Universidad de Jaén

Desarrollo de software para Seguidor Solar

Trabajo Fin de Máster

MASTER EN SOSTENIBILIDAD Y EFICIENCIA ENERGETICA EN LOS EDIFICIOS Y EN LA INDUSTRIA

Alumna: Zarin López Millanes

Tutor: Prof. D. Pedro J. Casanova Pelaez

Noviembre, 2015

Universidad de Jaén

TRABAJO FIN DE MASTER

DESARROLLO DE SOFTWARE PARA SEGUIDOR SOLAR

MASTER EN SOSTENIBILIDAD Y EFICIENCIA ENERGETICA EN LOS EDIFICIOS Y EN LA INDUSTRIA

Vº Bº

Vº Bº

Alumna: Zarin López Millanes

Tutor: Pedro D. Casanova Peláez

Noviembre, 2015

"Cualquier destino, por largo y complicado que sea, consta en realidad de *un solo momento:* el momento en que el hombre sabe para siempre quién es"

J. L. Borges

Gracias a mi familia, profesores y amigos por su apoyo incondicional.

TABLA DE CONTENIDO

INTRODUCCION

El siguiente proyecto final de máster consiste en la realización de un software para control de un sistema de radiación solar., desarrollado por medio de la plataforma ARDUINO y una interfaz gráfica en entorno MATLAB. El programa será capaz de controlar la trayectoria de un sistema de medida de radiación solar, obtener una señal y enviarla a un ordenador para ser almacenada.

Para facilitar la realización de este proyecto se ha utilizado equipo del Departamento de Ingeniería Mecánica y Minera de la Escuela Politécnica Superior de la Universidad de Jaén. Los equipos son los siguientes:

- Prototipo de Seguidor Solar
- Placa con microcontrolador ARDUINO

Para este fin, en este documento se realiza una introducción a la energía solar, se describe el funcionamiento del Seguidor Solar así como el funcionamiento de los sensores de radiación solar., exponiendo la teoría básica sobre células solares.

Para continuar se realiza una introducción a la plataforma ARDUINO así como al sistema de programación que será el responsable de los movimientos de los motores. Para permitir la comunicación con la computadora se utilizará el puerto USB simulando un puerto serial COM. La interfaz gráfica está diseñada a través de la opción GUIDE de Matlab. Se detalla el aplicativo desarrollado, realizando un barrido a las opciones que permite utilizar y cómo manejarlo.

El proyecto se ha estructurado de la siguiente forma:

- Se ha estudiado la documentación disponible, principalmente de los trabajos publicados sobre los equipos, concretamente proyectos de fin de carrera y manuales.
- Se hizo una búsqueda adicional de información a través de otros medios para comprender sobre la programación que se llevaría a cabo.
- Se evaluó la parte electrónica y de potencia comprobando su buen funcionamiento
- Se probó experimentalmente la comunicación del controlador Arduino para el movimiento de los motores y la recepción de las señales de entrada de optoacoplador y final de carrera.
- Se realizó la programación del controlador Arduino con las diferentes opciones de funcionamiento para el Seguidor: Manual y Secuencia automática
- Se probó la comunicación Arduino con Matlab para el envío y recibo de información por puerto serie.
- Se realizó la intefaz de GUIDE en Matlab para crear una comunicación más amigable con el usuario.

OBJETIVO

El objetivo de este proyecto es la programación de una placa Arduino equipada con sensores y que sea capaz de interaccionar con los motores que darán el movimiento al seguidor solar. Esta placa deberá ser capaz de controlar los movimientos y posición de los motores para conseguir la mayor precisión, así mismo enviará la información obtenida por el sensor de radiación por medio del puerto serial a un ordenador donde se graficarán los datos obtenidos y se almacenarán para futuras aplicaciones e investigaciones.

- La plataforma base para el desarrollo del proyecto, se está utilizando Arduino, concretamente Arduino Duemilanove.
- El lenguaje de programación utilizado en el proyecto es lenguaje de bajo nivel.

TEORIA

Introducción

El modelo actual de desarrollo se ha basado históricamente en el uso y explotación de los recursos energéticos de origen fósil. Estos combustibles han suministrado las fuentes energéticas del desarrollo económico del planeta, de manera intensiva desde el nacimiento de la Revolución Industrial hasta nuestros días.

Este acelerado desarrollo, sin embargo, también ha generado voces de alerta sobre impactos ambientales que genera la explotación de los recursos que, por su lenta velocidad de generación respecto de su explotación, son clasificados como no renovables. Los impactos ambientales que estos combustibles generan (cambio climático, lluvia ácida, capa de ozono), aunado a la creciente incertidumbre respecto del suministro de combustibles fósiles, ha obligado a buscar un nuevo modelo de desarrollo (Desarrollo Sostenible), sin comprometer las necesidades de futuras generaciones.

En este ámbito, España ha sido pionera en la búsqueda de nuevas alternativas de suministro energético, en especial, en la energía solar.

Energía Solar

El sol es una esfera compuesta por gases a alta temperatura que está situado a una distancia aproximada de $1.5 \cdot 10^8$ km de la tierra. Se estima que la temperatura de la fotósfera solar es la misma que "la temperatura efectiva del cuerpo negro" la cual es 5762. La fotosfera es la parte más exterior del sol, y es considerada opaca, por lo que emite la mayor parte de la radiación.

La energía emitida por un cuerpo negro viene dada por la Ley de Stefam-Boltzman. Esta ley sostiene que la energía radiada por un cuerpo negro es directamente proporcional a la cuarta potencia de su temperatura. La radiación solar que llega a la tierra se considera constante y depende de la temperatura del cuerpo negro, de la distancia de la tierra al sol, la relación de áreas entre la esfera solar y la terrestre y la constante de Stefam-Boltzman.

Al atravesar la atmósfera, los rayos solares pueden sufrir muchas variaciones antes de alcanzar la superficie terrestre. Reflexión, absorción y difusión son los tres factores que afectan a la radiación a su paso por la atmósfera. La radiación solar puede llegar a la tierra de tres maneras diferentes: directa, difusa y reflejada. La radiación directa es la que va a ser útil para obtener energía, y es aquella que proviene directamente de la esfera solar sin sufrir ningún cambio.

Las distintas tecnologías solares existentes son:

- Energía solar térmica: utilizada para producir agua caliente para uso sanitario y calefacción.

- Energía solar fotovoltaica utilizada para la producción de electricidad mediante paneles fotovoltaicos que generan energía eléctrica a partir de la radiación.
- Energía solar termoeléctrica: utilizada para la producción de electricidad mediante un ciclo termodinámico convencional a partir de un fluido calentado por la radiación solar térmica.
- Energía solar híbrida: combina la energía solar con otra energía.
- Energía eólico-solar: funciona con el aire calentado por el sol, que sube por una chimenea donde están los generadores.

Geográficamente, España, es uno de los sectores más desarrollados debido a sus características climáticas.

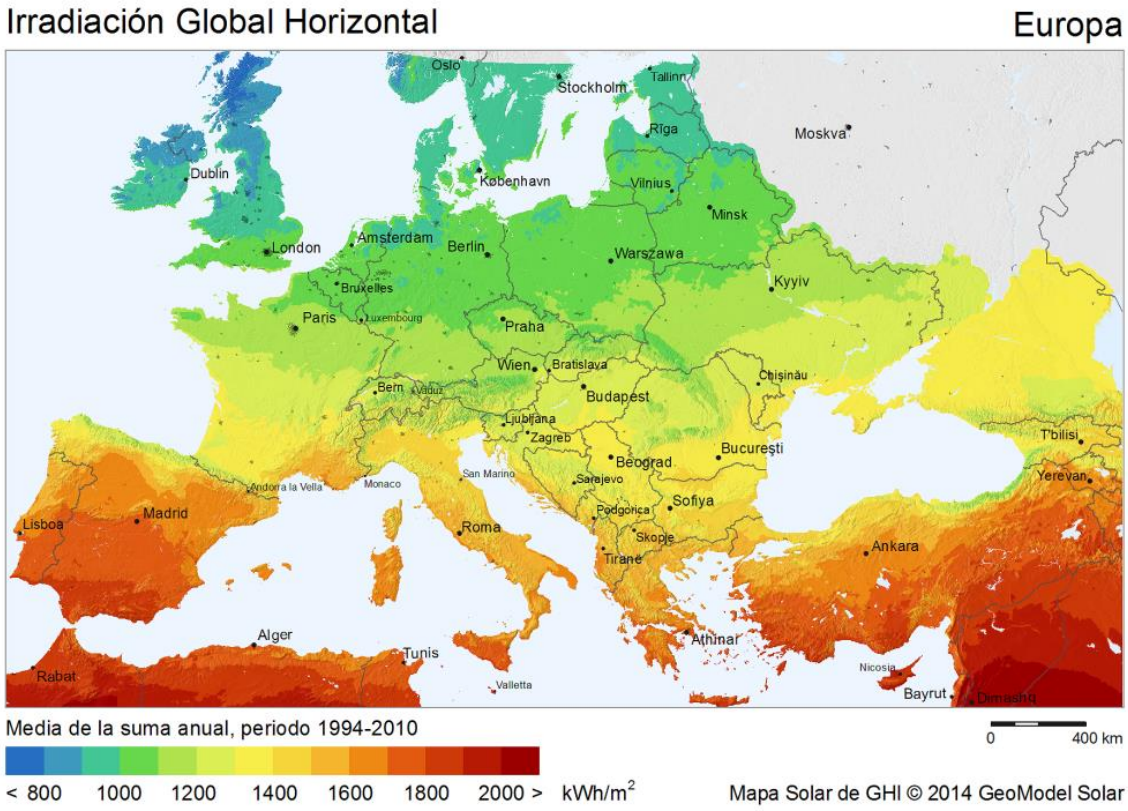


Figura 1. Mapa de irradiación solar en Europa

Energía Solar Fotovoltaica

La energía solar fotovoltaica consiste en la obtención de electricidad directamente a partir de la radiación solar mediante un dispositivo semiconductor denominado célula fotovoltaica.

La célula fotovoltaica es un dispositivo semiconductor capaz de convertir los fotones procedentes del sol en electricidad de una forma directa e inmediata, es decir, es el dispositivo responsable del efecto fotovoltaico. Cuando incide la luz sobre una célula, se produce un efecto caótico en la unión P-N del semiconductor que libera electrones, dando lugar a una corriente eléctrica.

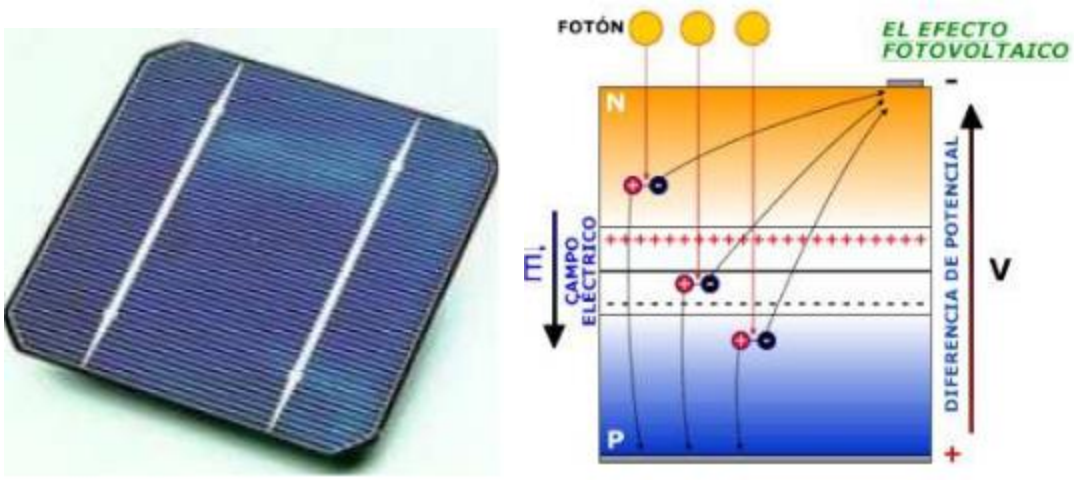


Figura 2. Célula fotovoltaica y efecto fotovoltaico

Este tipo de energía se usa para alimentar innumerables aparatos autónomos. Debido a la creciente demanda de energías renovables, la fabricación de células e instalaciones fotovoltaicas ha avanzado considerablemente en los últimos años.

Según un estudio publicado en 2007 por el World Energy Council, para el año 2100 el 70% de la energía consumida será de origen solar según informes de Greenpeace.

Existen diferentes tipos de instalaciones fotovoltaicas, entre las que se encuentran:

- Instalación solar fotovoltaica fija: Se denomina de esta forma a las plantas fotovoltaicas cuyos paneles permanecen en la misma posición a lo largo del tiempo.
- Instalación solar fotovoltaica de un eje: Aquí se empieza a utilizar el concepto de Seguidor Solar, una máquina con una parte fija y otra móvil que dispone de una superficie de captación solar lo más perpendicular al sol posible a lo largo del día y dentro de sus rangos de movimiento. Estos seguidores solo gozan de un grado de libertad en su movimiento.

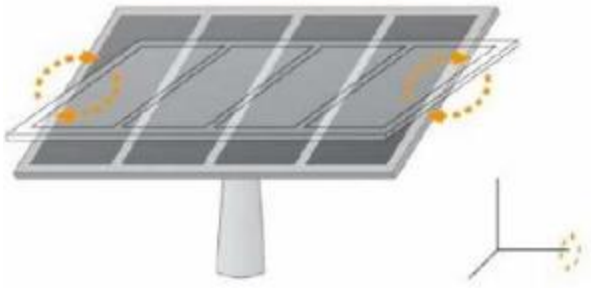


Figura 3. Seguidor Solar de un eje

- Instalación solar fotovoltaica de dos ejes: Se trata de seguidores con dos grados de libertad, capaces de hacer un seguimiento solar más preciso.

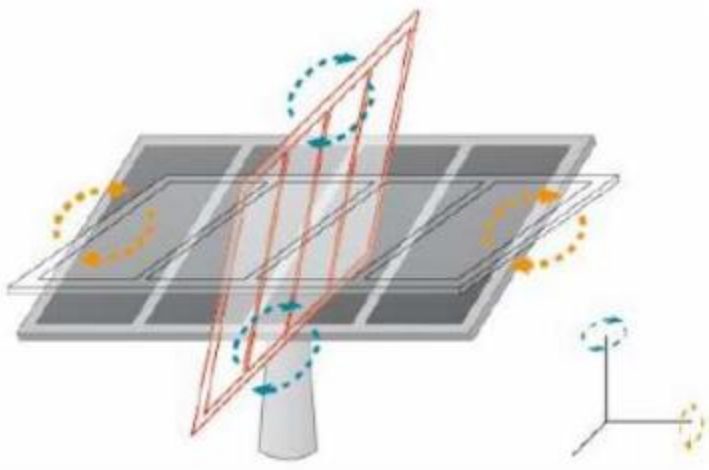


Figura 4. Seguidor Solar de dos ejes

Este tipo de sistemas siguen al sol tanto en dirección como en elevación, lo que implica tener dos actuadores, para variar la inclinación del panel de forma horizontal y vertical. Al orientar los paneles fotovoltaicos de forma perpendicular al sol se incrementa la energía recibida.

Utilizando el seguimiento, la energía total recibida en un día puede ser del orden de un 35% mayor que para el mismo colector estático. El seguimiento se puede realizar por distintos métodos:

- Seguimiento por reloj solar: este tipo está sujeto a la unidad de tiempo de 24 horas, variando su posición respecto al ciclo de esta unidad, con un seguimiento efectivo de 12 horas.
- Seguimiento por sensores: es el que permite la detección o medida que falta en el correcto ángulo entre la radiación solar y la superficie del panel solar.
- Seguimiento por coordenadas calculadas: este tipo de seguimiento sigue la trayectoria del sol entre cada posición mediante el cálculo de sus coordenadas astronómicas, no precisa de la presencia de radiación, los sistemas de coordenadas son inmunes a los días nublados y otro tipo de circunstancias que puede producir errores; como por ejemplo los destellos.

Equipos de medición de radiación existentes en la universidad

En esta parte se abordará la información disponible, principalmente de los trabajos publicados sobre el seguidor solar y los sensores para medir de radiación existentes en el departamento de Ingeniería Mecánica y Minera de la Universida de Jaén.

Sistema de medida de radiación solar de dos ejes

El mecanismo para el sistema de medida está basado en la misma idea de un seguidor solar. La única diferencia reside en que lleva incorporado un sensor de radiación solar en vez de un panel fotovoltaico.

Gracias a la configuración del seguidor solar con dos ejes rotacionales perpendiculares, el sensor es capaz de alcanzar, en teoría, infinitos puntos de medición. Cada eje es accionado mediante un motor paso a paso que conecta el tornillo sinfín de su extremo con el engranaje fijado al eje.

Para aumentar el par transmitido por el motor con el fin de que el seguidor sea capaz de realizar los movimientos con carga, el seguidor incluye un sistema de engranaje-tornillo sinfín. Cada eje cuenta con rodamientos, engranaje, enconder de 72 dientes y casquillos de unión.

La caja contenedora es capaz de proteger al sistema de la lluvia y de agentes externos. El grado de protección IP de la caja es un IP65, es decir, protegido contra la penetración de cuerpos sólidos de más de 1mm y protegido contra la entrada perjudicial de agua en todas direcciones.

Esta estructura se basa en dos soportes ajustados al eje horizontal, esto para dar estabilidad suficiente a la estructura, es decir, su peso es suficiente para que la inercia en el movimiento y paro de los motores no provoque que el sensor se mueva a una posición no deseada.

Para obtener una retroalimentación y conocer la posición del seguidor, cada eje tiene instalado un encoder de 72 dientes que indica un avance de 5º por cada diente, esto se lee a través de un optoacoplador que va enviando un valor para hacer el conteo adecuado. El encoder se ha fabricado con un saliente para activar el final de carrera.

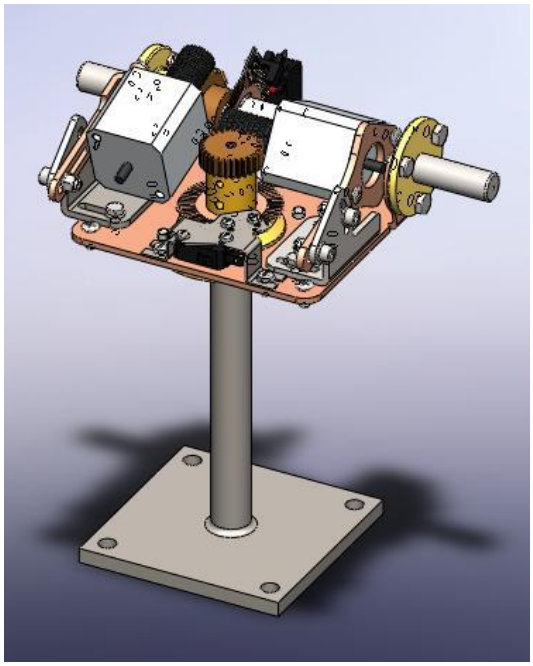


Figura 5. Montaje del mecanismo del seguidor solar

Sensor de radiación solar

Este sensor es un mecanismo diseñado por un equipo de investigadores del Departamento de Ingeniería Mecánica y Minera de la Universidad de Jaén. Actualmente está patentado por la

Universidad desde 2008. Del documento de solicitud internacional de patente se obtiene la siguiente descripción:

"Permite medir la radiación incidente global recibida en cada elemento diferencial de la superficie de una semiesfera, en todo momento, mediante una estructura simple, compacta y carente de elementos electromecánicos para la obtención de dicha medida. Está constituido por una superficie semiesférica, sobre la que se establecen unas células de manera que cada una presenta un ángulo acimutal terrestre e inclinación diferentes, habiéndose previsto que junto a cada una de las células esté asociado un sensor de temperatura, cuya función es la de corregir derivas térmicas en las señales de medida, de manera que las señales de ambos elementos se conectan a una etapa de amplificación para su conexión a un sistema de adquisición, evaluación y gestión de datos captados, tal como un ordenador, a través del que se podría establecer el control del sistema solar que se encuentren junto al dispositivo."

El objetivo de este sensor es medir la radiación en todo el hemisferio, a través de 31 pequeñas celdas fotovoltaicas instaladas en una estructura metálica en forma de semiesfera.

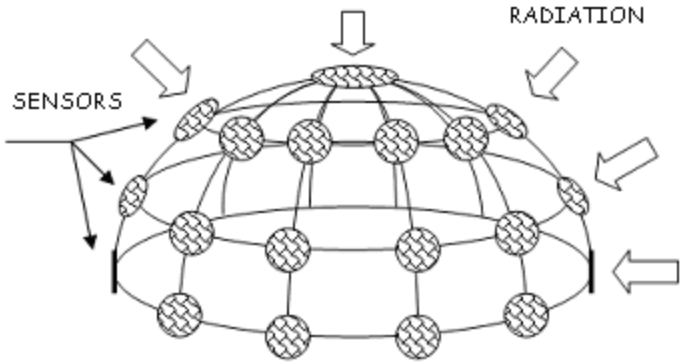


Figura 6. Sensor de radiación solar

Piranómetro

Un piranómetro es un instrumento para medir la radiación solar a una superficie plana. Se constituye por una pila termoeléctrica contenida en un alojamiento con dos semiesferas de cristal. La pila termoeléctrica está constituida por una serie de termopares colocados horizontalmente, cuyos extremos están soldados con unas barras de cobre verticales solidarias a una placa de latón maciza. El conjunto está pintado con un barniz negro, para absorber la radiación. El flujo de calor originado por la radiación se transmite a la termopila, generándose una tensión eléctrica proporcional a la diferencia de temperatura entre los metales de los termopares.

El piranómetro de la universidad es el modelo CMP3 de la casa Kipp & Zonen. Su construcción mecánica se basa en una sola cúpula, dimensiones de carcasa más pequeñas y no tiene cartucho de secado.

Este sensor se montará sobre el seguidor solar para poder medir la radiación en diferentes puntos, con el fin de hacer una interpolación con los datos del sensor de radiación y verificar si se requiere agregar más sensores fotovoltaicos.

Figura 7. Piranómetro

CONTROL DE SEGUIDOR SOLAR

El seguidor solar debe ser controlado para que el sensor esté posicionado directamente hacia el sol y de esta forma realizar la medición de la radiación solar de forma continua. Para el control de este seguidor se ha decidido utilizar el controlador Arduino y el programa Matlab para crear una interfaz gráfica y la adquisición de datos.

Controlador Arduino

Arduino es una plataforma electrónica abierta para la creación de prototipos y aplicaciones basadas en microcontroladores, donde tanto el software como el hardware son libres, flexibles y fáciles de usar.

Arduino puede tomar información del entorno que lo rodea a través de sus pines de entrada, a los que se le puede conectar una amplia gama de sensores y transductores. Del mismo modo, puede actuar sobre dicho entorno mediante sus pines o líneas de salida. Con ellas es posible controlar luces, motores, relés, altavoces y todo tipo de actuadores.

El microcontrolador de la placa Arduino se programa mediante el lenguaje de programación Arduino. Es un lenguaje de alto nivel con unas sentencias y sintaxis muy similares a las de lenguaje C. También posee un entorno de desarrollo basado en Processing, que permite la edición de un programa con el lenguaje Arduino, su verificación, compilación y la grabación sobre el controlador. Dicho entorno es código abierto y está disponible para plataformas Windows, Mac, etc. Una vez grabado el programa en la memoria del controlador, éste se ejecuta sin necesidad de estar conectado a un ordenador. Tenemos así un proyecto hardware/software totalmente autónomo.

Se ha elegido utilizar la plataforma Arduino para el control del seguidor porque proporciona una facilidad de uso muy grande y la programación que exigen es bastante sencilla e intuitiva. A diferencia con otros microcontroladores, esta plataforma facilita la configuración del microcontrolador gracias a unas librerías que podemos encontrar en la página oficial de Arduino.

Hardware

El hardware de Arduino está basado en el procesador ATMEL, un chip sencillo y de bajo coste que permite el desarrollo de múltiples diseños. Según ha ido avanzando el tiempo, el

procesador se ha ido actualizando hasta llegar al modelo Atmega1280 con mejores prestaciones, como por ejemplo más memoria flash.

Figura 8. Placa Arduino

Al ser hardware libre, tanto su diseño como distribución, es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido licencia alguna.

Una de las peculiaridades de la plataforma Arduino es que poseen placas que ayudan a la configuración del microcontrolador. En la figura anterior tenemos un ejemplo de una de las placas más sencillas y más utilizadas, esta placa incorpora un conector USB ya instalado, un regulador de tensión, un conversor puerto serie/USB, sistemas de protección y una fácil accesibilidad a las entradas y salidas del microcontrolador.

Para este proyecto se ha elegido una placa Arduino Duemilanove ya que esta ofrece una gran compatibilidad con todos los módulos que exiten en el mercado enfocados a Arduino, es de bajo coste y fácil de programar. En la siguiente tabla se muestran las características de la placa Duemilanove:

Microcontrolador	ATmega 328
Voltaje (recomendado)	7-12V
Digital E/S	14 (6 salidas PWM)
Entradas analógicas	6
Corriente por I/O	40 mA
Memoria Flash	32 kB con 2 kB para el bootloader
SRAM	2 kB
EEPROM	1 kB
Velocidad de reloj	16 MHz

Dicha placa será utilizada para el control de los motores a través de dos salidas digitales y dos entradas digitales, así como también será utilizada para leer la información del sensor de radiación

por medio de una entrada analógica. La siguiente tabla muestra la configuración de entradas y salidas del controlador.

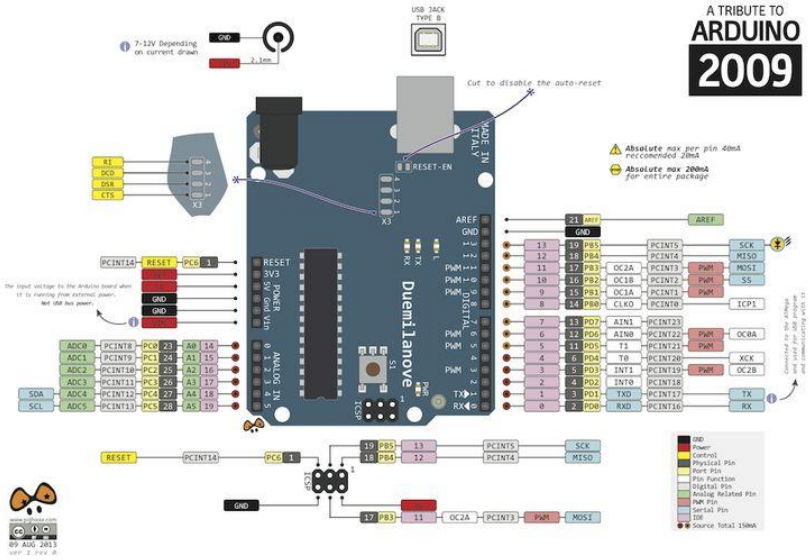


Figura 9. Diseño de placa Arduino Duemilanove

Entradas Digitales	
Dirección de motor vertical	Pin 6
Pulsos de motor vertical	Pin 7
Dirección de motor horizontal	Pin 10
Pulsos de motor horizontal	Pin 11
Salidas Digitales	
Sensor optoacoplador motor vertical	Pin 8
Sensor final de carrera motor vertical	Pin 9
Sensor optoacoplador motor horizontal	Pin 12
Sensor final de carrera motor horizontal	Pin 13
Entradas Analógicas	
Sensor de radiación solar	A0

El puerto USB permite una comunicación serie con el ordenador mediante el estándar de los controlades USB COM, sin necesidad de controlador externo. La placa avisa que la comunicación se está llevando a cabo con un parpadeo de los leds Rx y Tx.

El conector plug hembra de 2.1 mm lo podemos usar para alimentar la placa externamente, evitando así el uso del USB, si el programa ya está cargado en el microcontrolador, no necesitamos el ordenador para que funcione, basta con alimentarlo.

Software

Para programar la placa Arduino, se requiere descargar el software de la pagina web (http://www.arduino.cc/) en el sistema operativo deseado. Este software, como ya se ha mencionado antes, es fácil de usar y bastante intuitivo. Es de licencia con distribución y uso gratuito. En esta misma página podemos acceder a un foro en que se puede ser ayudado por la gran comunidad de usuario de Arduino para cualquier duda con la programación.

El entorno de desarrollo Arduino lo constituye un editor de texto, donde se plasmará el código; una consola de texto, un área de mensajes y la barra de herramientas con sus menús.

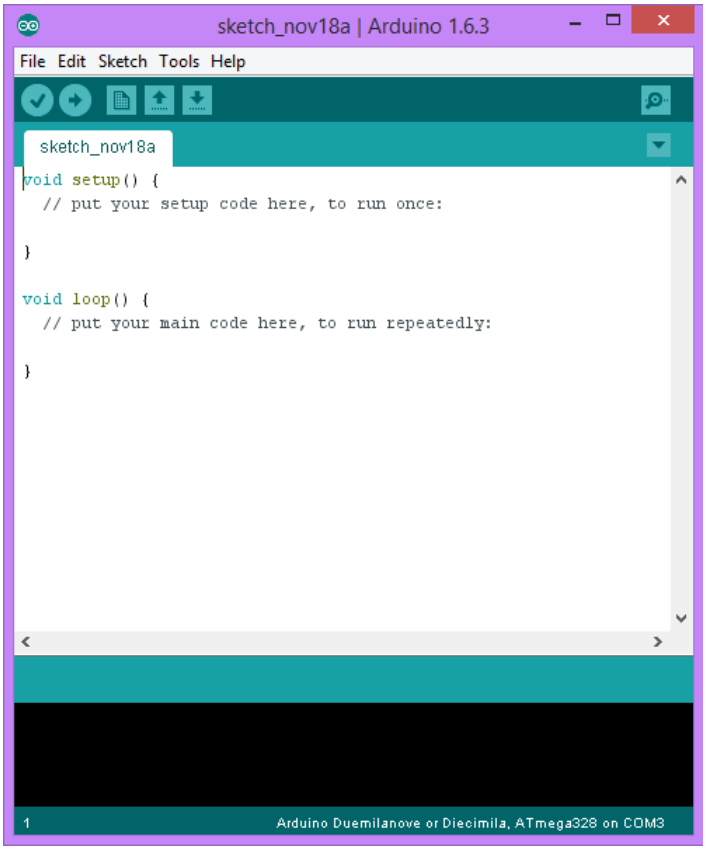


Figura 10. Pantalla principal de software Arduino

Estructura de un programa

Arduino se programa en C++, admitiendo la gran mayoría de librerías usadas en C++ y todas sus estructuras básicas. Todo sketch tiene siempre la misma estructura:

```
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

- Setup(). La función *setup()* se establece en cuanto se inicia el sketch. Se usa para iniciar variables declaradas anteriormente, asignar pines, cargar librerías, etc. Esta función solo se ejecuta una vez desde que se conecta la placa al ordenador o se reinicia.
- Loop(). Una vez inicializados y preparados todos los valores y funciones necesarias, esta función se ejecuta sucesivamente hasta la desconexión de la placa. Es por esta función que controlamos activamente la placa.

Una vez visto la estructura típica de cada programa, las funciones específicas que posteriormente se emplearán en el desarrollo del software se resumen en la siguiente tabla:

NOMBRE	DESCRIPCION	SINTAXIS	PARAMETROS	DEVOLUCIÓN
Pin mode()	Configura el pin especificado como entrada o salida	pinMode(pin,modo)	Pin: número de pin a definir Modo: INPUT (entrada), OUTPUT (salida)	-
Digital Write()	Escribe un valor eléctrico en el pin seleccionado	digitalWrite(pin,valor)	Pin: número de pin Valor: HIGH o LOW	-
Digital Read()	Lee el valor eléctrico del pin seleccionado	digitalRead(pin)	Pin: número de pin a leer	HIGH o LOW
Delay()	Pausa al programa por un determinado espacio de tiempo	delay(tiempo)	Tiempo en milisegundos	-
Analog Read()	Lee el valor de tensión en el pin analógico.	analogRead(pin)	Pin: número de pin a leer	Entero entre 0 y 1023

Comunicación de Arduino por puerto serie

Los puerto serie son la forma principal de comunicación de la placa Arduino con el ordenador. Un puerto es el nombre genérico para denominar las interfaces, físicas o virtuales, que permiten la comunicación entre dos ordenadores o dispositivos.

Un puerto serie envía la información mediante una secuencia de bits. Para ello se necesitan al menos dos conectores para realizar la comunicación de datos, RX (recepción) y TX (transmisión).

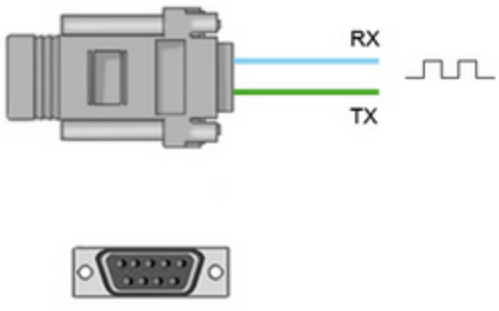


Figura 11. Comunicación puerto serie

Un ordenador convencional dispone de varios puertos de serie. Los más conocidos son el popular USB (Universal Serial Port) y el ya casi olvidado RS-232. En ocasiones se refiere a los puertos serie como UART (Universally Asynchronous Receive/Transmitter), ésta es una unidad que incorporan ciertos procesadores, encargada de realizar la conversión de los datos a una secuencia de bits y transmitirlos o recibirlos a una velocidad determinada.

Prácticamente todas las placas Arduino disponen al menos de una unidad UART. La placa Arduino Duemilanove dispone de 4 unidades UART TTL 0V/5V. Físicamente, los puertos serie están unidos a distintos pines de la placa arduino. Lógicamente, mientras usamos los puertos de serie no podemos usar como entradas y salidas digitales los pines asociados con el puerto serie en uso.

Para realizar la conexión mediante puerto serie es necesario conectar la placa Arduino empleando la misma interface que se utiliza para programar. Las funciones utilizadas para la programación del uso de comunicación por puerto serie se resumen en la siguiente tabla:

NOMBRE	DESCRIPCION	SINTAXIS	PARAMETROS	DEVOLUCIÓN
Begin()	Establece la velocidad para la transmisión de datos	Serial.Begin(baudios)	Baudios: velocidad en bits/seg	-
Available()	Informa del número de bytes disponibles para ser leídos	Serial.Available()	-	-
Read()	Recoge los datos del puerto serie	Serial.Read()	-	El primer byte disponible recibido por PS
Flush()	Vacía el búfer de entrada de datos al puerto serie	Serial.Flush()	-	-
Print()	Escribe los datos por puerto serie	Serial.print(valor) Serial.print(valor,formato)	Valor: valor que queremos imprimir	-

	como código ASCII		Formato: especifica el número de la base para entero o el número de posiciones para float	
Println()	Escribe los datos por puerto serie en código ASCII añadiendo un retorno de carro y un avance de línea	Serial.println(valor) Serial.println(valor,formato)	Valor: valor que queremos imprimir Formato: especifica el número de la base para entero o el número de posiciones para float	
Write()	Escribe datos binarios en el puerto serie	Serial.Write(valor)	Valor: puede ser un valor a enviar como un solo byte, un *string* o un *array*	-

El software de Arduino, IDE Standard, dispone de un monitor de puerto serie que nos permite enviar y recibir fácilmente información a través del puerto serie. Su uso es muy sencillo, y dispone de dos zonas, una que muestra los datos recibidos y otra para enviarlos.

Debido a su sencillez, este monitor de puerto serie no es suficiente para completar el control del seguidor solar ya que solo permite enviar y recibir información pero no podemos almacenar o graficar la información recibida desde el sensor de radiación. Por este motivo se ha buscado una alternativa para elaborar la segunda parte del control del seguidor.

Adquisición de datos por Matlab

Para la recepción, gráfica y almacenamiento de los datos se ha elegido el programa Matlab. Hay múltiples programas con los que se podría haber desarrollado la misma aplicación, como por ejemplo Labview, capaz de realizar una programación de objetos mucho más intuitiva. La elección de Matlab fue principalmente por la capacidad de cálculo frente a la comodidad que pudiese otorgar Matlab GUIDE para la creación de aplicación.

MATLAB es un lenguaje de programación y un entorno de desarrollo combinado. Como lenguaje de programación, es de alto nivel y orientado a problemas de cálculo técnico intensivo. Como entorno de desarrollo, está orientado a desarrollar aplicaciones y prototipos rápidos, en un entorno interpretado, que permite el desarrollo de aplicaciones técnicas de manera más eficiente que programando en lenguajes tradiciones como C, C++ o Fortran.

El sistema de desarrollo de Matlab trabaja interpretando las órdenes dadas al sistema, tanto desde la consola interactiva, como através de ficheros de texto. El sistema de Matlab no diferencia entre rutinas internas de la aplicación y rutinas creadas por el usuario.

Todas las funciones se deben declarar en el fichero abierto en el momento, o accederá a través de un fichero con el mismo nombre. Esto permite ir creando funciones y saltos del programa a través de ficheros.

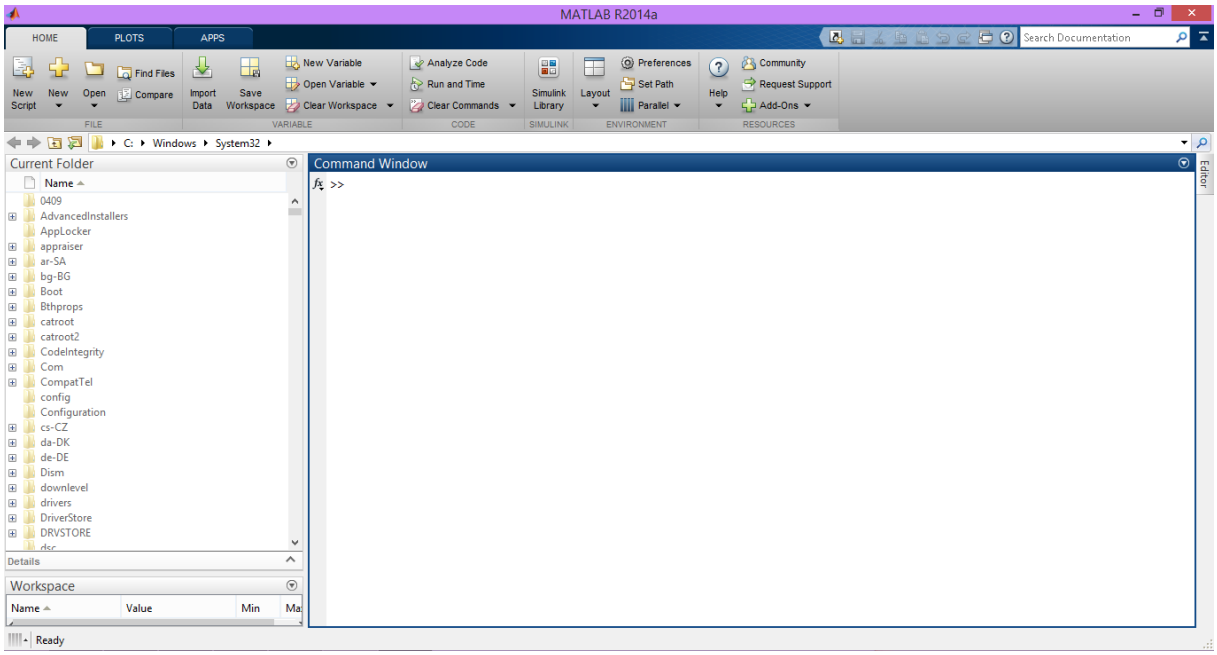


Figura 12. Pantalla principal de Matlab.

Además de los sistemas de programación descritos, Matlab incorpora un sistema de creación de aplicaciones rápida, basada en eventos y formularios GUI, similares a lo que han sido hasta las fechas los entornos "visuales".

Matlab GUIDE

Matlab Guide es un entorno de programación que ofrece Matlab para poder realizar y ejecutar programas de simulación a medida de forma simple. Tiene las características básicas de todos los programas visuales como Visual Basic o Visual C++. Este sistema, permite la creación gráfica de entornos de formularios, construyendo la estructura de programación que los dibuja en la pantalla y permitiendo activar una función del evento generado.

Este entorno permite la creación de interfaces gráficas de usuario, con cierta comodidad, programando únicamente las llamadas a los eventos producidos.

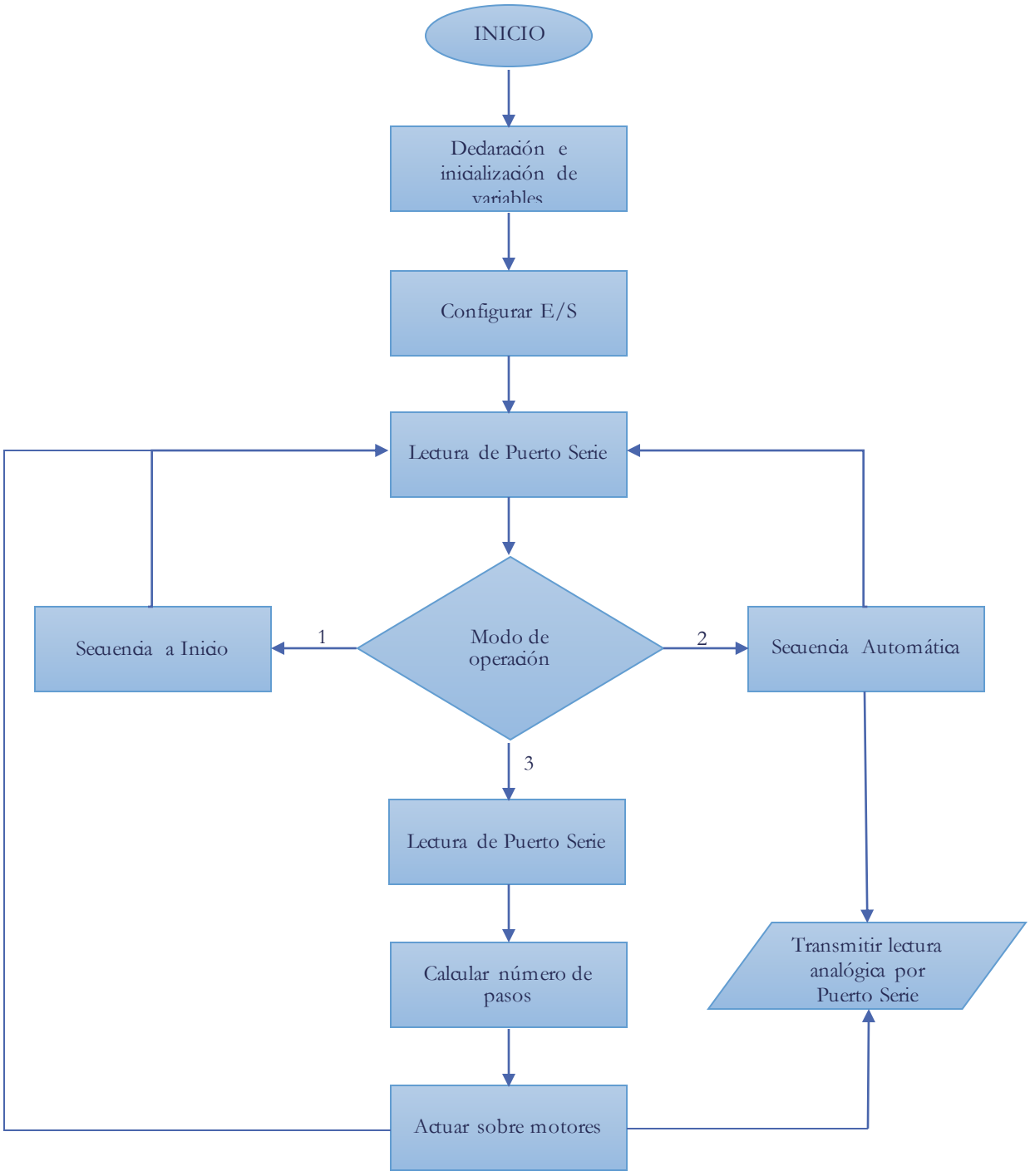
Una aplicación GUIDE consta de dos archivos: *.m* y *.fig*. El archivo *.m* es el que contiene el código con las correspondencias de los botones de control de la interfaz y el archivo *.fig* contiene los elementos gráficos.

Cada vez que se adiciona un nuevo elemento a la interfaz gráfica, se genera automáticamente código en el archivo *.m*.

DESARROLLO DE PROGRAMA ARDUINO

En esta sección de la memoria del proyecto se detallará el desarrollo de la programación de la placa Arduino, la cual es responsable del control del movimiento del seguidor solar.

El sistema tiene que controlar los dos movimientos posibles del seguidor. El sistema tiene la opción de operar en dos modos diferentes: Automático y Manual. Se ha realizado un diagrama de bloques para visualizar de manera general las fases de la programación de Arduino.



Se abordará a detalle cada una de las partes que componen la estructura del programa:

- Declaración de variables
- Posición inicial
- Modo de operación
- Detección de errores
- Lectura analógica de sensor
- Puerto serie

Declaración de variables

Este es el inicio del programa, donde se declaran las variables y se lleva a cabo una primera fase de inicialización de variables donde se aprovecha para igualar los valores a cero. Toda variable declarada fuera de las funciones tiene ámbito global, es decir, puede ser accedida desde cualquier parte del programa, es por esta razón que han sido declaradas al inicio.

Para hacer más sencillo el manejo de la programación, se le ha asignado un nombre a cada entrada y salida que se va a utilizar en el programa. Antes se ha mencionado en una tabla las entradas y salidas correspondientes de la placa Arduino, ahora para definirlas en el programa se le han asignado los siguientes nombres:

Nombre de la variable	Número de pin correspondiente
DIR1	6
CLK1	7
OPTO1	8
FINAL1	9
DIR2	10
CLK2	11
OPTO2	12
FINAL2	13

Las funciones correspondientes son Dirección, Pulsos, Optoacoplador y Final de carrera. El indicativo 1 corresponde al motor1 o motor de movimiento vertical; el indicativo 2, al motor 2 o motor de movimiento horizontal.

```
//Declaración de variables de entradas y salidas
//Motor 1 es del eje horizontal y Motor 2 es del eje vertical
int DIR1 = 6;
int CLK1 = 7;
int OPTO1 = 8;
int FINAL1 = 9;
int DIR2 = 10;
int CLK2 = 11;
int OPTO2 = 12;
int FINAL2 = 13;
```

Las siguientes variables que se declaran son las que se utilizarán para las diferentes operaciones dentro del programa, ya sea para almacenar un valor enviado a través del puerto serie o para el control de un bucle.

```
//Declaración de variables para utilizar puerto serial
int matlabDato;
int valor1;
float p1;
int direccion1;
int valor2;
float p2;
int direccion2;


//Declaración de variables para conteo de optoacoplador
int contador1 = 0;
int edoanterior1 = 0;
int valoropto1;
int contador2 = 0;
int edoanterior2 = 0;
int valoropto2;
int contcero1 = 0;
int contcero2 = 0;
int error = 0;

//Declaración de variables para sensor
int out1 = 0;
int sensor;
```

Una vez declaradas todas las variables a utilizar dentro de la programación, relacionando los nombres con la funciona que vayan a desempeñar, se inicia la función *setup()*. Dentro de esta función se dará a las primeras variables declaradas la asignación de entrada o salida. Cómo ya se ha mencionado anteriormente, esta declaración se hace dentro de *setup()* y solo se realiza una vez cada vez que inicia el programa.

Nombre de la variable	Descripción	Pin	Declaración
DIR1	Dirección de motor vertical	6	SALIDA
CLK1	Pulsos de motor vertical	7	SALIDA
OPTO1	Optoacoplador de motor vertical	8	ENTRADA
FINAL1	Final de carrera de motor vertical	9	ENTRADA
DIR2	Dirección de motor horizontal	10	SALIDA
CLK2	Pulsos de motor horizontal	11	SALIDA
OPTO2	Optoacoplador de motor horizontal	12	ENTRADA
FINAL2	Final de carrera de motor horizontal	13	ENTRADA

DIR1, *CLK1*, *DIR2* y *CLK2* son los pines que controlan el movimiento de los motores, es por esto que se configuran como salidas. La dirección (DIR) nos permite seleccionar el sentido de giro del motor, mientras que los pulsos (CLK) envía una secuencia de unos y ceros para que el motor realice los pasos indicados.

DIR1	Giro de motor	Movimiento seguidor
1 (HIGH)	Derecha	Abajo
0 (LOW)	Izquierda	Arriba
DIR2		
1 (HIGH)	Derecha	Izquierda
0 (LOW)	Izquierda	Derecha

OPTO1, *FINAL1*, *OPTO2* y *FINAL2* son los sensores que ayudarán a controlar la posición del seguidor. Los sensores optoacopladores dan la lectura del encoder para ir contando los grados que se debe de mover el seguidor. Por lado, el sensor de Final de carrera envía una señal cada vez que se llega a la posición inicial o se ha dado una vuelta de 360º. Al ser estos sensores que nos enviarán información al microcontrolador para tomar las decisiones correspondientes, se declaran como entradas.

```
void setup() {
  //Declaración de entradas y salidas
  pinMode(DIR1, OUTPUT);
  pinMode(CLK1, OUTPUT);
  pinMode(OPTO1, INPUT);
  pinMode(FINAL1, INPUT);
  pinMode(DIR2, OUTPUT);
  pinMode(CLK2, OUTPUT);
  pinMode(OPTO2, INPUT);
  pinMode(FINAL2, INPUT);
```

Posición inicial

Es necesario que antes de iniciar ninguna operación con el seguidor este se posicione automáticamente en la posición inicial. La secuencia de instrucciones para que realice esta tarea se define dentro de la función *setup()*, de este modo, cada vez se inicie el programa el seguidor volverá a su posición inicial para poder empezar a trabajar con él.

Los dos motores deben de moverse en la dirección LOW, que equivale al giro de motor a la izquierda. El motor vertical baja para volver a su posición de inicio, mientras que el motor horizontal gira a la derecha. Se conoce que está en su posición inicial cuando ha llegado al final de carrera y el sensor de final de carrera envía una señal de LOW.

```
//Posicionamiento inicial
while (digitalRead(FINAL1) == 1) {
  digitalWrite(CLK1, HIGH);
  delay(1);
  digitalWrite(CLK1, LOW);
  delay(1);
}
while (digitalRead(FINAL2) == 1) {
  digitalWrite(CLK2, HIGH);
  delay(1);
  digitalWrite(CLK2, LOW);
  delay(1);
}
```

Para terminar en la función *setup()*, se abre el puerto serie con la instrucción Serial.begin(), estableciendo la velocidad de bits por segundo en 9600 baudios.

Modo de operación del seguidor

Una vez inicializado el programa se ejecuta la función *loop()*, en la cual se encuentran todas las instrucciones que darán el control de movimiento al seguidor solar, así como también la lectura del sensor de radiación. Lo primero que realiza una vez dentro de *loop()* es comprobrar si el puerto serie tiene información disponible para ser leída con la función *Serial.available()*.

En esta parte, el programa espera a que el usuario seleccione un modo de operación para iniciar el movimiento del seguidor. Los modos de operación que ejecutará el seguidor se resumen a continuación:

- Secuencia inicio: una de las opciones disponibles para el usuario es poder volver al posición de inicio siempre que lo desee sin necesidad de tener que reiniciar el microcontrolador. Esta secuencia sigue las mismas instrucciones que las que se encuentran dentro de la función *setup()*. Mientras se ejecutarse la Secuencia de inicio no se toman lecturas del sensor de radiación.
- Secuencia automática: esta secuencia tiene una rutina de trayectoria preestablecida que ejecutará el seguidor solar para tomar las lecturas del sensor de radiación solar en las diferentes posiciones que cubre la semiesfera. La rutina inicia a partir de la posición inicial con el movimiento del motor vertical subiendo hasta 90º. Una vez que ha llegado a la máxima posición vertical, el motor horizontal gira a la derecha 10º. Posteriormente, el motor vertical cambia de sentido para bajar los 90º. El motor horizontal avanza de nuevo 10º hacia la derecha, y vuelve a empezar el movimiento el motor vertical. Esto se repite hasta que el motor horizontal da un giro completo de 360º. Cada motor se mueve en pasos de 10º, ya que es el mínimo que permite monitorear el encoder. Una vez terminada la Secuencia automática se debe seleccionar la Secuencia inicio para volver a la posición inicial.

- Secuencia Manual: dentro de esta secuencia es posible controlar el seguidor solar operado manualmente por el usuario. Es posible seleccionar la dirección de ambos motores y establecer los grados que cada motor deberá moverse.

Las instrucciones de programación siguen el tipo de estructura de selección con el comando de condición *if*. El usuario elegirá una de las tres opciones por medio de la interfaz gráfica de Matlab, Matlab enviará el dato con la opción elegida del modo de operación a travñes del puerto serie. El programa en Arduino leerá el dato del puerto serie y lo almacenará en una variable para poder utilizarla en el comando de condición y así ejecutar la tarea deseada.

- Si es 1 → Ejecuta Secuencia inicio

```
//Instrucciones para volver a la posición de inicio
if (matlabDato==1) {
  digitalWrite(DIR1, LOW);
  digitalWrite(DIR2, LOW);
  while (digitalRead(FINAL1) == 1) {
    digitalWrite(CLK1, HIGH);
    delay(1);
    digitalWrite(CLK1, LOW);
    delay(1);
  }
  while (digitalRead(FINAL2) == 1) {
    digitalWrite(CLK2, HIGH);
    delay(1);
    digitalWrite(CLK2, LOW);
    delay(1);
  }
}
```

- Si es 2 → Ejecuta Secuencia automática

La secuencia de pasos automática está definida para que cada motor avance en 10º cada vez hasta completar la rutina. El movimiento inicia con el eje vertial, el cual representa el ángulo azimut, elevándose hasta 90º. Este movimiento se realiza mediante bucles *for*. El programa calcula el número de pasos que corresponden a 10º en el encoder, cada encoder tienen 72 peines y cada paso es un peine, estos pasos definen el número de ciclos que ejecutará la función *for*. Para tener la retroalimentación del movimiento del motor se lee la señal del Optoacoplador que enviará una señal HIGH cada vez que lea un peine del encoder, aumentado el contador del bucle *for*. Este bucle *for* para avanzar 10º está dentro de otro bucle *for* que hace un conteo hasta 9 para llegar a los 90º del eje vertical.

Una vez que el motor vertical ha llegado a 90º se repite la rutina del bucle *for* de 10º para que se mueva el motor horizontal. Al concluir el bucle del motor horizontal, se inicia de nuevo los bucles para el motor vertical en sentido contrario para que el motor baje hasta la mínima posición. Por último, el motor horizontal avanza 10º continuando en el mismo sentido. Estos 4 movimientos de los motores, establecidos por cada bucle, se encuentran dentro de otro bucle *for* que cuenta 18 veces para que se repita la secuencia, de este modo se obtiene un giro de 360º en el motor horizontal, haciendo un barrido de 90º del eje horizontal cada 10º.

Cada vez que el motor avanza 10º, el programa hace una breve pausa para obtener la señal del piranómetro y enviar el dato por puerto serie.

```
//Instrucciones para funcionamiento automático
if (matlabDato==2) {
  Serial.println('1');
  p1 = 10 * 72 / 360; //número de pasos para 10 grados
  delay(200);
  out1 = analogRead(A1);
  Serial.println(out1);
  for (int a1 = 0; a1 < 18; ) {
    for (int b1 = 0; b1 < 9; ) {
      digitalWrite(DIR1, HIGH);
      contador1 = 0;
      for (int b2 = 0; b2 < p1; ) {
        digitalWrite(CLK1, HIGH);
        delay(1);
        digitalWrite(CLK1, LOW);
        valoroptо1 = digitalRead(OPTO1);
        if (valoroptо1 != edoanterior1) {
          if (valoroptо1 == HIGH) {
            contador1++; //Contador de pasos
            contcero1=0;
            contuno1++; //Contador para detectar atasco
            if(contuno1>1){
              goto error1;}
            b2++;
          }
        if (valoroptо1 == LOW) {
          contcero1++; //Contador para detectar atasco
          contuno1=0;
          if (contcero1>1) {
            goto error1;}
        }
        edoanterior1 = valoroptо1;
        delay(1);
      }
      delay(1);
    }
    b1++;
    delay(200);
    out1 = analogRead(A1);
    Serial.println(out1);
  }
  digitalWrite(DIR2, HIGH);
  contador2 = 0;
  for (int a2 = 0; a2 < p1; ) {
    digitalWrite(CLK2, HIGH);
    delay(1);
    digitalWrite(CLK2, LOW);
    valoropto2 = digitalRead(OPTO2);
    if (valoropto2 != edoanterior2) {
      if (valoropto2 == HIGH) {
        contador2++; //Contador de pasos
        contcero2 = 0;
```

```
      contuno2++; //Contador para detectar atasco
        if(contuno2>1){
          goto error2;}
      a2++;
    }
    if (valoropto2 == LOW) {
      contcero2++;
      contuno2=0;
      if (contcero2>1) {
        goto error2;}
    }
    edoanterior2 = valoropto2;
    delay(1);
  }
  delay(1);
}
outl = analogRead(A1);
 Serial.println(outl);
delay(200);
for (int b3 = 0; b3 < 9; ) {
  digitalWrite(DIR1, LOW);
  contador1 = 0;
  for (int b4 = 0; b4 < p1; ) {
    digitalWrite(CLK1, HIGH);
    delay(1);
    digitalWrite(CLK1, LOW);
   valoroptol = digitalRead(OPTO1);
   if (valoroptol != edoanterior1) {
     if (valoroptol == HIGH) {
       contador1++; //Contador de pasos
       contcero1 = 0;
       contunol++; //Contador para detectar atasco
       if(contunol>1){
         goto error1;}
       b4++;
     }
     if (valoroptol == LOW) {
       contcero1++;
       contunol=0;
       if (contcero1>1) {
         goto error1;}
     }
     edoanterior1 = valoroptol;
     delay(1);
   }
   delay(1);
 }
 b3++;
 outl = analogRead(A1);
 Serial.println(outl);
 delay(200);
}
```

```
    contador2 = 0;
    for (int a3 = 0; a3 < p1; ) {
      digitalWrite(CLK2, HIGH);
      delay(1);
      digitalWrite(CLK2, LOW);
      valoropto2 = digitalRead(OPTO2);
      if (valoropto2 != edoanterior2) {
        if (valoropto2 == HIGH) {
          contador2++; //Contador de pasos
          contcero2 = 0;
          contuno2++; //Contador para detectar atasco
            if(contuno2>1){
              goto error2;}
          a3++;
        }
        if (valoropto2 == LOW) {
          contcero2++;
          contuno2=0;
          if (contcero2>1) {
            goto error2;}
        }
        edoanterior2 = valoropto2;
        delay(1);
      }
      delay(1);
    }
     out1 = analogRead(A1);
     Serial.println(out1);
     delay(500);
   a1=a1+1;}
   Serial.println('0');
}
if((contuno1>1) || (contuno2>1) || (contcero1>1) || (contcero2>1)){
   if((contuno1>1) || (contcero1>1)){
   error1:
   Serial.println('Atasco en motor vertical');}
   if((contuno2>1) || (contcero2>1)){
   error2:
   Serial.println('Atasco en motor horizontal');}}
```

- Si es 3 → Ejecuta Secuencia manual

En esta secuencia, el usuario decide el sentido de giro del seguidor y el número de grados que deberá moverse, es por esto que en la programación se toma la lectura de estos datos que se reciben por puerto serie. Una vez obtenidos los datos y guardados en sus respectivas variables, se calculará el número de pasos que tiene que avanzar el motor.

El número de pasos se obtendrá por medio de una regla de tres con la relación que existe con los peines del encoder. Cada encoder tiene 72 peines que corresponden a 360º.

$$Pasos = No.Grados * 72/360$$

```
//Instrucciones para funcionamiento manual
else if(matlabDato==3) {
  direccion1 = Serial.parseInt();
  valor1 = Serial.parseInt();
  direccion2 = Serial.parseInt();
  valor2 = Serial.parseInt();
  Serial.println('1');
  //DIR1 = Motor 1 (Movimiento vertical); DIR2 = Motor 2 (Movimiento horizontal)
  p1=valor1*72/360;
  p2=valor2*72/360;
  contador1=0;
  contador2=0;
  //Instrucciones para movimiento eje vertical DIR1=HIGH= Movimiento hacia arriba; DIR1=LOW= Movimiento hacia abajo
  for(int i=0;i<p1; ){
    if(direccion1==1){
      if fin1==2{
        break;}
      digitalWrite(DIR1,HIGH);}
    if(direccion1==0){
      if fin1==1{
        break;}
      digitalWrite(DIR1,LOW);}
    digitalWrite(CLK1,HIGH);
    delay(1);
    digitalWrite(CLK1,LOW);
    valoroptol=digitalRead(OPTO1);
    if(valoroptol!=edoanterior1){
      if(valoroptol==HIGH){
        contador1++;
        contcero1=0;
        contuno1++;
      if(contuno1>1){
        goto error1;}
        i++;}
    if(valoroptol==LOW){
      contcero1++;
      contuno1=0;
      if(contcero1>1){
        goto error1;}}
    edoanterior1=valoroptol;
    delay(1);}
    Serial.println(contador1);
    if(digitalRead(FINAL1)==LOW){
      if(contador1>3){
        break;;}}}
  delay(1000);

//Instrucciones para movimiento eje horizontal DIR2=HIGH= Movimiento a la derecha; DIR2=LOW= Movimiento a la izquierda
  for(int x=0;x<p2; ){
    if(direccion2==1){
      if fin2==2{
```

```
      if fin2==2{
        break;}
      digitalWrite(DIR2,HIGH);}
    if(direccion2==0){
      if fin2==1{
        break;}
      digitalWrite(DIR2,LOW);}
    digitalWrite(CLK2,HIGH);
    delay(1);
    digitalWrite(CLK2,LOW);
    valoropto2=digitalRead(OPTO2);
    if(valoropto2!=edoanterior2){
      if(valoropto2==HIGH){
        contador2++;
        contcero2=0;
        contuno2++;
        if(contuno2>1){
          goto error2;}
        x++;}
      if(valoropto2==LOW){
        contcero2++;
        contuno2=0;
        if(contcero2>1){
           goto error2;}}
        edoanterior2=valoropto2;
        delay(1);}
      Serial.println(contador2);
    if(digitalRead(FINAL2)==LOW){
      if(contador2>3){
        break;;}}}

  if(digitalRead(FINAL1)==LOW && digitalRead(DIR1)==LOW){
    Serial.println('10');
    Serial.println("Punto de inicio del eje vertical");
    fin1=1;}
  if(digitalRead(FINAL1)==LOW && digitalRead(DIR1)==HIGH){
    Serial.println('10');
    Serial.println("El eje vertical ha llegado al límite, mover en sentido opuesto");
    fin1=2;}

  if(digitalRead(FINAL2)==LOW && digitalRead(DIR2)==LOW){
    Serial.println('11');
    Serial.println("Punto de inicio del eje horizontal");
    fin2=1;}
  if(digitalRead(FINAL2)==LOW && digitalRead(DIR2)==HIGH){
    Serial.println('11');
    Serial.println("El eje horizontal ha llegado al límite, mover en sentido opuesto");
    fin2=2;}

  Serial.println('0');
  out1=analogRead(A1);
  Serial.println(out1);
```

Detección de errores y advertencias

El programa incluye un sistema básico de retroalimentación para poder identificar que el conteo de pasos es correcto, así como identificar atascos en los motores. Para esto se hará uso del optoacoplador y el uso de variables para conteo. Cada vez que el optoacoplador cuente dos ceros o dos unos seguidos, el programa detendrá la secuencia y enviará un aviso de que ha ocurrido un atasco en el motor.

```
valoroptol=digitalRead(OPTO1);
if(valoroptol!=edoanterior1){
  if(valoroptol==HIGH){
    contador1++;
    contcero1=0;
    contuno1++;
    if(contuno1>1){
      goto error1;}
    i++;}
  if(valoroptol==LOW){
    contcero1++;
    contuno1=0;
    if(contcero1>1){
      goto error1;}}
  edoanterior1=valoroptol;

 valoropto2=digitalRead(OPTO2);
 if(valoropto2!=edoanterior2){
   if(valoropto2==HIGH){
     contador2++;
     contcero2=0;
     contuno2++;
     if(contuno2>1){
       goto error2;}
     x++;}
   if(valoropto2==LOW){
     contcero2++;
     contuno2=0;
     if(contcero2>1){
        goto error2;}}
   edoanterior2=valoropto2;

 error1:
 Serial.println('Atasco en motor vertical');}

 error2:
 Serial.println('Atasco en motor horizontal');}}
```

El seguidor solar cuenta con limitaciones físicas para el movimiento, es por esto que también se incluyen advertencias y protecciones dentro del programa para evitar que se produzcan contratiempos dentro del equipo. El eje horizontal tiene como máximo movimiento 360º, al alcanzar esta posición el programa detendrá la secuencia y enviará un texto que el usuario podrá visualizar: "El eje horizontal ha llegado al límite, mover en sentido opuesto". En caso de que el usuario cometa el error de enviar el dato incorrecto, el programa reconocerá que no se puede seguir moviendo en esa dirección para así evitar complicaciones en el equipo.

El programa enviará antes del mensaje un número para que el programa en Matlab reconozca que se trata de un mensaje

```
  if(digitalRead(FINAL2)==LOW){
    if(contador2>3){
      break;;}}}

if(digitalRead(FINAL1)==LOW && digitalRead(DIR1)==LOW){
  Serial.println('10');
  Serial.println("Punto de inicio del eje vertical");}
if(digitalRead(FINAL1)==LOW && digitalRead(DIR1)==HIGH){
  Serial.println('10');
  Serial.println("El eje vertical ha llegado al límite, mover en sentido opuesto");}

if(digitalRead(FINAL2)==LOW && digitalRead(DIR2)==LOW){
  Serial.println('11');
  Serial.println("Punto de inicio del eje horizontal");}
if(digitalRead(FINAL2)==LOW && digitalRead(DIR2)==HIGH){
  Serial.println('11');
  Serial.println("El eje horizontal ha llegado al límite, mover en sentido opuesto");}
```

Lectura analógica del sensor de radiación

El sensor de radiación solar cuenta con 31 sensores individuales que se conectarán mediante un solo cable a la entrada analógica de la placa Arduino A0.

La lectura analógica del sensor de radiación se toma cada 10º en ambos motores, para después ser enviada a través del puerto serie.

En cada posición que indique el usuario el programa tomará la lectura analógica del sensor de radiación y lo enviará por puerto serie.

En algunos casos se requiere enviar por puerto serie un '0' (cero) para que el software en Matlab reconozca que el siguiente dato a enviar corresponde al del sensor.

```
Serial.println('0');
outl=analogRead(A1);
Serial.println(outl);
```

Puerto Serie

Debido a que el uso de este puerto ha quedado en desuso a favor de la tecnología USB, Arduino cuenta con de serial a USB que permite a la placa ser reconocida por el ordenador como un dispositivo conectado a un puerto COM aun cuando la conexión física sea USB.

En el IDE de Arduino tenemos que configurar a través de que puerto se realizará la comunicación, esto se realiza en el menú de Herramientas. Una vez conectado la placa mediante el cable USB se habilitará la opción de Puerto para poder seleccionar el adecuado.

En la función *setup()* se inicia la comunicación serial con la sentencia Serial.begin(9600). El 9600 indica la velocidad establecida para cumplir el propósito de este proyecto.

Ahora, en la función *loop()* se verificará la disponibilidad del puerto a través de la sentencia Serial.available(), siempre que esté el puerto disponible el bucle estará ejecutándose. A partir de aquí, el programa esperará los datos que se enviarán a través de Matlab para ejecutar el modo de operación elegido por el usuario.

El dato enviado desde Matlab es leído a través de la sentencia Serial.parseInt(), este comando lee el dato del puerto serie y lo almacena en una variable en formato entero. El programa selecciona el modo de operación a través de sentencias if y ejecuta la secuencia indicada.

En el modo de operación *Manual*, se vuelve a utilizar el puerto serie para lectura para poder leer los datos de la dirección y número de grados que se moverán los motores. Esto también se hace por medio de la sentencia Serial.parseInt(). El número de grados se convierten al número de pasos que se contarán en el encoder.

```
direccion1=Serial.parseInt();
valor1=Serial.parseInt();
direccion2=Serial.parseInt();
valor2=Serial.parseInt();
p1=valor1*72/360;
p2=valor2*72/360;
```

En cada modo de operación se enviará información por puerto serie para informar sobre errores que puedan llegar a ocurrir en el sistema mecánico. Además, en la secuencias Automática y Manual, se enviará la información obtenida del sensor.

DESARROLLO DE INTERFAZ GRAFICA Y ADQUISICIÓN DE DATOS MATLAB

Matlab es un paquete de software orientado hacia el cálculo numérico científico e ingenieril. Integra cálculo numérico, computación de matrices y gráficos en un entorno de trabajo cómodo para el usuario.

Matlab es capaz de procesar de modo secuencial una serie de comandos previamente definidos, obteniendo de forma inmediata los resultados. Para que Matlab realice este proceso se crea un fichero con extensión *.m* donde incluirá todos los comandos a ejecutar.

El programa desarrollado en Matlab ha sido para crear una interfaz que permita al usuario controlar de forma fácil el seguidor, así como almacenar la información obtenida del sensor. El programa ha tenido que ser capaz de mantener una comunicación abierta por puerto serial para poder enviar y recibir datos desde la placa de Arduino.

Diseño GUIDE

La interfaz gráfica de usuario, o GUI, es un programa informático concebido para hacer más amigable la comunicación entre el usuario final y el software a ejecutar. La GUI se presenta como un conjunto de objetos gráficos de fácil manejo, a través de los cuales se expone la información y acciones disponibles. La GUI hace posible que el usuario final manipule de manera directa el control del seguidor solar.

El diseño de la interfaz gráfica consistirá en incluir los objetos necesarios para que el usuario sea capaz de modificar las funciones y movimientos del seguidor, a través de botones y cuadros de texto.

Los datos del sensor serán visualizados en tiempo real en una gráfica y el valor será visible en un cuadro de texto.

Por medio de un botón se hará la conexión serial para poder comunicar con Arduino.

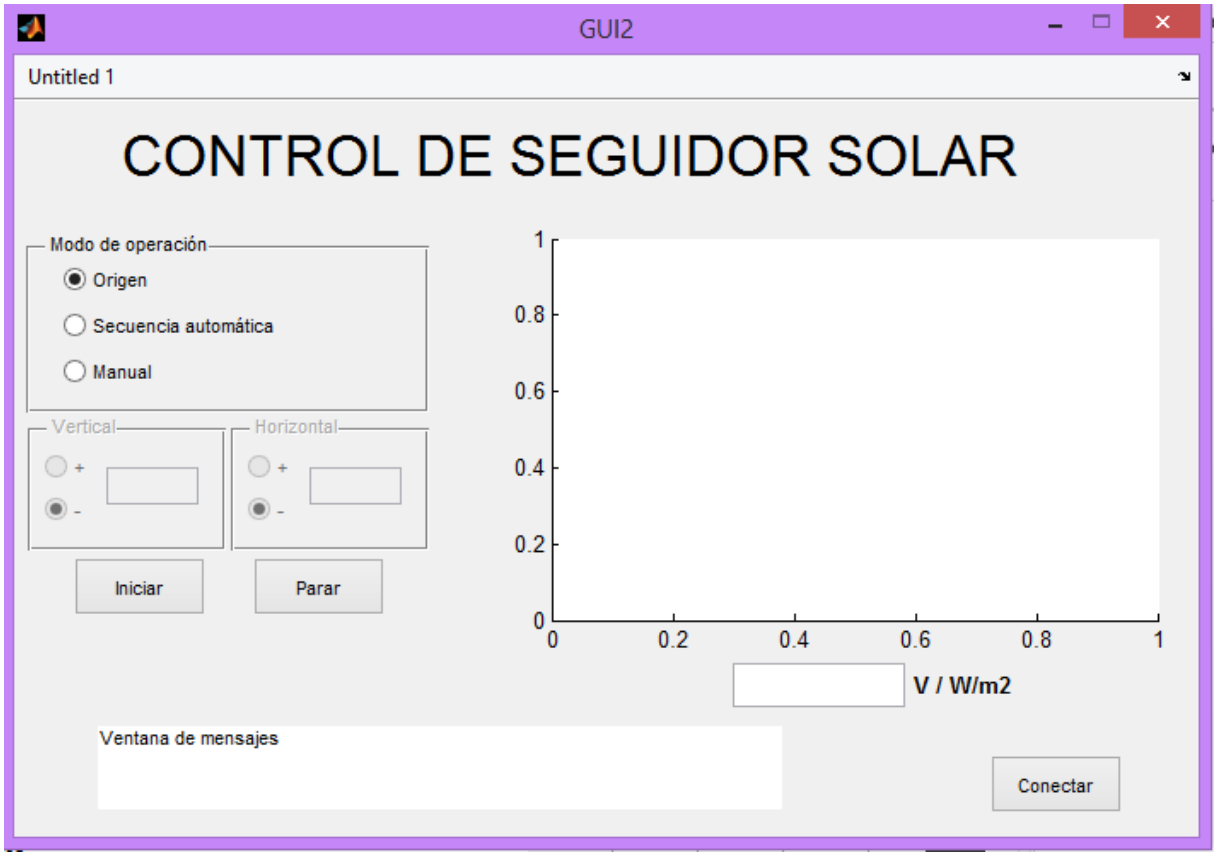


Figura 13. Pantalla de control del seguidor solar

En los siguienes apartados se detallará las funciones de cada elemento.

Puerto Serie

Por medio del puerto serie, como se ha mencionado anteriormente, se realizará la comunicación desde la placa de Arduino con el ordenador. En el GUI del seguidor solar existe un botón para iniciar la conexión por puerto serie.

Al pulsar el botón *Conectar* se llama a la función *Callback* para que el programa ejecute las instrucciones que permite abrir la comunicación serial.

```
function btConectar_Callback(hObject, eventdata, handles)
clc;
pserial=serial('COM3');
set(pserial,'BaudRate',9600);
set(pserial,'DataBits',8);
set(pserial,'Parity','none');
set(pserial,'StopBits',1);
set(pserial,'FlowControl','none');
fopen(pserial);
handles.CONECTADO=pserial;
guidata(hObject,handles);
```

Estas instrucciones definen el puerto simulado al que está conectada la placa Arduino, se define la velocidad que debe ser igual a la establecida en el programa de Arduino. La variable para controlar el puerto serial se ha llamado *pserial*.

El comando *fopen()* es el que inicia la comunicación serial. *handles.CONECTADO* corresponde a un identificador donde se almacenará la información de *pserial*, después con *guidata()* se salva la información para poder ser utilizado en cualquier otra parte del programa.

Para cerrar la comunicación de puerto serie existe el botón *Parar:*

```
function btParar_Callback(hObject, eventdata, handles)
pserial=handles.CONECTADO;
fclose(pserial);
delete(pserial)
clear pserial
```

Se llama al identificador *handles.CONECTADO* para utilizar la misma información antes guardada y con *fclose()* se cierra el puerto serie, *delete()* borra la información que haya quedado en el puerto.

Envío y adquisición de datos

Los datos que serán enviados por Matlab son los que requiere Arduino para ejecutar las secuencias de operación y para conocer los grados que se debe mover.

Una vez iniciada la conexión serial, se seleccionará un modo de operación:

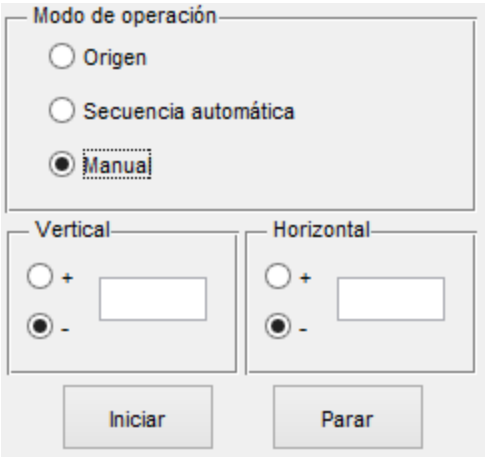


Figura 14. Modo de operación

Las opciones de *Vertical* y *Horizontal* solo se activarán si la opción *Manual* está seleccionada. El responsable de que los datos sean enviados por puerto serie es el botón *Iniciar*. Al pulsar el botón *Iniciar* se llama a la función *Callback* y se ejecuta el siguiente código:

```
function btInicio_Callback(hObject, eventdata, handles)
pserial=handles.CONECTADO;
```

Lo primero que realiza es obtener la información del puerto serie. A continuación hace una evaluación de acuerdo a la selección del modo de operación por medio de una condicional *if*:

- Origen:

  El programa lee el valor del botón *Origen* y lo guarda en la variable *estado*, si esta valor es igual a 1, envía por puerto serie el valor que corresponde a la ejecución de la secuencia para que el seguidor vuelva a su punto de inicio, en este caso es '1'.

```
estado = get(handles.modoInicio,'Value');
if estado==1 %Volver a posición de inicio
    fprintf(pserial,'1');
end
```

- Secuencia automática:

  El programa lee el valor del botón *Secuencia automática* y lo guarda en la variable *estado*, si este valor es igual a 1 envía por puerto serie el valor que corresponde a la ejecución de la secuencia para que el seguidor haga la rutina programada, en este caso es '2'.

```
estado = get(handles.modoAut,'Value');
if estado==1
    fprintf(pserial,'2');
    dato2=0;
```

En esta secuencia, el programa tendrá que recibir información enviada por la placa Arduino, para esto se requiere que el programa de Arduino envíe un '1' para que Matlab reconozca que a partir de este punto se recibirán datos. Se verifica que el puerto serie esté disponible y se procede a leer la información del puerto serie en binario, se almacena en la variable *dato2* convirtiéndola en tipo *doublé*.

```
while(~(pserial.BytesAvailable))
end

q=fgetl(pserial); %lectura del puerto
dato2=(str2double(q))
```

Si el valor recibido es un '1', se establecen los valores de la gráfica donde se mostrarán los valores obtenidos del sensor y se definen las variables que se utilizarán dentro de esta ejecución.

Los valores se leen por medio de un bucle while. El valor del puerto serie se lee por medio de *fscanf()* y lo guarda en la variable *sensor* en formato entero. Este valor es recibido como byte por lo que se debe hacer la conversión correspondiente al valor para que se muestre en voltaje. El nuevo valor de voltaje se va guardando dentro de la matriz *val*.

Para hacer la gráfica se utiliza la función *line* porque es más eficiente que la función *plot*. Con el comando *plot*, cuando pasan unos segundos, hay un cierto retardo en la representación gráfica, dejando de ser a tiempo real.

El programa de arduino, al terminar la secuencia automática, envía un '0' para que Matlab de por concluido el bucle *while*. Una vez finalizado el bucle se guardan los valores obtenidos en la matriz *val* en un archivo Excel.

```
if dato2==1
    axes(handles.axes1);
    xlabel('Muestra');
    ylabel('Voltaje de salida');
    val=zeros(500,1);
    i=1;
    lHandle=line(nan, nan)
    sensor =1;


    while sensor~=0
        ylim([0 13]);
        xlim([0 i+5]);
        sensor=fscanf(pserial,'%d')
        val(i)=sensor*12.75/1023;
        X = get(lHandle, 'XData');
        Y = get(lHandle, 'YData');
        x = [X i];
        y = [Y val(i)];
        set(lHandle, 'XData', x, 'YData', y);
        drawnow
        set(handles.Valor,'string',num2str(val(i)))
        i=i+1;
    end
    xlswrite('Libro1.xls', val, 'A1');
    end
```

- Secuencia Manual:

En esta secuencia se tiene que enviar el número '3' que indica el modo de operación en arduino, el sentido de dirección y los grados que debe moverse cada motor. Toda la información debe ser enviada en una línea para que el programa en arduino lo reconozca. Esto se consigue por medio de un vector que almacenará los 5 valores correspondientes. El vector *v* debe almacenar los datos de la siguiente forma: {No. 3, DIR1,GRADOS1,DIR2,GRADOS2} en donde el motor 1 es el vertical y el motor 2 el horizontal.

Una vez creado el vector, se envía por *fprintf* en formato string. El programa espera a que el puerto esté disponible y reciba un '1' desde arduino. El programa entra en un bucle para leer los datos enviados desde arduino en espera de que reciba un '0', el siguiente dato enviado después del '0' es el valor del sensor, este es almacenado en la variable *valor2* y se muestra en el cuadro de texto debajo de la gráfica.

```
if estado==1
    v=zeros(1,5);
    v(1)=3;
    Up= get(handles.verUp,'Value');
    if Up==1
        v(2)=1;
    end
    Down = get(handles.verDown,'Value');
    if Down==1
        v(2)=0;
    end
    v(3)=handles.verGrados;

    Right= get(handles.horRight,'Value');
    if Right==1
        v(4)=1;
    end
    Left = get(handles.horLeft,'Value');
    if Left==1
        v(4)=0;
    end
    v(5)=handles.horGrados;
    fprintf(pserial,num2str(v));
    dato2=0;
    valor=1;


  while(~(pserial.BytesAvailable))
  end

  q=fgetl(pserial); %lectura del puerto
  dato2=(str2double(q))

  if dato2==1
      while valor~=0
          valor=fscanf(pserial,'%d');
          if valor==12592;
              msj=fgetl(pserial);
              set(handles.txt5,'string',msj)
          end
          if valor==12593;
              msj=fgetl(pserial);
              set(handles.txt7,'string',msj)
          end
      end

      if valor==0
          valor2=fscanf(pserial,'%d')
          set(handles.Valor,'string',num2str(valor2))
      end
  end
```

CONCLUSIONES

Una vez que se ha llegado al final del proyecto, resulta necesario analizar si se han cumplido los objetivos que se fijaron en el mismo. Al comenzar el proyecto se planteó crear un programa capaz de mover el seguidor solar existente en la universidad.

Por medio de una placa Arduino Duemilanove y el software IDE de Arduino, se ha realizado un programa que cumple con el objetivo inicial planteado. El programa tiene una secuencia de operación automática para realizar un barrido y obtener la medición en diferentes puntos, puede ser operado de forma manual, tiene instrucciones para detectar fallos en el sistema y almacena la información obtenida por el sensor de radiación solar.

El tiempo de respuesta del seguidor solar, es decir, el movimiento de los motores paso a paso, en la secuencia automática puede ser modificado desde el algoritmo de arduino, de esta manera aumentar o disminuir la velocidad de giro.

Se realizó una aplicación para el usuario en Matlab, si bien esta parte escapa de los objetivos que se habían marcado, me pareció necesario realizar una interfaz más estética y que facilitara el uso del programa. De esta manera se realiza un proyecto completo, que abarca desde el control del seguidor solar hasta la señal tomada del sensor para ser representada gráficamente y almacenada en un archivo Excel.

Matlab cuenta con un paquete para realizar una comunicación directa con Arduino, desde la cual la comunicación serial es directa y los comandos utilizados desde Matlab configuran la placa Arduino. En esta aplicación el paquete Matlab-Arduino no pudo ser utilizado, ya que la velocidad de ejecución de los comandos es más lenta desde Matlab que la velocidad que tiene el microprocesador ATMega328 que tiene instalado la placa.

Se puede considerar que todos los objetivos han sido cumplidos, obteniendo como resultado un conjunto que funciona, un programa que es capaz de seguir una secuencia de movimientos automáticamente, así como moverse a un punto indicado por el usuario, obtener un valor desde el sensor de radiación solar y almacenarlo para ser utilizado en futuras aplicaciones o investigaciones. El programa tiene la posibilidad de seguir agregando diferentes opciones de acuerdo a las necesidades de la aplicación, tal como agregar nuevas secuencias automáticas o mejoras en la detección de errores, esto se podrá realizar una vez que el seguidor solar esté completamente instalado y se realicen las pruebas debidas.

Por otro lado, el desarrollo del programa ha implicado el aprendizaje y/o consolidación de conocimientos. Se han puesto en práctica conocimientos teóricos de energía solar, así como también de programación.

Aun teniendo en cuenta la gran variedad de TFMs a escoger, consideré oportuno hacer un proyecto basado en programación para ampliar mis conocimientos técnicos sobre el marco de trabajo de seguidores solares y Matlab.

BIBLIOGRAFIA

1. Manual de programación Arduino
   http://arduinobot.pbworks.com/f/Manual+Programacion+Arduino.pdf

2. ATmega328
   http://www.atmel.com/Images/doc8161.pdf

3. Documento Desarrollo de un sistema de medida de radiación solar basado en técnicas de seguimiento solar en dos ejes.
   Proyecto Fin de Carrera
   Celia Lorenzo García
   Universidad de Jaén

4. Documento Response fitting in low-cost radiation sensors
   Departamento de Ingeniería Electrónica
   Universidad de Jaén

5. Manual de Matlab
   https://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf

6. Manual de Matlab GUIDE
   https://www.mathworks.com/help/pdf_doc/matlab/buildgui.pdf

ANEXOS

PROGRAMA DE ARDUINO

```
//Declaración de variables de entradas y salidas
//Motor 1 es del eje horizontal y Motor 2 es del eje vertical
int DIR1 = 6;
int CLK1 = 7;
int OPTO1 = 8;
int FINAL1 = 9;
int DIR2 = 10;
int CLK2 = 11;
int OPTO2 = 12;
int FINAL2 = 13;

//Declaración de variables para utilizar puerto serial
int matlabDato;
int valor1;
float p1;
int direccion1;
int valor2;
float p2;
int direccion2;


//Declaración de variables para conteo de optoacoplador y errores
int contador1 = 0;
int edoanterior1 = 0;
int valoropto1;
int contador2 = 0;
int edoanterior2 = 0;
int valoropto2;
int contcero1 = 0;
int contuno1=0;
int contcero2 = 0;
int contuno2=0;
int fin1=0;
int fin2=0;

//Declaración de variables para sensor
int out1 = 0;

void setup() {
  //Declaración de entradas y salidas
  pinMode(DIR1, OUTPUT);
  pinMode(CLK1, OUTPUT);
  pinMode(OPTO1, INPUT);
  pinMode(FINAL1, INPUT);
  pinMode(DIR2, OUTPUT);
  pinMode(CLK2, OUTPUT);
  pinMode(OPTO2, INPUT);
```

```
  pinMode(FINAL2, INPUT);

  //Posicionamiento inicial
  while (digitalRead(FINAL1) == 1) {
    digitalWrite(CLK1, HIGH);
    delay(1);
    digitalWrite(CLK1, LOW);
    delay(1);
  }
  while (digitalRead(FINAL2) == 1) {
    digitalWrite(CLK2, HIGH);
    delay(1);
    digitalWrite(CLK2, LOW);
    delay(1);
  }

  Serial.begin(9600);
}


void loop() {
  if (Serial.available()) {
    matlabDato=Serial.parseInt();
    p1 = 0;
    p2 = 0;
    contador1 = 0;
    contador2 = 0;


    //Instrucciones para volver a la posición de inicio
    if (matlabDato==1) {
      digitalWrite(DIR1, LOW);
      digitalWrite(DIR2, LOW);
      while (digitalRead(FINAL1) == 1) {
        digitalWrite(CLK1, HIGH);
        delay(1);
        digitalWrite(CLK1, LOW);
        delay(1);
      }
      while (digitalRead(FINAL2) == 1) {
        digitalWrite(CLK2, HIGH);
        delay(1);
        digitalWrite(CLK2, LOW);
        delay(1);
      }
    }

    //Instrucciones para funcionamiento automático
    if (matlabDato==2) {
      Serial.println('1');
      p1 = 10 * 72 / 360; //número de pasos para 10 grados
      delay(200);
```

```
out1 = analogRead(A1);
Serial.println(out1);

for (int a1 = 0; a1 < 18; ) {
  for (int b1 = 0; b1 < 9; ) {
    digitalWrite(DIR1, HIGH);
    contador1 = 0;
    for (int b2 = 0; b2 < p1; ) {
      digitalWrite(CLK1, HIGH);
      delay(1);
      digitalWrite(CLK1, LOW);
      valoropto1 = digitalRead(OPTO1);
      if (valoropto1 != edoanterior1) {
        if (valoropto1 == HIGH) {
          contador1++; //Contador de pasos
          contcero1=0;
          contuno1++; //Contador para detectar atasco
          if(contuno1>1){
            goto error1;}
          b2++;
        }
        if (valoropto1 == LOW) {
          contcero1++; //Contador para detectar atasco
          contuno1=0;
          if (contcero1>1) {
            goto error1;}
        }
        edoanterior1 = valoropto1;
        delay(1);
      }
      delay(1);
    }
    b1++;
    delay(200);
    out1 = analogRead(A1);
    Serial.println(out1);
  }
  digitalWrite(DIR2, HIGH);
  contador2 = 0;

  for (int a2 = 0; a2 < p1; ) {
    digitalWrite(CLK2, HIGH);
    delay(1);
    digitalWrite(CLK2, LOW);
    valoropto2 = digitalRead(OPTO2);
    if (valoropto2 != edoanterior2) {
      if (valoropto2 == HIGH) {
        contador2++; //Contador de pasos
        contcero2 = 0;
        contuno2++; //Contador para detectar atasco
          if(contuno2>1){
            goto error2;}
```

```
      a2++;
    }
    if (valoropto2 == LOW) {
      contcero2++;
      contuno2=0;
      if (contcero2>1) {
        goto error2;}
    }
    edoanterior2 = valoropto2;
    delay(1);
  }
  delay(1);
}
out1 = analogRead(A1);
 Serial.println(out1);
delay(200);

for (int b3 = 0; b3 < 9; ) {
  digitalWrite(DIR1, LOW);
  contador1 = 0;
  for (int b4 = 0; b4 < p1; ) {
    digitalWrite(CLK1, HIGH);
    delay(1);
    digitalWrite(CLK1, LOW);
    valoropto1 = digitalRead(OPTO1);
    if (valoropto1 != edoanterior1) {
      if (valoropto1 == HIGH)  {
        contador1++; //Contador de pasos
        contcero1 = 0;
        contuno1++; //Contador para detectar atasco
        if(contuno1>1){
          goto error1;}
        b4++;
      }
      if (valoropto1 == LOW) {
        contcero1++;
        contuno1=0;
        if (contcero1>1) {
          goto error1;}
      }
      edoanterior1 = valoropto1;
      delay(1);
    }
    delay(1);
  }
  b3++;
  out1 = analogRead(A1);
  Serial.println(out1);
  delay(200);
}
```

```
    contador2 = 0;
    for (int a3 = 0; a3 < p1; ) {
     digitalWrite(CLK2, HIGH);
     delay(1);
     digitalWrite(CLK2, LOW);
     valoropto2 = digitalRead(OPTO2);
     if (valoropto2 != edoanterior2) {
      if (valoropto2 == HIGH) {
       contador2++; //Contador de pasos
       contcero2 = 0;
       contuno2++; //Contador para detectar atasco
        if(contuno2>1){
          goto error2;}
       a3++;
      }
      if (valoropto2 == LOW) {
       contcero2++;
       contuno2=0;
       if (contcero2>1) {
         goto error2;}
      }
      edoanterior2 = valoropto2;
      delay(1);
     }
     delay(1);
    }
    out1 = analogRead(A1);
    Serial.println(out1);
    delay(500);
  a1=a1+1;}
  Serial.println('0');
}

if((contuno1>1) || (contuno2>1) || (contcero1>1) || (contcero2>1)){
  if((contuno1>1) || (contcero1>1)){
  error1:
  Serial.println('Atasco en motor vertical');}
  if((contuno2>1) || (contcero2>1)){
  error2:
  Serial.println('Atasco en motor horizontal');}}

 //Instrucciones para funcionamiento manual
 else if(matlabDato==3) {
  direccion1 = Serial.parseInt();
  valor1 = Serial.parseInt();
  direccion2 = Serial.parseInt();
  valor2 = Serial.parseInt();
  Serial.println('1');

  //DIR1 = Motor 1 (Movimiento vertical); DIR2 = Motor 2 (Movimiento horizontal)
  p1=valor1*72/360;
  p2=valor2*72/360;
```

```
 contador1=0;
 contador2=0;
 //Instrucciones para movimiento eje vertical DIR1=HIGH= Movimiento hacia arriba;
DIR1=LOW= Movimiento hacia abajo
 for(int i=0;i<p1; ){
  if(direccion1==1){
   if fin1==2{
    break;}
   digitalWrite(DIR1,HIGH);}
  if(direccion1==0){
   if fin1==1{
    break;}
   digitalWrite(DIR1,LOW);}
  digitalWrite(CLK1,HIGH);
  delay(1);
  digitalWrite(CLK1,LOW);
  valoropto1=digitalRead(OPTO1);
  if(valoropto1!=edoanterior1){
   if(valoropto1==HIGH){
    contador1++;
    contcero1=0;
    contuno1++;
   if(contuno1>1){
    goto error1;}
    i++;}
  if(valoropto1==LOW){
   contcero1++;
   contuno1=0;
   if(contcero1>1){
    goto error1;}}
  edoanterior1=valoropto1;
  delay(1);}
  Serial.println(contador1);
  if(digitalRead(FINAL1)==LOW){
   if(contador1>3){
    break;;}}}
 delay(1000);

 //Instrucciones para movimiento eje horizontal DIR2=HIGH= Movimiento a la derecha;
DIR2=LOW= Movimiento a la izquierda
 for(int x=0;x<p2; ){
  if(direccion2==1){
   if fin2==2{
    break;}
   digitalWrite(DIR2,HIGH);}
  if(direccion2==0){
   if fin2==1{
    break;}
   digitalWrite(DIR2,LOW);}
  digitalWrite(CLK2,HIGH);
  delay(1);
  digitalWrite(CLK2,LOW);
```

```
      valoropto2=digitalRead(OPTO2);
      if(valoropto2!=edoanterior2){
       if(valoropto2==HIGH){
        contador2++;
        contcero2=0;
        contuno2++;
        if(contuno2>1){
         goto error2;}
        x++;}
       if(valoropto2==LOW){
        contcero2++;
        contuno2=0;
        if(contcero2>1){
          goto error2;}}
        edoanterior2=valoropto2;
        delay(1);}
       Serial.println(contador2);
      if(digitalRead(FINAL2)==LOW){
       if(contador2>3){
        break;;}}}

    if(digitalRead(FINAL1)==LOW && digitalRead(DIR1)==LOW){
     Serial.println('10');
     Serial.println("Punto de inicio del eje vertical");
     fin1=1;}
    if(digitalRead(FINAL1)==LOW && digitalRead(DIR1)==HIGH){
     Serial.println('10');
     Serial.println("El eje vertical ha llegado al límite, mover en sentido opuesto");
     fin1=2;}

    if(digitalRead(FINAL2)==LOW && digitalRead(DIR2)==LOW){
     Serial.println('11');
     Serial.println("Punto de inicio del eje horizontal");
     fin2=1;}
    if(digitalRead(FINAL2)==LOW && digitalRead(DIR2)==HIGH){
     Serial.println('11');
     Serial.println("El eje horizontal ha llegado al límite, mover en sentido opuesto");
     fin2=2;}

    Serial.println('0');
    out1=analogRead(A1);
    Serial.println(out1);
 }}
}
```

DESCRIPCION DE VARIABLES

NOMBRE VARIABLE	DESCRIPCION	VALORES
DIR1	Nombre del pin6, configurado como la salida para la dirección del motor	LOW= movimiento hacia abajo HIGH = movimiento hacia arriba
CLK1	Nombre del pin7, configurado como salida de los pulsos de motor	LOW HIGH
OPTO1	Nombre del pin8, configurado como entrada para la lectura de optoacoplador	LOW HIGH
FINAL1	Guarda el valor del sensor de final de carrera	0= el sensor no está pulsado 1= el sensor está pulsado
DIR2	Nombre del pin6, configurado como la salida para la dirección del motor	LOW= movimiento hacia derecha HIGH = movimiento hacia izquierda
CLK2	Nombre del pin7, configurado como salida de los pulsos de motor	LOW HIGH
OPTO2	Nombre del pin8, configurado como entrada para la lectura de optoacoplador	LOW HIGH
FINAL2	Guarda el valor del sensor de final de carrera	0= el sensor no está pulsado 1= el sensor está pulsado
matlabDato	Guarda los datos que se reciben por el puerto serie	Valores enteros
valor1	Guarda el valor de los grados para el movimiento vertical	Valores enteros
p1	Guarda el valor calculado de los peines que va a girar el encoder	Valores enteros
direccion1	Guarda el valor de la dirección seleccionada por el usuario	0= movimiento hacia abajo 1 = movimiento hacia arriba
valor2	Guarda el valor de los grados para el movimiento vertical	Valores enteros
p2	Guarda el valor calculado de los peines que va a girar el encoder	Valores enteros
direccion2	Guarda el valor de la dirección seleccionada por el usuario	0= movimiento hacia izquierda 1 = movimiento hacia derecha

contador1	Variable para el conteo de los peines del motor del eje vertical	Valores enteros de 1 a 72
edoanterior1	Guarda el anterior del optoacoplador para realizar el conteo de peines	Valores enteros 0 y 1
valoropto1	Guarda el valor actual del optoacoplador	Valores enteros 0 y 1
contcero1	Variable para el conteo de ceros para detector error en el motor del eje vertical	Valores enteros
contuno1	Varaible para el conteo de unos para detectar error en el motor del eje vertical	Valores enteros
contcero2	Variable para el conteo de ceros para detector error en el motor del eje horizontal	Valores enteros
contuno2	Varaible para el conteo de unos para detectar error en el motor del eje horizontal	Valores enteros
fin1	Variable para condición si el motor del eje vertical llega a un punto final	Valores enteros
fin2	Variable para condición si el motor del eje horizontal llega a un punto final	Valores enteros
out1	Guarda el valor leído del piranómetro	Valores analógicos

```
function varargout = GUI_seguidor_solar(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_seguidor_solar_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_seguidor_solar_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before GUI_seguidor_solar is made visible.
function  GUI_seguidor_solar_OpeningFcn(hObject,  eventdata,  handles,
varargin)

handles.output = hObject;

guidata(hObject, handles);

set( findall(handles.verPanel, '-property', 'Enable'), 'Enable', 'off')
set( findall(handles.horPanel, '-property', 'Enable'), 'Enable', 'off')

% --- Outputs from this function are returned to the command line.
function  varargout  =  GUI_seguidor_solar_OutputFcn(hObject,  eventdata,
handles)
varargout{1} = handles.output;


function vertGrados_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
   get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function horGrados_CreateFcn(hObject, eventdata, handles)
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function btInicio_Callback(hObject, eventdata, handles)
pserial=handles.CONECTADO;
estado = get(handles.modoInicio,'Value');
if estado==1 %Volver a posición de inicio
    fprintf(pserial,'1');
end

estado = get(handles.modoAut,'Value');
if estado==1
    fprintf(pserial,'2');
    dato2=0;

while(~(pserial.BytesAvailable))
end

q=fgetl(pserial); %lectura del puerto
dato2=(str2double(q))

if dato2==1
    axes(handles.axes1);
    xlabel('Muestra');
    ylabel('Voltaje de salida');
    val=zeros(500,1);
    i=1;
    lHandle=line(nan, nan)
    sensor =1;


    while sensor~=0
        ylim([0 13]);
        xlim([0 i+5]);
        sensor=fscanf(pserial,'%d')
        val(i)=sensor*12.75/1023;
        X = get(lHandle, 'XData');
        Y = get(lHandle, 'YData');
        x = [X i];
        y = [Y val(i)];
        set(lHandle, 'XData', x, 'YData', y);
        drawnow
        set(handles.Valor,'string',num2str(val(i)))
        i=i+1;
    end
    xlswrite('Libro1.xls',val);
    end
end

estado = get(handles.modoMan,'Value');
if estado==1
    v=zeros(1,5);
    v(1)=3;
    Up= get(handles.verUp,'Value');
    if Up==1
        v(2)=1;
    end
    Down = get(handles.verDown,'Value');
    if Down==1
        v(2)=0;
    end
    v(3)=handles.verGrados;
```

```
    Right= get(handles.horRight,'Value');
    if Right==1
        v(4)=1;
    end
    Left = get(handles.horLeft,'Value');
    if Left==1
        v(4)=0;
    end
    v(5)=handles.horGrados;
    fprintf(pserial,num2str(v));
    dato2=0;
    valor=1;

    while(~(pserial.BytesAvailable))
    end

    q=fgetl(pserial); %lectura del puerto
    dato2=(str2double(q))

    if dato2==1
        while valor~=0
            valor=fscanf(pserial,'%d');
            if valor==12592;
                msj=fgetl(pserial);
                set(handles.txt5,'string',msj)
            end
            if valor==12593;
                msj=fgetl(pserial);
                set(handles.txt7,'string',msj)
            end
        end

        if valor==0
            valor2=fscanf(pserial,'%d')
            set(handles.Valor,'string',num2str(valor2))
        end
    end

end


function verGrados_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function modoPanel_SelectionChangeFcn(hObject, eventdata, handles)
```

```
estado = get(handles.modoInicio,'Value');
if estado==1
   set( findall(handles.verPanel, '-property', 'Enable'), 'Enable',
   'off')
   set( findall(handles.horPanel, '-property', 'Enable'), 'Enable',
   'off')
end

estado = get(handles.modoAut,'Value');
if estado==1
    set( findall(handles.verPanel, '-property', 'Enable'), 'Enable',
    'off')
    set( findall(handles.horPanel, '-property', 'Enable'), 'Enable',
    'off')
end

estado = get(handles.modoMan,'Value');
if estado==1
    set( findall(handles.verPanel, '-property', 'Enable'), 'Enable',
    'on')
    set( findall(handles.horPanel, '-property', 'Enable'), 'Enable',
    'on')
end


function btConectar_Callback(hObject, eventdata, handles)
clc;
pserial=serial('COM3');
set(pserial,'BaudRate',9600);
set(pserial,'DataBits',8);
set(pserial,'Parity','none');
set(pserial,'StopBits',1);
set(pserial,'FlowControl','none');
fopen(pserial);
handles.CONECTADO=pserial;
guidata(hObject,handles);


function btParar_Callback(hObject, eventdata, handles)
pserial=handles.CONECTADO;
fclose(pserial);
delete(pserial)
clear pserial


function verGrados_Callback(hObject, eventdata, handles)
Val=get(hObject,'String');%Almacenar valor ingresado
NewVal = str2double(Val);%Transformar a formato double
handles.verGrados=NewVal;%Almacenar en identificador
guidata(hObject,handles);%Salvar datos de la aplicación


function horGrados_Callback(hObject, eventdata, handles)
Val=get(hObject,'String');%Almacenar valor ingresado
NewVal = str2double(Val);%Transformar a formato double
handles.horGrados=NewVal;%Almacenar en identificador
guidata(hObject,handles);%Salvar datos de la aplicación
```

```
function Valor_CreateFcn(hObject, eventdata, ~)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

DESCRIPCION DE VARIABLES DE MATLAB

NOMBRE VARIABLE	DESCRIPCION	VALORES
btInicio	Botón Inicio	0 y 1
btConectar	Botón conectar	0 y 1
btParar	Botón parar	0 y 1
modoInicio	Botón radio para modo Inicio	0 y 1
ModoAut	Boton radio para moto automático	0 y 1
ModoMan	Botón radio para modo manual	0 y 1
vertGrados	Cuadro de texto para los grados del movimiento vertical	Número en formato string
horGrados	Cuadro de texto para los grados del movimiento horizontal	Número en formato string
verUp	Botón radio para dirección hacia arriba del motor vertical	0 y 1
verDown	Botón radio para dirección hacia abajo del motor vertical	0 y 1
horLeft	Botón radio para dirección hacia izquierda del motor horizontal	0 y 1
horRight	Botón radio para dirección hacia derecha del motor horizontal	0 y 1
pserial	Guarda la información para realizar la conexión serial	
estado	Guarda el valor actual de la selección del modo de operación	0 y 1
q	Guarda el valor recibido por puerto serie	Valor en string 1
dato2	Guarda el valor recibido por puerto serie convertido. Dato que envía Arduino antes de enviar la información del sensor	Valor en double
val	Matriz para almacenar los valores del piranómetro	Matriz de 500 por 1
i	Contador para incrementar la fila en la matriz val	Valores enteros
sensor	Guarda el valor leido por puerto serie correspondiente al piranometro	
v	Vector para enviar los datos del modo manual por puerto serie	Vector de 1 por 5
Up	Guarda el valor del botón radio verUP	0 y 1
Down	Guarda el valor del botón radio verDown	0 y 1
Right	Guarda el valor del botón radio horRight	0 y 1
Left	Guarda el valor del botón radio horLeft	0 y 1

valor	Guarda el valor recibido por puerto serie para identificar mensajes	Valores enteros
msj	Guarda el mensaje enviado por puerto serie	Texto String
valor2	Guarda el valor recibido por puerto serie correspondiente al piranómetro	Valores enteros