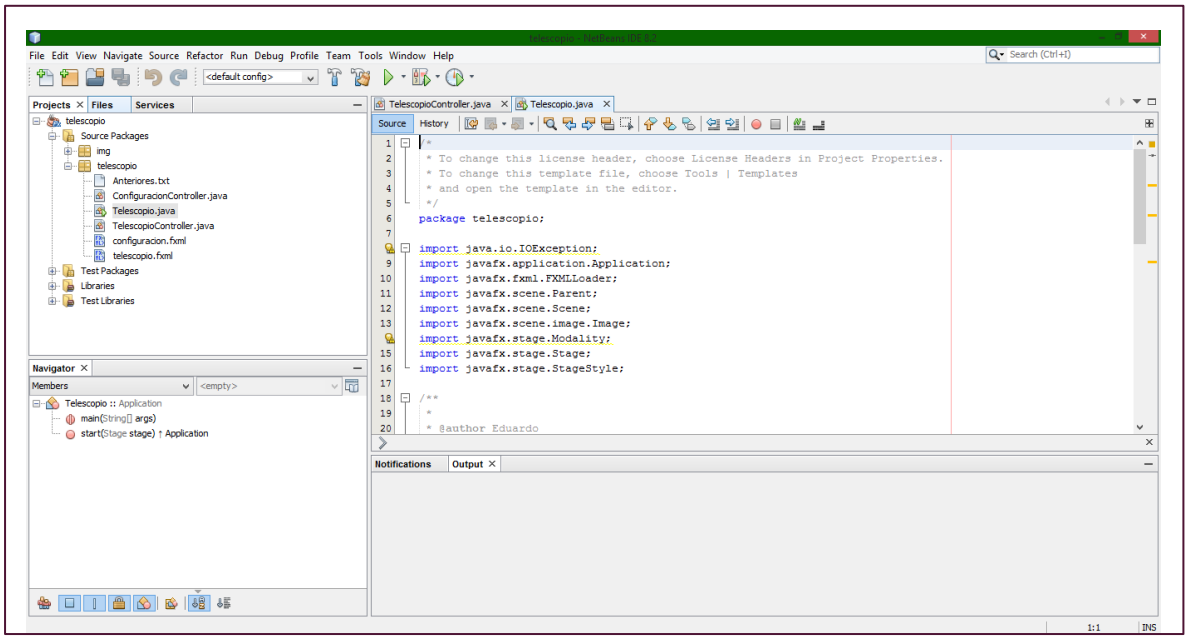


*Figura 41 Entorno de desarrollo NetBeans. Edición propia*

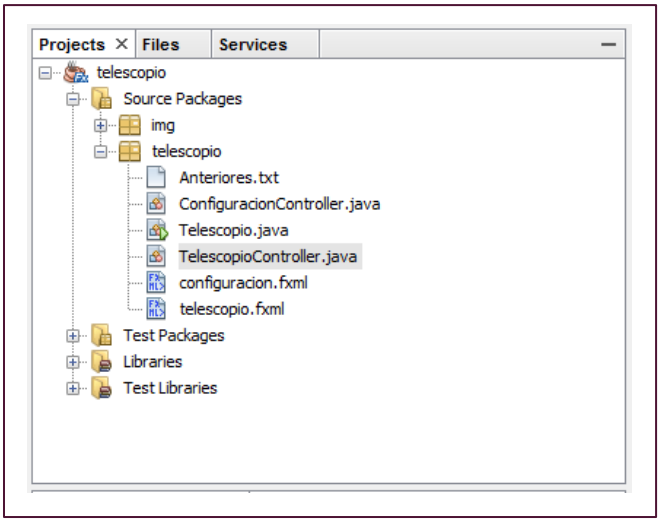


*Figura 42 Esquema de proyectos. Edición propia.*

En concreto dentro de las posibilidades que nos permitía la aplicación se ha usado el lenguaje de programación JavaFX.

### 4.1.1 JavaFX

Es un lenguaje tipado basado en **XML** y con secuencias de comando para construir objetos gráficos de Java. Proporciona una alternativa conveniente para construir gráficos en programación por procedimientos, es ideal para la definición de la interfaz de usuario de una aplicación JavaFX, ya que la estructura jerárquica de un documento XML es similar a la estructura del grafico de escena JavaFX.

Decidimos usar esta plataforma de desarrollo debido a que durante el paso por la universidad se ha trabajado desde el principio con el lenguaje de programación Java y porque durante uno de los años se estudiaba NetBeans en la asignatura Interfaces Persona Computador (IPC) por lo que estaba más familiarizado con su uso. Otra de las características más importantes que me determinado para trabajar con NetBeans es por la facilidad para comunicarse con Arduino lo que es un paso fundamental en el desarrollo y poder ayudarnos en la parte grafica de la interfaz mediante SceneBuilder. Además, Netbeans tiene una gran comunidad que está muy desarrollada y extendida por lo que normalmente se pueden encontrar ejemplos de acciones que no tengo conocimientos previos o para obtener más información sobre algún método.

## 4.2 JAVAFX SCENE BUILDER

Es una herramienta de diseño visual para los usuarios que permite la creación de interfaces de usuario de aplicaciones JavaFX de manera más rápida y sin codificación. Donde los usuarios pueden manejar **componentes de la IU** arrastrándolos y soltándolos en un área de trabajo, aplicar hojas de estilo, hacer modificaciones en sus propiedades y crea el código FXML del diseño de manera automática en segundo plano. El resultado que nos proporciona es un archivo FXML que luego se combina con un proyecto Java al vincular la interfaz de usuario a la lógica de la aplicación.

La interfaz (Figura 43) de Scene Builder se conformada por:

- Una parte central donde se colocan los **componentes** que se vayan a utilizar para la interfaz que se está diseñando.
- **Sección Library**. Donde podemos encontrar los todos los componentes de los que disponemos para diseñar la interfaz

- **Sección Document**. Se pueden ver los diferentes elementos que se están usando en nuestra interfaz y aquellos que están dentro de ellos, pero de una manera escrita para un rápido acceso a sus características.
- **Sección Inspector**. Contiene tres ventanas properties donde se pueden cambiar las propiedades del elemento seleccionado, layout nos permite cambiar las medidas del elemento y code que vincula al componente a unos controles para usarlos en la lógica del programa.

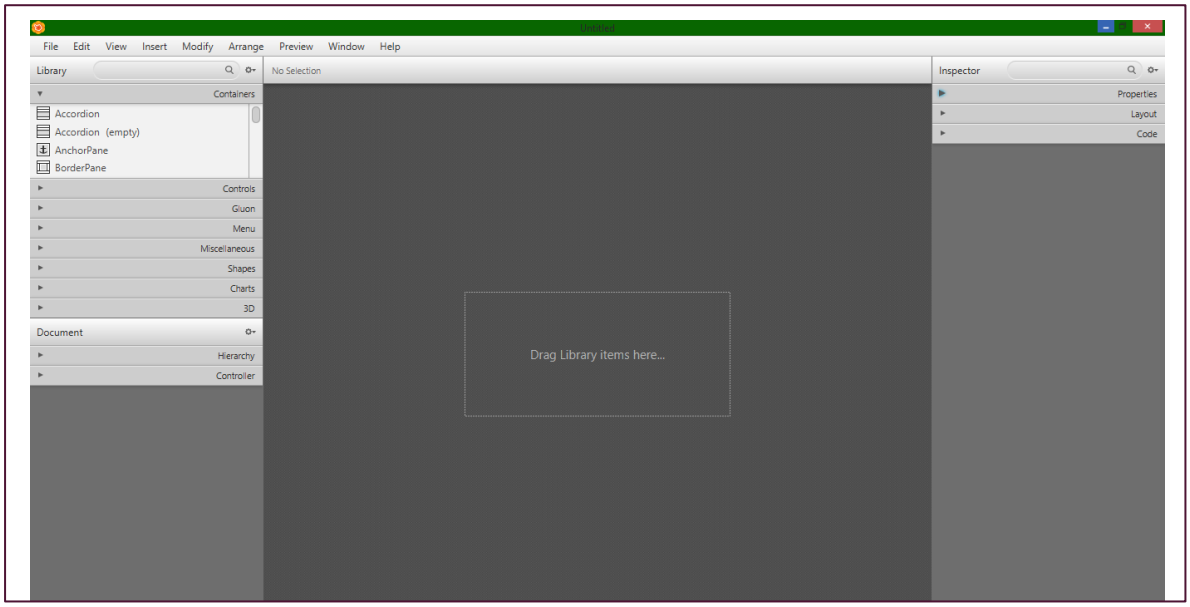


*Figura 43 Entorno de edición grafica Scene Builder. Edición propia*

Una vez acabada la interfaz requerida es necesario que vayamos a la IDE de Netbeans y en el lugar donde se controlan los proyectos y sus módulos, se debe pulsar con el botón derecho del ratón sobre el documento FXML para obtener la opción make controller. Al hacer esta acción se actualiza la información que se tiene sobre el archivo FXML en la plataforma y se actualiza a la última interfaz modificada. En la Figura 44 se muestra esta acción.

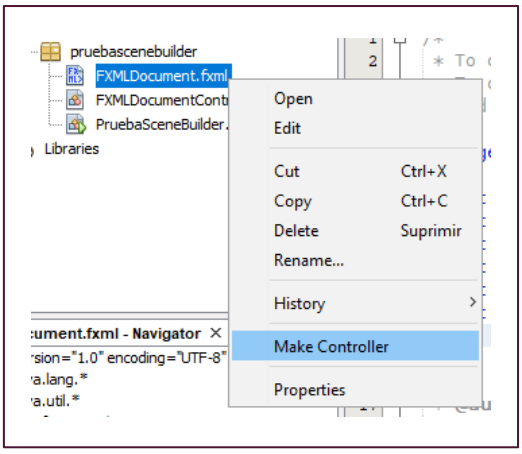


*Figura 44 Función Make Controller. Edición propia*

# 5 DESARROLLO DEL PROYECTO Y RESULTADO FINAL

En este apartado vamos a explicar el proceso que se ha realizado durante el TFG y mostrara el resultado final que se ha obtenido tras las diferentes y múltiples decisiones que hemos barajado y experimentado.

## 5.1 DESARROLLO

Se comenzó con un estudio sobre la placa MEGA, los drivers DRV8825 y los motores paso a paso, los cuales son los elementos fundamentales del proyecto, qué se han explicado anteriormente. Es necesario tener unos conocimientos básicos del DRV8825 porque son unos elementos frágiles que se pueden sobrecalentar e incluso quemar muy fácilmente. En la Figura 35 podemos observar las conexiones de los pines para poder vincularlos a la fuente de alimentación, al Arduino y motor.

Antes de conectar los motores, se debe comprobar mediante un polímetro las salidas del motor para identificar el bobinado y si tiene un correcto funcionamiento o le falla algo. Esta acción se realiza ubicando dos de las cuatro salidas en el polímetro colocándolo en modo continuidad, si nos da continuidad significa que corresponden al mismo bobinado. También es necesario comprobar la potencia suministrada a los motores y si hiciera falta modificarla, disponemos de un potenciómetro en los DRV8825 se puede observar en la Figura 35, lo encontramos a la derecha del pin nENABLE.

Una vez conectados los tres dispositivos. Para hacer girar el motor debemos pasarle un programa desde el PC a nuestra placa. La comunicación PC-Arduino se realiza a través del puerto serie. Para controlar el puerto serie en Arduino debemos saber que es la comunicación serial y sus referencias.

La **comunicación serial o serie** es importante ya que gran parte de los protocolos utilizados actualmente son seriales, muchos de los dispositivos con comunicación inalámbrica la usan para trasmitir información con el Arduino como los módulos bluetooth y los xbee o usarla directamente con el PC.

La comunicación serial **envía la información en una secuencia de bits.** Para ello hace uso de dos conectores o pines para conseguir la comunicación de datos, estos pines son denominados RX (recepción) y TX (transmisión). En la Figura 45 se puede observar un puerto serie físicamente.


*Figura 45. Conexión y comunicación Serie. Fuente [39]*

Todas las placas Arduino contienen un puerto serie habilitado en los pines digitales 0(RX) y 1(TX) compartido con el USB, por lo que no se usan estos pines como entradas/salidas. La placa MEGA dispone de tres puertos adicionales que son el Serial1 en los pines 19(RX) y 18(TX), Serial2 en los pines 17(RX) y 16(TX) y Serial3 en los pines 15(RX) y 14(TX). Estos pines no se encuentran conectados a la interfaz USB del Arduino.

Con el puerto serie conectado a nuestro circuito electrónico, se puede comenzar a enviar programas para realizar pruebas con los motores y observar su velocidad, su fluidez y probar los diferentes modos que tiene el DRV8825. Para poder realizar los diferentes pasos, inicialmente se utilizó el método delay. En la Figura 46 se visualiza el código para conseguir un movimiento en una dirección, que va a realizar 512 pasos y con un retardo(delay) de 100 milisegundos.

```
for (int i = 0;i < 512; i++){
  digitalWrite(Dir, HIGH);
  digitalWrite(Steps ,HIGH);
  digitalWrite(Steps ,LOW);
  delay(100);

}
```

*Figura 46 Código para mover motor con una velocidad y una dirección. Edición propia*

En este punto surge uno de los primeros problemas, que es poder controlar la dirección que ha de tomar el motor, dependiendo de si se quiere que gire en un sentido o en otro. Para conseguir esta acción al comienzo de la ejecución o cada vez que termine de dar los pasos requeridos, la placa va a quedarse esperando un valor para indicarle el sentido. Este valor se envía desde el monitor serial, explicado en el apartado 3.1.2 Entorno de desarrollo ARDUINO. Para ello debemos configurar en el código que la placa espere un valor, esta acción se realiza mediante las líneas de código vemos en la Figura 47.

```
if (Serial.available() > 0)
{

  String data = Serial.readStringUntil('\n');
```

*Figura 47. Para el programa y espera una entrada para continuar. Edición propia*

En estas líneas de código, en primer lugar comprueba mediante el método Serial.available() si el buffer del puerto está ocupado en el caso de que no lo este, la ejecución se espera a que le introduzcamos un valor. Este valor se guardará en nuestra variable String denominada data, guardándose hasta el salto de línea.

Hasta ahora estábamos trabajando con el método Delay () para obtener el movimiento del motor y que este lo haga a la velocidad que deseamos, pero no es un mecanismo suficientemente potente para algunas de las velocidades que se querían obtener en el movimiento del telescopio, decidimos comenzar a trabajar otra opción y esta era mediante interrupciones. Además, delay () congela la ejecución durante el tiempo que se encuentra activado.

En Arduino podemos encontrar dos tipos de eventos para definir **las interrupciones**. Unas son las interrupciones **programadas o timers (temporizadores),** estas se disparan cuando ha pasado un tiempo programado y las **interrupciones hardware**, que son una señal física que interrumpe la actividad normal de nuestro microprocesador que salta a atenderla. En la Figura 48 se ve el procesamiento de una interrupción.

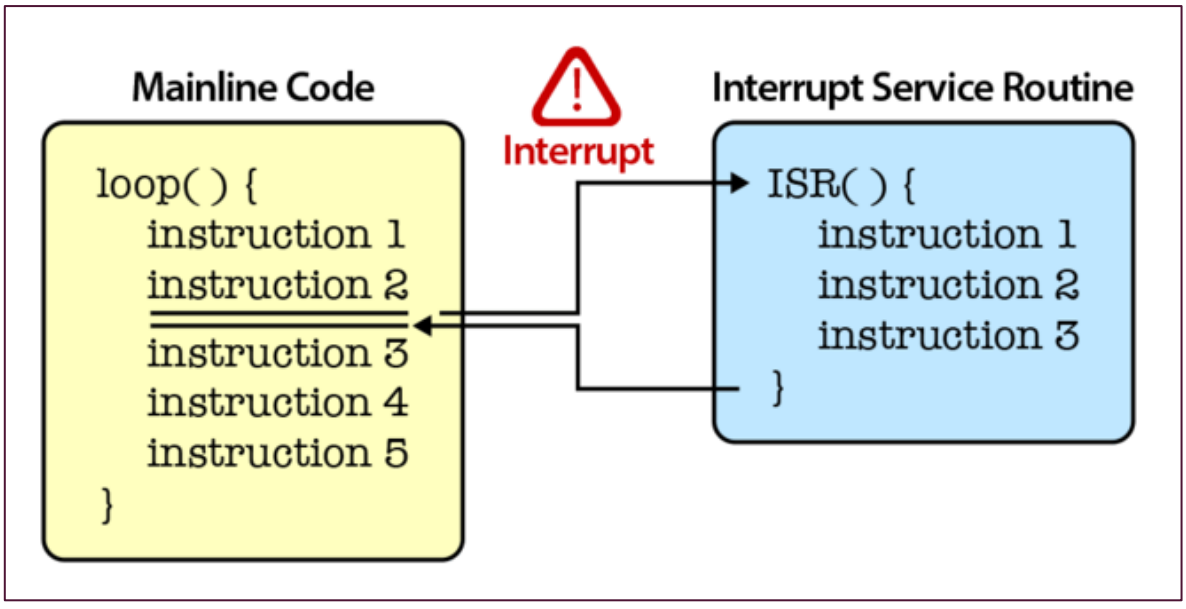


*Figura 48 Esquema de actuación de una interrupción. Fuente [40]*

Cuando salta una interrupción, tiene una función asociada que se ejecutará y que cuando acabe seguirá ejecutando el programa, normalmente esta función no recibe ni devuelve nada. A esta función se le llama **ISR (Interruption Service Routines)**.

Generalmente, no pueden ejecutarse dos ISR simultáneamente. Las variables que utiliza la ISR son volátiles, que significa que la variable debe ser consultada siempre antes de ser usada.

Para el control del tiempo se ha utilizado la librería TimerOne. Una de las razones por la que he se ha usado es la compatibilidad con la placa MEGA y porque hace uso de código sencillo y más intuitivo para la realización de interrupciones, otra de las razones es que funciona con Microsegundos. Como ya se ha mencionado anteriormente para trabajar con las librerías en la plataforma Arduino tenemos que escribir en código la línea "#include <TimerOne.h>", está línea concretamente es para usar la librería TimerOne y creamos un objeto denominado timer1. Para usar la interrupción siempre debemos inicializarla usando la línea de código "Timer1.initialize(tiempo);". Donde Timer1 es el objeto creado anteriormente y el tiempo son los segundos que tarda en saltar nuestra interrupción. Para coger la interrupción y activarla debemos usar el método "attachInterrupt ()". La línea de código que utilizamos es "Timer1.attachInterrupt(ISR);" que saltara cada vez que pase el tiempo puesto en el método initialize () y llamara a nuestra función ISR. Para deshabilitar la interrupción se debe usar el método "dettachInterrupt ()" un ejemplo de uso es "Timer1.detachInterrupt()", que detiene la interrupción creada con el objeto Timer1.

El tiempo que usaremos para inicializar nuestras interrupciones dependerá de la velocidad a la que queramos mover el telescopio. Generalmente el telescopio no se moverá a la misma velocidad que nuestro telescopio. Esto sucede porque no van conectados directamente el motor y telescopio, tendrán diferentes engranajes, acoplamientos o mecanismos para trasladar su movimiento y normalmente se obtendrán perdidas. Los motores que se usan para la montura Skywatcher EQ5 viene con un kit de engranajes, que tiene una reducción 120:1, que quiere decir que cada 120 pasos del motor darán una 1 paso el eje de salida del motor. La montura tiene una reducción de 144:1. Por ello podemos saber que para que dé un paso el eje de la montura el motor debe realizar 120*144 pasos que son 17.280 pasos. También se deberá saber el número de pasos que requiere el motor para dar una vuelta, nuestros motores dan una revolución en 48 pasos, debido a esto se necesitaran 17280x48 pasos para obtener el número de pasos que va tiene que dar el motor para conseguir una vuelta entera del eje de la montura que son 829440 pasos esto sucede en el caso de que usemos el modo full-step en el DRV8825.

Saber estos datos siempre serán necesarios para la configuración de nuestro programa, gracias a esto surgió la idea de la primera ventana que debía tener la aplicación. La **primera ventana** que encontramos por tanto es la de **configuración**. Se puede ver en la Figura 49 el primer prototipo de esta interfaz. La interfaz consta de tres partes los datos del motor del eje **AR**, datos del motor del eje **DEC** y dos botones **GET** y **ACEPTAR**.


*Figura 49 Primera modelo de interfaz gráfica de la configuración. Edición propia*

Tanto la parte del **motor AR y DEC** nos piden tres valores numéricos mediante unos Textfields, permitiendo solo valores numéricos. Estos Textfields tienen un id a la izquierda que los identifica y son:

1. Motor. Hay que introducir el número de pasos por vuelta que debe dar el motor.
2. M-1E. Introducir la reducción entre el motor y su salida. Por ejemplo, si es 120:1 solo hay que introducir el 120.
3. 1E-T. Introducir la reducción entre la salida del motor y el eje de la montura del telescopio. Por ejemplo, si es 144:1 solo hay que introducir el 144.

Cuando estén los tres Textfield con valores, tenemos una flecha para hacer las reducciones y decirnos cual será la relación entre el motor y su salida que llamamos R1 y la salida con el eje de la montura R2.

El **botón "GET"** sirve para coger los valores del ultimo uso que hicimos de la aplicación, así en el caso de que usemos los mismos dispositivos y no otros, no tendremos que introducir todos los datos y el botón **"ACEPTAR"** que nos permite continuar y pasar a la siguiente ventana que sería la de operativa.

En la Figura 50 fue el sucesor a la anterior interfaz de configuración que hemos explicado, pero realizando un cambio. Observamos que el usuario no necesita saber las reducciones que tiene el motor, sino que es más interesante conocer el delay que tendrá nuestro motor entre paso y paso a la velocidad estándar (1X) que es con el modo full-step.


*Figura 50. Segundo modelo de la interfaz gráfica de configuración. Edición propia*

La **velocidad 1X** que tendrá nuestro motor por defecto es la velocidad solar, anteriormente explicada en este documento, es 1 vuelta completa del eje en un día o de dicho de otra manera una vuelta cada 24 horas. En la tabla delay podemos ver el delay necesario para cada paso con los diferentes modos del DR8825 en los motores que hemos usado en microsegundos ya que será con la medida que trabajaremos en nuestra librería Arduino.

| **Delay (segundos)** | **Pasos** | **Modo** |
|---|---|---|
| 0,10416 | 829440 | 1 |
| 0,05234 | 1.658.880 | 1/2 |
| 0,02604 | 3.317.760 | 1/4 |
| 0,01302 | 6.635.520 | 1/8 |
| 0,00651 | 13.271.040 | 1/16 |
| 0,00326 | 26.542.080 | 1/32 |

*Tabla 8 Tabla de Delay por segundo para dar una vuelta dependiendo de la resolución. Edición propia*

El **delay o retraso** se obtiene de la fórmula que se ve en la Figura 51. Donde T es el tiempo que tarda el eje de la montura en dar una vuelta, Nº Pasos son los pasos que tiene quedar el motor para conseguir una vuelta del eje de la montura y el modo dependerá del que tengamos en nuestro driver seleccionado, usualmente será 1/32 porque es el más preciso. En la Figura 52 podemos ver como seria el método que traduce esta ecuación en nuestro código.

$$\frac{T}{Nº\ Pasos} * modo\ = delay$$

*Figura 51 Formula para obtener el delay. Edición propia*

```
// Operacion para las relaciones entre los engranages, se pone el 1000000
// abajo porque si no provoca overflow
public double relacion(double x, double y,double z){
    // numero de dientes i = Ze/Zs
    x = x * 32;//multiplicacion del modo que usamos
    res = x * y * z;//multiplica las reducciones
    float tiempo;
    tiempo = 24 * 60 * 60;//numero de segundos en un dia
    res = tiempo / res *1000000; //nos pasa el valor en microsegundos del delay
    return res;//valor final de la relaciones (periodo)

}
```

*Figura 52.Código para obtener el delay. Edición propia*

Otra de las cualidades nuevas que se le añadieron a la interfaz, es que no se puede pulsar el botón aceptar hasta que no se dispongan de los dos delay, ya que serán necesarios en la siguiente ventana, que es la operativa.

Desarrollada la ventana de configuración, se podía comenzar a trabajar la interfaz gráfica de nuestra **ventana principal** en la aplicación. Ya que la **operativa** necesita, los valores obtenidos en la interfaz anterior, para tener un control correcto de los motores con los que se va a trabajar.

El primer prototipo de la ventana principal donde se encuentra la operativa del programa la podemos contemplar en la Figura 53.

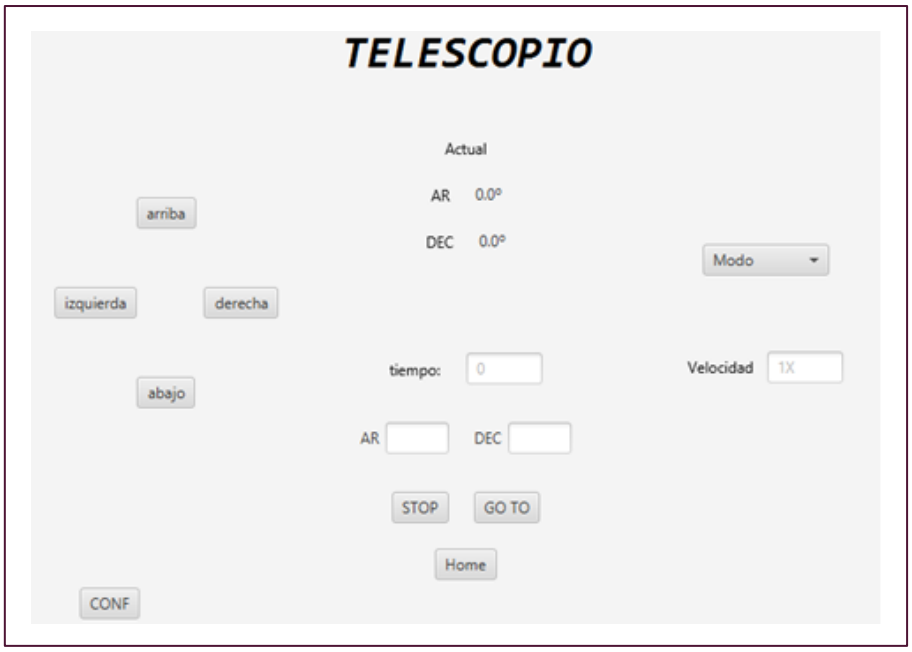


*Figura 53 Primer modelo de interfaz gráfica de la operativa. Edición propia*

La interfaz se compone de varias zonas que son:

- Zona **Manual**. A la izquierda tenemos cuatro botones en formato de cruz, que se usan para tener un control manual sobre el motor a velocidad 1X, donde moveremos los ejes a la posición donde queramos, mientras estemos pulsando el botón.
- **CONF**. Debajo de la zona manual, se puede encontrar un botón con la identificación CONF. Este botón nos permite volver a la ventana de configuración.
- Zona **GOTO**. En el centro de la interfaz, se debe observar la posición actual de nuestro telescopio, en ambos ejes. Aquí se realiza el método GOTO para ir a la posición que se indicase en los Textfields AR y DEC. También encontramos el botón Stop, que pausa los motores y el botón Home que lleva el telescopio a la posición 0,0. Y se daba la opción de poner un tiempo para que se moviera el telescopio.
- Zona **Velocidad**. La encontramos a la derecha y nos permite seleccionar el modo de velocidad del telescopio ya sea sideral, solar o lunar y la velocidad del motor respecto al modo.

En esta interfaz, faltaba por introducir gráficamente el método Follow y la configuración dependiendo del hemisferio que nos encontremos, esto es necesario porque el movimiento de AR no es el mismo, en el hemisferio sur que en el norte, dependiendo de donde nos encontremos el eje girara en un sentido o en otro. Llevándonos a la siguiente versión de la interfaz gráfica que visualizamos en la Figura 54.

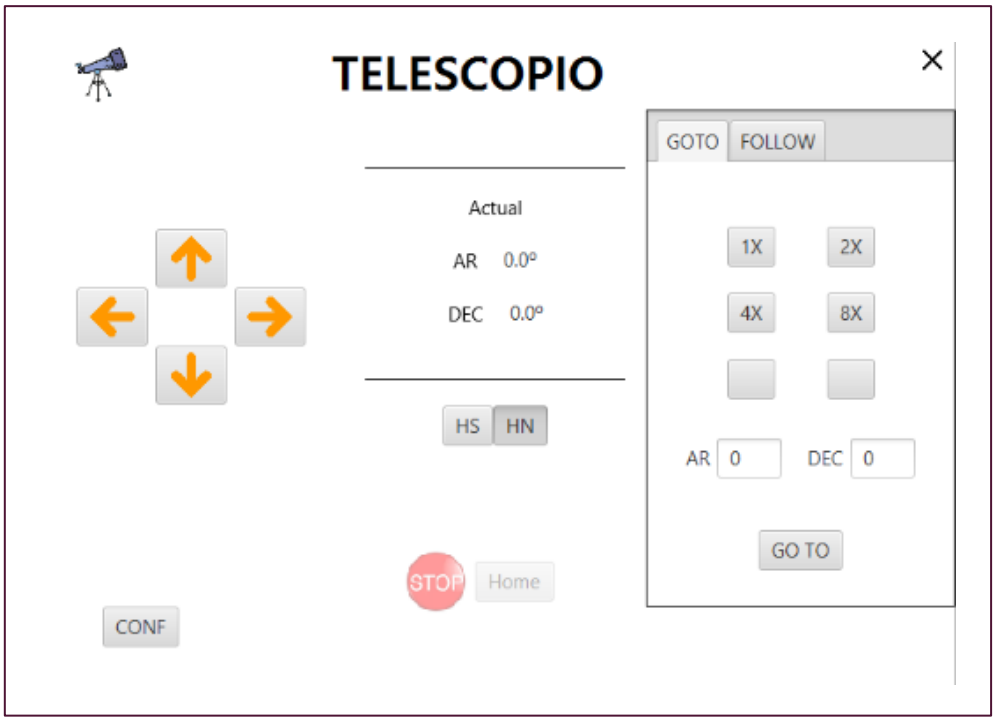


*Figura 54 Segundo modelo de la interfaz gráfica de la operativa. Edición propia*

Aun siendo muy semejante a la anterior versión, se ha cambiado el método GOTO a la derecha, introduciéndolo en una barra tabuladora, donde podemos cambiar de método fácilmente, este método nos permite elegir la velocidad a la que queremos que se posicione el telescopio desde 1X a 32X. Otras diferencias, son los botones de los hemisferios, las flechas del modo manual, en lugar del texto y el botón Stop, por una imagen insertada.

A través de la ventana de operativa, nos comunicaremos con la placa Arduino. Para conseguir, vincular la placa Arduino y nuestro proyecto de Java, hacemos uso de la librería PanamaHitek_Arduino. Si queremos trabajar esta librería debemos, importarla a nuestro proyecto en Netbeans. Esta librería, nos permite una comunicación mucho más intuitiva y sencilla con Arduino.

La librería cuenta con 4 clases:

- **PanamaHitek_Arduino** nos permite controlar la conexión con el Arduino a través del puerto serie. Cuenta con los métodos para inicia y para la conexión, los parámetros de la comunicación y las funciones para el envío y recepción de los datos.

- **PanamaHitek_Multimessage** gracias a ella podemos recibir múltiples datos de forma simultánea.
- **PanamaHitek_SpreadSheetHandler** sirve para el almacenamiento de datos y exportación a ficheros MS Excel.
- **ArduinoException** gestiona las excepciones que se puedan producir. Es una herramienta auxiliar de las primeras tres clases nombradas.

Para saber más información de PanamaHitek_Arduino se puede ver en la referencia [42], donde también se podrá encontrar un ejemplo, sencillo de la instalación y uso de esta librería.

A parte de la vinculación Arduino-NetBeans, se necesita comunicación entre las dos ventanas de la interfaz, puesto que en la configuración introducimos valores de nuestro dispositivo y obtenemos, los pasos totales que tiene quedar el motor, para un giro del eje de la montura y el delay, entre pasos que debe llevar a nuestra velocidad 1X.

Para conseguir esta comunicación, tenemos que trabajar con las clases controladoras de las interfaces, cuya creación se explicaba en el apartado 4.2 JavaFX Scene Builder y también se pueden observar en la Figura 44, se diferencia por el identificativo Controller pegado al nombre de la clase y la extensión ".java" , por ejemplo, "telescopioController.java". Si queremos conseguir que pase la información de la configuración, a la operativa se tiene que crear un método en la operativa para recibir los valores y dentro de la configuración debemos crear un controlador para usar el método de la operativa.

A partir de este momento, comenzamos a trabajar con dos motores y nos dimos cuenta de que trabajar con los dos drivers y los motores conectados en una placa protoboard no era lo más eficiente, limpio y ordenado. Se decidió introducir un módulo shield, en concreto el explicado en el apartado 3.2 AZ-Delivery CNC Shield V3 Pinout cambiando nuestro circuito electrónico al de la Figura 55.

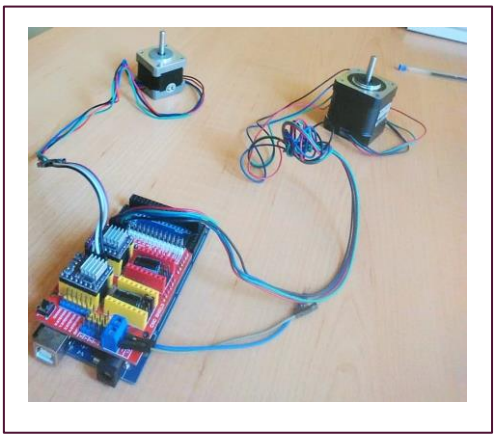

*Figura 55. Circuito completo . Edición propia*

Esta circunstancia, nos obliga a elegir el modo de microstepping de manera definitiva, sin poder ir cambiándolos en el programa, porque en el módulo CNC se hace físicamente poniendo los jumpers, como se explica en el apartado 3.2 AZ-Delivery CNC Shield V3 Pinout. Se eligió usar el modo 1/32 de microstepping para conseguir una mayor precisión en el movimiento del motor, aunque tengamos que hacer más pasos por vuelta, para alcanzar la posición deseada.

Al hacer uso de dos motores surgen varios problemas, uno de lo más importante es poder mover los dos motores sin perder pasos ni velocidad. La primera opción fue hacer el movimiento de manera continua, que cuando acabase uno empezase el otro, pero esto no resulto eficaz. Esta idea surgió porque la librería TimerOne no tiene la capacidad de crear más de un objeto y no puede usar más de una interrupción al mismo tiempo. La segunda opción y la que definitivamente adoptamos es que los motores se muevan **simultáneamente**. El problema como se ha dicho es que TimerOne no toleraba esta segunda opción.

Se buscaron diferentes opciones, desde crear las propias interrupciones sin hacer uso de librerías, crear una nueva librería parecida a TimerOne con más objetos, intentar crear más objetos TimerOne y la que fue nuestra solución hacer uso de otra librería que no interceda con TimerOne y que se pueda usar paralelamente. Esta librería se llama TimerThree es similar a TimerOne, pero cambiando el objeto al nombre Timer3 en lugar de Timer1 y cuando la declaramos debemos poner "#include <TimerThree.h>". Al tener esta librería todo se convierte más sencillo en este programa donde solo quedaba realizar el código correctamente.

Gracias a la obtención de esta librería se puede desarrollar el método **FOLLOW**. Este método solo tiene que mover el eje AR de la montura en un sentido, a una velocidad y un tiempo constante. Por ello mediante TimerOne realizamos el movimiento del motor y con TimerThree hacemos un temporizador que salte, cuando acabe para el motor en la posición que se quede. El problema que encontramos es que las dos librerías Timers tienen una máxima frecuencia o mínimo periodo de 1 microsegundo o 1 megahercio (MHZ) y un máximo periodo o mínima frecuencia de 8388480 microsegundos. Para ello se hace uso de una variable, que se incrementa cada vez que salta TimerThree, se inicializada la interrupción con un segundo de retardo, hasta llegar al tiempo deseado.

Otro problema que tenemos es como realizar él envió de datos desde NetBeans y leer desde Arduino. Hasta ahora solo se envía un mensaje desde Netbeans de un carácter o un numero de Sting fijo que representan un valor único, para obtener una respuesta de un motor. En Netbeans se enviaban los datos al suceder eventos en la interfaz, por ejemplo, al pulsar un botón. Si se pone más de un envió en ese evento Arduino los lee como el mismo. En Arduino al entrar datos por el Serial se coge hasta el salto de línea y se guarda todo en una variable que es

la que vamos a utilizar. Por lo que no podemos enviar toda la información o datos necesarios y si lo hacemos no se cogen en el orden correcto o cambian su valor. Se intento resolver mediante una cola de mensajería (MQTT, RabbitMQ), pero no se consiguió el efecto deseado y no obteníamos los resultados correctos. La otra manera que se puso en práctica fue usar la librería Parser.

La librería **Parser** implementa funciones, para interpretar de manera más simple los datos que están introducidos en un char Array o un String. Dispone de métodos para poder leer solo números, cadenas, búsqueda de caracteres, la capacidad de moverse o buscar en el buffer. Está diseñado para trabajar con información, llegada por un medio de comunicación (puerto serie). No modifica en ningún momento el buffer por lo que es seguro usarlo con cualquier Array. Se debe declarar en el código, como las demás librerías vistas anteriormente, debe tener al lado "#include".

Al implementar esta librería en Arduino conseguimos, que la variable donde guardamos la información enviada desde NetBeans al suceder un evento, pudiendo separar y seleccionar el trozo que queramos, para usarlo de una manera determinada. Por ejemplo, si necesitamos el retraso de ambos motores para mover los motores, se puede enviar un String con el formato "DelayMotor1-DelayMotor2" y guardar en la variable del motor1 hasta el guion y después de este en el motor2. De esta manera podemos ir cogiendo cada valor y guardarlo en las variables de la forma que deseamos y nos permite tener un código más simple y ordenado.

Con el funcionamiento de las tres librerías (TimerOne, TimerThree, Parser) tenemos la capacidad de realizar el método **GOTO** y **FOLLOW** ya que podemos usar las interrupciones de manera paralela, sin que se molesten entre ellas y separar los datos recibidos para poder guardarlos en puntos del código. Consiguiendo un movimiento simultaneo de los motores, en diferentes direcciones o que se pare el motor AR cada cierto tiempo.

Una vez conseguidos estos dos métodos, que nos permiten realizar el posicionamiento y el seguimiento. para darle mayor versatilidad, funcionalidad o precisión a la aplicación y al movimiento de los motores implementamos el modo Manual y el modo Stop. Estos dos métodos se explican en el apartado 5.2 resultado final.

## 5.2 RESULTADO FINAL

Este apartado muestra, el resultado final de nuestra interfaz gráfica y las funciones de cada uno de sus elementos y su interacción, con el código de Arduino que efectúa el movimiento de los motores.

Al iniciar la aplicación, lo primero que vamos a encontrar es la ventana de configuración, esta se puede observar en la Figura 56. Esta interfaz gráfica está compuesta por 4 partes.

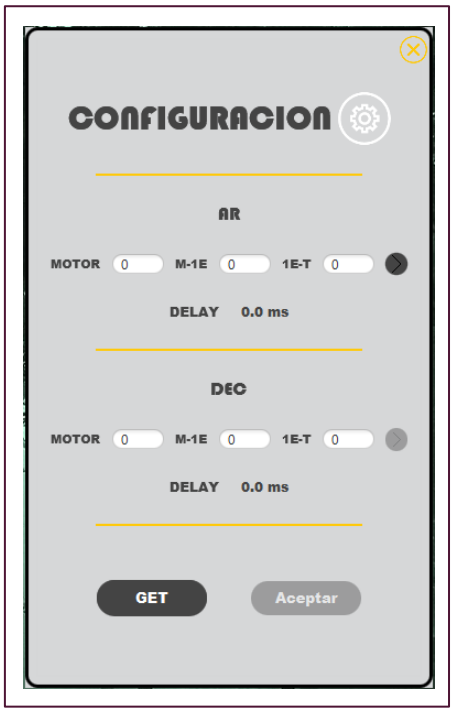


*Figura 56. Ventana de configuración. Edición propia*

- **Título de la ventana**. Encontramos el titulo con su icono correspondiente a la derecha y arriba en la esquina encontramos un botón con la imagen de una x para poder cerrar la ventana.
- **Configuración AR**. Está compuesta por 3 Textfields donde introducimos, los pasos que necesita para dar una vuelta. El motor identificado con el texto a la Izquierda MOTOR, la primera relación que tenemos del motor y su salida que se identifica como M-1E y la salida final entre la relación de salida del motor y el telescopio que se observa como 1E-T. A la derecha, se encuentra un botón para realizar la conversión de todos los pasos que tenemos que dar y pasarlo al delay estándar. que va a tener nuestro motor y lo introduce en el label que se encuentra abajo con el identificativo DELAY. Figura 51 encontramos la fórmula para este evento y en la Figura 52 el código para la conversión.

- **Configuración DEC**. Es igual a la configuración anterior solamente que pondremos los valores del motor DEC y el botón para realizar la conversión de pasos a Delay, este se encuentra deshabilitado hasta que no tengamos la configuración de AR.
- **Botones GET y ACEPTAR**. El botón GET nos permite usar la configuración de los motores, que empleamos la vez anterior, al pulsarlo nos saldrán directamente sus datos puestos en cada uno de los apartados correspondientes. En la Figura 57 observamos el pseudocódigo que realiza la aplicación para coger estos valores.

```
01. METODO GET(EVENTO).
02. VARIABLES
03.     FILE f, SCANNER s, STRING[] l, INT i
04.
05. TRY
06.     Leemos el fichero con el Scanner
07.
08.     WHILE (Tenga lineas el fichero)
09.             Se guarda esa linea en el STRING[] en nuestra posicion i
10.             se suma uno esa posicion.
11.     CATCH
12.
13.     FINALLY
14.             cerramos el SCANNER.
15.
16.     GUARDAMOS CADA POSICION DEL ARRAY STRING EN UNA VARIABLE
17.     ESTA VARIABLE LA MOSTRAMOS EN LA INTERFAZ DONDE CORRESPONDE.
```

*Figura 57 Pseudocódigo método GET. Edición propia.*

El botón ACEPTAR se inicializa como deshabilitado y solo se habilitará cuando tengamos el delay de las dos configuraciones anteriores. Al pulsar ACEPTAR el programa realiza siempre dos acciones, guarda los valores en un fichero txt para usarlo después con GET y nos envía a la pestaña de la operativa. En el caso de que no estuviera conectado el ARDUINO o estuviera mal puesto nos sale una ventana de aviso como la figura 58.

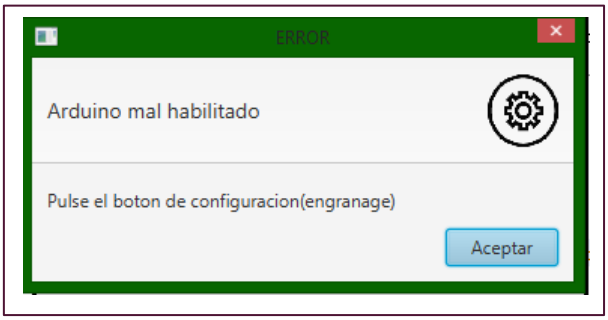


*Figura 58 Ventana de alerta, Arduino mal puesto. Edición propia*

La siguiente ventana que nos sale, es el estado inicial de nuestra operativa, el estado de nuestra interfaz gráfica varía dependiendo del evento que suceda. En la figura 59 vemos el estado inicial al pasar a la interfaz de la operativa.

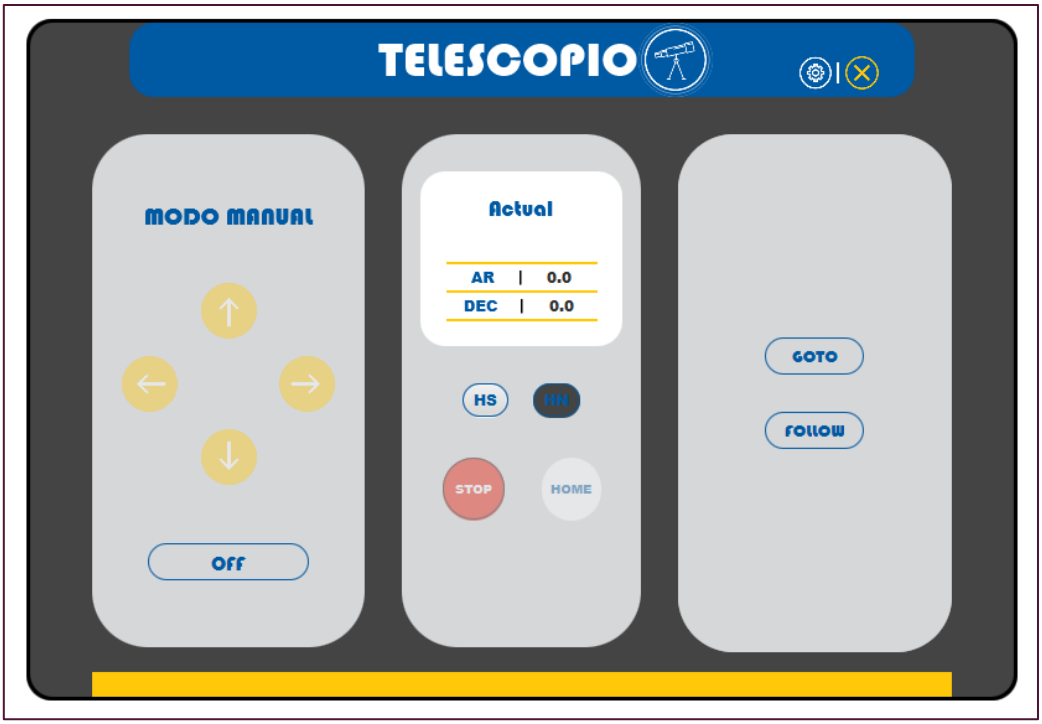


*Figura 59 Estado inicial ventana operativa. Edición propia*

Podemos observar en la parte superior el título de la ventana que es telescopio y a la derecha vemos el botón de configuración (Figura 60) que nos permite volver a la ventana de configuración al pulsarlo y un botón con una X para cerrar la ventana. Nuestra interfaz consta de tres secciones diferentes (MODO MANUAL, ACTUAL, métodos GOTO y FOLLOW).

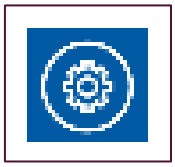

*Figura 60 Botón de configuración. Edición propia.*

- **Sección de MODO MANUAL.** En la parte superior tenemos un identificativo para saber que vamos a usar en este caso el modo manual. En su parte central hay 4 botones puestos en cruz, que representan hacia donde mover la montura del telescopio por lo que si se pulsan los botones en dirección horizontal se moverá en el eje AR y en vertical en DEC. Para poder pulsarlos primero debe activarse el modo manual, pulsando el botón OFF de la parte inferior y esta sección pasaría a tener el formato de la Figura 61. Donde podemos observar que los botones se han puesto de un color más intenso al habilitarse y el botón OFF ha pasado a llamarse ON indicando que está activo el modo MANUAL y deshabilitando el sector de los métodos GOTO y FOLLOW. Al pulsar OFF la aplicación le envía un mensaje al código Arduino donde le indicara que debe entrar al método manual. Tenemos cuatro opciones.

1. **Botón AR positivo.**
2. **Botón AR negativo**
3. **Botón DEC positivo**
4. **Botón DEC negativo**

En las cuatro opciones se envía un String con el formato "Dirección-Delay entre pasos" que hace girar el motor mientras estemos pulsando el botón. Si dejamos de pulsar el botón la aplicación de NetBeans envía el mensaje "para", diciéndole a la placa que debe parar el movimiento, además actualiza el valor de las posiciones AR y DEC con los pasos que hemos dado.


*Figura 61. Ventana operativa con MODO MANUAL activo. Edición propia*

Al pulsar el botón ON este pasara a ser OFF deshabilitando, las líneas de movimiento manual. Y dependiendo de en qué situación este la sección de métodos GOTO y FOLLOW envía un mensaje al Arduino u otro.

- **Sección ACTUAL.** Se encuentra en el centro de la interfaz. Nos debe mostrar donde están posicionados los ejes en grados y puede ir desde -179º a 180º. Se observan cuatro botones, HS y HN donde se selecciona el hemisferio en el que se encuentra, esta predefinido con hemisferio norte, STOP para el método que se esté ejecutando en el Arduino y home que nos lleva a la posición 0(AR), 0(DEC).

- **Sección métodos GOTO Y FOLLOW.** Esta es la sección más compleja y en ella están contenidos los dos métodos principales de la aplicación. La primera ventana que tiene esta sección, son dos botones para elegir el método que se va a emplear.

    1. Si pulsamos **GOTO** nos cambia la apariencia de esta ventana y pasa a ser como la Figura 63, además envía información a través del serial al Arduino para acceder al método GOTO. En el método GOTO Arduino se queda esperando que le pasemos el delay que llevara nuestro motor para ello se dispone de seis botones en la interfaz que van desde 1-32. Cuando presionamos uno divide los Delays obtenidos en la ventana de configuración entre el número del botón seleccionado, enviando estos nuevos Delays al Arduino. Al recibir esta información, Arduino necesita saber los pasos que debe dar cada motor, para comenzar a moverlos, estos parámetros los obtenemos al pulsar el botón GOTO que tenemos abajo, que envía los pasos y la dirección que van a seguir los ejes. Los pasos y la dirección los introducimos en los Textfields con el identificativo AR y DEC, en estos Textfields se introducen los valores con el formato de grados, van desde -179 a 180 y al pulsar GOTO se convierten en pasos.


*Figura 62 Ventana GOTO. Edición propia*

    2. Si pulsamos **FOLLOW** nos llevara al formato de la Figura 64 y seleccionara el método Follow dentro del código ARDUINO, el cual se queda esperando que le pasemos el tiempo que debe moverse el motor y delay entre pasos que va a llevar. Para ello en la interfaz gráfica lo primero que necesitamos es decidir qué modo vamos a usar, los diferentes modos han sido explicados en 2.1.2 Velocidades de seguimiento/translación. Lo siguiente es poner la velocidad con la queremos movernos como máximo son 32 y solo nos queda poner el tiempo

de trabajo de los motores, se introduce en los Textfields con los identificativos h(horas), m(minutos) y s(segundos), el valor total se pasara a segundos. Teniendo todos estos valores podemos enviarle la información al Arduino pulsando el botón FOLLOW que envía un mensaje con el formato String "dirección-velocidad-tiempo".


*Figura 63 Ventana método FOLLOW. Edición propia.*

Dentro de esta sección cuando nos encontramos dentro de una de las ventanas explicadas anteriormente, podemos movernos entre ellas mediante la barra que hay en la parte superior que nos lleva entre las ventanas.

### 5.2.1 Diagramas de flujo métodos ARDUINO.

- **Elección del modo**.

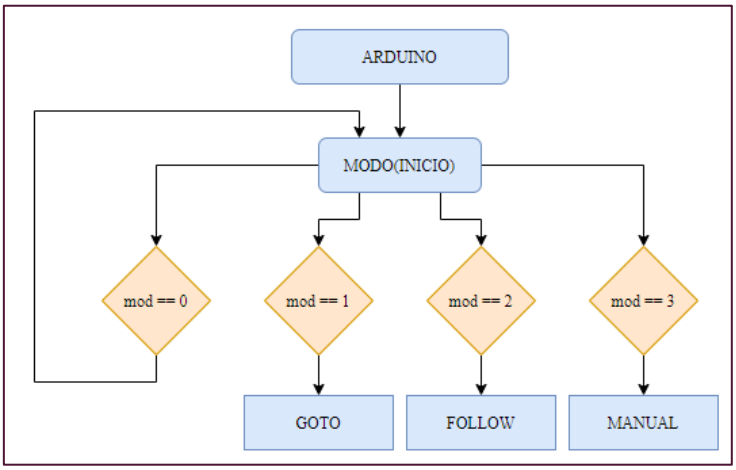


*Figura 64 diagrama flujo de elección de modo. Edición propia.*

- **Método GOTO.**

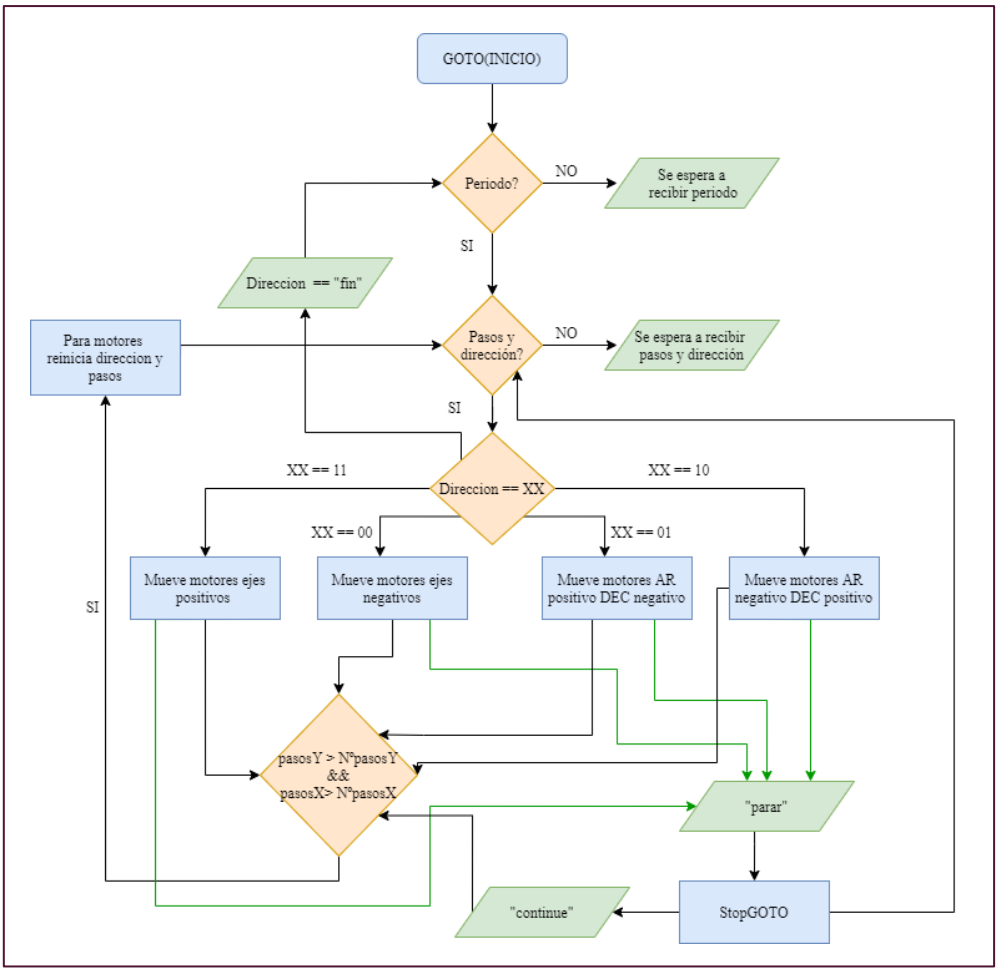


*Figura 65 Diagrama de flujo del método GOTO. Edición propia*

- **Método FOLLOW.**

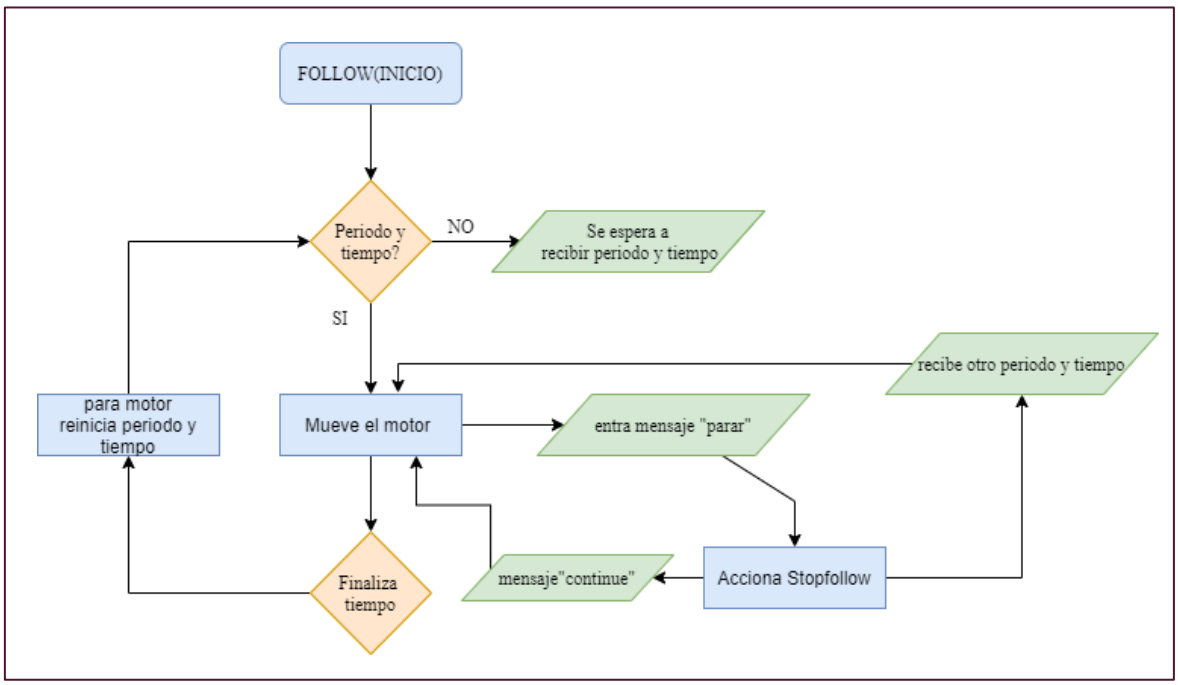


*Figura 66 Diagrama de flujo del método FOLLOW. EDICION PROPIA.*

- **Método MANUAL**

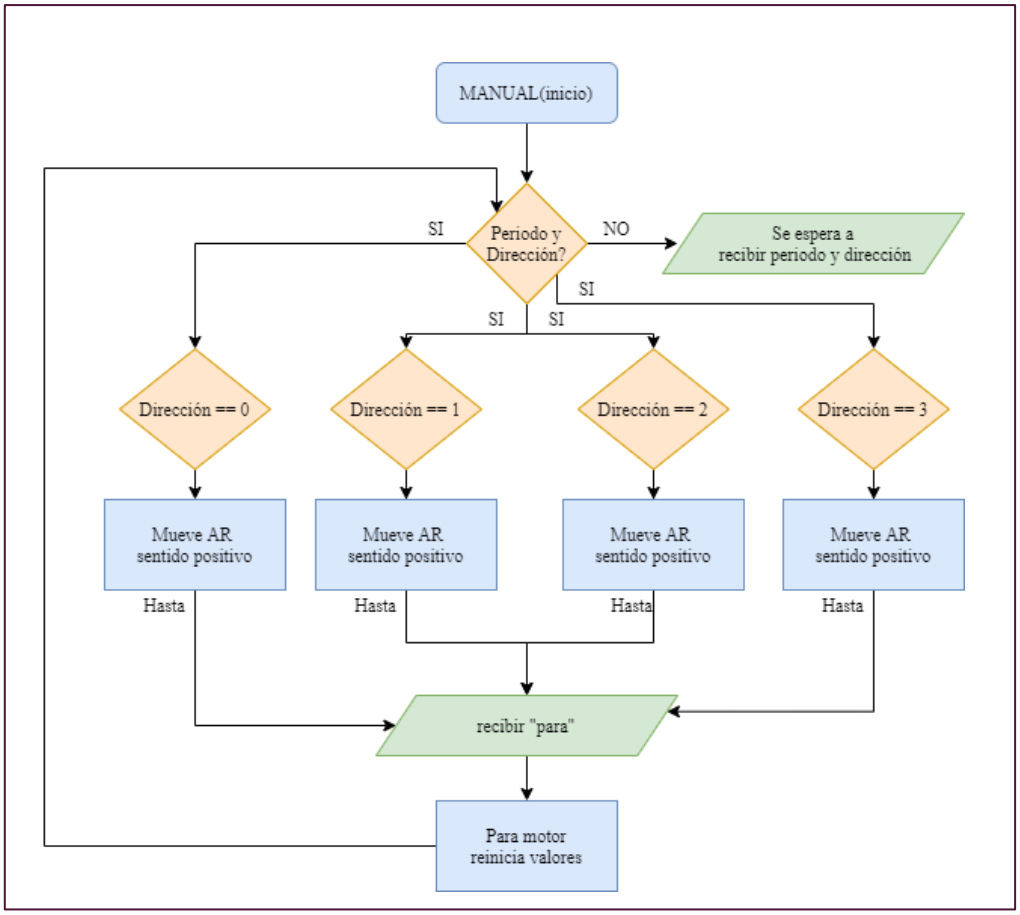


*Figura 67 Diagrama de flujo del método Manual. Edición propia.*

Se puede observar el código de todos estos diagramas de flujo en el punto 8 Anexos.

# 6 CONCLUSIONES

En este capítulo vamos a expresar unas posibles líneas futuras que se podrían desarrollar con la utilización de este programa y una evaluación particular del progreso del TFG, durante el tiempo que se ha estado elaborando y que debido a las incidencias por todos conocidas, ha tenido que ejecutarse de forma especial, pues sea visto mermada la posibilidad de una demostración práctica, que seguramente hubiera aportado perspectivas distintas y que al mismo tiempo me ha obligado a flexibilizar el itinerario habitual de elaboración de este tipo de proyectos.

## 6.1 LÍNEAS FUTURAS

Dentro del TFG podemos encontrar otras maneras diferentes de mejorar o desarrollar nuevas ideas con respecto a este tipo de instrumento científico, ya que en los telescopios no solo tenemos la capacidad de automatizar la montura, si no que muchos de sus elementos, se pueden controlar mediante un microcontrolador. Pero muchas de estas ideas, requieren un tiempo de desarrollo del que no se disponía,  por lo que no se quedaron sin poderse contemplar dentro de este proyecto.

Una de estas ideas, era la posibilidad de hacer la aplicación para usarla desde el móvil, además del ordenador, proporcionando así un acceso alternativo a la ejecución del programa. Sería necesario instalar un módulo de Wifi o Bluetooth en nuestra placa mega o cambiar nuestra placa a un Bluetooth o UNO Wifi. Esto proporcionaría la vía necesaria para la comunicación entre el móvil y la placa. La interfaz no tendría que cambiar mucho, lo primero que nos seguiría apareciendo seria la ventana de configuración, porque es necesario tener la información de nuestros motores. Al pulsar aceptar, dejaría como la ventana principal la sección actual y con la opción de poder deslizar hacia ambos lados para tener la capacidad de acceder tanto a la sección de manual como a la sección de métodos GOTO y FOLLOW. Pero aparte de que las secciones se verían en diferentes ventanas, solo tendríamos que quitar la parte superior del título y el botón X para salir, dejando el botón de configuración en otro lugar más cómodo para su uso en un móvil.

Otra mejora que se podría conseguir seria perfeccionar, la forma de comunicación entre Java y Arduino y la manera en que depura y coge, los datos Arduino. Ya que con esta librería es necesario que pase un evento para enviar la información, esta circunstancia ocasiona muchos problemas como la de falta de datos o que entre un dato cuando no debe, produciendo un

colapso en nuestro programa, haciendo que actúe por su cuenta y no realice las acciones esperadas

Una línea futura, seria añadir más funcionalidades a la aplicación, por ejemplo, poniendo una cámara en el telescopio mostrándonos la imagen que recoge el telescopio en todo momento a través de la aplicación o que los métodos GOTO y FOLLOW tuvieran la capacidad mediante una API de conseguir las coordenadas de un astro y moverse a su posición, para seguirlo durante un periodo elegido.

## 6.2 EVALUACIÓN PARTICULAR

Para mi persona el TFG ha sido un proyecto donde desarrollar las capacidades y conocimientos adquiridos durante el proceso de formación de los años pasados en la universidad y conseguir nuevos conocimientos de nuevos ámbitos y temas, que me pueden ser muy útiles en un futuro. Además, hay que considerarlo desde la perspectiva de la situación especial en que se ha tenido que desarrollar, siendo un proyecto con gran necesidad de hardware y de elementos, encontrados en la universidad y a los que no hemos tenido acceso, como la montura del telescopio, obligándonos al uso de una sola forma de comunicación con el profesor que sin suponer una anomalía si hizo más desventajosa la intercomunicación y el feedback ,ya que en ningún momento pudo ser física, debido a la pandemia de COVID^19 que en estos momentos afrontamos.

El ámbito de este TFG en el grado de ingeniería estaría dentro de la rama de ingeniería de computadores, encargada del desarrollo a nivel hardware. Siendo la rama por la que opte para mi especialización.

Me gustaría destacar varios de los temas que se han tocado en este proyecto.

1. El uso del lenguaje Java, que se ha necesitado para el desarrollo de la aplicación, con el que comenzamos a trabajar desde el primer día que pisamos la universidad y donde podemos apreciar una gran mejora con mayor seguridad y agilidad en la aplicación de este lenguaje de desarrollo.
2. Trabajar con la plataforma Arduino y su lenguaje, que es muy parecido a C. Otro lenguaje que desde el comienzo del grado universitario hemos trabajado y visto en clase.
3. La combinación de dos aspectos fundamentales para el desarrollo de este proyecto, que es tener que integrar los dispositivos hardware dentro de una aplicación software.

4. Ampliar mis conocimientos sobre microcontroladores y herramientas físicas, que me serán útiles a la hora de automatizar otros dispositivos.
5. Tener la posibilidad de aprender sobre un dispositivo científico, tan antiguo y al tiempo tan evolucionado, como es el telescopio y que nos da incalculables oportunidades de alcanzar nuevos descubrimientos sobre uno de los mayores y más inquietantes enigmas de nuestra sociedad, como es el espacio. Además, de adquirir conocimientos sobre astronomía una ciencia que conmueve como he podido comprobar multitud de mentes mas o menos brillantes o más o menos especializadas o aficionadas que sienten una especial atracción por el espacio.

Es importante remarcar que no habría sido capaz de realizar este proyecto sin los conocimientos adquiridos durante mi paso por la universidad, aunque algunas asignaturas no estén visibles explícitamente en el proyecto, me han ayudado a mi forma de abarcar los problemas de una manera implícita. Gracias a ellas he sido capaz de poder solucionar problemas de manera más lógica con la aplicación de conocimientos y referencias alternativas que permiten una apertura mental que conduce a la solución de las complicaciones que encontramos.

# 7 BIBLIOGRAFÍA

[1] "Astronomía", www.ecured.cu, 2020, [Online]. Disponible: https://www.ecured.cu/Astronom%C3%ADa (Accessed on 05/2020).

[2] "Mecánica Celeste, sobre el movimiento de los cuerpos celestes", www.universidadviu.es, 2017, [Online]. Disponible: https://www.universidadviu.es/mecanica-celeste/ (Accessed on 05/2020).

[3] "Telescopio", es.wikipedia.org, 2019, [Online]. Disponible: https://es.wikipedia.org/wiki/Telescopio (Accessed on 05/2020).

[4] ESTELA, RAFFIÑO, MARIA "Telescopio", 2019, [Online]. Disponible: https://concepto.de/telescopio/ (Accessed on 05/2020).

[5] "Telescopios, Características generales", www.saracco.com, 2020, [Online]. Disponible: https://www.saracco.com/telescopios-caracteristicas-generales/ (Accessed on 05/2020).

[6] DIAZ, JOSE VICENTE "Telescopios: Tipos, parámetros y uso", 2015, [Online]. Disponible: https://josevicentediaz.com/el-universo/telescopios-tipos-parametros-y-uso/ (Accessed on 05/2020).

[7] BRAVO, ROBERTO "Tipos de Telescopios astronómicos", 2011, [Online]. Disponible: https://astroaficion.com/2011/02/22/tipos-de-telescopios/ (Accessed on 05/2020).

[8] "MONTURAS PARA TELESCOPIOS, RECOMENDACIONES PRÁCTICAS" www.airelibre.es, 2020, [Online]. Disponible: https://www.airelibre.es/blog/monturas-telescopios-recomendaciones-practicas/ (Accessed on 05/2020).

[9] BRAVO, ROBERTO "Tipos de Monturas para un telescopio", 2011, [Online]. Disponible: https://astroaficion.com/2011/03/04/tipos-de-monturas/ (Accessed on 05/2020).

[10] "Tipos de Montura. Elegir Montura (ecuatorial o azimutal)", planetastronomia.com, 2015, [Online]. Disponible: https://planetastronomia.com/monturas/tipos-de-montura-telescopio/ (Accessed on 05/2020).

[11] "SKY-WATCHER EQ5", www.duoptic.com.ar, 2018, [Online]. Disponible: https://www.duoptic.com.ar/skywatcher-eq5.html (Accessed on 05/2020).

[12] "Aplicaciones espaciales con microcontroladores comerciales resistentes a la radiación", www.automaticaeinstrumentacion.com, 2019, [Online]. Disponible: http://www.automaticaeinstrumentacion.com/es/notices/2019/04/aplicaciones-espaciales-con-microcontroladores-comerciales-resistentes-a-la-radiacion-45370.php#.XtFItlUzbIV (Accessed on 05/2020).

[13] "SAMV71Q21RT", microchip.com, 2020, [Online]. Disponible: https://www.microchip.com/wwwproducts/en/SAMV71Q21RT (Accessed on 05/2020).

[14] “SAMRH71”, microchip.com, 2020, [Online]. Disponible: https://www.microchip.com/wwwproducts/en/SAMRH71 (Accessed on 05/2020).

[15] COSIJOPI, LOBO, DAVID “APLICACIONES DE LOS MICROCONTROLADORES”, sites.google.com, 2017, [Online]. Disponible: https://sites.google.com/site/dcosijopil/unidad-5-arquitecturas-embebidas-o-microcontroladores/5-4-aplicaciones-de-los-microcontroladores (Accessed on 05/2020).

[16] DUNBAR, BRIAN “About The Hubble Space Telescope” 2020, [Online]. Disponible: https://www.nasa.gov/mission_pages/hubble/about (Accessed on 05/2020).

[17] CESARI, THADDEUS “First Look: NASA’s James Webb Space Telescope Fully Stowed” 2020, [Online]. Disponible: https://www.nasa.gov/mission_pages/hubble/about (Accessed on 05/2020).

[18] *SMITH, YVETTE* “Carina Nebula's 'Mystic Mountain'”,2019, [Online]. Disponible: https://www.nasa.gov/image-feature/carina-nebulas-mystic-mountain (Accessed on 05/2020).

[19] *ALVAREZ, RAUL* “Por primera vez podemos ver en todo su esplendor el enorme telescopio espacial James Webb, el sucesor del Hubble”,2017, [Online]. Disponible: https://www.xataka.com/espacio/por-primera-vez-podemos-ver-en-todo-su-esplendor-el-enorme-telescopio-espacial-james-webb-el-sucesor-del-hubble (Accessed on 05/2020).

[20] “Y4, un cometa que se desintegra durante la pandemia en la Tierra”, www.efe.com, 2020, [Online]. Disponible: https://www.efe.com/efe/espana/destacada/c-2019-y4-un-cometa-que-se-desintegra-durante-la-pandemia-en-tierra/10011-4239516 (Accessed on 05/2020).

[21] “¿Que es Arduino?”, arduino.cl, [Online]. Disponible: https://arduino.cl/que-es-arduino/ (Accessed on 05/2020).

[22] NOVAS, PEÑA, DESPRABEL “Microcontroladores”, [Online]. Disponible: https://www.aiu.edu/Spanish/Microcontroladores.html (Accessed on 05/2020).

[23] “MICROCONTROLADOR 32 BITS / DE USO GENERAL / DE BAJA POTENCIA”, www.directindustry.es, [Online]. Disponible: https://www.directindustry.es/prod/atmel/product-13779-584831.html (Accessed on 06/2020).

[24] “Arduino Duemilanove”, www.arduino.cc, 2020, [Online]. Disponible: https://www.arduino.cc/en/Main/arduinoBoardDuemilanove (Accessed on 06/2020).
[25] JADIAZ “PLACA ARDUINO UNO”, 2016, [Online]. Disponible: http://www.iescamp.es/miarduino/2016/01/21/placa-arduino-uno/ (Accessed on 06/2020).

[26] DELGADO, CRESPO, MANUEL “Arduino Mega 2560”, 2017, [Online]. Disponible: http://manueldelgadocrespo.blogspot.com/p/arduino-mega-2560.html (Accessed on 06/2020).

[27] “ARDUINO DUE”, www.arduino.cc, 2020, [Online]. Disponible: https://store.arduino.cc/arduino-due (Accessed on 06/2020).

[28] “ARDUINO NANO”, www.arduino.cc, 2020, [Online]. Disponible: https://store.arduino.cc/arduino-nano (Accessed on 06/2020).

[29] “ARDUINO BT(BLUETOOTH)”, www.arduino.cc, 2020, [Online]. Disponible: https://www.arduino.cc/en/Main/ArduinoBoardBT (Accessed on 06/2020).

[30] "ARDUINO UNO WIFI REV2", www.arduino.cc, 2020, [Online]. Disponible: https://store.arduino.cc/arduino-uno-wifi-rev2 (Accessed on 06/2020).

[31] "Arduino Software (IDE)", www.arduino.cc, 2020, [Online]. Disponible: https://www.arduino.cc/en/Guide/Environmen (Accessed on 06/2020).

[32] "Stepper Motor: Bipolar, 200 Steps/Rev, 28×32mm, 3.8V, 0.67 A/Phase" www.pololu.com, 2020, [Online]. Disponible: https://www.pololu.com/product/1205(Accessed on 06/2020).

[33]JORGE "Motor paso a paso con Arduino" 2016, [Online]. Disponible: https://paletosdelaelectronica.wordpress.com/2016/02/18/motor-paso-a-paso-con-arduino/ (Accessed on 06/2020).

[34] "Puente H (electrónica)", es.wikipedia.org, 2020, [Online]. Disponible: https://es.wikipedia.org/wiki/Puente_H_(electronica) ) (Accessed on 06/2020).

[35] "DRV8825 Driver Microstepper + Disipador térmico", www.3djake.es, 2020, [Online]. Disponible: https://www.3djake.es/e3d/drv8825-driver-microstepper-disipador-termico (Accessed on 06/2020).

[36] "DRV8825 Stepper Motor Driver Carrier, High Current (md20a)", www.pololu.com, 2020, [Online]. Disponible: https://www.pololu.com/product/2132 (Accessed on 06/2020).

[37] ISAAC "DRV8825: el driver para motores paso a paso",2019, [Online]. Disponible: https://www.hwlibre.com/drv8825/ (Accessed on 06/2020).

[38] "CNC-Shield V3 Datenblatt", AZ-delivery, [Online]. Disponible: https://cdn.shopify.com/s/files/1/1509/1638/files/CNC-Shield_V3_Datenblatt.pdf?12861848950625246088 (Accessed on 06/2020).

[39] LLAMAS, LUIS "COMUNICACIÓN DE ARDUINO CON PUERTO SERIE", 2014, [Online]. Disponible: https://www.luisllamas.es/arduino-puerto-serie/ (Accessed on 06/2020).

[40] CRESPO, ENRIQUE "Interrupciones",2016, [Online]. Disponible: https://aprendiendoarduino.wordpress.com/2016/11/13/interrupciones/ (Accessed on 06/2020).

[41] "Timer1", playground.arduino.cc, 2018, [Online]. Disponible: https://playground.arduino.cc/Code/Timer1/ (Accessed on 06/2020).

[42] GARCIA, GONZALEZ, ANTONY "Librería PanamaHitek_Arduino, v3.0.0", 2017, [Online]. Disponible: http://panamahitek.com/libreria-panamahitek_arduino/ (Accessed on 06/2020).

# 8 ANEXOS

## Código de programación Arduino

```
#include "ParserLib.h"
#include <TimerThree.h>
#define DEBUG(a) Serial.println(a);
#include <TimerOne.h>
//Programa motores
//pin dirección
int Dir = 5;
int DirY = 6;
//pin de pasos
int Steps = 2;
int StepsY = 3;
//int dir;
String Direccion;
//String per;
double pasos = 0;
double pasosY = 0;
// enable
char mod;
const int enPin = 8;
//PERIODOS
volatile double Periodo;
volatile double PeriodoY;
volatile double Time;
int Ti = 0;
boolean nuevoMod = false;
boolean reset = false;
//PASOS
volatile double Numpasos;
volatile double NumpasosY;
Parser parser;
//poner salidas

void setup() {
  // put your setup code here, to run once:
  pinMode(Dir, OUTPUT);
  pinMode(Steps, OUTPUT);
  pinMode(enPin, OUTPUT);
  digitalWrite(enPin, LOW);
  Serial.begin(57600);
}

void loop() {
  if (mod == 0) {
    Modo();
  }
  if (mod == 1) {
    GOTO();
  }
  if (mod == 2) {
    Follow();
  }
```

```
  if (mod == 3) {
    Manual();
  }

}

void Modo() {
  if (Serial.available() > 0)
  {
    mod = Serial.read();

    if (mod >= '0' && mod <= '9')
    {
      mod -= '0';
      DEBUG((int)mod);

    }
    Serial.println("Periodo :");
    Serial.flush();
  }

}

void GOTO() {
  if (reset == true) {
    Serial.end();
    Serial.begin(57600);
    reset = false;
    Numpasos = 0;
    NumpasosY = 0;


  }
  //Mira si tenemos el periodo para los motores
  if (Periodo == 0 && Numpasos == 0) {
    cogerPeriodo();
  }
  // Si tenemos la velocidad si coge los pasos y la dirección
  else if (Periodo > 0) {
    cogerPasos();

  }

}

/*
  ##############################
   ##############################
   ##############################
*/
// En este metodo pide el periodo del motor AR
void cogerPeriodo() {
  if (Serial.available() > 4)
  {
    String data = Serial.readStringUntil('\n');
    DEBUG(data);
    parser.Init(data);
    String f = parser.Read_String('-');// Coge valores hasta el
carácter -
    parser.Skip(1);
```

```
    String N = parser.Read_String('-');// Coje valores hasta el
carácter -
    parser.Skip(1);
    String P = parser.Read_String('\n');//coje valores hasta el salto
de línea
    if (f == "mod" )
    {
      mod = 2;
      Serial.print("Periodo:");
      Periodo = 0;
      Numpasos = 0;
      pasos = 0;
      Ti = 0;
    }
    if (f == "3") {
      mod = 3;
    }
    else {
      PeriodoY = P.toDouble();
      Periodo = N.toDouble();
      Timer1.initialize(Periodo);
      Timer3.initialize(PeriodoY);
      Serial.println("Numero de pasos: ");
    }
  }
  parser.Reset();
  Serial.flush();
}
/*
   ##############################
   ##############################
   ##############################
*/
// coge los pasos a dar por el motor AR
void cogerPasos() {
  if (Serial.available() > 0)
  {
    pasosY = 0;
    pasos = 0;
    String data = Serial.readStringUntil('\n');
    DEBUG(data);
    parser.Init(data);
    Direccion = parser.Read_String('-'); // se guarda la dirección
    parser.Skip(1);
    String N = parser.Read_String('-'); // se guarda la dirección
    parser.Skip(1);
    String P = parser.Read_String('\n');//Se guarda el número de pasos
    Numpasos = N.toDouble() * 1000000;
    NumpasosY = P.toDouble() * 1000000;

    if (Direccion == "11" ) {
      AR11();
    }
```

```
    if (Direccion == "00") {

      ARN00();

    }
    if (Direccion == "10" ) {

      AR10();
    }

    //Con el rojo al lado de Vmot gira con el mismo del movimiento del
reloj
    if (Direccion == "01") {
      ARN01();
    }
    if (Direccion == "fin") {
      reset = true;
      Periodo = N.toDouble();
      PeriodoY = P.toDouble();
      Timer1.initialize(Periodo);
      Timer3.initialize(PeriodoY);
    }
    if (Direccion == "mod") {
      mod = 2;

      Periodo = 0;
      PeriodoY = 0;
      Numpasos = 0;
      NumpasosY = 0;
      pasos = 0;
      Ti = 0;
    }
    if (Direccion == "3") {
      mod = 3;

      Periodo = 0;
      PeriodoY = 0;
      Numpasos = 0;
      NumpasosY = 0;
      pasos = 0;
      Ti = 0;
    }
  }
  parser.Reset();
  Serial.flush();
}
/*
   ##############################
   ##############################
   ##############################

  // dirige el movimiento de los motores
  Es la ISR de la interrupción, hace subida y bajada de las bobinas,
*/
void Paso() {

  digitalWrite(Steps , HIGH);
  digitalWrite(Steps , LOW);
  pasos++;

}
```

```
void PasoY() {

  digitalWrite(StepsY , HIGH);
  digitalWrite(StepsY , LOW);
  pasosY++;

}
//método para movimiento AR;
void AR11() {
  digitalWrite(DirY, HIGH);
  digitalWrite(Dir, HIGH);
  while (pasos < Numpasos && pasosY < NumpasosY ) {
    Timer1.attachInterrupt(Paso);
    Timer3.attachInterrupt(PasoY);
    Serial.print("x");
    Serial.print(pasos);
    Serial.print("y");
    Serial.println(pasosY);
    if (Serial.available()) {
      String dat = Serial.readStringUntil('\n');
      DEBUG(dat);
      if (dat == "stop") {
        StopGo();

      }
    }
  }
  if ( NumpasosY == 0) {
    while (pasos < Numpasos) {
      Timer1.attachInterrupt(Paso);
      Serial.print("AR");
      Serial.println(pasos);
      if (Serial.available()) {
        String dat = Serial.readStringUntil('\n');
        DEBUG(dat);
        if (dat == "stop") {
          StopGo();
        }
      }
    }
    if (pasosY >= NumpasosY ) {
      Serial.println("finalizadoAR");
      Serial.println("finalizadoAR");
      Serial.println("finalizadoAR");
      Serial.println("finalizadoAR");
      Serial.println("finalizadoAR");
      Serial.println("finalizadoAR");
      Serial.println("finalizadoAR");
      Serial.println("finalizadoAR");
      Serial.println("finalizadoAR");
      Timer1.detachInterrupt();

    }
  }
```

```
if ( Numpasos == 0 ) {
  while (pasosY < NumpasosY) {
    Timer3.attachInterrupt(PasoY);
    Serial.print("DEC");
    Serial.println(pasosY);

    if (Serial.available()) {
      String dat = Serial.readStringUntil('\n');
      DEBUG(dat);
      if (dat == "stop") {

        StopGo();
      }
    }
  }
  if (pasosY >= NumpasosY ) {
    Serial.println("finalizadoDEC");
    Serial.println("finalizadoDEC");
    Serial.println("finalizadoDEC");
    Serial.println("finalizadoDEC");
    Serial.println("finalizadoDEC");
    Serial.println("finalizadoDEC");
    Serial.println("finalizadoDEC");
    Serial.println("finalizadoDEC");
    Timer3.detachInterrupt();

  }
}
if (pasosY >= NumpasosY ) {
  Timer3.detachInterrupt();
  if (pasos >= Numpasos) {
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Timer1.detachInterrupt();
  }
  while (pasos < Numpasos) {
    Serial.print("AR");
    Serial.println(pasos);
    if (Serial.available()) {
      String dat = Serial.readStringUntil('\n');
      DEBUG(dat);
      if (dat == "stop") {

        StopGo();
      }
    }
  }

}
if (pasos >= Numpasos ) {
  Timer1.detachInterrupt();

  while (pasosY < NumpasosY) {
    Serial.print("DEC");
    Serial.println(pasosY);
```

```
      if (Serial.available()) {
        String dat = Serial.readStringUntil('\n');
        DEBUG(dat);
        if (dat == "stop") {

          StopGo();
        }
      }
    }
    if (pasosY >= NumpasosY) {
      Serial.println("finalizadoDEC");
      Serial.println("finalizadoDEC");
      Serial.println("finalizadoDEC");
      Serial.println("finalizadoDEC");
      Serial.println("finalizadoDEC");
      Serial.println("finalizadoDEC");
      Serial.println("finalizadoDEC");
      Serial.println("finalizadoDEC");
      Timer3.detachInterrupt();

    }

  }
}

void ARN00() {
  digitalWrite(DirY, LOW);
  digitalWrite(Dir, LOW);
  while (pasos < Numpasos && pasosY < NumpasosY ) {
    Timer1.attachInterrupt(Paso);
    Timer3.attachInterrupt(PasoY);
    Serial.print("x");
    Serial.print(pasos);
    Serial.print("y");
    Serial.println(pasosY);
    if (Serial.available()) {
      String dat = Serial.readStringUntil('\n');
      DEBUG(dat);
      if (dat == "stop") {

        StopGo();

      }
    }
  }
  if ( NumpasosY == 0) {
    while (pasos < Numpasos) {
      Timer1.attachInterrupt(Paso);
      Serial.print("AR");
      Serial.println(pasos);

      if (Serial.available()) {
        String dat = Serial.readStringUntil('\n');
        DEBUG(dat);
        if (dat == "stop") {

          StopGo();
        }
      }
    }
    if (pasosY >= NumpasosY ) {
```

```
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Timer1.detachInterrupt();

  }
}
if ( Numpasos == 0 ) {
  while (pasosY < NumpasosY) {
    Timer3.attachInterrupt(PasoY);;
    Serial.print("DEC");
    Serial.println(pasosY);
    if (Serial.available()) {
      String dat = Serial.readStringUntil('\n');
      DEBUG(dat);
      if (dat == "stop") {

        StopGo();
      }
    }
  }
  if (pasosY >= NumpasosY ) {
    Serial.println("finalizadoDEC");
    Serial.println("finalizadoDEC");
    Serial.println("finalizadoDEC");
    Serial.println("finalizadoDEC");
    Timer3.detachInterrupt();
  }
}
if (pasosY >= NumpasosY ) {
  Timer3.detachInterrupt();
  if (pasos >= Numpasos) {
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Timer1.detachInterrupt();

  }
  while (pasos < Numpasos) {
    Serial.print("AR");
    Serial.println(pasos);
    if (Serial.available()) {
      String dat = Serial.readStringUntil('\n');
      DEBUG(dat);
      if (dat == "stop") {

        StopGo();
      }
    }
  }

}
```

```
  if (pasos >= Numpasos ) {
    Timer1.detachInterrupt();

    while (pasosY < NumpasosY) {
      Serial.print("DEC");
      Serial.println(pasosY);
      if (Serial.available()) {
        String dat = Serial.readStringUntil('\n');
        DEBUG(dat);
        if (dat == "stop") {

          StopGo();
        }
      }
    }
    if (pasosY >= NumpasosY) {
      Serial.println("finalizadoDEC");
      Serial.println("finalizadoDEC");
      Serial.println("finalizadoDEC");
      Serial.println("finalizadoDEC");
      Timer3.detachInterrupt();
    }

  }
}
void AR10() {

  digitalWrite(DirY, LOW);
  digitalWrite(Dir, HIGH);
  while (pasos < Numpasos && pasosY < NumpasosY ) {
    Timer1.attachInterrupt(Paso);
    Timer3.attachInterrupt(PasoY);
    Serial.print("x");
    Serial.print(pasos);
    Serial.print("y");
    Serial.println(pasosY);
    if (Serial.available()) {
      String dat = Serial.readStringUntil('\n');
      DEBUG(dat);
      if (dat == "stop") {

        StopGo();

      }
    }
  }
  if ( NumpasosY == 0) {
    while (pasos < Numpasos) {
      Timer1.attachInterrupt(Paso);
      Serial.print("AR");
      Serial.println(pasos);
      if (Serial.available()) {
        String dat = Serial.readStringUntil('\n');
        DEBUG(dat);
        if (dat == "stop") {

          StopGo();
        }
      }
    }
```

```
  if (pasosY >= NumpasosY ) {
    Serial.println("finalizadoAR");
    Timer1.detachInterrupt();

  }
}
if ( Numpasos == 0 ) {
  while (pasosY < NumpasosY) {
    Timer3.attachInterrupt(PasoY);
    Serial.print("DEC");
    Serial.println(pasosY);
    if (Serial.available()) {
      String dat = Serial.readStringUntil('\n');
      DEBUG(dat);
      if (dat == "stop") {

        StopGo();
      }
    }
  }
  if (pasosY >= NumpasosY ) {
    Serial.println("finalizadoDEC");
    Timer3.detachInterrupt();
  }
}
if (pasosY >= NumpasosY ) {
  Timer3.detachInterrupt();
  if (pasos >= Numpasos) {
    Serial.println("finalizadoDEC");
    Timer1.detachInterrupt();

  }
  while (pasos < Numpasos) {
    Timer1.attachInterrupt(Paso);
    Serial.print("AR");
    Serial.println(pasos);
    if (Serial.available()) {
      String dat = Serial.readStringUntil('\n');
      DEBUG(dat);
      if (dat == "stop") {

        StopGo();
      }
    }
  }

}
if (pasos >= Numpasos ) {
  Timer1.detachInterrupt();

  while (pasosY < NumpasosY) {
    Serial.print("DEC");
    Serial.println(pasosY);
    if (Serial.available()) {
      String dat = Serial.readStringUntil('\n');
      DEBUG(dat);
      if (dat == "stop") {

        StopGo();
      }
    }
```

```
    }
    if (pasosY >= NumpasosY) {
      Timer3.detachInterrupt();
      Serial.println("finalizadoDEC");

    }

  }
}
void ARN01() {

  digitalWrite(DirY, HIGH);
  digitalWrite(Dir, LOW );
  while (pasos < Numpasos && pasosY < NumpasosY ) {
    Timer1.attachInterrupt(Paso);
    Timer3.attachInterrupt(PasoY);
    Serial.print("x");
    Serial.print(pasos);
    Serial.print("y");
    Serial.println(pasosY);
    if (Serial.available()) {
      String dat = Serial.readStringUntil('\n');
      DEBUG(dat);
      if (dat == "stop") {

        StopGo();

      }
    }
  }
  if ( NumpasosY == 0) {
    while (pasos < Numpasos) {
      Timer1.attachInterrupt(Paso);
      Serial.print("AR");
      Serial.println(pasos);
      if (Serial.available()) {
        String dat = Serial.readStringUntil('\n');
        DEBUG(dat);
        if (dat == "stop") {
           StopGo();
        }
      }
    }
    if (pasosY >= NumpasosY ) {
      Serial.println("finalizadoAR");
      Timer1.detachInterrupt();

    }
  }
  if ( Numpasos == 0 ) {
    while (pasosY < NumpasosY) {
      Timer3.attachInterrupt(PasoY);
      Serial.print("DEC");
      Serial.println(pasosY);
      if (Serial.available()) {
        String dat = Serial.readStringUntil('\n');
        DEBUG(dat);
        if (dat == "stop") {
            StopGo();
        }
      }
```

```
        }
      if (pasosY >= NumpasosY ) {
        Serial.println("finalizadoDEC");
        Timer3.detachInterrupt();

      }
    }
    if (pasosY >= NumpasosY ) {
      Timer3.detachInterrupt();
      if (pasos >= Numpasos) {
        Serial.println("finalizadoAR");
        Timer1.detachInterrupt();

      }
      while (pasos < Numpasos) {
        Serial.print("AR");
        Serial.println(pasos);

        if (Serial.available()) {
          String dat = Serial.readStringUntil('\n');
          DEBUG(dat);
          if (dat == "stop") {

            StopGo();
          }
        }
      }

    }
    if (pasos >= Numpasos ) {
      Timer1.detachInterrupt();

      while (pasosY < NumpasosY) {
        Serial.print("DEC");
        Serial.println(pasosY);
        if (Serial.available()) {
          String dat = Serial.readStringUntil('\n');
          DEBUG(dat);
          if (dat == "stop") {

            StopGo();
          }
        }
      }
      if (pasosY >= NumpasosY) {
        Serial.println("finalizadoDEC");
        Timer3.detachInterrupt();
      }

    }

}
/*

#########################################################################
####################Funcion parar en Goto ###############################
#########################################################################
*/
```

```
void StopGo() {
  Timer1.detachInterrupt();
  Timer3.detachInterrupt();
  Serial.print("Parax");
  Serial.print(pasos);
  Serial.print("y");
  Serial.println(pasosY);
  while (!Serial.available()) {}
  String data = Serial.readStringUntil('\n');

  if (data == "continue") {
    Serial.println("Sigue");
    if (pasosY < NumpasosY && pasos < Numpasos ) {

      Timer1.attachInterrupt(Paso);
      Timer3.attachInterrupt(PasoY);
    }
    if (pasos >= Numpasos ) {

      Timer3.attachInterrupt(PasoY);
    }
    if (pasosY >= NumpasosY ) {

      Timer1.attachInterrupt(Paso);
    }

  }
  else {
    pasosY = 0;
    pasos = 0;
    String data = Serial.readStringUntil('\n');
    DEBUG(data);
    parser.Init(data);
    Direccion = parser.Read_String('-'); // se guarda la dirección
    parser.Skip(1);
    String N = parser.Read_String('-'); // se guarda la dirección
    parser.Skip(1);
    String P = parser.Read_String('\n');//Se guarda el número de pasos
    Numpasos = N.toDouble() * 1000000;
    NumpasosY = P.toDouble() * 1000000;
    if (Direccion == "mod") {
      mod = 2;
      Periodo = 0;
      PeriodoY = 0;
      Numpasos = 0;
      NumpasosY = 0;
      pasos = 0;
      Ti = 0;
    }
    if (Direccion == "3") {
      mod = 3;

      Periodo = 0;
      PeriodoY = 0;
      Numpasos = 0;
      NumpasosY = 0;
      pasos = 0;
      Ti = 0;
    }
    if (Direccion == "fin") {
      Numpasos = 0;
```

```
      NumpasosY = 0;
      reset = true;
      Periodo = N.toDouble();
      PeriodoY = P.toDouble();
      Timer1.setPeriod(Periodo);
      Timer3.setPeriod(PeriodoY);
    }
    if (Direccion == "11" ) {
      AR11();
    }
    //Con el rojo al lado de Vmot gira con el mismo del movimiento del
reloj
    if (Direccion == "00") {
      ARN00();
    }
    if (Direccion == "10" ) {

      AR10();
    }

    //Con el rojo al lado de Vmot gira con el mismo del movimiento del
reloj
    if (Direccion == "01") {

      ARN01();
    }

  }
  parser.Reset();
  Serial.flush();
}
/*

######################################################################
#################### FOLLOW  #########################################
######################################################################
*/
void Follow() {

  if (reset == true) {
    Timer3.detachInterrupt();
    Serial.end();
    Serial.begin(57600);

    reset = false;
    Periodo = 0;
    Numpasos = 0;
    pasos = 0;
    Ti = 0;
    Serial.print("Periodo: ");


  }
  if (Periodo == 0) {
    cogerPeriodoF();
  }
  else {
    ARF();
  }
```

```
}
void cogerPeriodoF() {

  if (Serial.available() > 4)
  {

    String data = Serial.readStringUntil('\n');
    DEBUG(data);
    parser.Init(data);
    String f = parser.Read_String('-');
    parser.Skip(1);
    String P = parser.Read_String('-');
    parser.Skip(1);
    String T = parser.Read_String('\n');
    if (f == "mod") {
      mod = 1;
      Serial.print("periodo:");
    }
    if (f == "3") {
      mod = 3;
    }
    if (f == "sur") {
      digitalWrite(Dir, LOW);
      Periodo = P.toDouble();
      Timer1.initialize(Periodo);
      Timer3.initialize(1000000);
      Serial.println(Periodo);
      Time = T.toDouble();

    }
    else {
      digitalWrite(Dir, HIGH);
      Periodo = P.toDouble();
      Timer1.initialize(Periodo);
      Timer3.initialize(1000000);
      Serial.println(Periodo);
      Time = T.toDouble();
    }
  }
  parser.Reset();
  Serial.flush();
}
/*
   ##############################
   ##############################
   ##############################

  // dirige el movimiento de los motores

*/
void Tiempo() {
  Ti++;
}
//método para ir AR-;
void ARF() {

  Timer1.attachInterrupt(Paso);
  Timer3.attachInterrupt(Tiempo);
  Serial.print("AR");
  Serial.println(pasos);
  if (Serial.available()) {
```

```
    String dat = Serial.readStringUntil('\n');
    DEBUG(dat);
    if (dat == "stop") {

      StopFoll();
    }
  }

  if (Ti >= Time + 1) {
    Timer1.detachInterrupt();
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    Serial.println("finalizadoAR");
    reset = true;
  }
}

/*
   ##############################
   ##############################
   ##############################

  // Para el método follow y el motor

*/


void StopFoll() {
  Timer1.detachInterrupt();
  Timer3.detachInterrupt();
  Serial.print("Parax");
  Serial.print(pasos);
  Serial.print("y");
  Serial.println(pasosY);
  while (!Serial.available()) {}
  String data = Serial.readStringUntil('\n');
  DEBUG(data);
  parser.Init(data);
  String f = parser.Read_String('-');
  parser.Skip(1);
  String P = parser.Read_String('-');
  parser.Skip(1);
  String T = parser.Read_String('\n');
  if (data == "continue") {
    Timer1.attachInterrupt(Paso);
    Timer3.attachInterrupt(Tiempo);
  }
  else {
    Ti = 0;
    Periodo = P.toDouble();
    Timer1.setPeriod(Periodo);
    Time = T.toDouble();
    ARF();
  }
}
/*
##########################################################################
##########      Manual            ################################
##########################################################################
#
```

```
*/
void Manual() {
  if (Serial.available() > 4)
  {
    String data = Serial.readStringUntil('\n');
    DEBUG(data);
    parser.Init(data);
    Direccion = parser.Read_String('-'); // se guarda la dirección
    parser.Skip(1);
    String N = parser.Read_String('\n'); // se guarda la dirección
    parser.Skip(1);
    Periodo = N.toDouble();
    pasos = 0;
    pasosY = 0;
    if (Direccion == "0") {
      ARNegativo();
    }
    if (Direccion == "1") {
      ARPositivo();

    }
    if (Direccion == "2") {
      DECNegativo();

    }
    if (Direccion == "3") {
      DECPositivo();

    }
    if (Direccion == "mod") {
      if ( N == "goto" ) {
        mod = 1;
      }
      if ( N == "follow" ) {
        mod = 2;
      }
      if ( N == "modo" ) {
        mod = 0;
      }
    }
  }
}
void ARPositivo() {
  digitalWrite(Dir, HIGH);
  Timer1.initialize(Periodo);
  Timer1.attachInterrupt(Paso);
  while (!Serial.available()) {
    Serial.println(pasos);
  }
  String data = Serial.readStringUntil('\n');
  DEBUG(data);
  if (data == "para") {
    Timer1.detachInterrupt();
  }


}
void ARNegativo() {
  digitalWrite(Dir, LOW);
  Timer1.initialize(Periodo);
  Timer1.attachInterrupt(Paso);
```

```
  while (!Serial.available()) {
    Serial.println(pasos);
  }
  String data = Serial.readStringUntil('\n');
  DEBUG(data);
  if (data == "para") {
    Timer1.detachInterrupt();
  }


}
void DECPositivo() {
  digitalWrite(DirY, HIGH);
  Timer3.initialize(Periodo);
  Timer3.attachInterrupt(PasoY);

  while (!Serial.available()) {
    Serial.println(pasosY);
  }
  String data = Serial.readStringUntil('\n');
  DEBUG(data);
  if (data == "para") {
    Timer3.detachInterrupt();
  }


}
void DECNegativo() {
  digitalWrite(DirY, LOW);
  Timer3.initialize(Periodo);
  Timer3.attachInterrupt(PasoY);
  while (!Serial.available()) {
    Serial.println(pasosY);
  }
  String data = Serial.readStringUntil('\n');
  DEBUG(data);
  if (data == "para") {
    Timer3.detachInterrupt();
  }


}
```