Graduado en Ingeniería de Computadores

Diseño y construcción de un prototipo de placa solar orientable para la docencia en asignaturas de Control Automático

Design and construction of an orientable solar panel for teaching in Control Engineering subjects

Realizado por

Javier Jaime Pérez

Tutorizado por

Juan Antonio Fernández Madrigal

Ana Cruz Martín

Departamento

Ingeniería de Sistemas y Automática

MÁLAGA, septiembre de 2024

Resumen

En este proyecto se ha realizado el diseño, construcción y caracterización de un prototipo de placas solares que pueden orientarse según la posición de una fuente de luz. Para ello, se ha hecho uso principalmente de dos placas solares, un motor de corriente continua y un Arduino UNO, así como de Matlab y su herramienta *rltool* para el diseño del controlador que después ha sido implementado en lenguaje C. El objetivo es que este proyecto pueda ser usado como material práctico para la docencia en asignaturas de Control Automático, de forma que complemente los contenidos teóricos en el modelado de sistemas físicos, su respuesta temporal y su control. Uno de los principales puntos que se han tenido en cuenta a lo largo del proyecto es conseguir la máxima linealidad posible en el sistema, ya que en las asignaturas introductorias esta es la base teórica que se enseña.

Palabras Claves: Control automático, Matlab, Arduino, Placas solares, Motor DC.

Abstract

In this project, the design, construction and characterization of a prototype of solar panels that can be oriented according to the position of a light source has been carried out. For this, we have made use of two solar panels, a DC motor and an Arduino UNO, as well as Matlab and its *rltool* for the design of the controller, that has been implemented in C language. The goal is that this project can be used as practical material for teaching in Automatic Control subjects, so that it complements the theoretical contents in the modeling of physical systems, their time response and their control. One of the main points that have been considered throughout the project is to get the maximum possible linearity in the system, since this is the theoretical basis that is taught in introductory courses.

Keywords: Automatic: Automatic control, Matlab, Arduino, Solar panels, DC Motor.

Índice

Índice de Figuras

Capítulo 1

Introducción

En el último curso de la titulación del grado en Ingeniería de Computadores se imparte la asignatura "Control por Computador", donde se aborda tanto el contenido teórico como práctico necesario para diseñar controladores para sistemas físicos. El enfoque principal del curso se centra en sistemas lineales e invariantes en el tiempo (LTI). Durante el curso se aprende a trabajar con diversas herramientas matemáticas para el análisis de dichos sistemas y el diseño de controladores.

Al finalizar la asignatura, los alumnos son capaces de, con la herramienta *rltool* de Matlab, diseñar estos controladores y determinar con lo visto en clase si esos diseños son adecuados. Sin embargo, implementar estos diseños en el mundo físico resulta complicado por el coste del equipamiento necesario, principalmente. En los últimos cursos se han utilizado robots Lego Mindstorms EV3, pero tienen bastantes no linealidades y además hace algunos años que ya no se fabrican.

Así pues, iniciamos este trabajo fin de grado con el propósito de ofrecer un recurso práctico para las asignaturas de Control Automático. El objetivo principal ha sido diseñar y construir un sistema físico de bajo coste capaz de seguir automáticamente una fuente de luz externa, utilizando para ellos dos pequeñas placas solares, un motor

DC y un controlador basado en Arduino UNO. Para lograr esto, hemos caracterizado los componentes del sistema e implementado controladores en lenguaje C.

Las tareas más importantes del proyecto han sido las siguientes:

- Se han considerado criterios de linealidad en el diseño del sistema para poder aplicar técnicas de control lineal clásico.
- Se han llevado a cabo diferentes tipos de pruebas para realizar la caracterización de los componentes; estos datos se han recopilado y han sido analizados con Matlab para obtener información sobre la respuesta temporal.
- Se han utilizado técnicas de modelado con ecuaciones diferenciales, diagramas de bloques y transformada de Laplace para obtener las funciones de transferencia de esos componentes.
- Se ha utilizado la herramienta *rltool* de Matlab para diseñar el controlador basado en el lugar de las raíces una vez obtenidas las funciones de transferencia.
- Se han implementado controladores lineales en lenguaje C y se han comparado y ajustado según su rendimiento real.

Uno de los principales desafíos del proyecto ha sido lograr que el sistema sea lo más lineal posible; a lo largo de esta memoria se detallarán los obstáculos encontrados y las no linealidades inevitables, así como los resultados finales obtenidos.

1.1 Estado del arte

Los seguidores solares (*solar trackers*) son dispositivos mecánicos que orientan los paneles solares buscando mantener estos perpendiculares a los rayos del sol, con el objetivo de maximizar la exposición de los paneles a la luz y con esto poder generar más energía. Existen seguidores uniaxiales (ver Figura 1) que solo siguen la luz en el horizonte y biaxiales (ver Figura 2), que también pueden modificar el ángulo de inclinación de las placas solares [1].

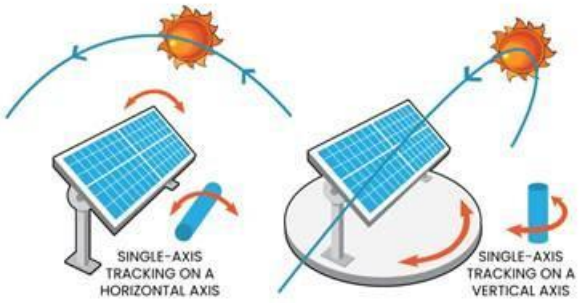


Figura 1. Seguidor solar uniaxial [1].

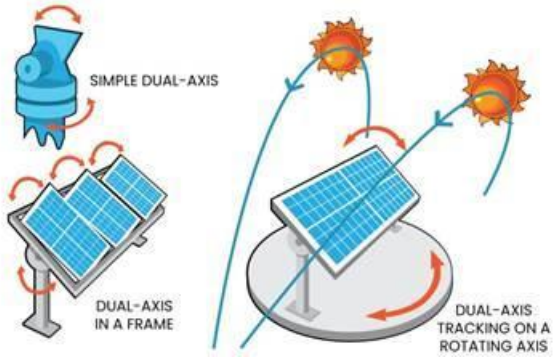


Figura 2. Seguidor solar biaxial [1].

Uno de los trabajos más interesantes es un Agile Eye (ver Figura 3), esto es, un proyecto avanzado en términos de mecánica y de control cuyo objetivo es hacer movimientos rápidos y precisos en la orientación [2].

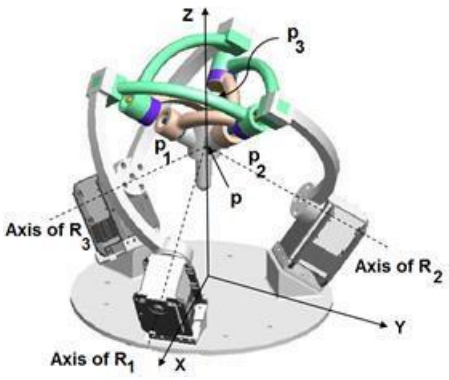


Figura 3. Agile Eye [1].

Otro trabajo implementa un seguidor solar biaxial [3].

Como nuestro propósito es proporcionar un material práctico para las asignaturas de Control Automático, hemos implementado un seguidor solar uniaxial, es decir nuestro objetivo será hacer girar las placas solares en el eje horizontal, ya que con esto sería suficiente para poder realizar el diseño e implementación de diferentes tipos de controladores lineales clásicos. Además, hemos perseguido mantener el coste más limitado posible para que pueda ser asequible para un departamento universitario, y el desarrollo ha tenido en todo momento en cuenta las necesidades concretas de la asignatura en el contexto de la cual surgió.

1.2 Estructura del documento

En este capítulo hemos descrito la motivación para desarrollar este proyecto, así como algunas de las tareas que se han realizado a lo largo de este; también en qué situación se encuentra en el mundo actual dicho problema, las diferentes formas de afrontarlo, varios proyectos que hemos examinado como referencia y cómo vamos a afrontar nosotros el trabajo fin de grado. Finalmente, en este apartado describimos la estructura del resto del documento.

En el capítulo de herramientas utilizadas (capítulo 2) presentamos aquellos recursos hardware y sobre todo software que han sido necesarios para desarrollar el proyecto. Se explica el motivo de la elección de estos, para qué y cómo se usan y las configuraciones que han sido necesarias para el funcionamiento correcto del trabajo.

En el diseño hardware (capítulo 3) describimos aquellos materiales que se han adquirido y en qué nos hemos basado para escogerlos. En un segundo apartado veremos la caracterización de dos de los componentes, las placas solares y el motor DC; con ellos pretendemos realizar diferentes experimentos para ver cuál es su comportamiento en

diferentes situaciones con el objetivo de obtener un modelo del sistema a partir de sus funciones de transferencia.

Para el diseño de controladores (capítulo 4) veremos cómo haciendo uso de las funciones de transferencia calculadas en el capítulo 3 podemos obtener diferentes tipos de controladores para nuestro sistema, haciendo uso de la herramienta *rltool*.

En los experimentos reales (capítulo 5) implementaremos los controladores diseñados sobre la placa Arduino, y comprobaremos su comportamiento en el sistema real.

Por último, en el capítulo de conclusiones y futuros trabajos (capítulo 6) enumeramos todos aquellos aspectos que hemos conseguido cubrir en el proyecto, y también aquellos que han quedado pendientes y que se podrían abordar en el futuro para mejorar el proyecto.

Capítulo 2

Herramientas utilizadas

2.1 Arduino UNO

La placa de desarrollo que hemos elegido para desarrollar el proyecto ha sido un Arduino UNO (ver Figura 4) [4].

Uno de los motivos que han llevado a seleccionar esta placa, además de su coste reducido y facilidad de programación, es porque a lo largo de la carrera hay varias asignaturas en las que se trabaja con ella. De esta manera, el contenido de las asignaturas de Control Automático se podrá centrar en el aprendizaje del control, ya que los estudiantes estarán familiarizados con el uso de esta herramienta.

Para programarla existe la opción de utilizar el IDE (*Integrated Development Environment*, entorno integrado de desarrollo) de Arduino y aprovechar así los recursos de las múltiples bibliotecas de las que este dispone; sin embargo, cuando empezamos a plantear el proyecto sabíamos que íbamos a tener que hacer uso de *timers* y ondas PWM, entre otros elementos, y al usar librerías implementadas por terceros podríamos encontrar problemas de compatibilidad, con lo que habríamos tenido que replantear todo el software del proyecto.

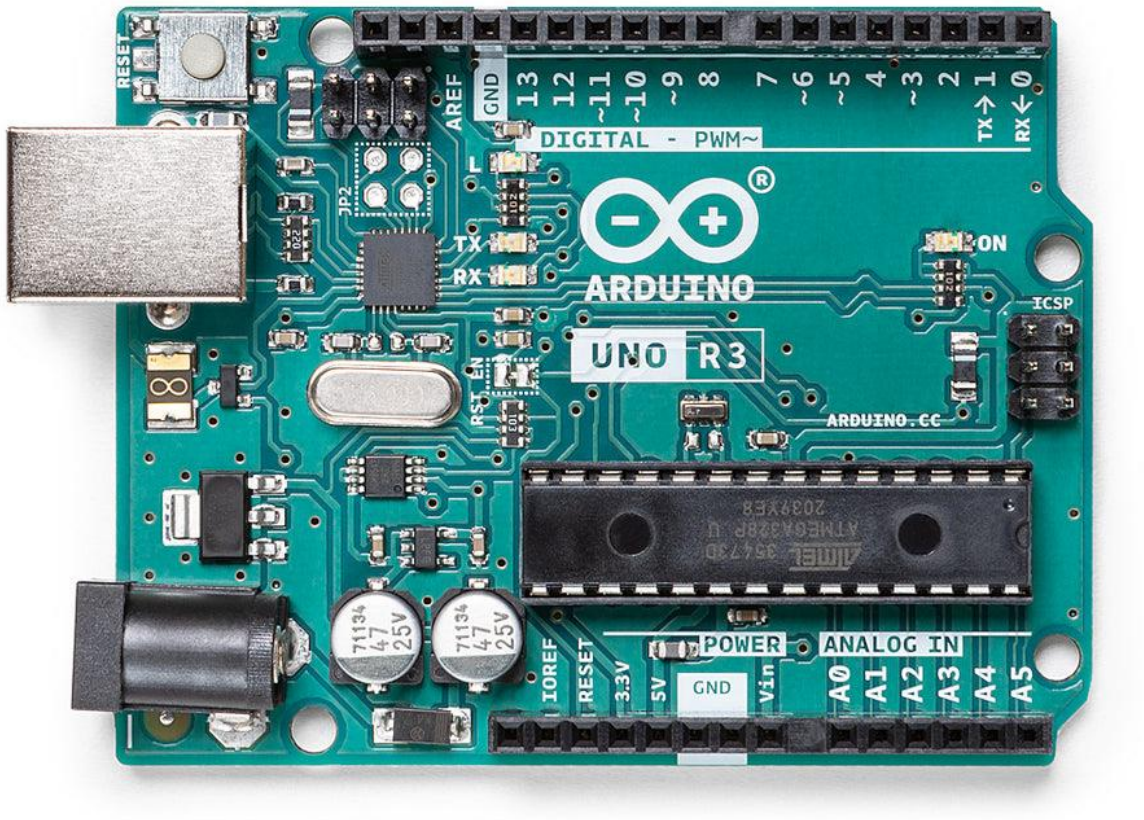


Figura 4. Arduino Uno en su revisión 3 [4].

Por esta razón se opta por programar el Arduino en *bare-metal*, es decir, programando directamente el microcontrolador ATmega328P [5] utilizando sus registros y periféricos en lugar de funciones y abstracciones proporcionadas por el *framework* de Arduino. Esto nos ha aportado una serie de ventajas, como es tener un control completo sobre el hardware, poder optimizar el rendimiento y el uso de recursos, y también hacer códigos más eficientes.

En el Grado de Ingeniería de Computadores en la asignatura de Sistema de Tiempo Real se enseña cómo programar el Arduino UNO en *bare-metal*. Para ello, una de las cosas que nos hace falta es conocer la correspondencia de los pines del Arduino UNO con los del microcontrolador (ver Figura 5).

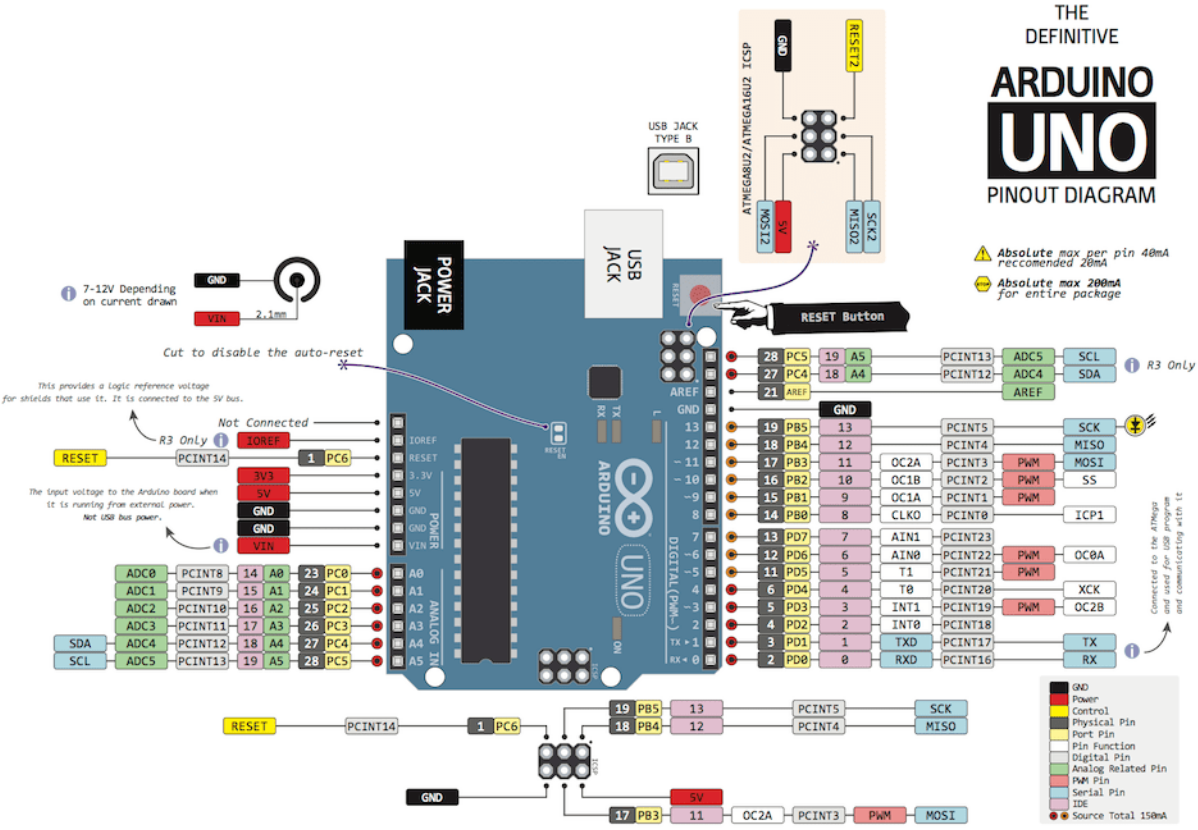


Figura 5. Correspondencia de pines Arduino UNO [6].

Hemos usado la hoja de especificaciones del microcontrolador [7], que contiene información detallada sobre los registros, periféricos, memoria y funcionamiento general del microcontrolador, así como algunas librerías de AVR [8].

2.2 Microchip Studio

El microcontrolador ATmega328P pertenece a la familia de AVR del fabricante Atmel, ahora parte de Microchip Technology [9]. Esta compañía ha desarrollado el IDE Microchip Studio [10], empleado en el proyecto para programar y depurar la aplicación del microcontrolador. Actualmente está migrando sus herramientas de desarrollo a MPLAB, pero aún es útil y funcional el primero.

Para crear un proyecto en Microchip Studio debemos realizar una serie de configuraciones iniciales. Primero, seleccionamos “New Project”. Como nuestra

aplicación será en C, elegimos la opción "GCC C Executable Project", nombramos el proyecto y seleccionamos el directorio de destino. Luego tendremos que seleccionar el dispositivo, en este caso, el ATmega328P, que se puede buscar desde una pestaña en la que tenemos enlaces a la *web* del microcontrolador y su *datasheet (ver* Figura 6*)*.

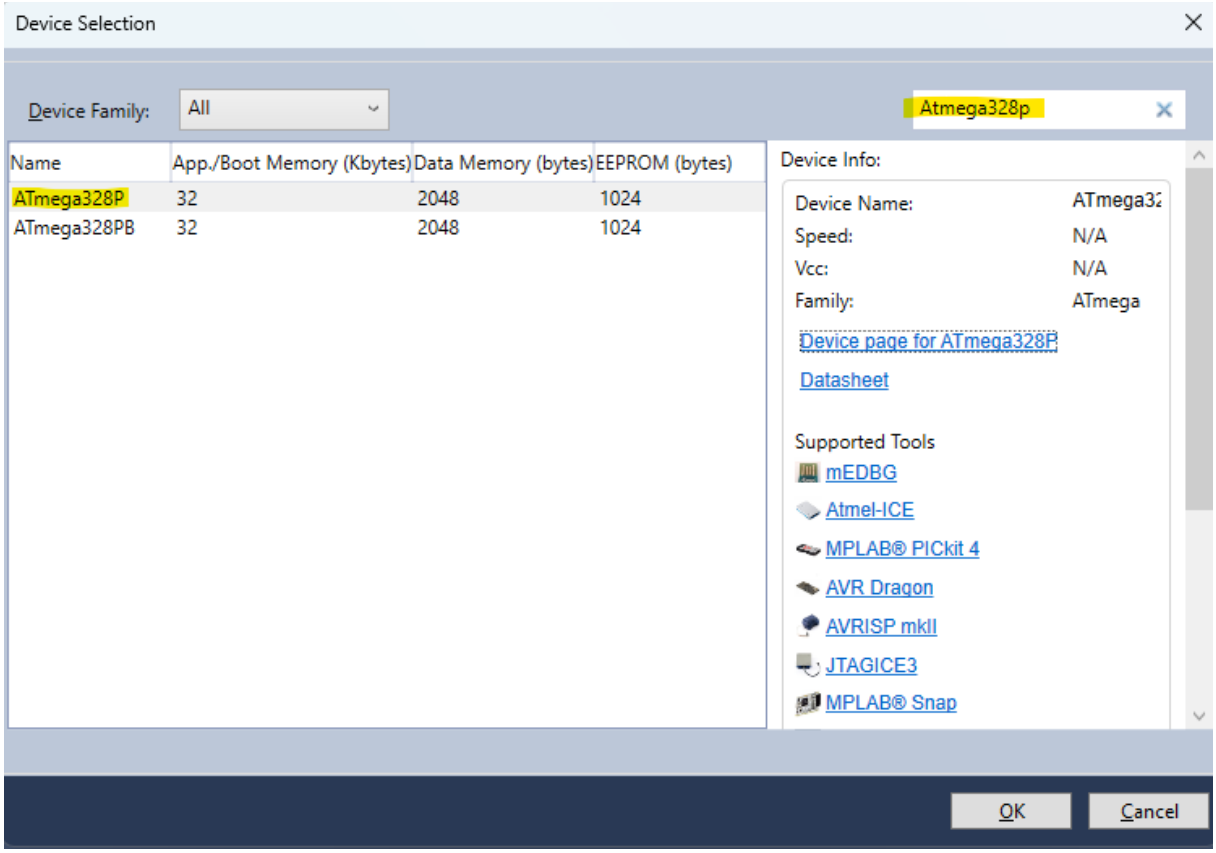


Figura 6. Microchip Studio – Selección de microcontrolador.

Dado que estamos utilizando un microcontrolador AVR, necesitaremos un compilador específico para convertir el código fuente en código máquina que pueda ser ejecutado por ese microcontrolador. En este caso, utilizamos AVR-GCC, el cual ya viene incluido con el IDE de Microchip Studio. El compilador AVR-GCC convierte nuestro código fuente en un archivo binario. Este archivo es el *firmware* que contiene las instrucciones que el microcontrolador ejecutará.

Una vez que tenemos el archivo binario generado por el compilador, necesitamos cargar este *firmware* en el microcontrolador. Para ello, utilizamos un programador de hardware (ver Figura 7). Haremos uso de AVR-Dude [11] una herramienta de línea de comandos. Esta herramienta deberemos descargarla [11] por separado.

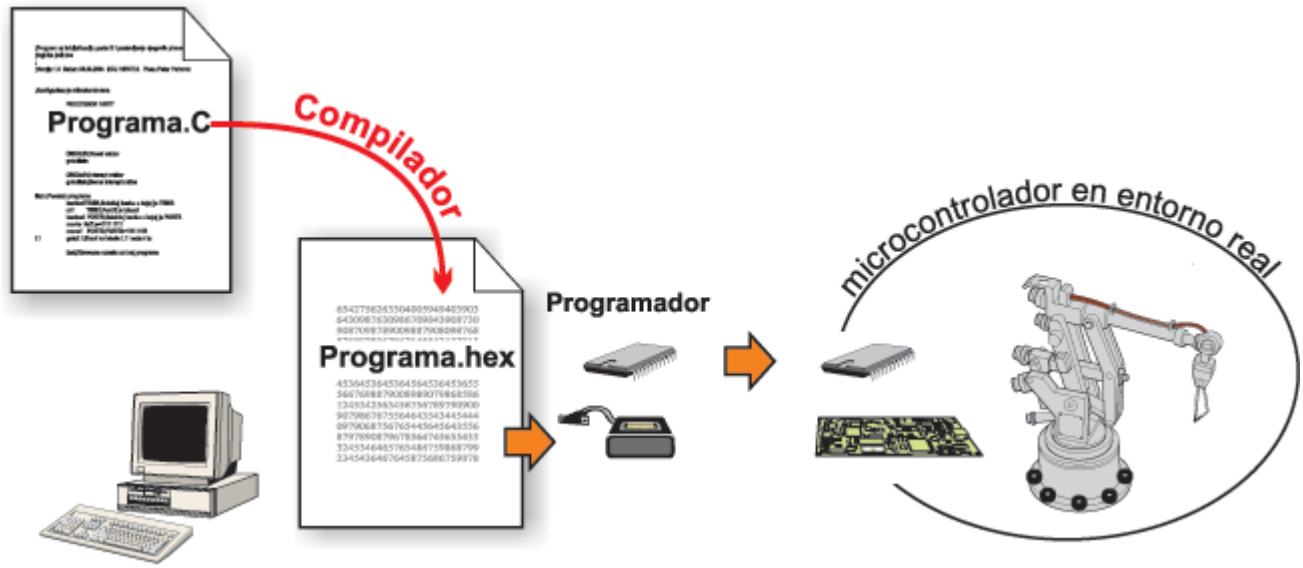


Figura 7. Compilación y carga del firmware en el ATmega328P [7].

En el IDE se puede configurar el uso de la herramienta de AVR-Dude; para ello, en la barra superior iremos a "Tools" -> "External Tools", seleccionamos "Add" para crear un nuevo *script*, en "Title" podemos poner el nombre que queramos (vamos a referirnos a él como ArduinoDownload).

La ruta del comando se ejecutará cuando lancemos el *script* desde AVR Studio; aquí deberemos poner la ubicación del programador que hemos descargado de AVR-Dude, el fichero `avrdude.exe`. La ruta, en caso de estar en Windows, debería ser similar a: `C:\...\avrdude-vX.X\avrdude.exe`

En el apartado de argumentos, le indicaremos a `avrdude.exe` qué programa debe cargar en Arduino. De igual manera que antes, en este caso deberemos indicarle la ruta

donde tengamos el fichero `avrdude.conf` y sus parámetros (ver Figura 8). El campo será parecido al siguiente:

```
-C C:\...\ardude-vX.X\avrdude.conf -v -pATmega328p -carduino -
P\\.\COM5 -b115200 -
Uflash:w:$(ProjectDir)Debug\$(TargetName).hex:i
```

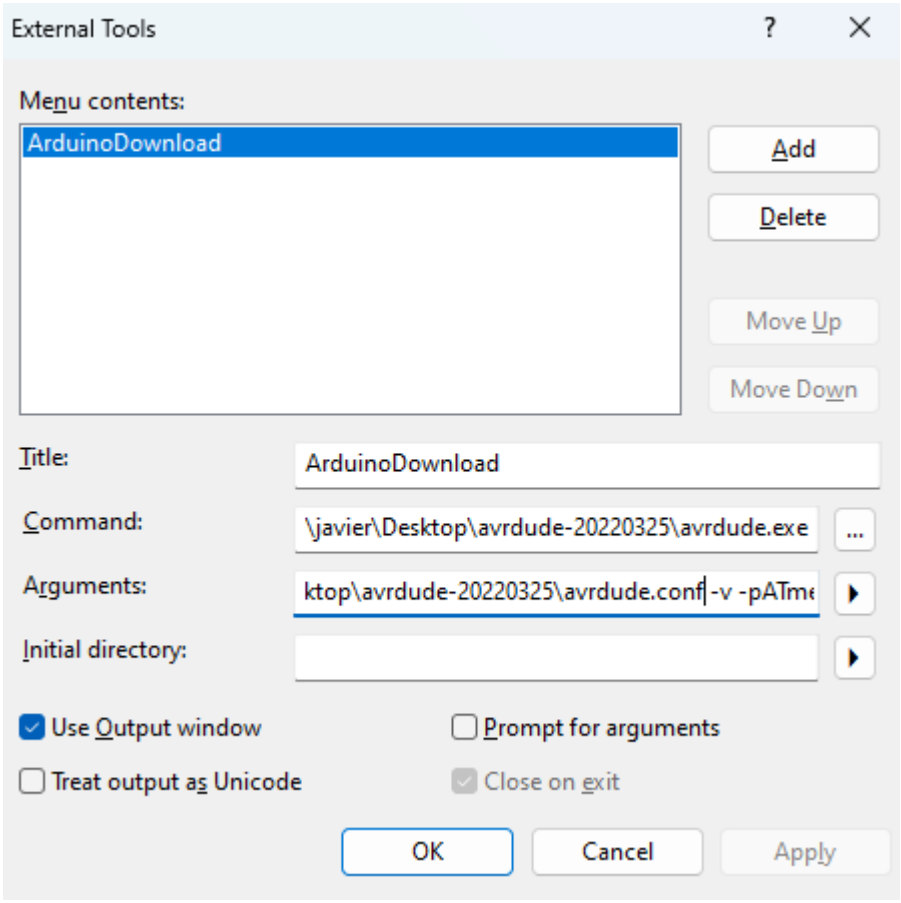


Figura 8. Configuración de AVR-Dude en el IDE de Microchip Studio.

-C *ruta* : fichero de configuración requerido por `avrdude.exe`.

-v : incrementa verbosidad (más información durante la programación).

-pATmega328p -carduino : indica que se trata de un Arduino con un ATmega328P.

--P\\.\COM5 : indica el puerto serie del Arduino. Hay que consultar el puerto COM de la placa conectada.

-b115200 : velocidad del puerto.

-Uflash:w:$(ProjectDir)Debug\$(TargetName).hex:i : dónde se encuentra el código generado por el IDE.

Después de compilar el código, cada vez que queramos cargarlo deberemos ir a la pestaña "Tools" y pulsar sobre "ArduinoDownload".

El IDE proporciona, entre otras herramientas, la opción de ver el código en ensamblador, un depurador y un simulador que permite ver el estado de los registros sin necesidad de tener el hardware físico. Hay un manual de referencia desarrollado por la compañía donde se explica cómo hacer uso de estas herramientas [12].

2.3 Matlab

Durante el desarrollo del proyecto, se han llevado a cabo una serie de experimentos que requieren la recopilación, procesamiento y análisis de datos con el objetivo de validar los resultados obtenidos. Para realizar estas tareas se ha optado por utilizar Matlab [13], un entorno de programación y simulación numérica de sistemas.

Para poder analizar los datos producidos por el Arduino, éstos se almacenaron en ficheros tras ser recibidos por el puerto serie. Una vez recopilados, Matlab permitió su procesamiento. Uno de los aspectos destacados de Matlab en este proyecto ha sido su capacidad para generar visualizaciones claras y significativas mediante la creación de gráficos. Esto ha permitido representar los resultados obtenidos de manera visual, y ha facilitado la interpretación de los datos y la validación de los resultados experimentales. Además, con estas visualizaciones y haciendo uso de otras herramientas de Matlab hemos podido obtener más datos relevantes para continuar con el desarrollo del proyecto, como la identificación de sistemas o el diseño de controladores.

Matlab cuenta con una sección llamada "add-ons"; desde ahí podemos instalar extensiones o complementos que nos permitirán acceder a funciones y herramientas

adicionales diseñadas para tareas específicas. Nosotros hemos hecho uso de los siguientes:

- ***System Identification Toolbox*** [15]**:** esta herramienta se ha utilizado con el propósito de construir modelos matemáticos de nuestro sistema. Contábamos con los datos de entrada y salida de nuestro sistema, obtenidos a partir de experimentos previos; el objetivo era conseguir las funciones de transferencia partir de esos datos experimentales. Para lograr esto, primero se importan los datos usando herramientas de la *toolbox*, se utilizan métodos de identificación de sistemas y, una vez obtenido el modelo se realiza su validación comparando su respuesta con los datos obtenidos en los experimentos. Tras asegurarnos de que el modelo matemático es correcto, podemos pasar a diseñar el controlador.
- ***Control System Toolbox*** [14]**:** esta *toolbox* permite analizar, diseñar y ajustar sistemáticamente sistemas de control lineal. Esta herramienta es la que nos facilita la teoría y práctica del Control Automático, permitiéndonos diseñar controladores, analizar su estabilidad y rendimiento.

A lo largo de este documento se irán viendo y explicando el uso de funciones que pertenecen a estas herramientas.

2.3.1 *Rltool*

Para un sistema de control cuya función de transferencia en bucle abierto se defina como:

$$TF_{OL}(s) = KG(s)H(s)$$

su función de transferencia en bucle cerrado se definirá de la siguiente manera:

$$TF_{CL}(s) = \frac{KG(s)}{1 + KG(s)H(s)}$$

donde K representa la ganancia del controlador, G(s) la función de transferencia de la planta y H(s) la del sensor. En este caso el controlador se está representando como una

constante K que es una ganancia; más adelante será una función de transferencia más general que denotaremos como C(s). El diagrama de bloques de esta función de transferencia de bucle cerrado es el que tenemos en la Figura 9.

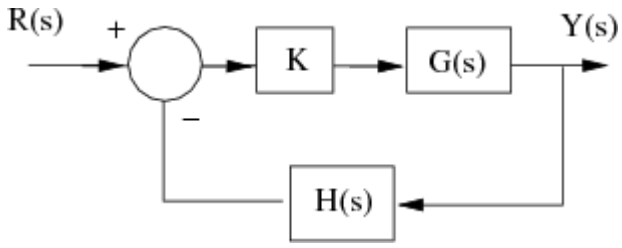


Figura 9. Diagrama de bloques de una F.T. en bucle cerrado [13].

Los polos de bucle cerrado son las raíces del denominador de la función de transferencia de bucle cerrado.

El lugar de las raíces muestra gráficamente cómo cambian las raíces de la función de transferencia de bucle cerrado (ver Figura 9) a medida que se modifica la ganancia del controlador. Este gráfico representa el movimiento de los polos del sistema de control en el plano complejo, lo que permite visualizar la región del plano complejo donde pueden ubicarse los polos de bucle cerrado. La posición de estos polos determina la respuesta temporal del sistema, su estabilidad y rendimiento.

La herramienta *rltool* [16] forma parte de la *Control System Toolbox*. Proporciona una interfaz gráfica que facilita el diseño y análisis de controladores para sistemas lineales utilizando la técnica del "lugar de las raíces" (*root locus,* Figura 10). Esta herramienta es fundamental para ajustar la ganancia K y observar cómo afecta a la respuesta temporal, la estabilidad y el rendimiento del sistema.

Estos pueden ser movidos en el plano complejo variando la ganancia K en *rltool*.

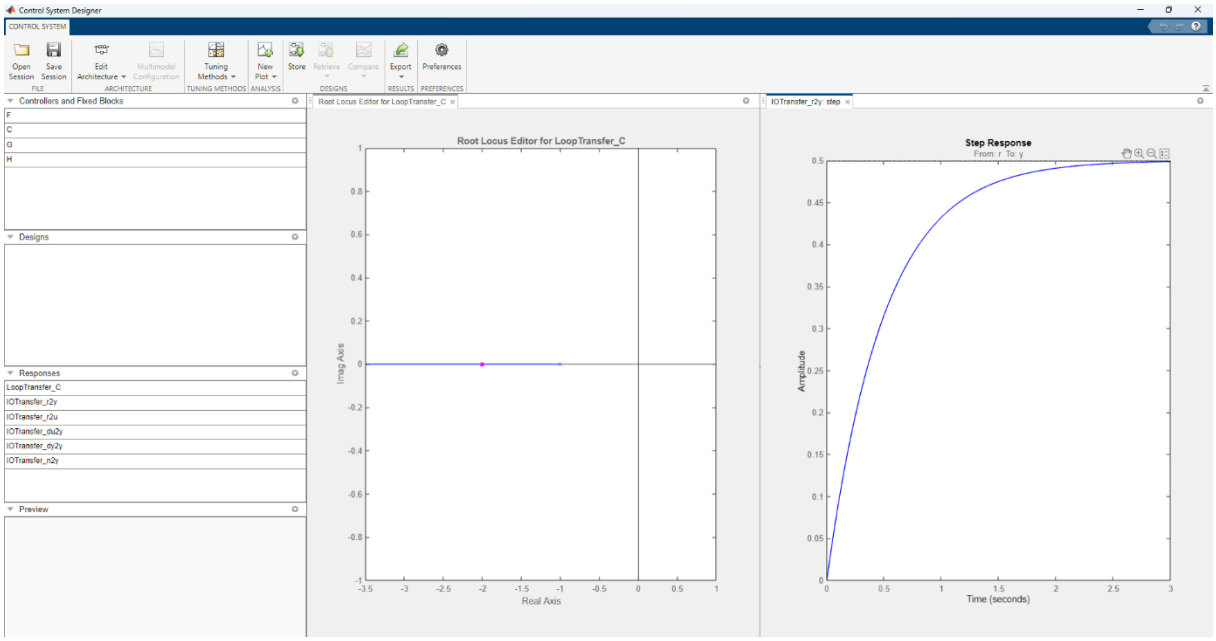


Figura 10. Lugar de las raíces en la rltool.

Capítulo 3

Diseño hardware

En este apartado detallaremos todo el proceso de construcción del hardware del dispositivo, desde la compra de los materiales hasta la caracterización de los componentes seleccionados. Dado su importancia para el desarrollo del trabajo fin de grado, nos detendremos con más detalle en el punto donde obtenemos las funciones de transferencia.

3.1 Elección de componentes

Como la placa de desarrollo que se ha usado es un Arduino UNO, para la compra de los materiales se debían tener en cuenta sus características y limitaciones.

Para empezar, para alimentar el Arduino UNO podemos usar un cable USB a 5V o el conector *Jack*, con el que se recomienda trabajar con voltajes entre 7 y 12V (aunque el rango completo es de 6-20V). También se puede usar una fuente de alimentación externa conectada a los pines Vin y GND. El voltaje que puede replicar la placa a través de sus pines de alimentación es de 3,3V y 5V.

Los pines de entrada/salida (E/S) de la placa trabajan con un voltaje máximo de 5V y pueden suministrar o recibir hasta 20mA por pin (aunque el límite absoluto es de 40mA por pin). La corriente máxima total que la placa puede manejar a través de

todos los pines de E/S combinados no debe exceder los 200mA. Por tanto, no podemos conectar a la placa directamente un motor de corriente continua.

El Arduino UNO dispone de entradas analógicas, es decir, señales cuya magnitud puede tomar un valor dentro de un intervalo entre $-V$ y $+V$. Estas entradas son generalmente más escasas, lentas y debido a su complejidad más caras en cuento a costes respecto a las entradas digitales. La representación de una entrada analógica se realiza con un valor digital formado por N bits, lo cual se lleva a cabo con un conversor analógico digital (ADC). Cuanto mayor sea el número de bits, mayor será el número de intervalos, menor el ancho de cada intervalo y, en consecuencia, mejorará la precisión de la medición [17].

3.1.1 Placas solares

Para hacer la selección de las placas solares debíamos fijarnos principalmente en el voltaje y la intensidad máximos que podía recibir el Arduino. Teniendo esto en cuenta podíamos realizar un filtrado entre las diferentes opciones. Aparte de buscar unas placas solares que pudieran cumplir estas características, otro de los aspectos que tuvimos en cuenta fue que hubiera una hoja de datos o *datasheet*, que, en la medida de lo posible, estuviera lo más detallada posible por parte del fabricante. Esto finalmente nos llevó a escoger las placas solares AM-5904CAR (ver Figura 11) [18].

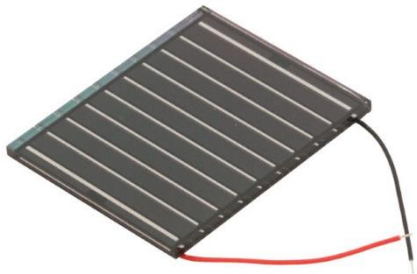


Figura 11. Placa solar AM-5904CAR.

Para poder trabajar con esta placa solar hay que tener en cuenta una serie de detalles, por lo que deberemos consultar el *datasheet* [19] del fabricante.

La placa, AM-5904, tiene un área aproximadamente de $13\ cm^2$ $(40{,}1\text{mm} \times 33{,}1\text{mm})$. Las siglas AM nos están indicando que se trata de una placa solar de vidrio de tipo *Glass-Type* (ver Figura 12), el número 5 que está fabricada para exteriores y el número 9 es el número de celdas.

En el documento se indica la manera de calcular la potencia máxima de la placa solar, sabiendo que proporciona $7{,}89mW/cm^2$. Como ya sabemos cuál es el área, calculamos que la potencia máxima es de 0,10W. Dado que la potencia es $P = V \cdot I$, entonces debe cumplirse que $V \cdot I \leq 0{,}1W$.

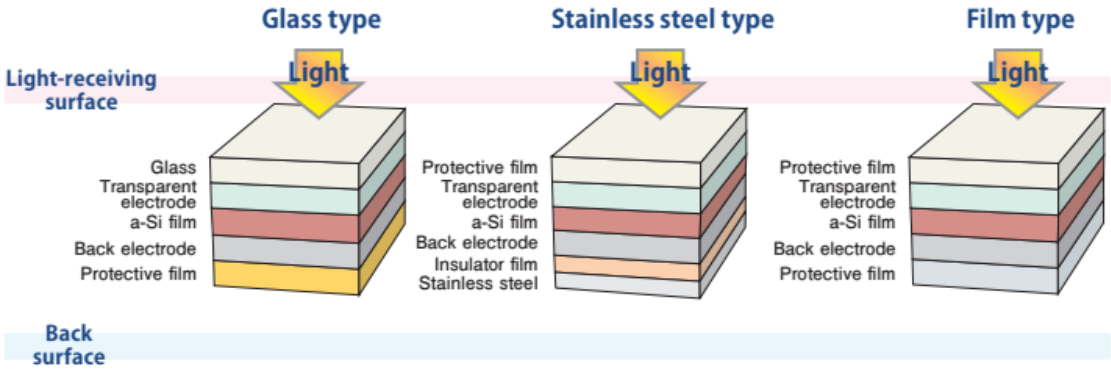


Figura 12. Diseño de diferentes paneles solares.

De igual manera, en el documento se encuentra la información necesaria para calcular cuál es el voltaje máximo que puede proporcionar la placa cuando tenemos el circuito en abierto. Este es de $0{,}89V \cdot\ nº\ Celdas = \ 0{,}89V \cdot\ 9 = 8{,}01V$.

Como hemos dicho, tenemos que asegurarnos de que la potencia suministrada sea menor de 0,1W. Dado que el voltaje en abierto supera los 5V soportados por la placa de Arduino, debemos colocar una resistencia que limite la corriente. En nuestro caso, hemos usado una de 680Ω. Haciendo uso de la Ley de Ohm, $V = R \cdot I$, podemos calcular la intensidad que circula por el circuito. Con los valores de los voltajes ya conocidos y

los de la intensidad calculados, podemos realizar el cálculo de la potencia y asegurarnos que en ningún momento se superará el valor máximo de 0,1W.

La manera en que se obtuvieron los voltajes será explicada más adelante, pero para ejemplificar este desarrollo adelantaremos los resultados del voltaje máximo, que sería el peor de los casos.

Cuando la luz incide de forma directa sobre las placas solares, obteniendo así el voltaje máximo, al realizar las medianas de los datos se obtiene 0,6V. Con este valor y la resistencia de 680Ω, la intensidad máxima será:

$$I = \frac{0{,}6V}{680\Omega} = 8{,}82 \cdot 10^{-4}A$$

Una vez que tenemos la intensidad calculamos la potencia suministrada por las placas solares con la cantidad de luz máxima que usaremos en nuestros experimentos:

$$P = 0{,}6V \cdot \ 8{,}82 \cdot 10^{-4}A = 5{,}29 \cdot \ 10^{-4}W \leq 0{,}1W$$

3.1.2 Motor DC

A la hora de buscar un motor hemos cogido uno de corriente continua (DC) por su facilidad de uso. Necesitamos alimentarlo, lo cual se puede lograr montando un circuito adecuado o usando una fuente de alimentación externa, porque, como hemos explicado antes, la placa de Arduino no puede suministrar la potencia necesaria.

Lo ideal habría sido utilizar un motor sin reductoras, ya que estas afectan el comportamiento lineal al modificar las características de torque y velocidad; sin embargo, esto no fue posible debido a las elevadas velocidades a las que funcionan y a que queríamos asegurar un torque suficiente para hacer rotar el soporte de las placas solares. Finalmente, el motor elegido es el que se muestra en la Figura 13.

Figura 13. Motor DC - Polulu Gearmotor 37D [16] [17].

Según el *datasheet* [20] del fabricante, este motor tiene una reductora de 150:1, una velocidad de 67 RPM y es de 12V.

Como funciona a 12V, para alimentarlo se ha utilizado un *driver* de motores y un adaptador de corriente de 12V y 1A, conectado al conector *Jack* del Arduino UNO; esta alimentación se extrae por los puertos Vin y GND de la placa Arduino y se lleva al *driver* de motores mostrado en la Figura 14.

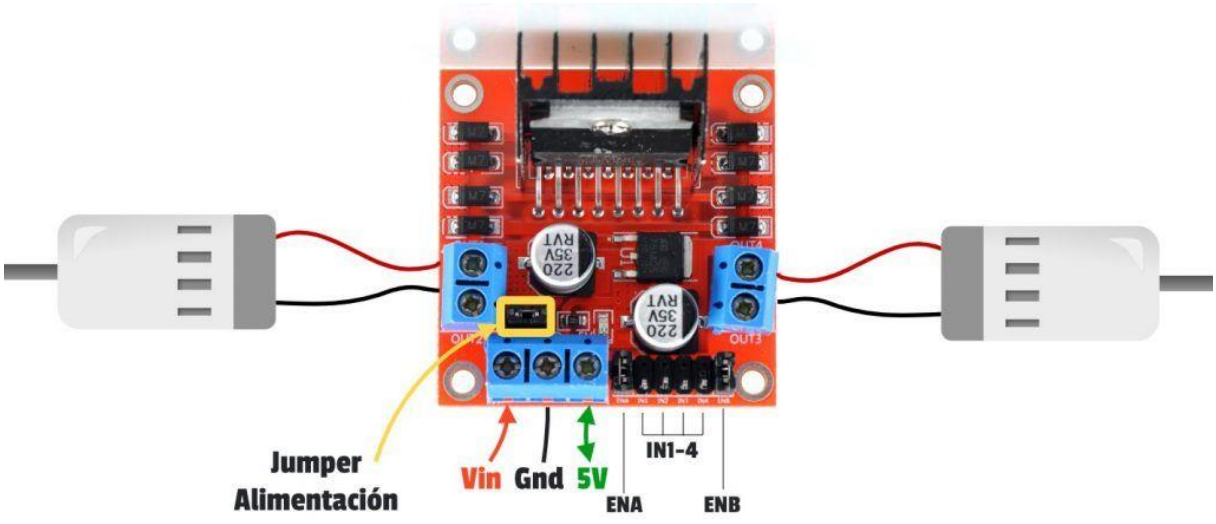


Figura 14. Driver de motores L298N [19].

Este *driver* de motores incorpora el puente H L298N [21]. El *driver* es un componente que permite controlar de manera precisa y segura la velocidad y dirección de motores eléctricos regulando la cantidad de corriente que se suministra al motor evitando así dañar el microcontrolador por las altas corrientes que los motores necesitan. Por otra parte, el puente H es un circuito que nos permite controlar el sentido de giro del motor sin necesidad de cambiar físicamente las conexiones del motor.

En el próximo subapartado de este capítulo se entrará en más en detalle acerca de cómo se ha configurado esta parte del proyecto.

3.1.3 Otros componentes

Nuestro objetivo es que el motor haga girar a las placas solares en busca de la fuente de luz. Para completar el montaje se han tenido que adquirir otros materiales, como poleas [22] [23] y una correa [24]. Además, otra pieza que fue necesaria fue un anillo de cables deslizante [25], el cual evita que los cables se enreden mientras las placas solares giran (ver Figura 15).

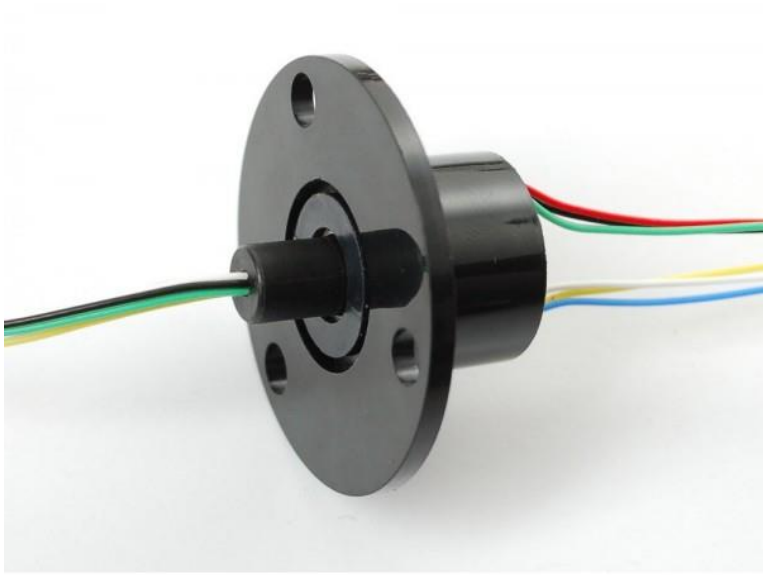


Figura 15. Anillo de cables deslizante.

Todo esto necesitaba, por último, una estructura física para ser montado. Esta se diseñó y construyó con una impresora 3D. Finalmente, se diseñaron también las poleas, ya que las que se compraron no eran del diámetro adecuado.

Para la estructura se hicieron dos diseños. En el primero, el soporte de las placas solares estaba en paralelo y el dentado de las poleas no era lo suficientemente profundo, por lo que la correa deslizaba; además, la parte en la que se colocaba el motor estaba abierta por la parte superior para poder introducirlo, provocando que no estuviera lo suficientemente sujeto, como resultado teníamos oscilaciones a la hora de poner en funcionamiento el motor. Esta estructura es la que se observa en la Figura 16.

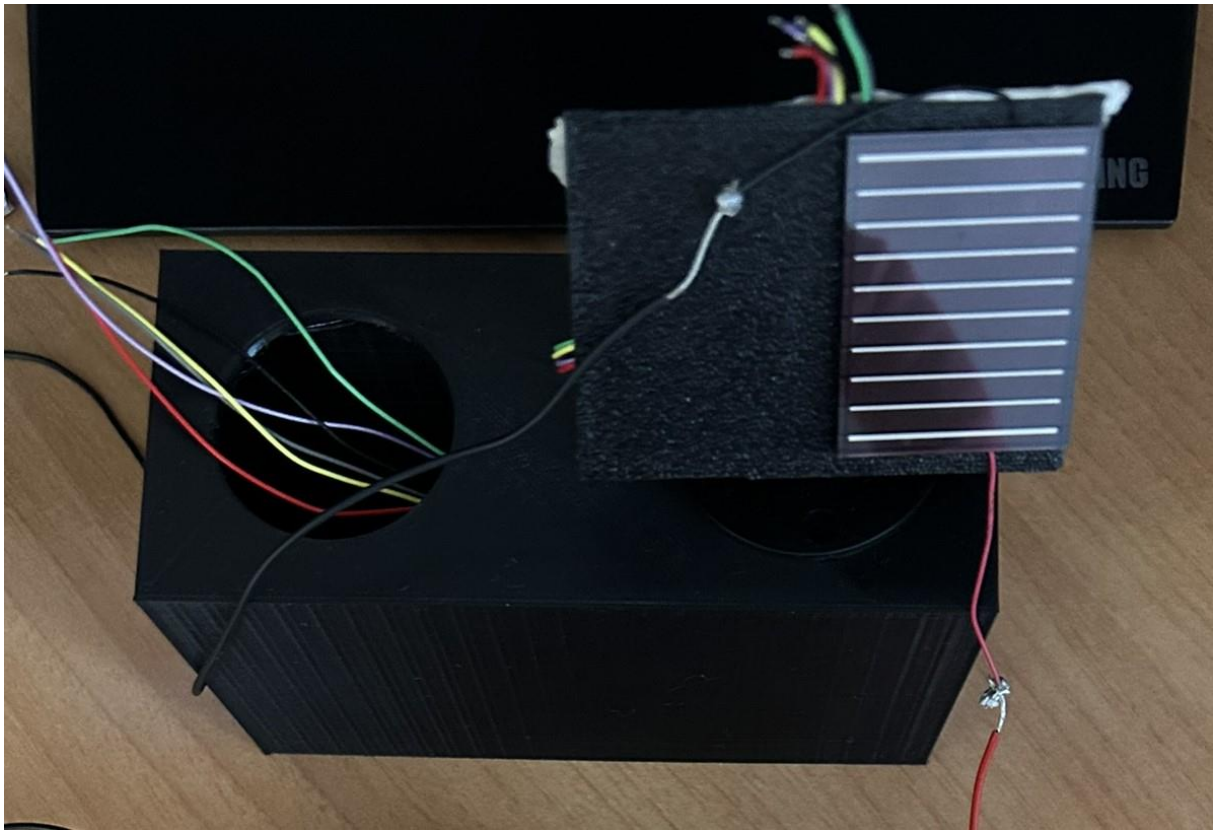


Figura 16. Diseño original de la estructura.

En el siguiente diseño, la estructura de las placas solares se colocó para que estuviera formando entre ellas un ángulo de 90º, se realizaron otros diseños de poleas y la parte del motor se introducía por uno de los costados y era atornillado por la parte superior y consiguiendo así que quedara completamente fijo. La estructura completa quedó como se puede apreciar en la Figura 17.

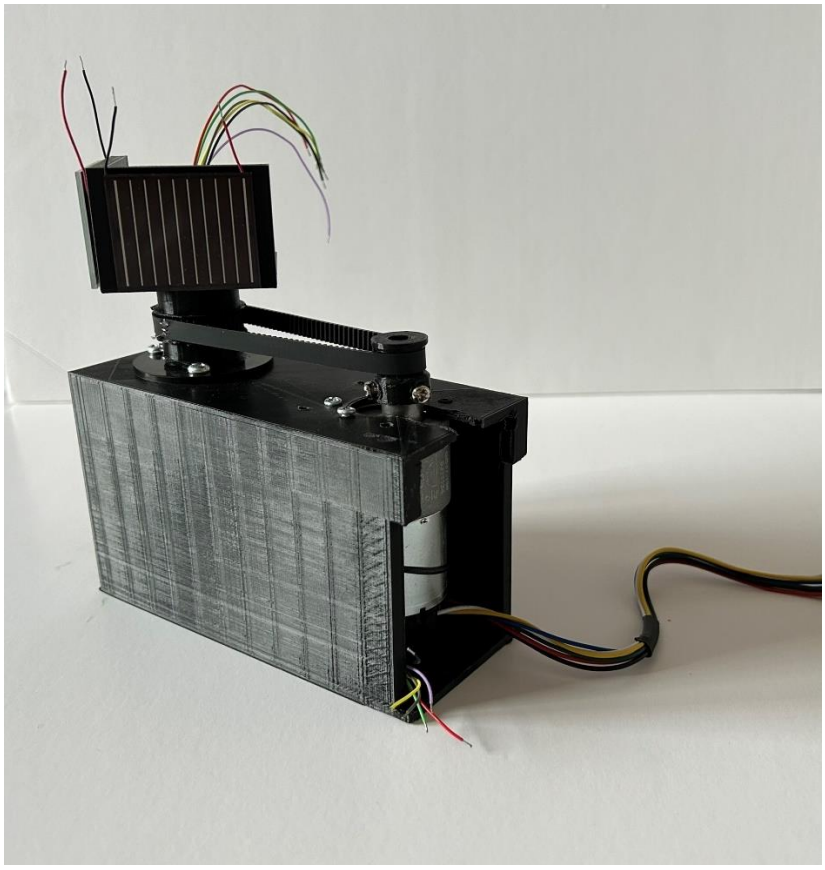


Figura 17. Estructura para las placas solares.

En el capítulo 7 de apéndices se incluirá información sobre las dimensiones de la caja y cómo deben colocarse los componentes.

3.2 Caracterización de los componentes

Para realizar la caracterización de las placas solares y del motor se han implementado varios programas en lenguaje C que permiten dar y leer voltajes de los mismos. Estos códigos difieren en qué periféricos se están configurando: ADC para las placas y PWM para el motor. Para ambos se configura el *Timer 1*; su rutina de interrupción de servicio (ISR) es el único cambio significativo entre experimentos, ya que lo que hacemos con ella es definir el comportamiento que buscamos con nuestro programa.

3.2.1 Placas Solares

Para realizar la caracterización de las placas solares se realizó un experimento en el que se situaba una luz en una posición fija y se iba girando la placa solar un ángulo conocido para medir el voltaje generado según la cantidad de luz recibida.

Para ello, sobre un cartón se trazó una circunferencia, indicando sobre ella ángulos de 15º en 15º, y se situó la estructura con la placa solar en el centro. Un flexo se colocó aproximadamente a 20 cm de distancia de la placa solar. Para poder seleccionar los ángulos con la mayor precisión posible, se utilizó una varilla de metal flexible, enroscada en la parte giratoria de la estructura, que servía de guía en el cartón para seleccionar los ángulos.

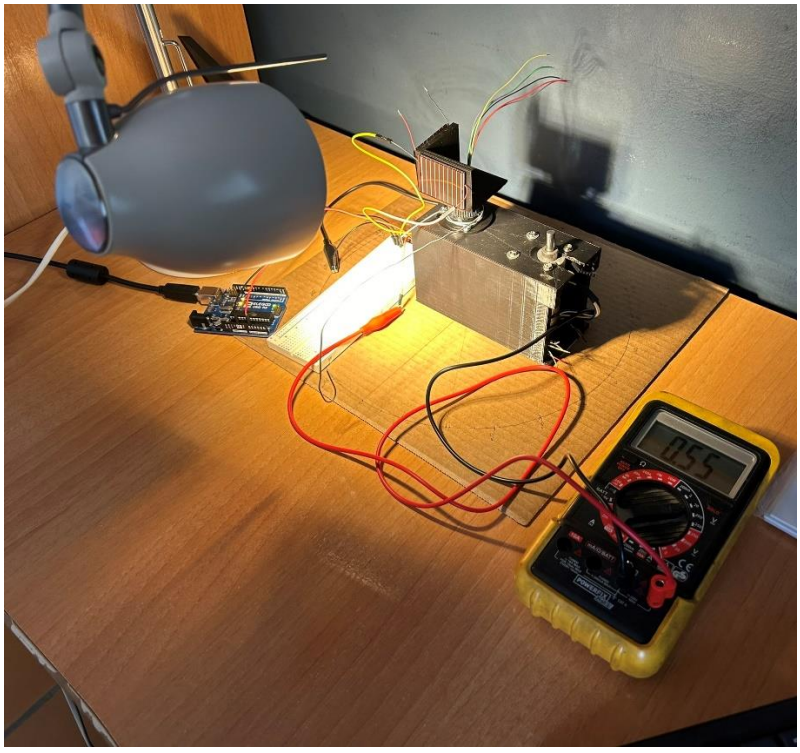


Figura 18. Experimento para caracterizar las placas solares.

Como vimos anteriormente en el apartado 3.1.1, el voltaje en abierto de esta placa era de 8,01V superando el voltaje que puede soportar el Arduino UNO. Esto se comprobó también haciendo uso de un multímetro según se aprecia en la Figura 19.

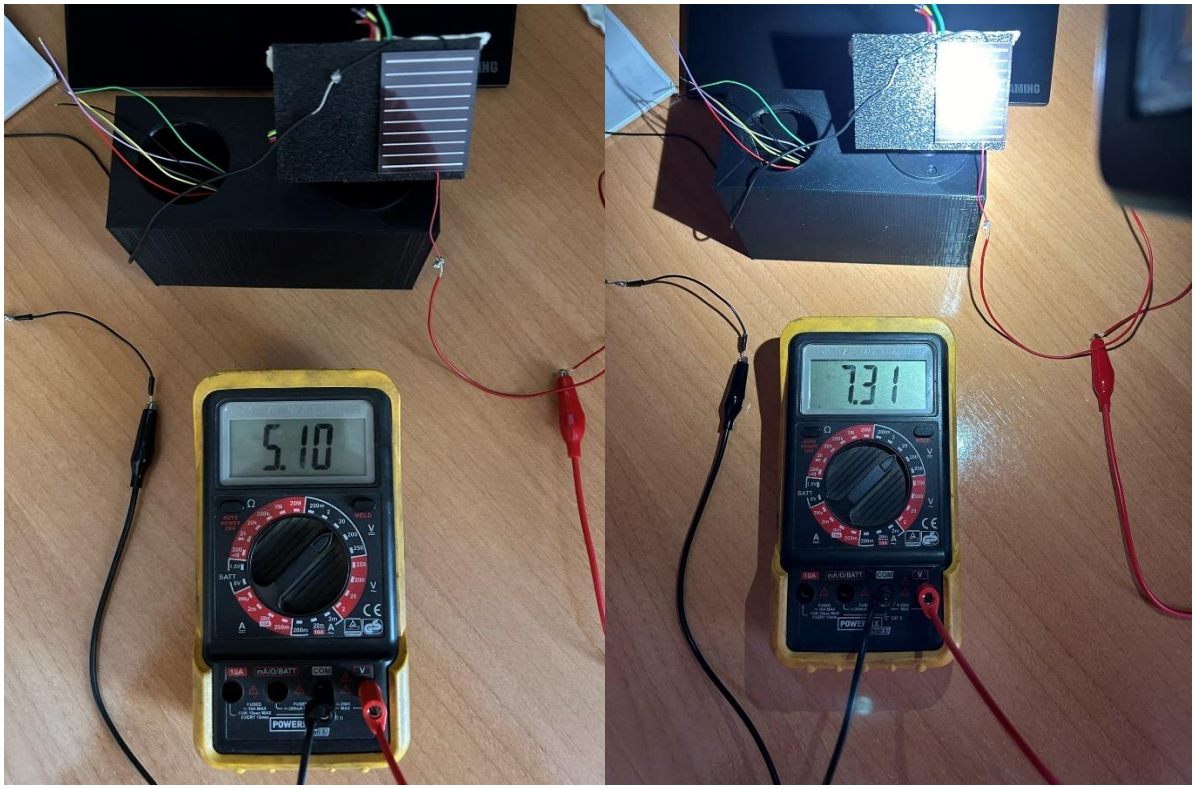


Figura 19. Voltaje en abierto sin y con luz.

Como se comentó previamente, se hizo uso de una resistencia de 680Ω para limitar la corriente. El circuito que hemos montado se muestra en la Figura 20, este sirve para medir el voltaje generado por la placa solar utilizando la entrada analógica del Arduino

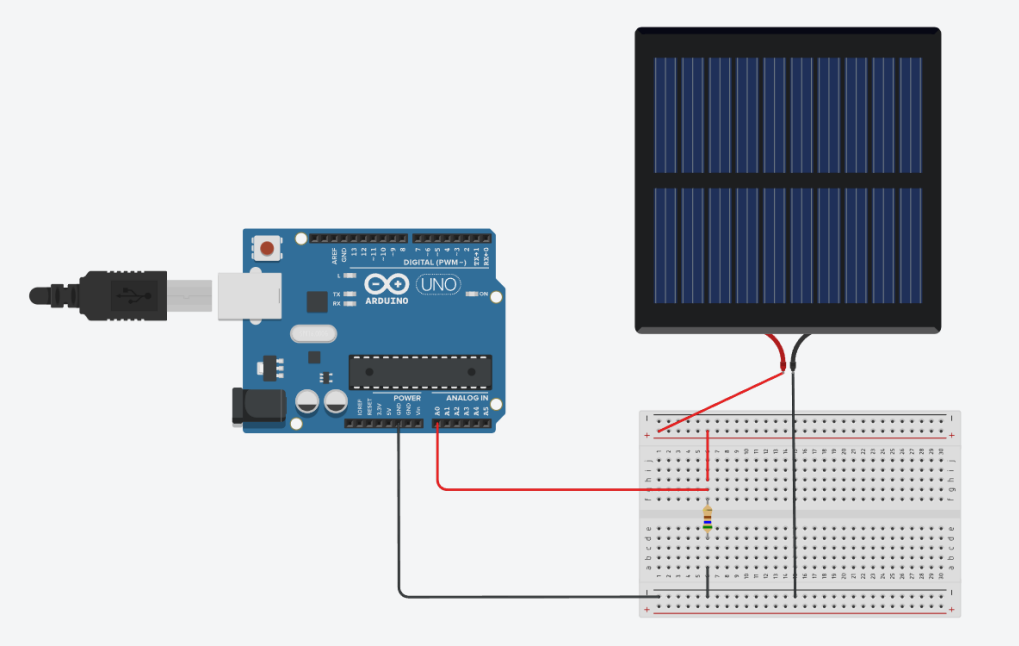


Figura 20. Circuito para las placas solares.

En la Figura 21 podemos ver cómo sería el modelado de una placa solar eléctricamente:

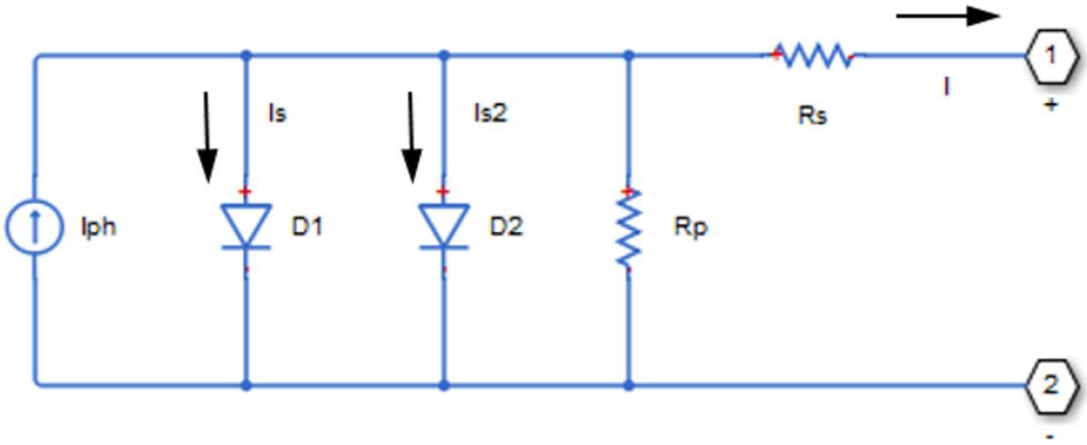


Figura 21. Modelado eléctrico de una placa solar [26].

En cuanto a la configuración del Arduino, podemos destacar que en el conversor ADC se está usando como referencia el voltaje interno de 1,1V. Esto lo hacemos ya que, en nuestro caso, nunca superamos 1,1V, lo que nos permite obtener una mayor resolución en las lecturas. El ADC lee con una resolución digital de 10 bits, es decir, distingue 1024 valores posibles (de 0 a 1023), ya que usamos tanto los bits altos como bajos del

registro ADC del microcontrolador. Si tenemos 1.1V como referencia y lo dividimos entre 1024 valores, cada paso del ADC corresponderá aproximadamente a 1,074 mV. En cambio, si usáramos el voltaje de referencia de 5V, cada paso del ADC correspondería aproximadamente a 4,88 mV.

También hemos usado un temporizador de 0,5 segundos del ATmega328P configurado en modo CTC. En el modo CTC se cuenta hasta el valor almacenado en el registro OCRnX y, cuando el contador llega a este valor, se reinicia y, en nuestro caso, se genera una interrupción. Para configurar un tiempo de muestreo de 0,5 segundos hemos realizado los siguientes cálculos:

$$OCRnX = \frac{Tiempo\ que\ queremos}{\frac{1}{f_{timer}}} < Tamaño\ timer$$

$$f_{timer} = \frac{f_{CPU}}{Preescalado}$$

La frecuencia de la CPU (f_{CPU}) es 16 MHz. Como estamos usando el Timer1, el tamaño del timer es de 2^{16} $(16bits)$. El preescalado es un valor que debemos consultar en la hoja de datos del microcontrolador y entre las opciones que tenemos disponibles debemos elegir aquella que satisface que OCRnX < Tamaño timer, en nuestro caso este valor es 256. En esta ocasión el código implementado en la rutina de interrupción de servicio (ISR) es el que podemos ver en la Figura 22.

```
78 ISR(TIMER1_COMPA_vect) {
79     if(listosEnviar == 0){
80         valorA0 = leerADC(0); //Leer el valor ADC para el PIN A0
81         valoresA0[indice] = valorA0; //Escribir valor en el array
82         indice++; //Incrementar indice
83 
84         if(indice>=250){
85             indice = 0;
86             listosEnviar = 1;
87 
88         }
89     }
90 }
91 
```

Figura 22. ISR para la caracterización de las placas solares.

Esta ISR se ejecutará cada 0,5 segundos. En ella lo que se hace es leer el valor del canal del ADC 0 al que está conectada la placa solar; este valor lo almacenamos en un *array*. Cuando este *array* se llena, lo mandamos por el puerto serie; mientras se están enviando datos por el puerto serie no se almacenan nuevos datos en el *array*.

Para enviar los datos por el puerto serie se está haciendo uso de una librería proporcionada por el tutor del TFG (Juan Antonio Fernández Madrigal [27]).

Para almacenar los datos en un fichero, se podría haber utilizado *PuTTY* para generar un archivo .log, o cualquier otro programa que permitiera almacenar los datos de una terminal en un fichero. En nuestro caso, hemos utilizado un *script* de Matlab.

El fichero obtenido contiene los voltajes medidos para cada ángulo concreto. En cada ángulo se tomaron 1000 medidas con un intervalo de 0,5 segundos entre cada una. Se consideraron 13 ángulos desde 0º hasta 180º, siendo 0º la posición donde la placa recibe la luz de forma directa y 180º donde la recibe por la parte trasera. El experimento no se realizó para ángulos de 180º a 360º, ya que los resultados deben ser simétricos a los obtenidos en los otros ángulos.

Una vez que en Matlab tenemos los vectores de los voltajes de cada ángulo, podemos empezar a trabajar con ellos para analizar el comportamiento de la placa solar.

Anteriormente a estas pruebas, nos encontramos un problema con la caracterización de las placas. Este fue el que podemos observar en la Figura 23:

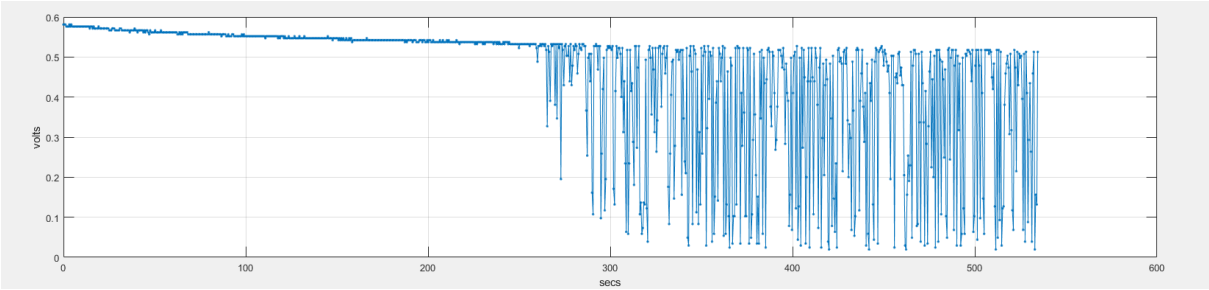


Figura 23. Pruebas iniciales de la caracterización de las placas solares

Al principio las placas solares mostraban un comportamiento completamente lineal. Sin embargo, a mitad del experimento, tras unas 250 medidas aproximadamente, empezaba a aparecer mucho ruido en la señal, con diferencias de 0,6V de pico a pico. Buscamos el error en el código, el montaje del circuito, problemas con el Arduino, etc. Tras realizar una prueba en el despacho de los tutores, donde todo salió bien, volvimos a probar en casa y los resultados volvieron a ser incorrectos. La única diferencia era la fuente de luz que estábamos utilizando. Por comodidad, facilidad de transporte y montaje, estábamos usando una linterna led, y esta era la causa del ruido en la señal. Finalmente, se decidió usar la luz de una lámpara de escritorio. También se probó con la linterna del móvil, que funcionó bien, pero el problema era que tenía que estar demasiado cerca de las placas solares, lo que hacía imposible realizar medidas en otros ángulos.

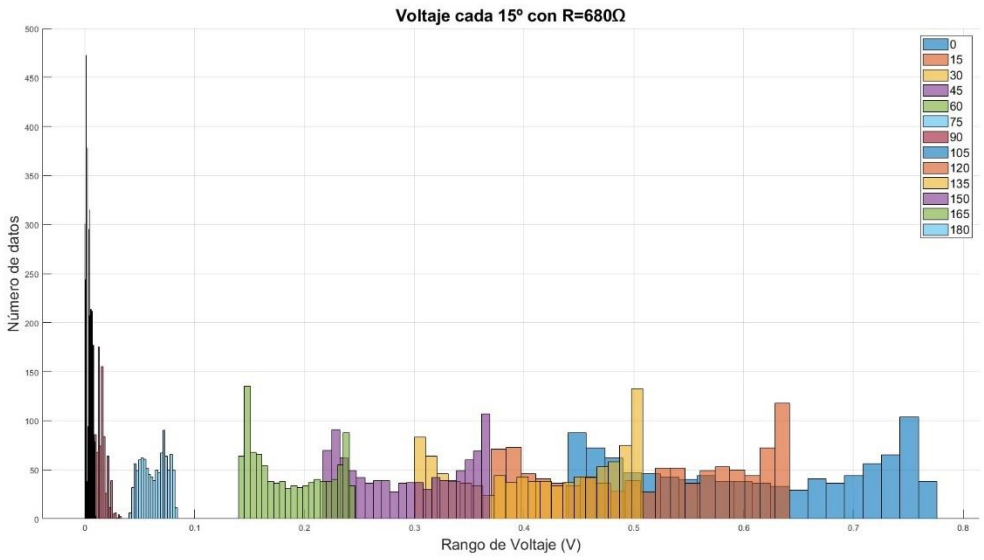


Figura 24. Histograma de voltajes recibido de la placa solar para distintos ángulos.

En el histograma de voltajes de la Figura 24 se representan los datos de los 13 ángulos. Se puede observar que el voltaje disminuye a medida que aumenta el ángulo, lo cual es correcto. Sin embargo, también se puede notar que nuestras placas solares no tienen una distribución uniforme en aquellos ángulos donde reciben mucha luz, lo que indica la presencia de ruido. Si ampliamos la parte izquierda del histograma (Figura 25), que

corresponde a los ángulos que reciben menos luz, veremos que tienen más varianza y los datos se asemejan más a una distribución uniforme.

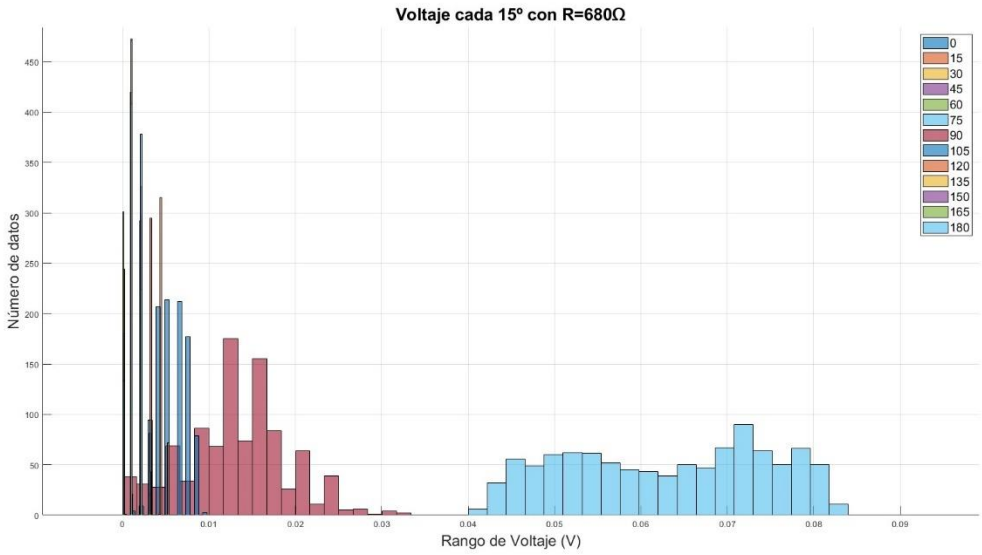


Figura 25. Ampliación de la Figura 24.

Utilizando la función boxplot de Matlab obtenemos una gráfica con las medianas y desviaciones típicas de los mismos datos (ver Figura 26).

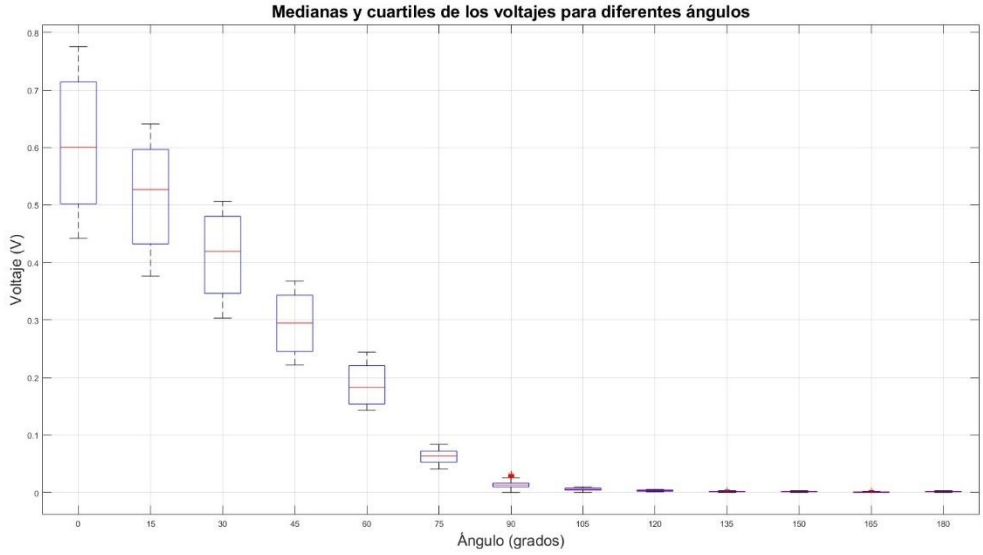


Figura 26. Boxplot de los datos de la Figura 24.

Al igual que con el histograma, pero de manera más clara, se puede ver cómo el voltaje disminuye conforme aumenta el ángulo. En el *boxplot* se observa además claramente que a partir de 90º las medidas tienden a ser casi siempre iguales, con un valor cercano a 0. Este dato será importante más adelante, ya que ignoraremos los ángulos superiores a 90º.

Otro dato importante que tiene impacto en el proyecto son las desviaciones típicas que vemos en la Figura 26. Más adelante justificaremos por qué nuestro rango de trabajo será en torno a los 0,3V; ahora mismo, esto significa que trabajaremos alrededor del ángulo de 45º. Como podemos apreciar, hay unas variaciones de voltaje considerables ahí; lo ideal habría sido trabajar en una situación parecida a la que tenemos en los ángulos de 75º y 90º, donde el voltaje siempre tiene un valor más constante. ¿Por qué no se ha hecho esto? También explicares esto más en detalle posteriormente, pero adelantaremos que se debe a que alrededor de 45º el comportamiento del sistema es más lineal.

En la Figura 27 se puede apreciar la variación del voltaje proporcionado por las placas solares cuando están recibiendo la luz solar de forma directa, en el ángulo de 0º.

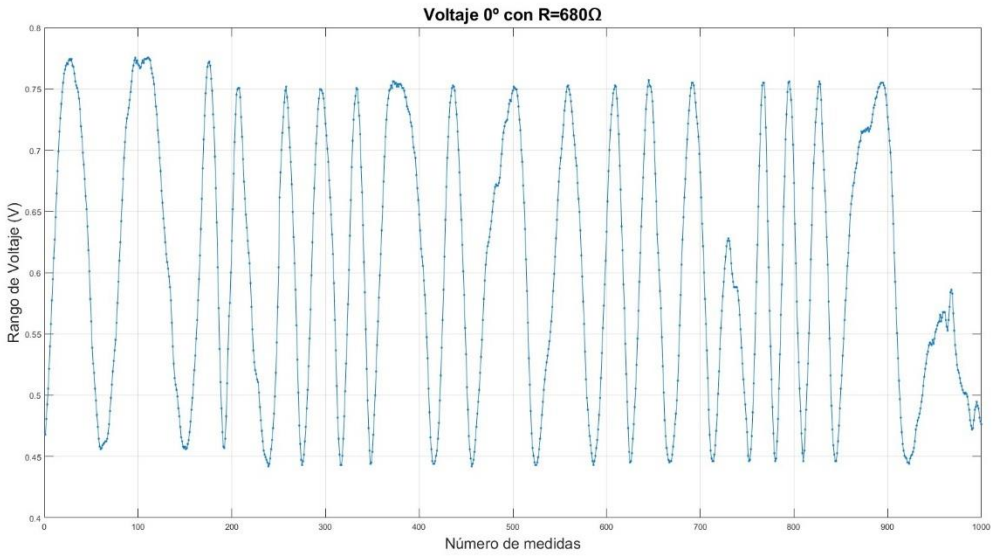


Figura 27. Voltaje en el ángulo 0º.

El motivo por el que se producen estas oscilaciones y su posible solución no se han contemplado en el estudio de este proyecto, pero ha sido uno de los principales problemas que hemos tenido; esto sería una de las áreas para tener en cuenta a mejorar en un futuro. Para continuar nuestro trabajo hemos considerado estas oscilaciones como ruido.

Continuando con el análisis de los datos del experimento, con las medianas que van desde el ángulo 0º hasta el ángulo 90º se puede obtener la ecuación de una recta de regresión donde el eje X represente el ángulo en radianes y el eje Y represente los voltajes para cada ángulo (ver Figura 28).

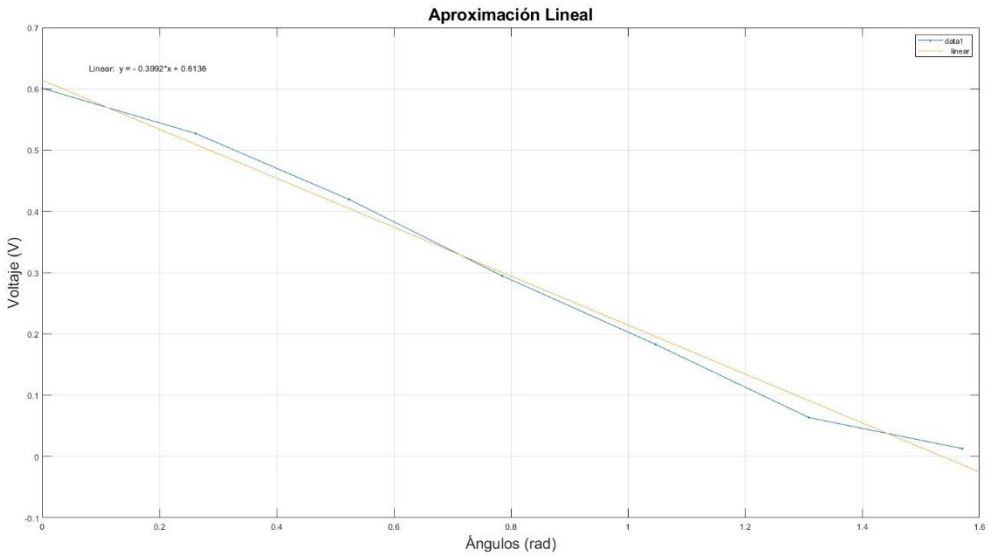


Figura 28. Ecuación de la recta de regresión de las placas solares.

Se obtiene así una ecuación de recta decreciente con un intercepto distinto de 0, demasiado grande. Para deducir esta ecuación de la recta, una vez se ha realizado el plot, una gráfica, en Matlab, se selecciona "Tools" -> "Basic Fitting" -> "Linear".

Una recta decreciente que no pasa por el punto (0,0) no nos sirve como modelo lineal del sistema de las placas, ya que no tiene función de transferencia. En este punto cabe señalar que para conseguir una función de transferencia válida necesitamos modificar

ligeramente este diagrama (no hay problema puesto que la modificación puede considerarse interior al bloque H) (Figura 29).

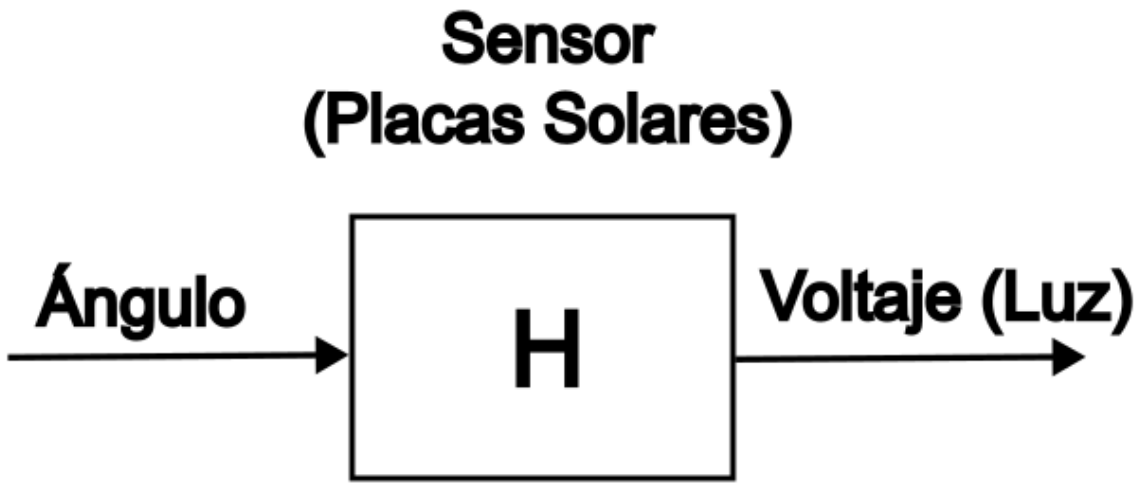


Figura 29. Diagrama de bloques original del componente de las placas solares.

El diagrama de bloques de las placas solares sería el de la Figura 29.

Lo que vamos a hacer es invertir el sentido de los ángulos para obtener una recta creciente y tener un intercepto próximo a cero.

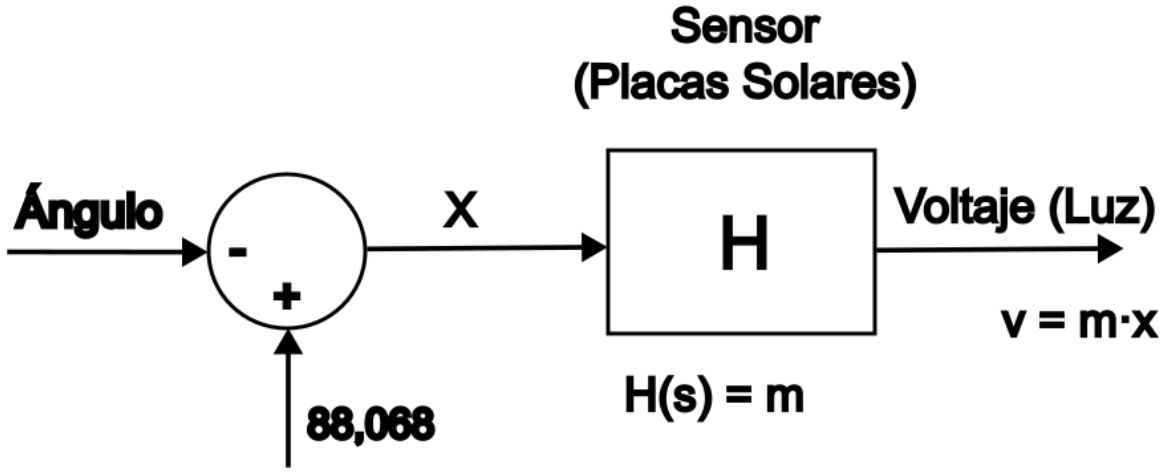


Figura 30. Diagrama de bloques invertido.

En la Figura 28 al hallar la ecuación de la recta de regresión obtenemos $y = -0{,}3992x + 0{,}6136$. Lo primero que debemos hacer con dicha ecuación es encontrar el punto de corte con el eje Y; esto nos dará como resultado $x = 1{,}5370 \cdot \left(\frac{180}{\pi}\right) = 88{,}068$ rad.

Invirtiendo el eje horizontal como ilustramos en la Figura 30 (equivalente a restar el ángulo donde tenemos voltaje nulo), podemos hacer una representación según lo indicado en la Figura 30.

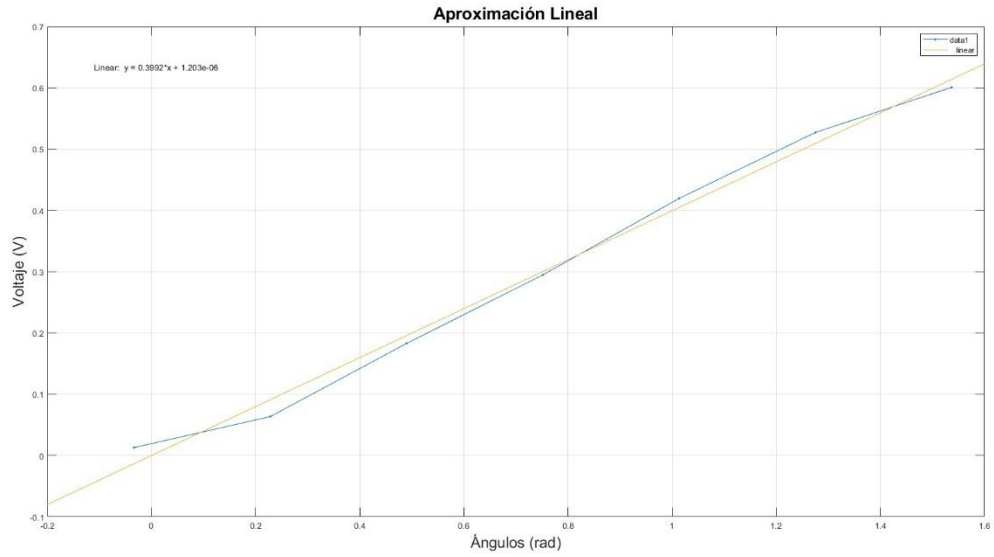


Figura 31. Ecuación de la recta creciente como modelo lineal de las placas solares.

Esta ecuación de la recta de regresión tiene la forma $y = 0{,}3992x + 1{,}203 \cdot 10^{-6}$. Seguimos teniendo intercepto, aunque ahora es despreciable. En cualquier caso, forzamos una regresión lineal [28] [29] para obtener una ecuación de la recta que tenga la forma $y = m \cdot x$. Tras realizar estos cálculos, nuestro resultado final es $y = 0{,}3992x$. El objetivo de eliminar el intercepto es porque, para obtener la función de transferencia, por definición, no podemos tener un término independiente.

En la Figura 32 podemos ver el resultado de hacer la regresión lineal sin intercepto. Como podemos observar, no se ve a simple vista ningún cambio al comparar las gráficas

de los datos originales con la regresión lineal; deberemos ampliar sobre el punto (0,0) para comprobar que esta pasa por el origen.

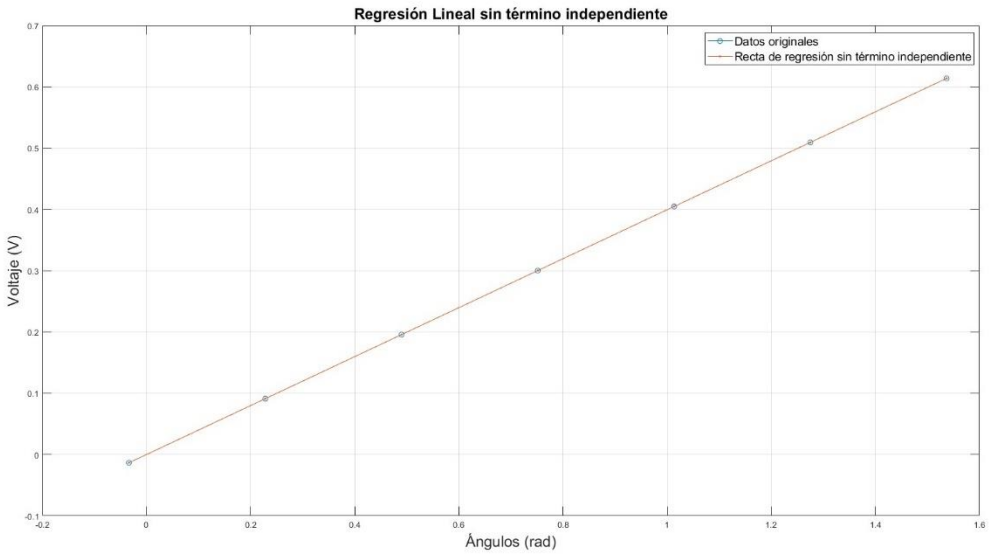


Figura 32. Datos originales vs regresión lineal forzada.

Si nos fijamos en las Figura 26 y Figura 31, vemos que nuestras ecuaciones de la recta vienen de una aproximación lineal; también vemos como esta aproximación tiende a $\pm\infty$, lo cual no es cierto: por el lado izquierdo de la gráfica tendremos en realidad valores de 0V, ya que serán los puntos donde menos luz está incidiendo, y por el lado derecho, una vez se alcance el punto máximo, tendremos una pendiente decreciente, ya que entramos en ángulos donde la luz deja de incidir de forma directa sobre las placas solares. Es decir, este componente tiene importantes no linealidades.

Para evitarlas, lo que vamos a hacer es trabajar en el punto medio de la recta, donde la linealidad es mayor, evitando llegar a los extremos, ya que, si se produjeran oscilaciones o errores, el controlador podría entrar en puntos de no linealidad y tener un comportamiento no modelado.

Con la ecuación de la recta $y = 0{,}3992x$ lo que debemos hacer es obtener la función de transferencia de las placas; para ello haremos uso de la transformada de Laplace:

$$y(t) = 0.3992x(t) \rightarrow [\mathcal{L}] \rightarrow Y(s) = 0{,}3992X(s)$$

$$TF(s) = \frac{Y(s)}{X(s)} = 0.3992 = H(s)$$

3.2.2 Motor DC

En la Figura 33 podemos ver cuáles son las conexiones que tiene el motor. Este dispone de dos *encoders*; nosotros solo hemos hecho uso de uno.

Lead Color	Function
Red	Motor power
Black	Motor power
Green	Encoder ground
Blue	Encoder Vcc (3.5 V to 20 V)
Yellow	Encoder A output
White	Encoder B output

Figura 33. Conexiones del motor.

El conexionado con el *driver* de motores L2398N y el Arduino UNO se puede ver en la Figura 34. Aquí usamos los 5V y GND para alimentar el *encoder* del motor. Además, como en el *Jack* hay un cable de alimentación de 12V, los pines Vin y GND se conectan a la entrada del *driver*. Los pines de salida del *driver* son los que se conectan al motor para alimentarlo con los 12V. El cable amarillo, salida del *encoder* A, estará conectado al pin digital 2 del Arduino; este pin es uno de los que se puede configurar para tener interrupciones externas. Los otros tres cables son para entradas del motor: el rojo (IN1) y marrón (IN2) para configurar el sentido de giro del motor (ver Tabla 1), y el verde (ENA) para la PWM.

IN1	IN2	Sentido de Giro
0	0	Parado
0	1	Sentido antihorario
1	0	Sentido horario
1	1	Cortocircuito

Tabla 1. Sentidos de giro del motor.

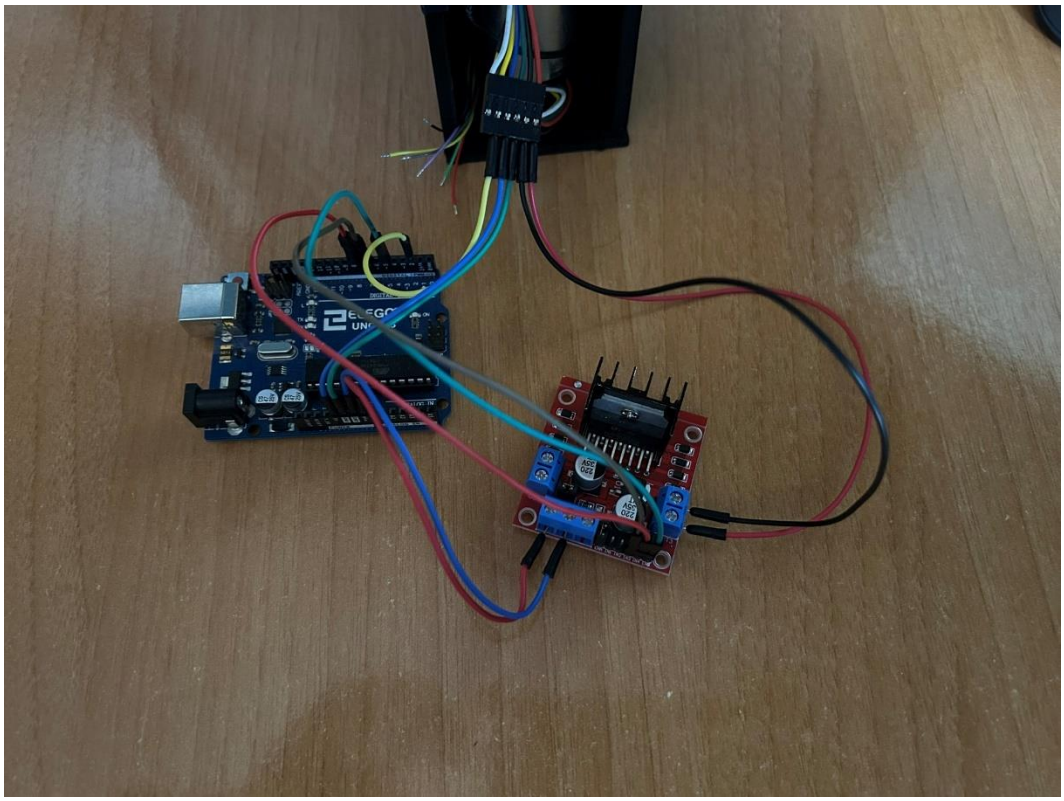


Figura 34. Conexiones del motor con Arduino y driver del motor.

Para realizar la caracterización del motor se llevaron a cabo diferentes experimentos de los cuales se obtuvieron varias conclusiones.

Lo primero que tuvimos que hacer fue generar una salida digital, mediante la configuración de una PWM (modulación por ancho de pulsos), que sirviera para producir movimiento en el motor a través del *driver* (puente H). Partíamos de una señal de 100 KHz, suficiente para producir un movimiento de suficiente resolución.

Sabiendo esto debíamos buscar un *timer* del ATmega328P y un modo que fuera capaz de generar señales con esta frecuencia.

Dado el periodo de la señal, decidimos utilizar el *Timer* 0, de 8 bits, y configurarlo para que su valor máximo (*TOP*) se estableciera en el registro *OCR0A*. De esta manera, el temporizador contaría hasta el valor especificado en *OCR0A* y luego se reiniciaría a 0, en lugar de contar hasta 255 (2^8).

Los cálculos que se realizaron para configurar la PWM son los siguientes:

$$N = preescalado$$

$$f_{base} = 100KHz \rightarrow T = \frac{1}{f_{base}} = 10^{-5}s$$

1. $Buscamos\ el\ preescalado$: $\frac{T \cdot f_{CPU}}{Tam\_timer} = \frac{10^{-5} \cdot 16 \cdot 10^6}{256} = 0.627 \rightarrow N = 1$
2. $Frecuencia\ del\ timer$: $f_{timer} = \frac{f_{CPU}}{N} = 16\ MHz$
3. *Frecuencia PWM:* $f_{PWM} = \frac{f_{CPU}}{N \cdot Tam\_Timer} = \frac{16 \cdot 10^6}{256} = 62500Hz < 100000Hz\ (f_{base})$

   El valor de frecuencia era aceptable. Continuamos con la configuración del modo de cuenta.

Inicialmente, configuramos el *timer* en modo *Fast PWM*, donde éste cuenta hasta el valor especificado en el registro *OCR0A* y luego regresa a 0 para empezar de nuevo. Estos cálculos quedaban de la siguiente manera:

4. Valor de TOP: $f_{PWM} = 100000 = \frac{16 \cdot 10^6}{x} \rightarrow x = 160 = TOP = OCR0A$

Para poder generar señales con un periodo de 100 KHz debíamos establecer el valor de *TOP* en 160, por lo que el *timer* contaría de 0 a 160 y se reiniciaría.

Para establecer los *duty cycle* (ancho positivo de la señal) debemos establecer valores en el registro *OCR0B*. De esta manera lo que estamos haciendo es que en OCR0A definimos el periodo de la señal y en OCR0B modificamos los valores para establecer cuánto tiempo queremos que esta señal esté establecida en alto.

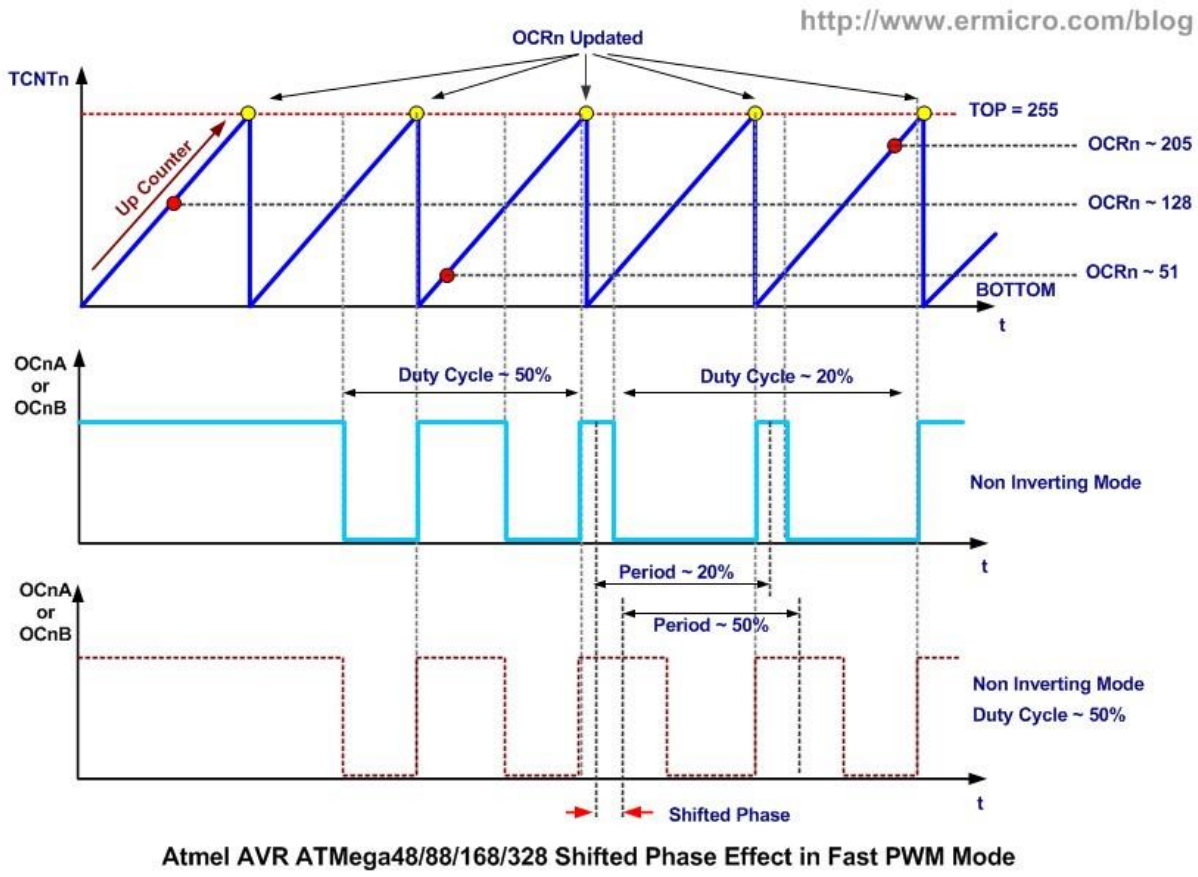


Figura 35. Diagrama de tiempo para el modo Fast PWM [28].

Como estamos trabajando con un motor, finalmente se terminó optando por configurar el modo *Phase Correct PWM*. Esto se debe a que este modo proporciona una señal más simétrica, contando de 0 a TOP y de TOP a 0; dicha simetría puede ayudar a que tengamos menos ruido haciendo que este modo sea preferible para aplicaciones de control de motores. Al usar este modo, debemos tener en cuenta que el valor de *TOP* debe ser la mitad del calculado para *Fast PWM* para mantener la frecuencia deseada. Por lo tanto, para generar señales con un periodo de 100 KHz, el valor de *TOP* debe ser 80.

Con esto que acabamos de ver podemos generar señales PWM con diferentes *duty cycle* que especifiquen la velocidad a la que va a girar el motor.

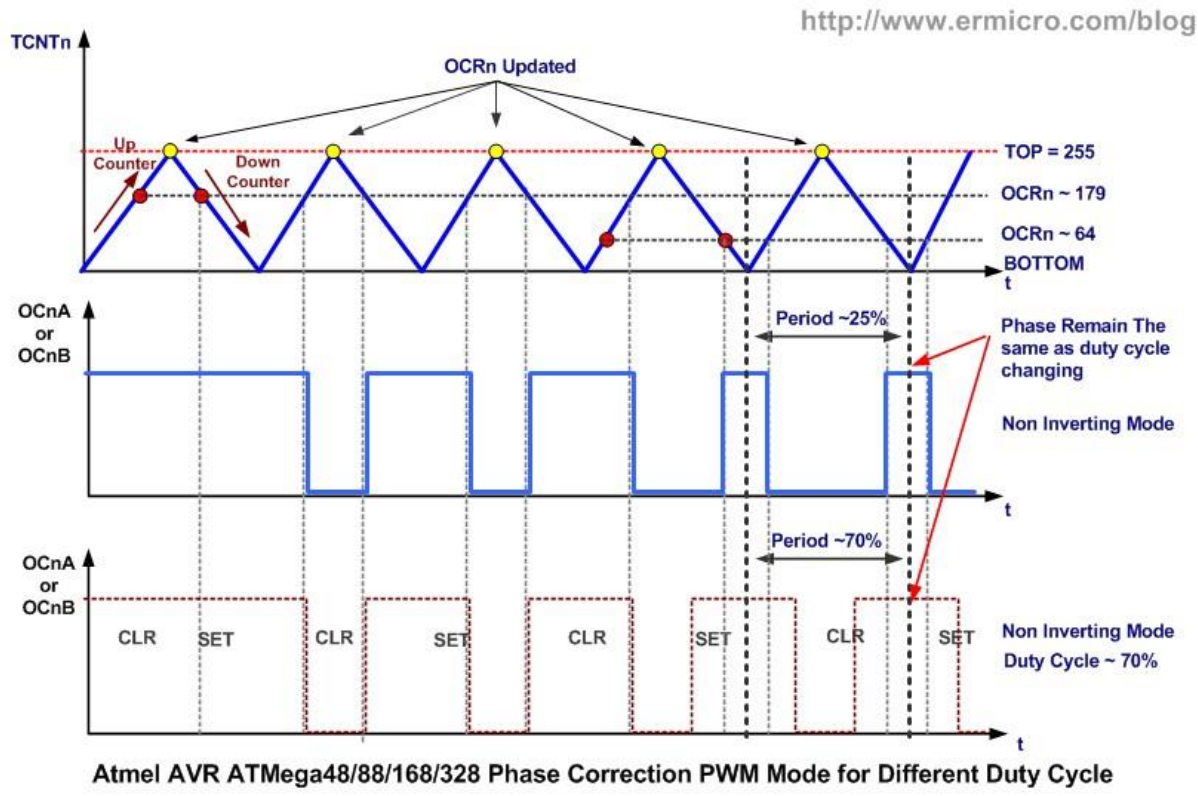


Figura 36. Diagrama de tiempo para el modo Phase Correct PWM [28].

Como el motor tenía un *encoder,* también se hizo uso de él para contar las RPS (revoluciones por segundo) del motor que proporcionaba al aplicarle diferentes *duty cycle*. Un *encoder* es un dispositivo que se utiliza para convertir movimientos mecánicos en señales eléctricas. En nuestro caso, al producirse los movimientos rotativos del motor, el *encoder* genera pulsos eléctricos que representan la velocidad. Esto es necesario para obtener una gráfica de las velocidades reales alcanzadas por el motor para una velocidad determinada dada por el *duty* de la PWM. En este caso la ISR del código del programa es la que vemos en la Figura 37:

```
90  ISR(TIMER1_COMPA_vect){
91      if(listoEnviar == 0){
92          pulsos_seg[indice_pulsos] = cnt_pulsos;
93          indice_pulsos++;
94          //Inicialmente esta 1seg con el motor parado
95          if (estado < 5*2 - 1){      //El timer es de 0,1s -> 5*2 = 10 -> 10*0.1 = 1
96              finPWM();
97              estado++;
98          } else{                     //Activamos la PWM para que el motor se mueva
99              if (estado == 5*2 - 1)
100                 TCCR0B |= (1<<CS00);
101             if(indice_pulsos >= 100){
102                 indice_pulsos = 0;
103                 listoEnviar = 1;
104             }
105         }
106         cnt_pulsos=0;
107     }
108
109 }
```

Figura 37. ISR para la caracterización del motor.

Previamente se ha configurado un ISR de interrupciones externas por flanco de subida, que permiten leer los flancos de subida del *encoder*. Aquí lo que se hace es incrementar una variable, cnt_pulsos, para contar el número de pulsos que lee el *encoder*. El *timer* es de 0.1 segundos en la ISR del *timer* (Figura 37); esta almacena la cantidad de pulsos que se han leído durante ese tiempo en un *array*. Para los experimentos, inicialmente dejamos el motor parado, en este caso durante 1 segundo; pasado este segundo activamos la PWM para que el motor empiece a moverse según el valor que hayamos asignado a *OCR0B*. De esta manera veremos el estado transitorio y estacionario del motor. Lo que vamos a tener en cada posición del *array* son los pulsos leídos cada 0.1 segundos. Cuando este *array* está lleno, se envía por el puerto serie para trabajar con los datos en Matlab.

Una de las pruebas que hemos hecho pretende comprobar cómo se comportaba el motor al recibir diferentes anchos positivos de la PWM y ver cuál es el mínimo necesario para iniciar el movimiento. Realizando estas pruebas se encontraron tres no linealidades. La primera es que el motor tiene una zona muerta, es decir, existe un voltaje no nulo para

el que no se mueve; el motor empieza a moverse cuando el ancho positivo de la PWM es $\geq 5\%$. Esta no linealidad no se trata; es ignorada en el modelado de Matlab.

La otra no linealidad que hallamos es que llegados a un cierto punto se produce una saturación. La saturación es una no linealidad que ocurre cuando la salida del sistema alcanza un límite máximo debido a restricciones físicas. Cuando se alcanza este límite máximo pueden aparecer problemas de oscilaciones, precisión o incluso daños. Por ello habrá que diseñar el controlador para que evite producir voltajes en esta zona de saturación. Esta no linealidad se puede ver en las curvas superiores de la Figura 38 donde independientemente del voltaje de entrada que apliquemos obtenemos la misma salida.

Para ver la tercera no linealidad primero vamos a comentar como realizamos los experimentos. Para ver el comportamiento del motor estos se realizaron tanto sin carga como con carga. Se montaron las poleas y correa y se hizo girar la estructura con las placas solares.

Analizaremos primero la gráfica de la Figura 38 en la que se aprecia la velocidad angular sin carga.

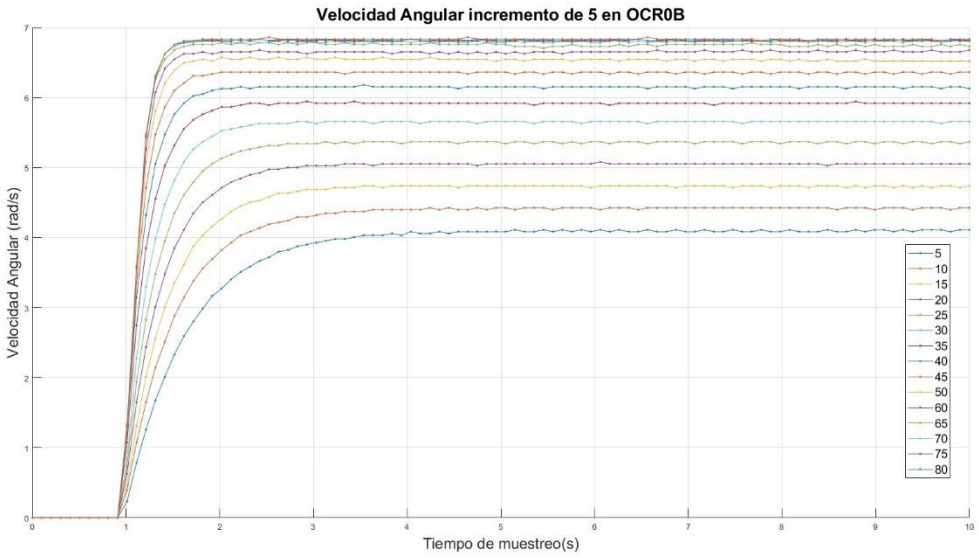


Figura 38. Respuesta del motor sin carga ante distintas velocidades.

Como se ha visto en la ISR, inicialmente se dejaba el motor parado para luego empezar a moverlo y así poder ver bien cuál es su estado transitorio y en qué momento llega al estacionario para cada una de las PWM. Para obtener estos resultados, previamente realizamos varias pruebas para determinar el tiempo adecuado del *timer* que nos permitiera capturar un número suficiente de pulsos durante el transitorio. Si el tiempo del *timer* era demasiado corto, existía el riesgo de que se omitieran algunos pulsos del *encoder*. En cambio, si era demasiado largo, los pulsos estarían demasiado separados entre sí, lo que dificultaría la observación clara del transitorio.

Lo que estamos contando con el Arduino son los pulsos del *encoder* cada décima de segundo. Esto habrá que convertirlo en velocidad angular; para hacer esta conversión necesitaremos usar los datos de la hoja de fabricante del motor. Los cálculos serían los siguientes:

1. Pulsos por segundo: $pps = 10 \cdot \frac{pulsos}{0{,}1\ seg}$

   Convertimos nuestros pulsos cada décima de segundo en pulsos por segundo.

2. Revoluciones por segundo: $rps = \frac{pps}{CPR \cdot Reductora}$

   Donde CPR (*counts per revolution*) es el número de pulsos por revolución. En nuestro caso, como solo estamos usando un canal y solo el flanco de subida, son 16. El motor tiene una reductora de 150:1. Estos datos los proporciona el fabricante del motor.

3. Velocidad angular (rad/s): $\omega = rps \cdot 2\pi$

En la Figura 38 se observa esta tercera no linealidad: la dependencia de los polos del sistema con respecto a la entrada. En un sistema idealmente lineal, los polos no deberían depender de la magnitud de la entrada. Sin embargo, durante los estados transitorios se observa que la respuesta del sistema varía con diferentes niveles de voltaje. Debido a esto lo que hemos tenido que hacer es seleccionar una de las respuestas

como modelo y suponer que es válida para todos los voltajes de entrada. Esta solución complica el diseño del controlador, ya que va a estar diseñado para un rango especifico de voltajes y puede que no funcione adecuadamente para otros rangos.

La gráfica de la Figura 39 muestra los resultados de un experimento similar realizado con carga. Para que los resultados con carga fueran más o menos precisos, se tuvieron que hacer algunos ajustes en el diseño de la estructura, ya que al añadir las correas se producían oscilaciones que eran visibles en las gráficas: el anillo de cables deslizante y la pieza de las placas solares están unidos por un agujero, y este es un poco más grande de lo que debería ser. Se intentó solucionar esto haciendo otra pieza con un diámetro más pequeño, pero la diferencia era tan mínima que o no encajaba o terminábamos con el mismo problema que teníamos inicialmente. Finalmente, se colocó un trozo de plástico, sellado con calor, para rellenar esa pequeña diferencia, logrando que la pieza no oscilara.

Otro motivo por el que se nos estaban produciendo oscilaciones era por la longitud de la correa. Se requiere que esta esté lo más tensa posible, pero esta tensión de la correa nos estaba produciendo oscilaciones. La correa usada inicialmente tenía una longitud de 22,3 cm; se probó también con una de 22,4 cm y finalmente se ha usado una correa que mide 22,5 cm con la que hemos obtenido unos buenos resultados.

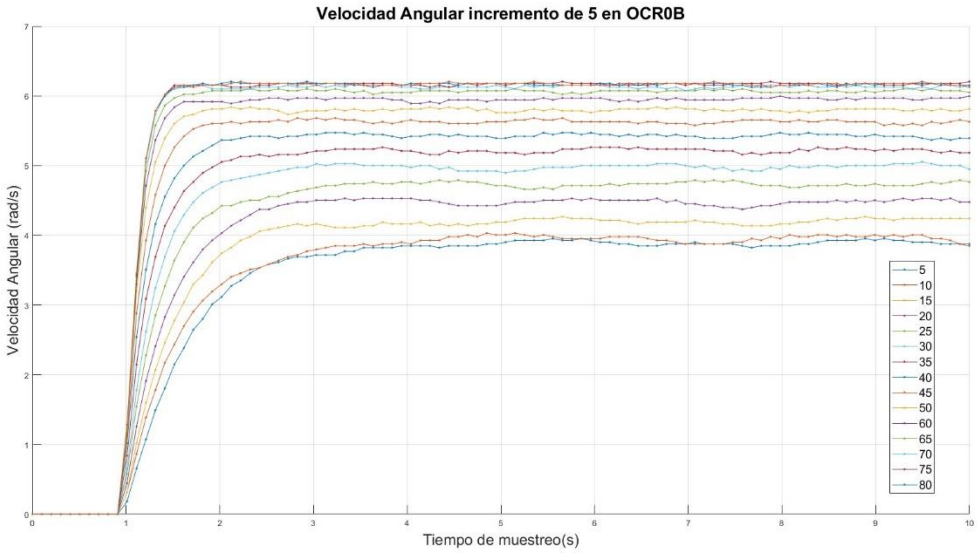


Figura 39. Respuesta del motor con carga.

Estas curvas representan la respuesta del motor ante diferentes entradas, como se ha mencionado; nos quedaremos con una de ellas debido a la no linealidad ya explicada y de ahí se sacará un modelo.

Tanto en la Figura 38 como en la Figura 39, podemos observar una respuesta muy parecida a la de un sistema de segundo orden sobreamortiguado. Esta clasificación implica que el modelado del sistema está formado por una ecuación diferencial de segundo orden. Hemos decidido usar una respuesta de 2º orden en lugar de una de primer orden, que también podría ser candidata, porque hemos observado que, al aplicar una entrada, la velocidad al motor, la derivada de la salida respecto al tiempo inicialmente se mantiene en cero $(\frac{dy}{dt} = 0)$ durante un breve período antes de que la respuesta comience a cambiar, una característica típica de los sistemas de segundo orden.

El comportamiento sobreamortiguado se puede deducir del análisis de la respuesta transitoria y estacionaria del sistema. En un sistema sobreamortiguado, la respuesta se caracteriza por la ausencia de oscilaciones. Esto significa que, ante una perturbación,

la salida del sistema regresa al valor de equilibrio sin sobrepasarlo, y lo hace de manera gradual.

La forma general de una ecuación diferencial de segundo orden sería la siguiente:

$$k \cdot f(t) = a\frac{d^2y(t)}{dt^2} + b\frac{dy(t)}{dt} + cy(t)$$

donde:

$k \cdot f(t)$ sería la entrada

$y(t)$ es la variable del sistema (posición)

$\frac{dy(t)}{dt}$ es la primera derivada (velocidad)

$\frac{d^2y(t)}{dt^2}$ es la segunda derivada (aceleración)

$a, b\ y\ c$ son constantes que dependen del sistema

Si obtenemos la función de transferencia de esta ecuación diferencial en su formato polinómico, tendríamos lo siguiente:

$$TF(s) = \frac{Y(s)}{X(s)} = \frac{k}{as^2 + bs + c}\ donde\ k\ es\ una\ ganancia$$

En el formato estándar tendría la siguiente forma:

$$TF(s) = \frac{k}{s^2 + 2\omega_n \varsigma s + \omega_n^2}$$

$donde$:

$\omega_n\ representa\ la\ frecuencia\ natural\ \left(\frac{rad}{s}\right)$

$\varsigma\ representa\ el\ factor\ de\ amortiguamiento$

Para identificar el sistema se puede usar la representación de la curva. Teóricamente se deben buscar algunos valores clave, como el tiempo de establecimiento y la sobreoscilación, para calcular cuánto valen ω_n y ς.

En nuestro caso, por simplicidad, hemos usado Matlab con la herramienta *System Identification Toolbox*. El procedimiento es el siguiente:

Paso 1. Creación del objeto de datos.

Con `iddata` se crea un objeto que contiene los datos necesarios para la identificación. Esta función necesita:

- Un vector con la salida del sistema (las velocidades angulares en rad/s).
- Un vector con la entrada del sistema. Representará el voltaje aplicado al motor durante el muestreo. Esta señal es un escalón cuya amplitud varía según la curva elegida. La amplitud del escalón se calcula multiplicando el voltaje de entrada del motor (12V) por el porcentaje del ancho positivo de la señal de entrada.
- El tiempo de muestreo del experimento.

Paso 2. Identificación del modelo.

El objeto `iddata` generado se pasa a la función `tfest`, especificando el número de polos y ceros del sistema que queremos identificar. Dado que estamos trabajando con un sistema de segundo orden sobreamortiguado, especificamos 2 polos y 0 ceros.

Paso 3. Verificación del modelo.

La función `tfest` devuelve un modelo del sistema identificado. Posteriormente, verificamos si el modelo es adecuado comparando su respuesta con los datos experimentales. Para hacer esta comprobación le aplicamos la misma entrada escalón a la función de transferencia y comparamos la respuesta con los resultados de los datos experimentales.

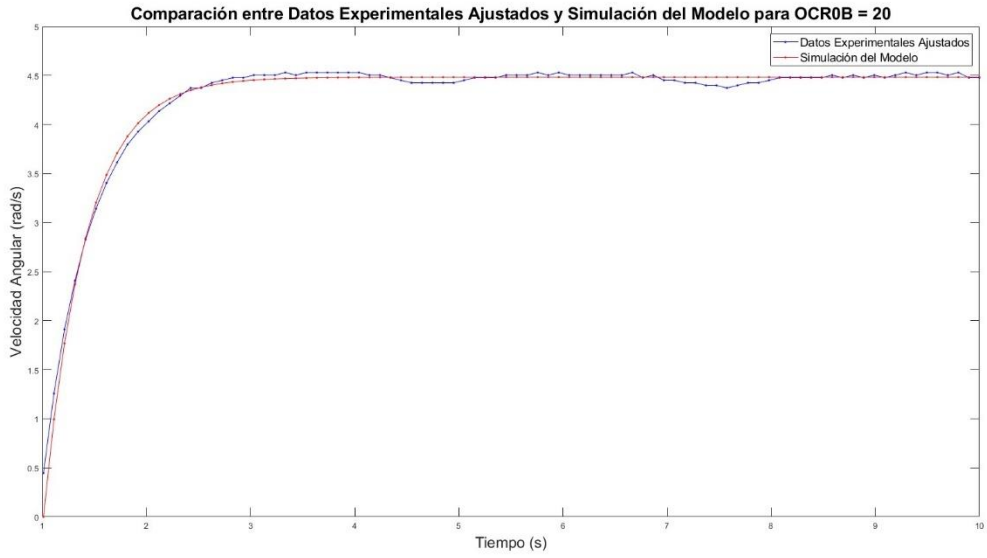


Figura 40. Verificación del modelo del motor.

En la Figura 40 mostramos gráficamente la verificación del modelo para el cual la señal tiene una PWM con un ancho positivo del 20%. Se seleccionó esta señal para modelar el motor y obtener una buena velocidad, evitando acercarnos demasiado a valores superiores o inferiores debido a la no linealidad de saturación mencionada anteriormente.

La función de transferencia de la planta que hemos obtenido para esta señal usando `tfest` es la siguiente:

$$TF(s) = G(s) = \frac{5{,}395 \cdot 10^6}{s^2 + 1{,}454 \cdot 10^6 s + 3{,}611 \cdot 10^6}$$

Capítulo 4

Diseño de controladores

En el capítulo anterior hemos deducido las funciones de transferencia de nuestro sensor y del motor, que son:

$$H(s) = 0{,}3992$$

$$G(s) = \frac{5{,}395 \cdot 10^6}{s^2 + 1{,}454 \cdot 10^6 s + 3{,}611 \cdot 10^6}$$

Con estas funciones de transferencia podremos montar nuestro sistema de bucle de control cerrado y haciendo uso de la rltool realizar el diseño y análisis de nuestros controladores.

La salida de la función de transferencia del motor es la velocidad. Para obtener la posición debemos añadir un integrador $(\frac{1}{s})$. Teniendo esto en cuenta, el diagrama de nuestro bucle de control es el que se puede apreciar en la Figura 41.

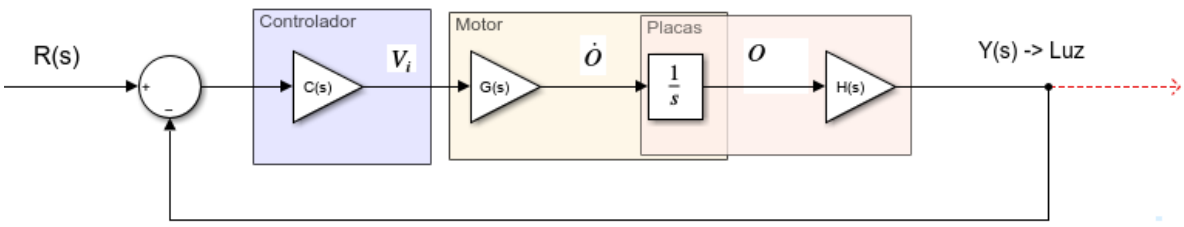


Figura 41. Diagrama de bucle cerrado del sistema controlado.

En la *rltool* se pueden configurar diferentes tipos de diagramas de bloques, pero el que nosotros vamos a usar es el de la Figura 42. Para utilizar la *rltool*, debemos pasarle como parámetro la función de transferencia de la planta, lo que creará el modelo con un sensor unitario. Por eso, la función de transferencia que nosotros le proporcionamos será el resultado de multiplicas $G(s) \cdot \frac{1}{s} \cdot H(s)$ en la Figura 41.

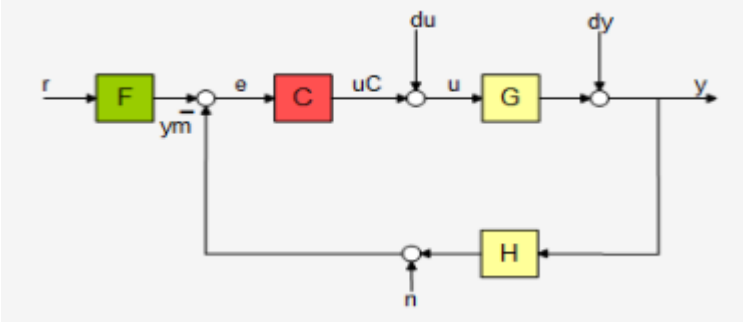


Figura 42. Diagrama de sistema de bucle de control cerrado en rltool.

También hay que configurar el valor de la pre-alimentación, F. A este subsistema le asignaremos nuestro valor de referencia, consiguiendo así que la entrada escalón unitario que aplica la *rltool* pase a tener la magnitud del escalón que realmente hay. En la Figura 26, en el *boxplot*, observamos que el valor de la mediana en el ángulo que más luz recibe es de 0,6V. Debido a la no linealidad ya explicada de las placas, la ecuación de la recta que las aproxima sugiere tomar como referencia 0,3V.

Otro ajuste que deberemos establecer es en "Preferences" -> "Options", seleccionando "Zero/pole/gain". Una vez realizado este paso, podemos hacer el diseño de nuestro controlador.

Inicialmente, en la *rltool* vemos dos gráficas. La de la izquierda representa el lugar de las raíces del sistema y la de la derecha la respuesta ante escalón. A medida que modifiquemos el lugar de las raíces, la respuesta se ajustará automáticamente. Nuestro sistema, inicialmente, se comporta tal como se muestra en la Figura 43. En esa

situación, la realimentación es unitaria y el controlador un simple proporcional de ganancia 1, es decir, C(s) = 1, por lo que la respuesta no tiene por qué ser idónea.

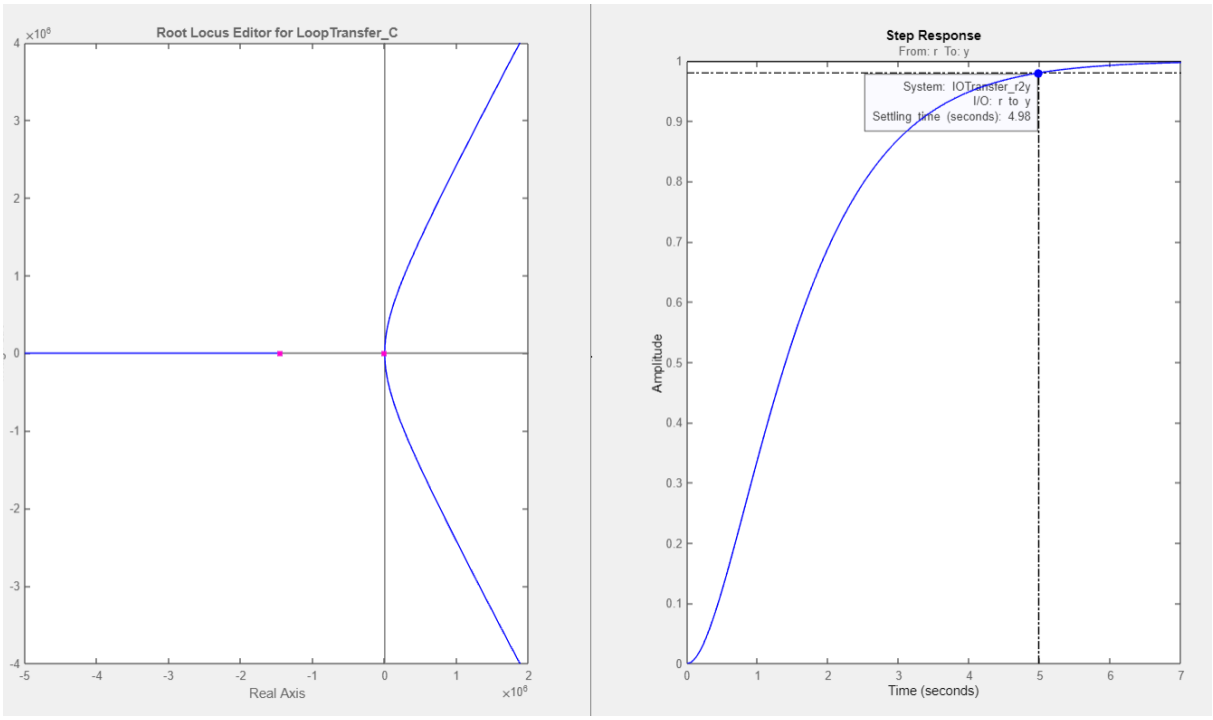


Figura 43. Estado inicial del sistema de bucle cerrado modelado en la rltool.

Para saber si se ha diseñado un buen controlador, debemos tener en mente tres objetivos:

1. *Estabilidad.*

   Para que un sistema lineal invariante en el tiempo sea estable, sus polos deben situarse en el semiplano izquierdo del plano complejo. Esto se debe al comportamiento exponencial de la respuesta del sistema: la función de transferencia se puede expresar como:

$$Y(s) = TF(s)X(s) = \frac{N(s)}{D(s)}X(s)$$

   Los polos del sistema son las raíces del denominador D(s). La descomposición en fracciones parciales es:

$$Y(s) = \frac{A_1}{s - p_1} + \cdots + \frac{A_n}{s - p_n} + \frac{B_1}{s - p'_1} + \cdots + \frac{B_m}{s - p'_m}$$

Donde p_n y p'_m son los polos del sistema. Aplicando la transformada inversa de Laplace, tenemos que:

$$\mathcal{L}^{-1}\left[\frac{A_i}{(s - p_i)^{r_i}}\right] = \frac{A_i}{(r-1)!}\, e^{(\sigma + \omega j)t} t^{r_i - 1} u(t)$$

Aquí, σ representa la parte real (eje x) del polo p_i y determina la estabilidad del sistema y ω representa la parte imaginaria (eje y) y determina las oscilaciones del sistema.

Para que $e^{\sigma t}$ sea decreciente y el sistema sea estable, es necesario que σ sea negativo ($\sigma < 0$), lo que significa que los polos deben estar en el semiplano izquierdo del plano complejo. Si σ está en el semiplano derecho ($\sigma > 0$), la exponencial será creciente, haciendo que el sistema sea inestable.

2. *Error en el estado estacionario.*

Para que no se produzca error en el estacionario, buscamos que la señal de referencia R(s) sea igual en el tiempo infinito a la señal M(s), que es la salida del sensor. Matemáticamente y haciendo uso del teorema del valor final, esto lo podemos expresar como:

$$\lim_{s \to 0} s\big(R(s) - M(s)\big) = \lim_{s \to 0} sE(s) = 0$$

En nuestro caso, al haber introducido un integrador para convertir la salida de la planta de velocidad a posición, no debemos preocuparnos por que este error pueda no ser cero. Lo que estamos haciendo al introducir un integrador es poner un polo en el origen, y al resolver los límites obtenemos el resultado de que el error en el estacionario es cero siempre.

Esto lo podemos demostrar de la siguiente manera:

$$E(s) = \frac{R(s)}{1 + G(s)H(s)}$$

Si tenemos una entrada escalón $R(s) = \frac{A}{s}$ y un sistema con un polo en el origen, que tiene la forma general $G(s)H(s) = \frac{K}{s} \cdot P(s)$

$$\lim_{s\to 0} sE(s) = \lim_{s\to 0} s\left(\frac{\frac{A}{s}}{1+\frac{K}{s}\cdot P(s)}\right)\lim_{s\to 0}\frac{A}{1+\frac{K}{s}\cdot P(s)} = \frac{A}{1+\lim_{s\to 0}\frac{K}{s}\cdot P(s)} = \frac{A}{\infty} = 0$$

3. *Buen transitorio*

Un buen transitorio dependerá de la aplicación y objetivos que tengamos que cumplir. Generalmente, buscamos tener un tiempo de establecimiento corto y poca oscilación.

Los sistemas LTI pueden tener unos polos más dominantes que otros: son aquellos más cercanos al eje imaginario, ya que por su posición producen componentes exponenciales decrecientes en la señal de salida (suponiendo inestabilidad) que tardan más en desaparecer. Estos polos dominantes afectan significativamente el comportamiento del sistema, mientras que los polos menos dominantes tienen una influencia mínima porque esta se desvanece muy rápido. Como ya hemos mencionado, un sistema de segundo orden se caracteriza por tener una función de transferencia de la siguiente forma:

$$TF(s) = \frac{k}{s^2 + 2\omega_n \varsigma s + \omega_n^2}$$

Si resolvemos las raíces del denominador podemos sacar los polos del sistema, siendo estos:

$$p_1 = -\varsigma\omega_n + \omega_n\sqrt{1-\varsigma^2}i$$
$$p_2 = -\varsigma\omega_n - \omega_n\sqrt{1-\varsigma^2}i$$

Dado que los polos se pueden expresar como $p = \sigma \pm \omega j$, tenemos:

$$\sigma = -\varsigma\omega_n \text{ y } \omega = \pm\omega_n\sqrt{1-\varsigma^2}$$

El tiempo de establecimiento, de pico y la sobreoscilación se calculan como:

$$T_s \cong \frac{4}{|\sigma|}$$

$$T_p \cong \frac{\pi}{|\omega|}$$

$$\%OS = 100e^{\frac{\varsigma\pi}{\sqrt{1-\varsigma^2}}} \ (\%)$$

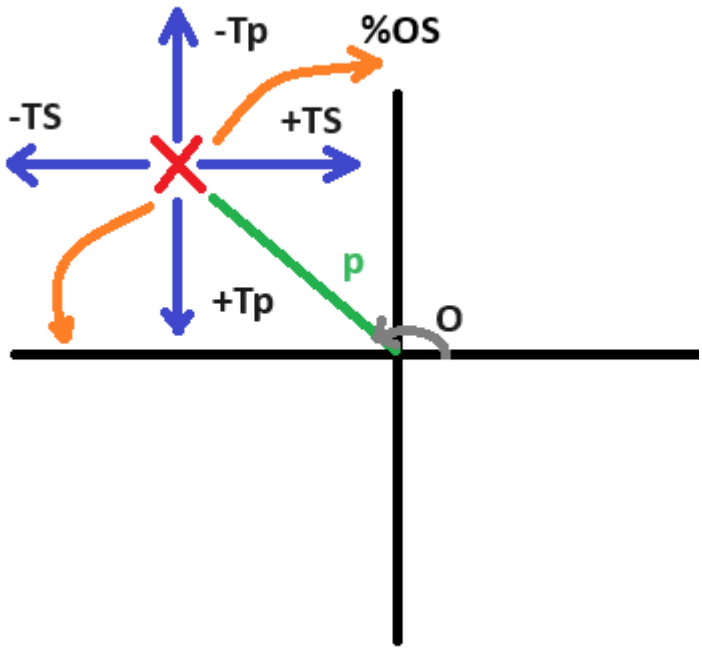


Figura 44. Efectos en el transitorio de mover un polo de un sistema por el plano S.

Para saber bien qué debemos hacer con la %OS, debemos calcular qué pasa con el ángulo del polo respecto al eje real, que se obtiene de la siguiente manera:

$$\theta = \text{atan}\left(-\frac{\sqrt{1-\varsigma^2}}{\varsigma}\right) + \pi$$

Viendo cómo se calculan estas variables sabemos que si nos desplazamos hacia la izquierda en el plano disminuirá el tiempo de establecimiento; si nos desplazamos arriba, disminuirá el tiempo de pico, y cuanto más nos acerquemos al eje real, menos oscilaciones se producirán. Todo esto se resume en la Figura 44.

Por lo tanto, el objetivo de diseñar un controlador en el lugar da las raíces es situar los polos del sistema de bucle cerrado en lugares donde estos parámetros (tiempo de establecimiento, sobreoscilación, etc) sean los más parecidos a los buscados.

4.1 Control P (proporcional)

Nuestro sistema es estable. Sin embargo, debemos tener cuidado al elegir los parámetros de un controlador proporcional, cuya función de transferencia es $C(s) = K_p$, ya que podríamos mover los polos hacia el semiplano derecho, lo que haría que el sistema se volviera inestable. También hemos visto que no tenemos error en estado estacionario. El problema que tenemos entonces es su tiempo de establecimiento, que con la configuración inicial de la *rltool* es de casi 5 segundos (Figura 43).

Para abordar esto, intentaremos encontrar un mejor controlador proporcional (P); son los controladores más simples, pero pueden mejorar la respuesta transitoria del sistema. Como desventaja, no pueden quitar el error, pero al no ser ese uno de nuestros problemas, pueden ofrecernos una buena solución.

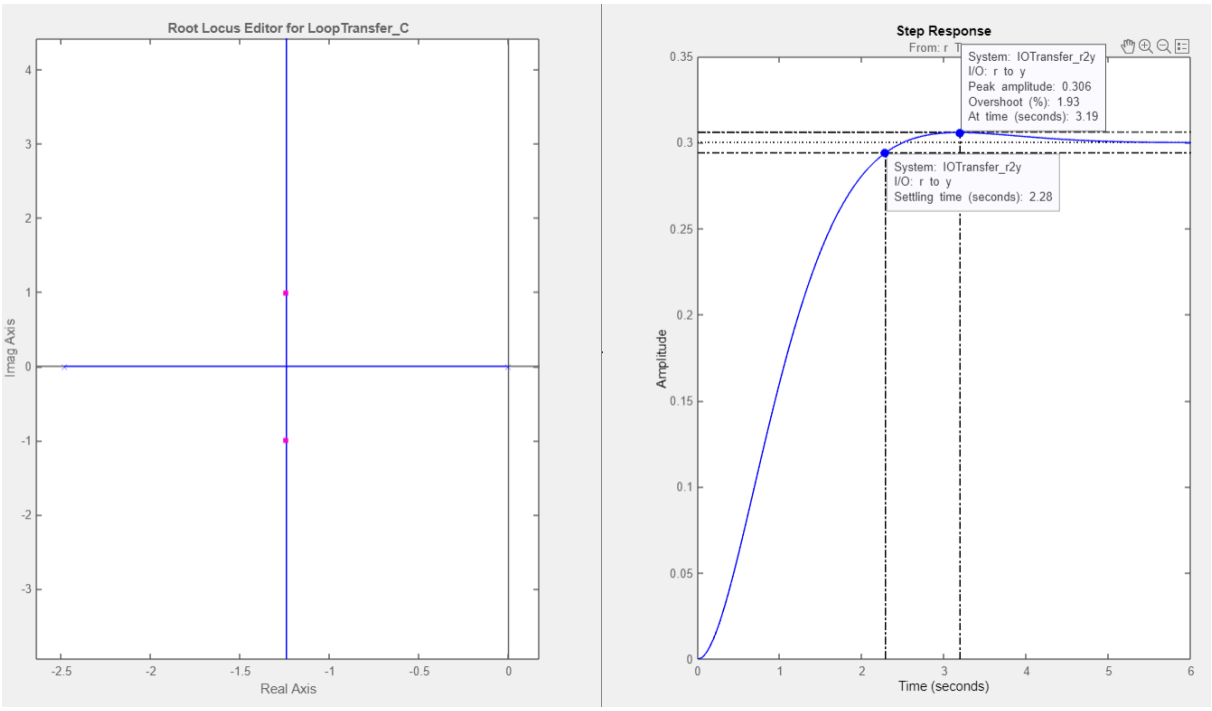


Figura 45. Controlador proporcional diseñado con la rltool.

En la Figura 45 se muestra el resultado final del diseño del controlador P. En la gráfica de la izquierda, correspondiente al lugar de las raíces, hemos realizado una ampliación centrada en los dos polos más dominantes, aquellos que están más cercanos al origen. Para mejorar el tiempo de establecimiento, hemos movido el polo más próximo al origen hacia la izquierda. Sin embargo, este ajuste ha introducido algo de oscilación debido al alejamiento del eje real. Una regla práctica es que mientras no tengamos más de un 20% de OS, se puede considerar trabajar con el modelo realizado. En nuestro caso conseguimos bajar el tiempo de establecimiento a 2,28 segundos, más de la mitad que el inicial. Esto se ha logrado con un controlador proporcional con $K_p = 1.7$.

4.2 Control PD (proporcional + derivativo)

Para seguir mejorando el tiempo de establecimiento, hemos implementado también un controlador proporcional derivativo (PD). Este tipo de controlares tienen la forma:

$$C(s) = K_p + K_d s$$

Al introducir un término derivativo, el controlador considera la derivada del error actual. Esto le permite anticipar el comportamiento futuro del sistema; al prever la tendencia del error, puede aplicar correcciones antes de que el error aumente significativamente, lo que mejora la respuesta transitoria del sistema. Como la introducción del cero real negativo en el controlador atraerá ramas hacia el lado izquierdo y eje real del lugar de las raíces, es posible con este tipo de controlador mejorar la estabilidad y la velocidad de respuesta del sistema, aumentar el amortiguamiento y reducir las oscilaciones. Este cero puede ser ajustado moviendo el valor de K_p. Dependiendo de su posición, afectará a la distribución de las ramas en el plano complejo de distintas maneras.

En nuestro caso, probamos varias posiciones para este cero con el fin de obtener resultados específicos.

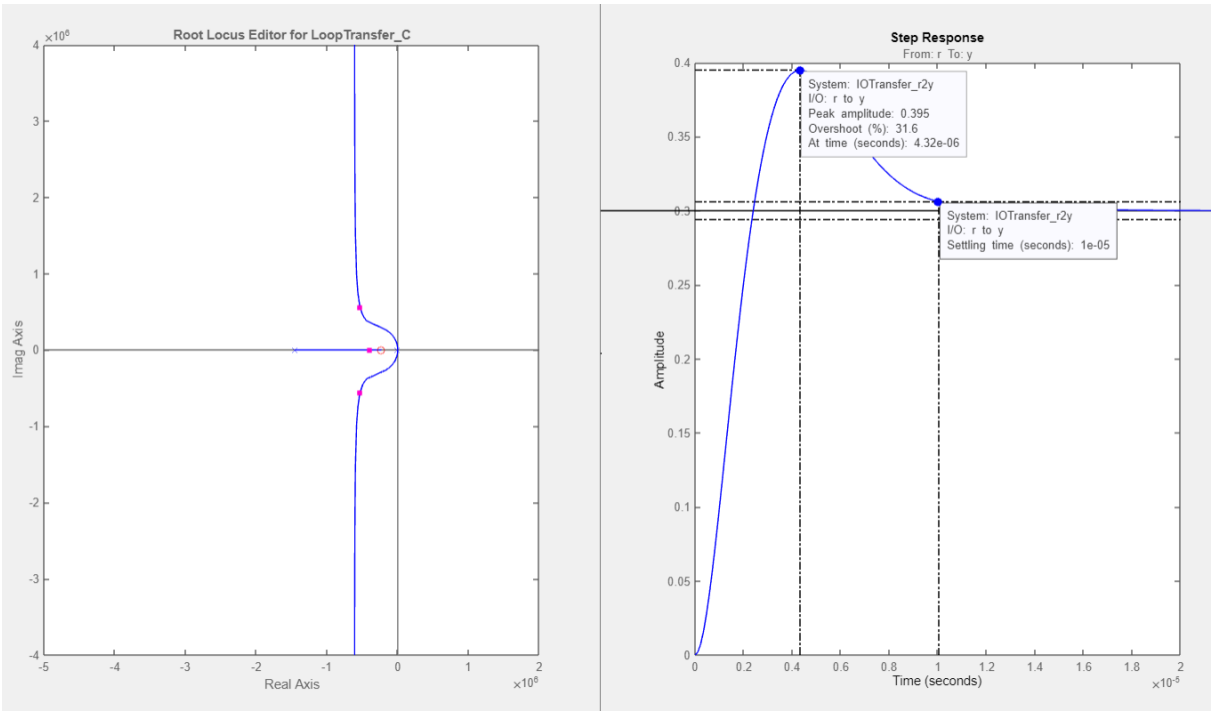


Figura 46. Control PD, opción 1.

Una de las opciones es la que vemos en la Figura 46. Si colocábamos el cero para atraer la rama situada más a la izquierda podíamos moverlo a una posición en la que conseguíamos curvar las otras dos ramas para que se vinieran al semiplano izquierdo. Sin embargo, tenemos dos problemas: %OS = 31,6% (mayor que el 20%), $K_d = 4{,}6779 \cdot 10^5 s$ y $K_p = 1{,}0792 \cdot 10^{11}$, valores demasiados grandes para introducirlos en un microcontrolador de 8 bits.

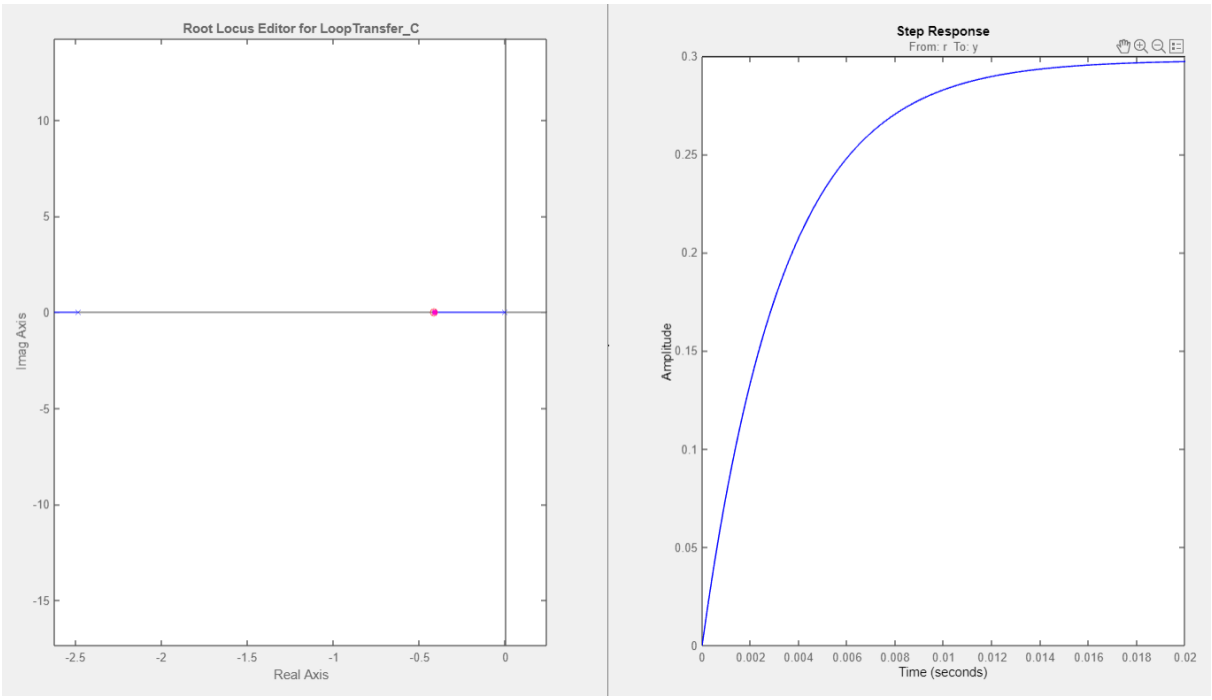


Figura 47. Control PD, opción 2.

La segunda opción es la que observamos en la Figura 47. En este caso ubicamos el cero cerca del polo que está situado en el origen y ajustamos el polo del bucle cerrado con K_d para que esté lo más cercano posible al cero, cancelando así su efecto de dominancia. De esta manera, los otros dos polos del sistema serán los que definan la respuesta.

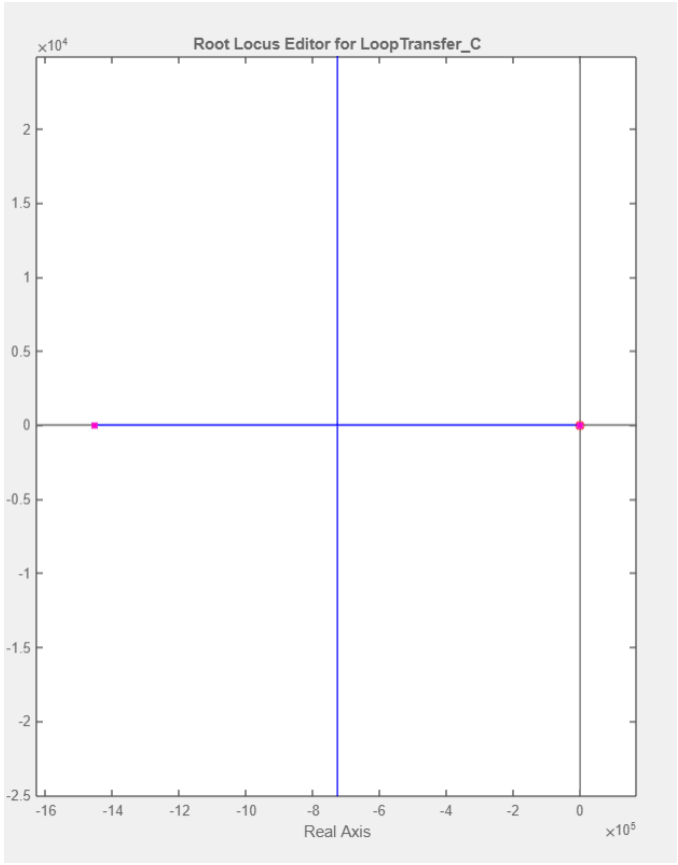


Figura 48. Lugar de las raíces de la Figura 47 sin ampliación.

Jugando con la ampliación sobre los dos polos que están más cercanos al eje imaginario, fuimos ajustando los valores hasta encontrar un buen tiempo de establecimiento sin comprometer otras características necesarias. Finalmente, llegamos al resultado de $K_d = 200s$ *y* $K_p = 82{,}8$. Con estos valores obtenemos un tiempo de establecimiento de 0,0145s como se puede ver en la Figura 49.

Con esta configuración colocamos el cero aproximadamente en -0,5, como se observa en la Figura 47. Este cero atrae las ramas del lugar de las raíces que hacían que el sistema pudiera ser inestable. Al situar este cero, logramos que la rama cuyo polo sería más dominante tenga menos influencia en el sistema, acercando dicho polo lo más posible al cero. Como resultado, tenemos un sistema en el que la dominancia viene dada por los otros dos polos, que están situados más a la izquierda y sobre el eje real.

Como es de esperar, esto se traduce en una reducción del tiempo de establecimiento y de la sobreoscilación.

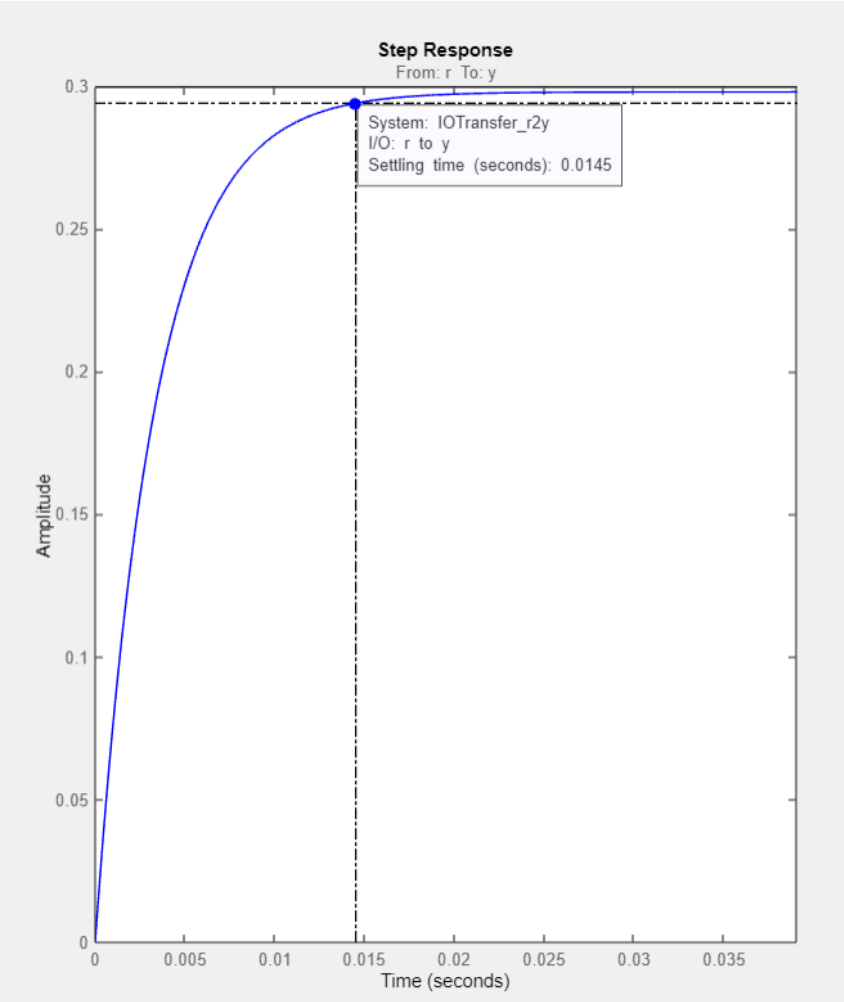


Figura 49. Tiempo de establecimiento para el controlador de la Figura 47.

Capítulo 5

Experimentos reales

En el capítulo 4 hemos obtenido nuestros controladores P y PD. Lo siguiente es comprobar si los resultados obtenidos con *rltool* proporcionan buenos controladores en el mundo real.

En la implementación de los controladores hay un comportamiento esperado que puede resultar extraño. Anteriormente, se mencionó que, debido a la no linealidad que surgía en los extremos del comportamiento del sensor, íbamos a trabajar en el punto medio de nuestra ecuación de recta de regresión. Esto nos supondría tomar como voltaje de referencia 0,3V. Al tomar este valor como referencia, nuestros controladores lo que harán es estar continuamente girando en ausencia de luz hasta encontrar una fuente de luz y poder estabilizarse en esos 0,3V.

El código desarrollado para ambos controladores, al igual que en los experimentos de caracterización, incluye la configuración del ADC y de la PWM. Esta configuración es la misma que se ha utilizado en los experimentos de caracterización del capítulo 3. La diferencia principal, una vez más, se encuentra en la ISR del *timer* 1.

En los experimentos solo se ha usado una de las placas solares para tomar las medidas de la fuente de luz. Dejamos como trabajo futuro utilizar dos placas, por ejemplo, restando sus medidas y usando como señal de referencia 0V en lugar de 3,3V.

5.1 Controlador P

La forma que tiene nuestro controlador proporcional es la siguiente:

$$C(s) = K_p = 1{,}7 \rightarrow \mathcal{L}^{-1} \rightarrow u(t) = K_p \cdot e(t)$$

Donde e(t) representa el error. Este se calcula haciendo uso del valor de referencia y el valor que se está leyendo del ADC, la cantidad de luz que está recibiendo la placa solar, convertida a voltios.

El código del controlador P podemos verlo en la Figura 50.

```
121 ISR(TIMER1_COMPA_vect){
122     /*Tenemos un timer de 0.01s (tiempo de muestreo), cuando pasa ese tiempo se comprueba el estado de las placas de solares y se aplica el control*/
123
124     if(listosEnviar == 0){
125
126         int16_t error;          //Señal de error
127
128         adc0_Valor = leerADC(0);
129
130         error = (int16_t) adc0_Valor - (int16_t) REFERENCIA;
131         U = (double)Kp * error;
132
133         vec_adc[indice] = adc0_Valor;
134         vec_error[indice] = error;
135         indice++;
136
137         //Aplicar el sentido de giro según el valor de la señal de control
138         if(U < 0){
139             sentido_giro(1);
140         }else{
141             sentido_giro(0);
142         }
143
144         //Ajustar la señal de control para el motor. Limitar U para que esté en el rango del PWM (0 a 60), no llego a 80 por la no linealidad de saturacion
145         U = U < 0 ? -U : U;
146         if (U > 60)
147         {
148             U = 60;
149         }
150
151         OCR0B = (uint8_t) U; //Ajustar el duty cycle del PWM
152
153         if(indice >= TAM)
154         {
155             listosEnviar = 1;
156             indice = 0;
157             OCR0B = 0; //Paramos el motor mientras se envian los datos por el puerto serie
158         }
159
160     }
161 }
162
```

Figura 50. Código de la ISR del controlador P.

Lo que se hace es leer la cantidad de luz que está recibiendo la placa solar (línea 128). Con este valor y el valor de referencia, se calcula el error, y este valor se multiplica por nuestra K_p para obtener nuestra señal de control U.

Según el signo que tenga esta señal de control, se hará girar a la placa solar en un sentido u otro. A su vez, aplicamos el límite máximo que puede obtener esta señal de

control. Al igual que antes, cuando estamos mandado los datos por el puerto serie detenemos el funcionamiento de la ISR, para lo que paramos el motor. Esto producirá cambios en la respuesta, pero no podemos evitarlo dadas las limitaciones de memoria del controlador. Nótese que no es un problema si se controla el sistema de manera continua, sin recoger datos del mismo.

El resultado usando $K_p = 1{,}7$ podemos observarlo en la Figura 51.

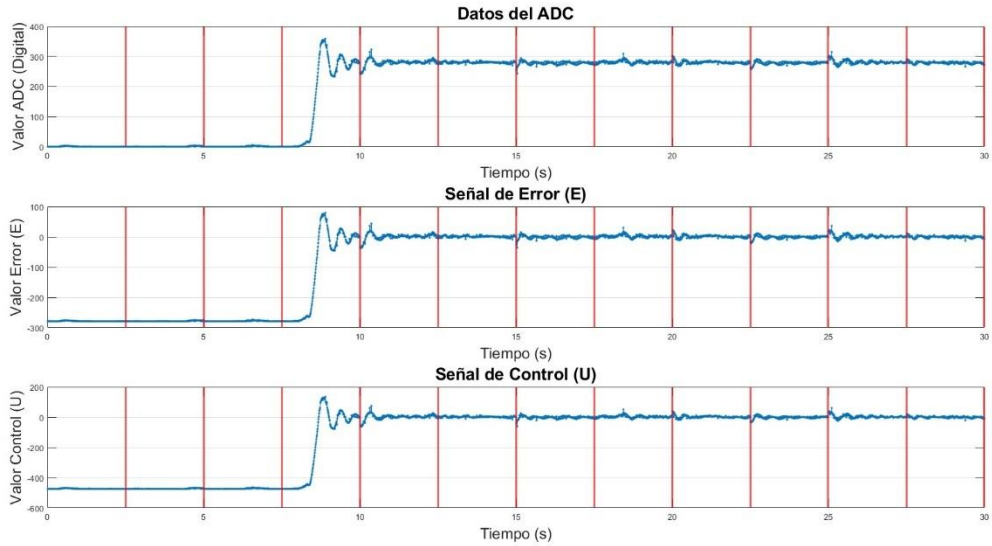


Figura 51. Control P con Kp = 1,7.

En este experimento lo que se hizo es tener al principio la fuente de luz apagada, esto es lo que vemos en las gráficas antes de que se produzca la subida. Una vez se produce la subida, vemos unas pequeñas oscilaciones hasta que el controlador consigue estabilizarse. Como podemos apreciar, la señal de control se estabiliza en el valor 0, indicando que no tenemos error en el estado estacionario, es decir, la placa solar se queda estabilizada de tal manera que está recibiendo luz para producir 0,3V.

También podemos observar pequeñas oscilaciones en el estado estacionario; estas oscilaciones pueden deberse al comportamiento de la placa solar explicado en la Figura 27.

Como podemos ver, el comportamiento del controlador que diseñamos con la *rltool* es bastante bueno. Aun así, como observamos que se producen oscilaciones, realizamos una prueba disminuyendo el valor de K_p a 0,5. De esta manera buscamos disminuir las oscilaciones sabiendo que aumentaremos el tiempo de establecimiento.

En la Figura 52 podemos observar los resultados obtenidos. Como podemos observar no conseguimos mejorar, sino todo lo contrario, obtenemos un peor controlador. Hay que tener en cuenta que el sistema real es no lineal y no tiene por qué obedecer las reglas que determinan el comportamiento de los sistemas LTI. Se realizaron pruebas con valores entre 1 y 1,7, obteniendo resultados similares a los vistos en la Figura 51.

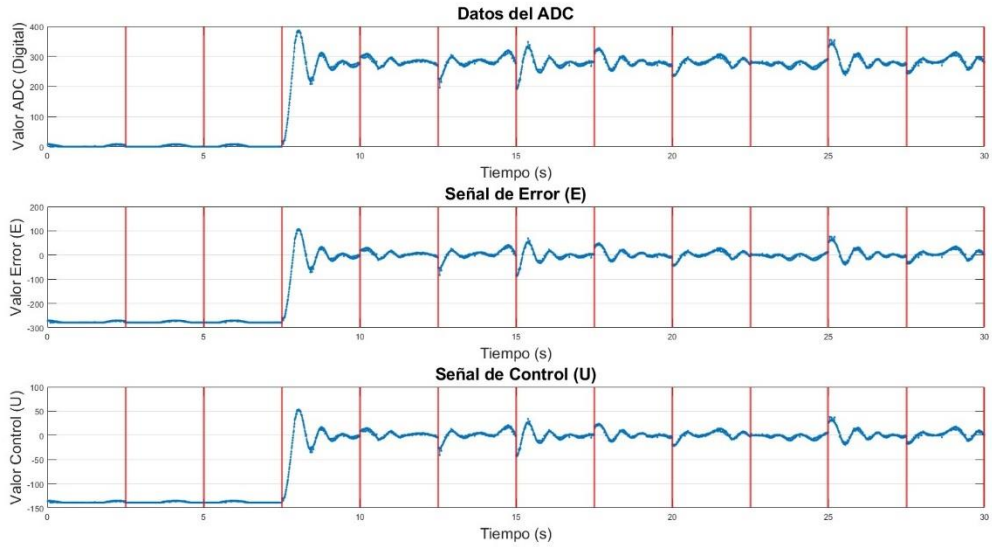


Figura 52. Control P con Kp = 0,5.

En todas estas figuras se pueden apreciar unas líneas rojas verticales. Estás líneas rojas se han añadido para señalar aquellos momentos en los que se ha parado el motor para poder mandar los datos por el puerto serie. En el controlador P los *arrays* que se pueden tener son de 250 datos; como estamos tomando 3000 datos y la ISR se ejecuta cada 0,01 segundos, el experimento dura 30 segundos. Lo que se está viendo es que cada 2,5 segundos se está dibujando una línea vertical roja, que es lo que tarda en llenarse el *array* para ser enviado ($250 \cdot 0{,}01 = 2{,}5$).

5.2 Controlador PD

La forma que tiene un controlador PD es la siguiente:

$$C(s) = K_p + K_d s \rightarrow \mathcal{L}^{-1} \rightarrow u(t) = K_p \cdot e(t) + K_d \frac{de(t)}{dt}$$

La parte del control derivativo se implementaría de la siguiente manera aproximadamente:

$$u_k = K_d \frac{e_k - e_{k-t}}{T} \; si \; k > 0$$

$$u_k = 0 \; si \; k = 0$$

Donde e_k es el error actual, e_{k-t} el error anterior y T el tiempo de muestro.

Los experimentos realizados son los mismos que en el controlador P, haciendo los ajustes necesarios en la ISR para convertirlo en un controlador PD, que podemos ver en la Figura 53.

```
/* ISR Timer_1 */
ISR(TIMER1_COMPA_vect){
    /*Tenemos un timer de 0.01s (tiempo de muestreo), cuando pasa ese tiempo se comprueba el estado de las placas de solares y se aplica el control*/

    if(listosEnviar == 0){

        int16_t error;          //Señal de error
        double error_derivada;   //Derivada del error

        adc0_Valor = leerADC(0);

        error = (int16_t) adc0_Valor - (int16_t) REFERENCIA;
        //error derivada = 0 primera vez y cuando se retome despues de enviar por rs232
        if(indice == 0)
        {
            error_derivada = 0;
        }else{
            error_derivada = (error - error_anterior)/t_muestreo;
        }
        error_anterior = error;

        U = (double)Kp * error + Kd * error_derivada;

        vec_adc[indice] = adc0_Valor;
        vec_error[indice] = error;
        vec_errorD[indice] = Kd*error_derivada;
        indice++;

        if(U < 0){
            sentido_giro(1);
        }else{
            sentido_giro(0);
        }

        //Ajustar la señal de control para el motor
        //Limitar U para que esté en el rango del PWM (0 a 60), no llego a 80 por la no linealidad de saturacion
        U = U < 0 ? -U : U;
        if (U > 60) U = 60;

        OCR0B = (uint8_t) U; //Ajustar el duty cycle del PWM

        if(indice >= TAM)
        {
            listosEnviar = 1;
            indice = 0;
            OCR0B = 0; //Paramos el motor mientras se envian los datos por el puerto serie
        }
    }
}
```

Figura 53. Código de la ISR controlador PD.

El tiempo de muestreo tiene que ser el tiempo con el que se ejecuta la ISR; en este caso se está ejecutando cada 0,01 segundos.

El problema que nos encontramos con este controlador es que el diseño obtenido en la *rltool,* al llevarlo al mundo real, no nos está dando un controlador con buenos resultados, como se ve en la Figura 54.

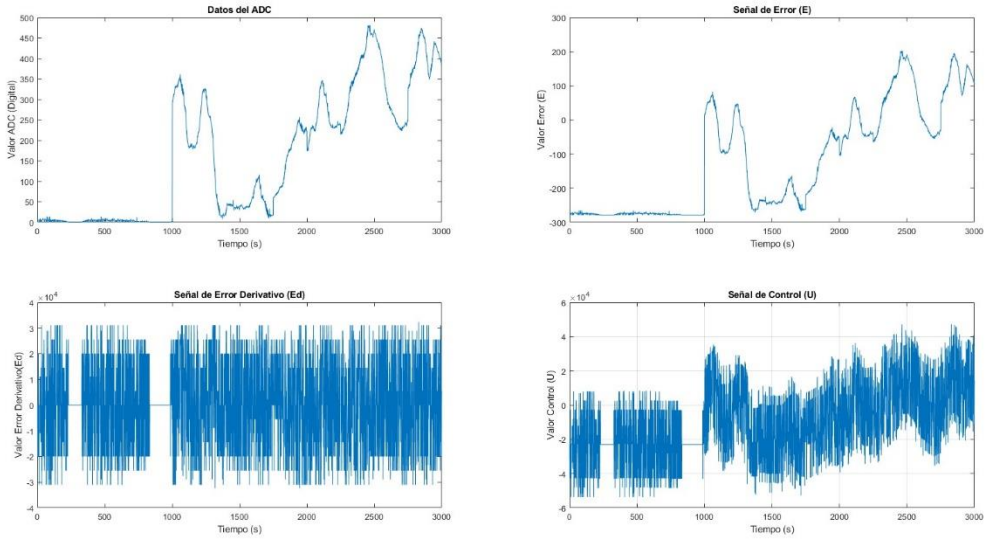


Figura 54. Implementación del controlador PD de la rltool.

Esto se puede deber a que lo que se modela en la *rltool* es ideal y lineal. Recordemos que nuestro sistema tenía un gran rango de inestabilidad, y que al añadir el cero curvábamos las ramas para convertirlo en un sistema estable (Figura 47). Poníamos el cero muy cerca del origen; cuanto más cerca estemos del origen, al traerlo al mundo físico puede producir qué se siga comportando como un sistema inestable.

Viendo el comportamiento del controlador que habíamos diseñado inicialmente y pensando que podía deberse a un comportamiento inestable, realizamos un nuevo diseño con la *rltool* donde situamos el cero más a la izquierda en el lugar de las raíces.

El controlador que obtenemos ahora es el que podemos ver en Figura 55. Como se puede ver, hemos situado el cero más a la izquierda, en -5. Esto ha producido que cambie cómo se mueven las ramas de nuestro sistema por el lugar de las raíces. Uno de

los cambios más notorios respecto a los que teníamos inicialmente es que aparece sobreoscilación y un aumento del tiempo de establecimiento.

Como resultado, ahora tenemos un controlador con la siguiente expresión:

$$C(s) = 50 + 10s$$

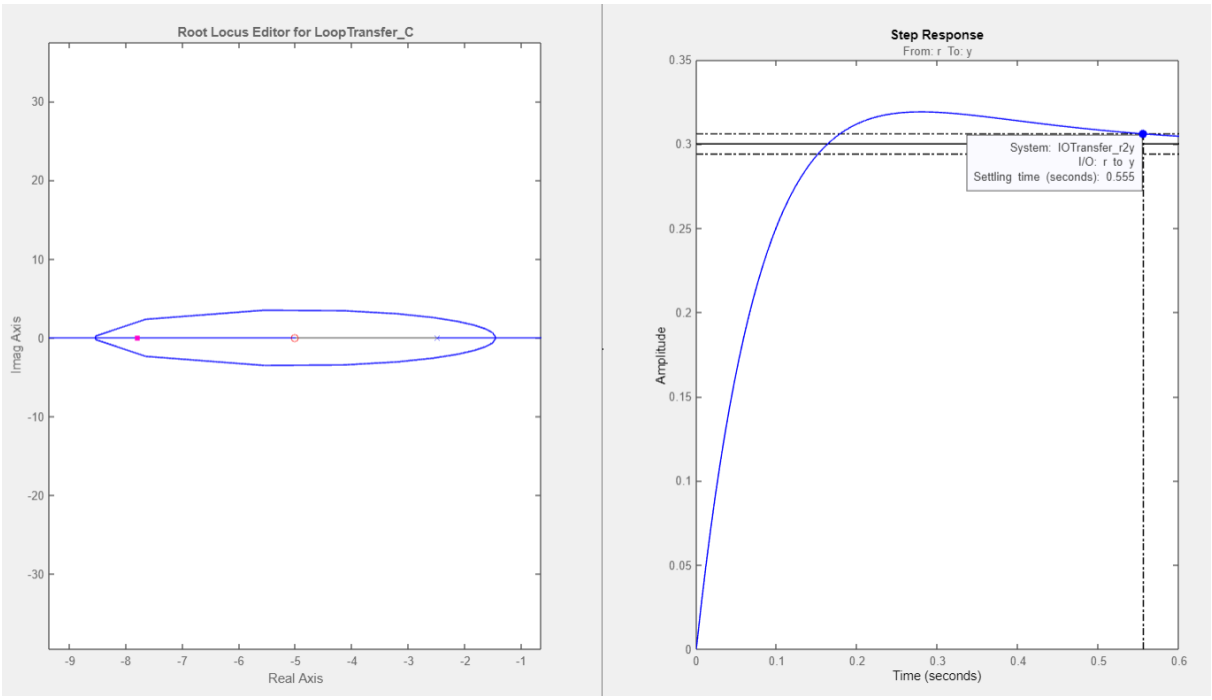


Figura 55. Corrección del controlador PD.

Al implementar este controlador en el Arduino obtenemos los resultados de la Figura 56. Ahora sí obtenemos un comportamiento mejor. Al igual que con el controlador P, introducimos las líneas rojas verticales; la diferencia es que aquí son cada 2 segundos, ya que los vectores en esta ocasión son de 200 datos. Esto se debe a que estamos mandando más datos por el puerto serie (los correspondientes al error derivativo) y a la memoria limitada del ATmega328P.

Las diferencias que podemos ver entre el controlador P y el PD son varias:

1. *Oscilaciones.* Podemos ver que en el controlador P tenemos oscilaciones antes de llegar al estado estacionario, pero en el PD no.

2. *Tiempo de establecimiento.* También podemos ver que el controlador PD tiene un estado transitorio más corto, alcanzando más rápido el estado estacionario.
3. *Estado estacionario.* Cuando el controlador P alcanza el estado estacionario es muy lineal; sin embargo, en el controlador PD podemos apreciar que hay ruido en todas las señales. Esto se debe al derivador. Esta cuestión sería otro de los aspectos a tratar en un futuro: reducir el ruido producido por el derivador en el controlador PD. Es bien sabido que las derivadas de una señal con ruido amplifican dicho ruido.

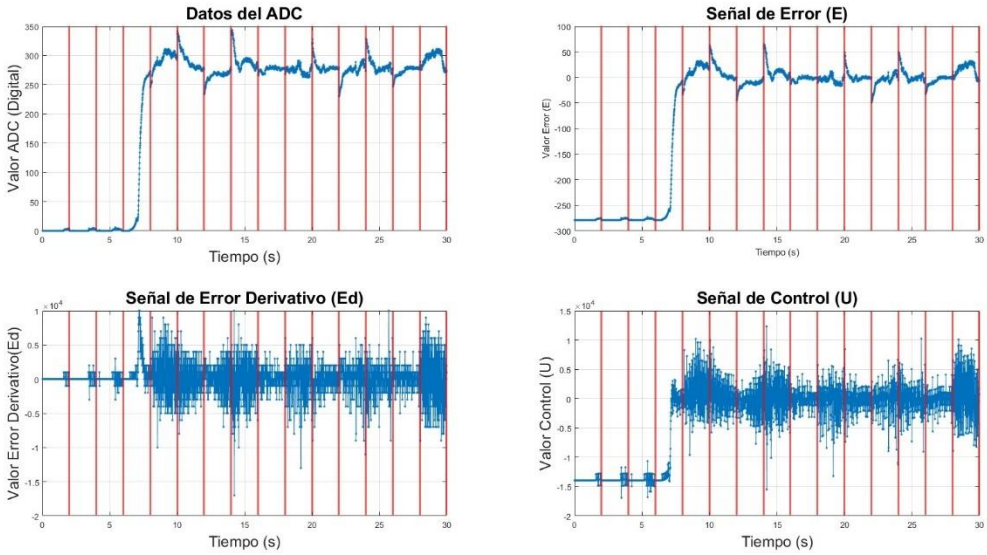


Figura 56. Controlador PD con Kp = 50 y Kd = 10.

Capítulo 6

Futuros trabajos

Este proyecto ha finalizado con el desarrollo de dos controladores, uno proporcional y otro proporcional derivativo, que, haciendo uso de una placa solar y un motor DC instalados en un montaje propio de bajo coste y programados en un microcontrolador popular, son capaces de buscar una fuente de luz y establecerse en un valor de referencia que esté fijado en 0,3V.

A lo largo de esta memoria hemos visto todo el proceso seguido para llegar a estos resultados:

1. Compra de materiales. Hemos visto qué materiales se han comprado para realizar el proyecto y los criterios seguidos para seleccionarlos.
2. Caracterización. Una vez adquiridos los materiales necesitábamos saber cómo se comportaban en distintos experimentos. Para ello los sometimos a diferentes tipos de pruebas y, haciendo uso de Matlab, pudimos analizar estos resultados. En esta etapa comenzamos a encontrar los primeros problemas/dificultades que íbamos a tener que enfrentar a lo largo del proyecto, y por los que se han tenido que ir tomando diferentes decisiones.
3. Diseño de controladores. Una vez que ya sabíamos cuál era el comportamiento de nuestros componentes y habíamos obtenido sus funciones de transferencia, pudimos pasar a usar la *rltool* para, haciendo uso del lugar de las raíces, diseñar diferentes tipos de controladores.

4. Implementación en el mundo físico. Los controladores diseñados en la *rltool* había que probarlos en el mundo físico y ver si su comportamiento era el esperado. Esto lo hemos podido ver más claramente en el controlador PD donde se ha observado cómo lo que obtenemos con la *rltool* son comportamientos ideales que, al llevarlos a la realidad, fallan y requieren realizar otro diseño.

6.1 Conclusiones

Al iniciar este proyecto, una de las propuestas principales fue desarrollar un material práctico para las asignaturas de Control Automático, con el objetivo de sustituir los Legos EV3, que presentan diversas no linealidades. A lo largo del desarrollo de la memoria hemos visto como en el proyecto nos hemos enfocado en aplicar técnicas para diseñar controladores lineales clásicos. Sin embargo, de manera recurrente nos hemos encontrado con diferentes tipos de no linealidades, tanto en las placas solares como en el motor, lo que nos ha obligado a trabajar bajo ciertas limitaciones.

A pesar de estas dificultades, hemos documentado todo el proceso necesario para llevar a cabo este tipo de trabajo, detallando las decisiones que tomamos debido a los problemas encontrados. Como resultado, hemos logrado implementar controladores de tipo P y PD, creando un sistema físico de bajo coste capaz de seguir automáticamente una fuente de luz externa.

6.2 Futuros trabajos

A lo largo del documento se han mencionado algunas tareas que deberían tenerse en cuanta y ser revisados en un futuro:

1. *Comportamiento placas solares.* Habría que estudiar mejor el comportamiento oscilatorio de las placas solares y ver porqué se está produciendo tanto ruido e intentar reducirlo.

2. *Ruido del controlador PD.* Cuando se añade un derivador en los controladores podemos tener ventajas al anticipar los cambios futuros de la señal de error, ya que aplica correcciones antes de que se produzca el error; sin embargo, como desventaja es más sensible al ruido. Para ello se podría estudiar implementar algún tipo de filtro para reducirlo.
3. *Placas solares.* En el montaje tenemos dos placas solares. La idea principal era hacer uso de las dos. Por cómo se han ido desarrollando las cosas, se tomó la decisión de probar inicialmente a implementar los controladores solamente con una de ellas, y, según como fuéramos avanzando, introducir la segunda.

   Las configuraciones de la placa Arduino están hechas desde el principio para poder trabajar con ambas, pero finalmente sólo se ha trabajado con una de ellas, ya que constatamos que los objetivos del proyecto se habían cumplido en el tiempo planificado para su desarrollo.

   Ahora mismo, que estamos trabajando con una sola placa, lo que hacemos en los controladores es leer el valor de su correspondiente ADC y realizar la operación que corresponda con este valor. En cuanto al código que se implementa del controlador, no hay que modificar gran cosa para adaptarlo a dos placas, ya que lo que implementaríamos sería el resultado de la resta de las dos señales de luz. El motivo de usar la resta es por cuestión de linealidad, ya que si restamos el valor del ADC0 y ADC1 empezaríamos con una curva en su amplitud máxima que sería decreciente y llegaría hasta la misma amplitud en negativo.

   Sobre esta curva deberíamos realizar nuestra nueva aproximación lineal para poder sacar una ecuación de la recta y así su función de transferencia para sacar un nuevo modelo.

Para obtener esa curva deberíamos hacerlo con la curva que tenemos de haber modelo una de las placas solares, Figura 28. En esta gráfica tenemos la parte decreciente hacia la parte derecha, pero por la parte izquierda debería a parecer lo mismo; recordemos que lo que aparece en la Figura 28 solo representa desde el ángulo 0º hasta el 180º. Si representamos desde el ángulo 0º al 360º obtendremos una gráfica con forma de campana (Figura 57). Esta gráfica representa el comportamiento de una de las placas. Para sacar el comportamiento de la otra placa deberemos desplazarlo 90º, ya que es el ángulo que forman entre sí las placas solares en la estructura. Con esto ya tendríamos las gráficas de nuestras dos placas solares y con ello podremos realizar la curva de la resta.

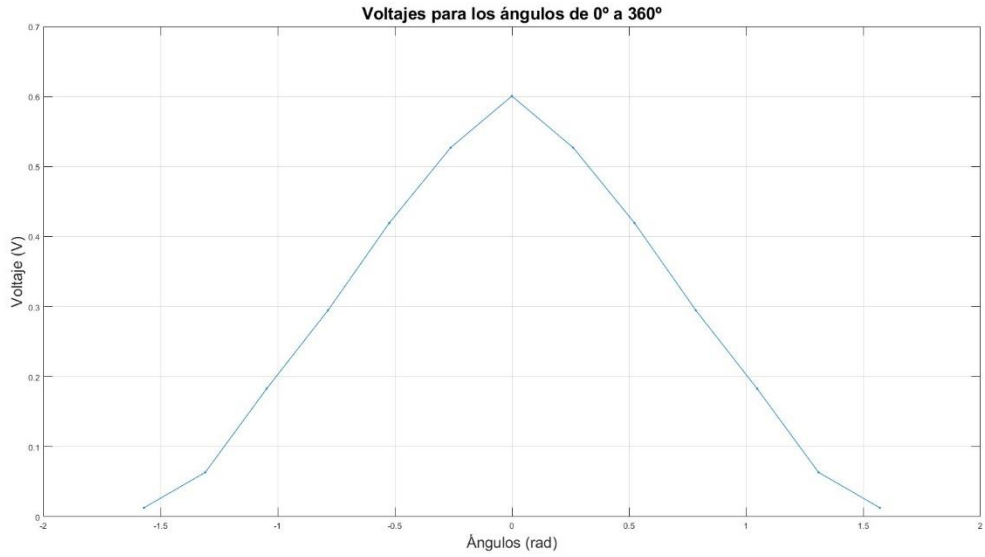


Figura 57. Representación de los voltajes de 0º a 360º para una placa solar.

Capítulo 7

Apéndices

7.1 Lista de componentes y presupuesto

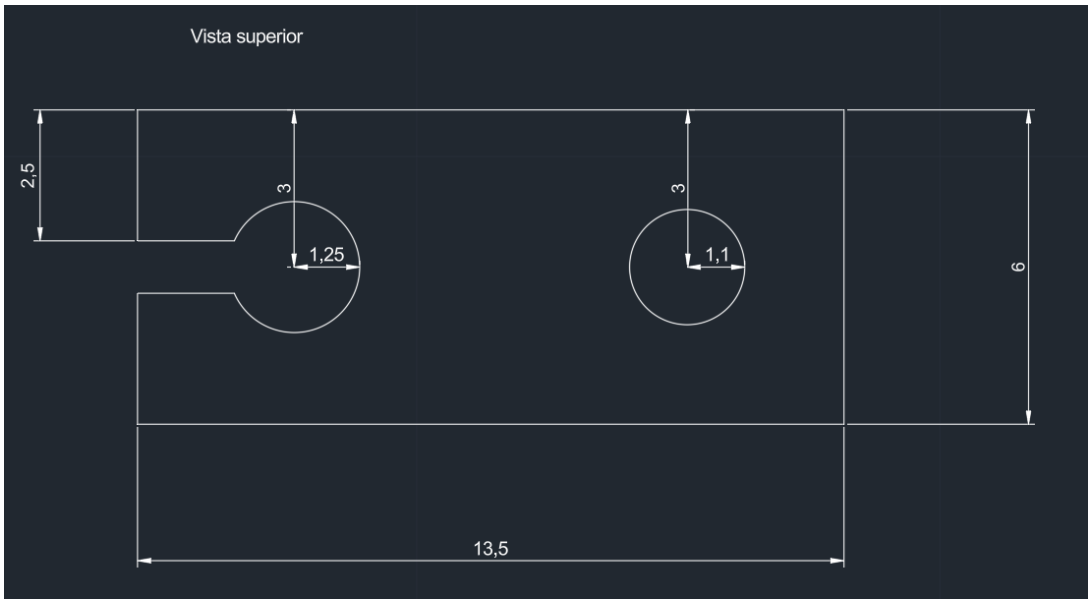
Componentes	**Precios/unidad (con IVA)**
Placa solar (x2)	11,20€
Motor DC	62,80€
Anillo de cables	21,72€
Poleas	1,82€ + 1,33€
Correa	5,99€
Resistencia 680Ω (x2)	> 1.00€
Puente H	8,00€
Cable de alimentación 12V	12,00€
Total	137,06€

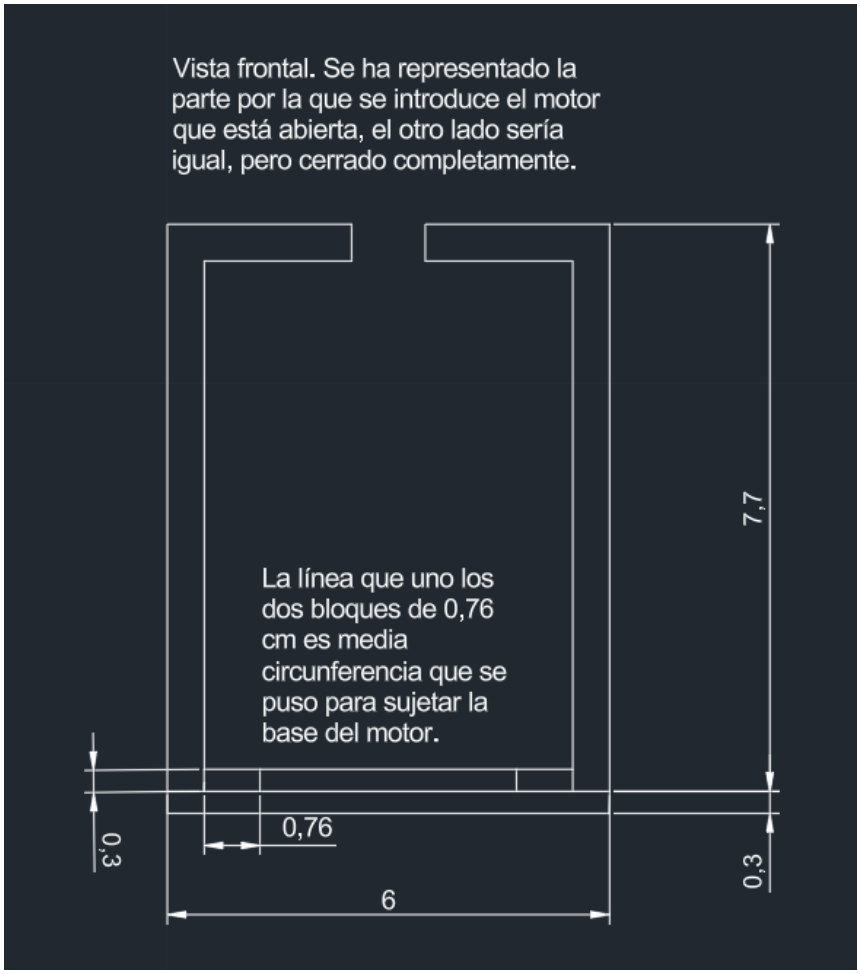
Los cables y el Arduino que se han usado en el proyecto ya se habían adquirido para realizar otros trabajos antes del TFG.

No se están teniendo en cuenta los precios de la estructura diseñada con la impresora 3D, ya que el costo adicional sería el del rollo de filamento, que ronda aproximadamente 20€ por 1 kg, siendo esta cantidad superior a la utilizada en el proyecto.

7.2 Medidas de la estructura

Todas las unidades de las figuras son en cm.





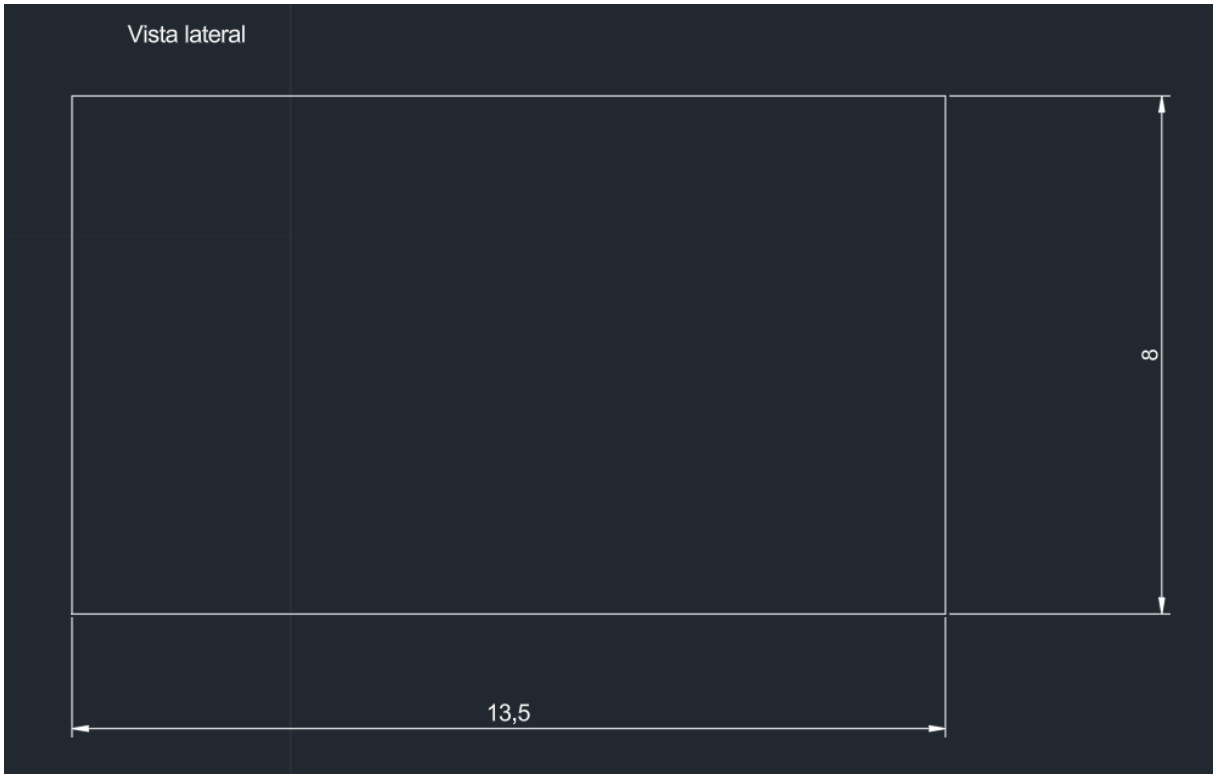


Figura 58. Vistas con las medidas de la caja.

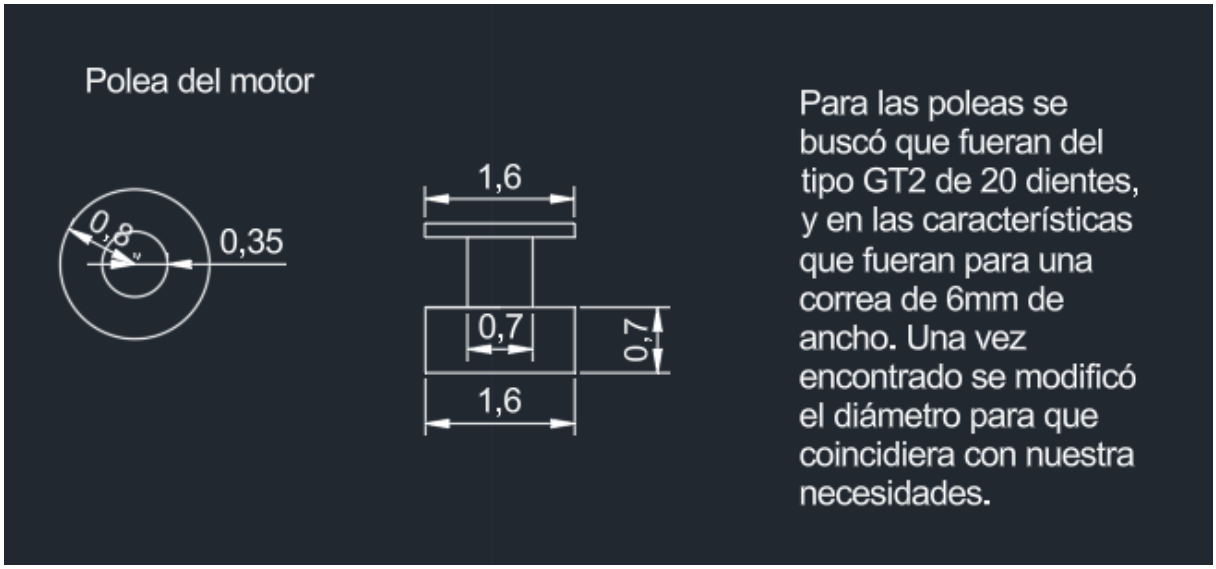


Figura 59. Medidas de la polea del motor.

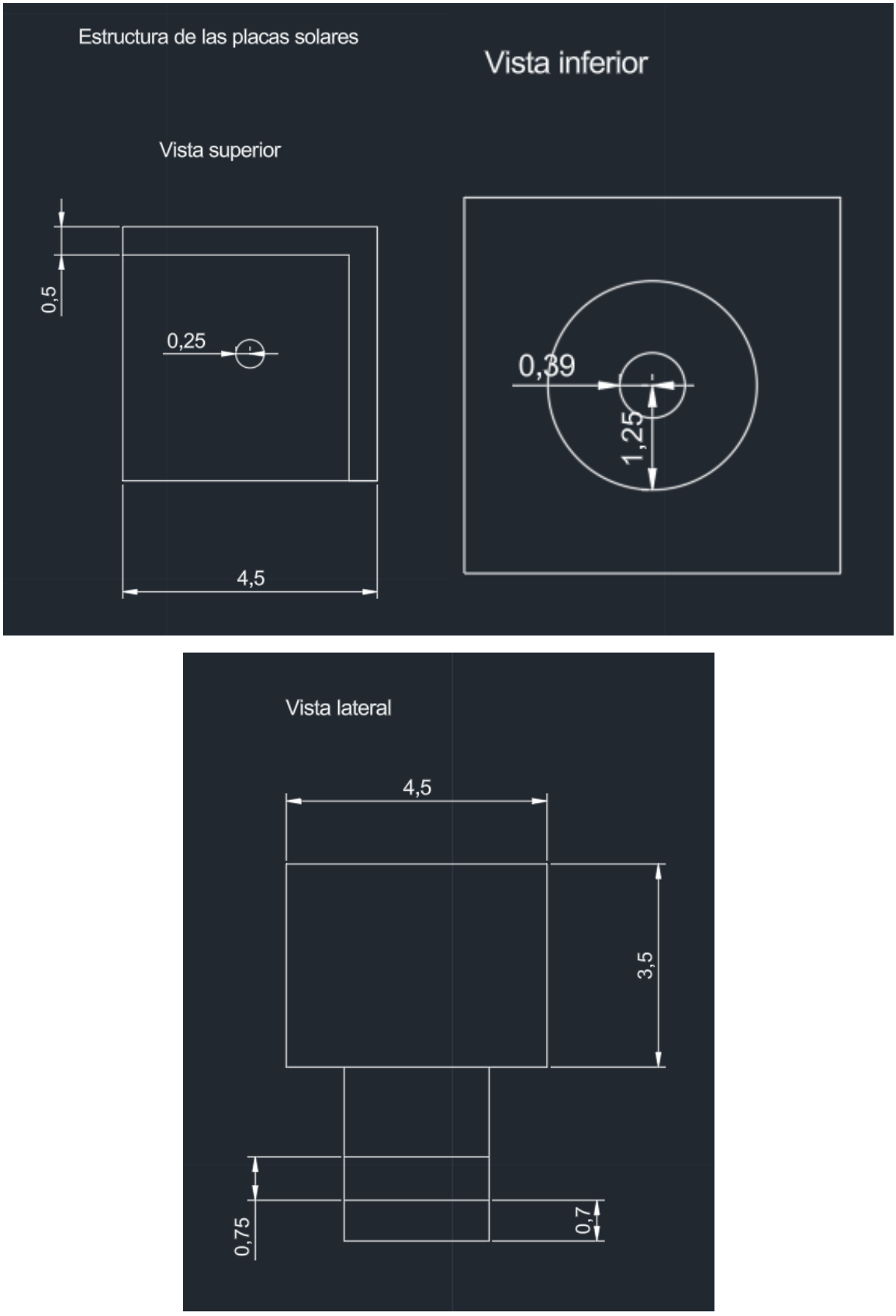


Figura 60. Vistas con las medidas de la estructura de las placas solares.

7.3 Instalación de Software

```c
/*
 * controlP.c
 *
 * Created: 06/07/2024 19:54:26
 * Author : Javi
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>
#include "rs232atmega.h"


/* -------------- Constantes y Macros --------------- */

#define MYRS232BUFSIZE 32           //Tamaño buffer
#define MYRS232ENDCHAR '\r'         //Final de un comando
#define MYRS232SENDSIZE 128         //Tamaño maximo del comando enviado
#define MYRS232BAUDS 38400L         //Baudios
#define vRef 1.1                    //Voltaje interno de 1.1V
#define Kp 1.7                      //Ganancia del controlador proporcional
#define TAM 250

char rs232inputbuf[MYRS232BUFSIZE];     //Buffer de recepcion de datos
RS232InputReport rs232report;           //Informacion de los resultados
char rs232sentcommand[MYRS232SENDSIZE]; //Buffer de envio de datos

const int16_t REFERENCIA = (0.3*1023)/1.1; //Voltaje referencía del sistema (lo convertimos en un valor digitial) -> 279

volatile uint16_t adc0_Valor = 0;    //Valor leído del ADC0
volatile uint16_t adc1_Valor = 0;    //Valor leído del ADC1
volatile int16_t U = 0;              //Señal de control -> U puede alcanzar valores que necesitan más de 8 bits para ser presentandados

volatile uint16_t vec_adc[TAM];      //Array ADC
//volatile int16_t vec_U[TAM];       //Array señal de control con signo
volatile int16_t vec_error[TAM];     //Array señal de error

volatile char listosEnviar = 0;      //Enviar los datos por el puerto serie
volatile uint8_t indice = 0;         //Indice del array


/* -------------- Funciones --------------- */

/* ADC */
void configADC(){
    ADMUX = (0x01<<REFS1) | (0x01<<REFS0) | (0x01<<ADLAR); // Voltaje interno 1,1V y justificación a la izquierda

    ADCSRA = (0x01<<ADEN) | (0x01<<ADPS2) | (0x01<<ADPS1) | (0x01<<ADPS0); //Habilito ADC + divisor de reloj en 128
}
```

```
52  uint16_t leerADC(uint8_t canal){
53      ADMUX &= 0xF0; //Limpiamos los bits de la seleccion de canal
54
55      //Seleccionamos el canal (0 o 1)
56      ADMUX |= canal;
57
58      //Comenzamos la conversión
59      ADCSRA |= (0x01<<ADSC); //Hacemos una OR (|) para solo poner un 1 en el bit que queremos en caso de que este a 0
60
61      //Esperamos a que termine la conversión
62      while(!(ADCSRA & (0x01<<ADIF))); //Eperamos que ADIF sea 1
63
64      //Para limpiar el bit ADIF para la próxima conversión hay que escribir un uno lógico
65      ADCSRA |= (1<<ADIF);
66
67      //El resultado esta almacenado en el registro ADCH y ADCL
68      uint16_t adcValue = (ADCL>>6);
69      adcValue |= (ADCH<<2);  //Valor de ADC de 10 bits -> Voltaje rango 0-1023
70
71      return adcValue;
72  }
73
74
75  /* Motor */
76  void configPWM(){
77      //PWM de 100KHz, modo Phase Correct con Top=OCRA
78      DDRD |= (0x01<<DDD5);  //Pin 5 configurado como Salida para la PWM
79      TCCR0A |= (1<<COM0B1); //No Invertido
80      TCCR0A |= (1<<WGM00);  //Phase Correct con TOP = OCRA
81      TCCR0B |= (1<<WGM02);
82  }
83
84  void iniPWM(){
85      TCCR0B |= (1<<CS00);    //Sin preescalado
86      OCR0A = 80;             //80 -> Para no llegar a 255 y tener una PWM de 100KHz
87      OCR0B = 20;             //20 -> Duty 25% -> Voltaje 6.8V (Teorico 3V)
88  }
89
90  void finPWM(){
91      TCCR0B &=~ ((0x01<<CS02) | (0x01<<CS01) | (0x01<<CS00)); //Apagar PWM
92  }
93
94
95  /* Timer_1 */
96  void configTimer1(){
97      //Tiempo de muestreo -> Si o si al timer 1 que es de 16 bits
98      TCCR1B |= (0x01<<WGM12) | (0x01<<CS12);     //CTC con TOP = OCR1A + Preescalado = 256
99      OCR1A = 625;                                //Periodo de 0.5s = 31250 | 0.1s = 6250 | 0.01s = 625
100     TIMSK1 |= (1 << OCIE1A);                    //Habilitar flag de interrupcion por comparación con OCR1A
```

```
}


 /* Auxiliares */
void sentido_giro(char adc){
    /*Dependiendo de que placa reciba más luz nos movemos en un sentido u otro*/
    if(adc == 0)
    {
        PORTD |= (0x01<<PORTD7);
        PORTB &= ~(0x01<<PORTD0);
    }else{
        PORTD &= ~(0x01<<PORTD7);
        PORTB |= (0x01<<PORTD0);
    }
}


 /* -------------- Interrupciones --------------- */

 /* ISR Timer_1 */
ISR(TIMER1_COMPA_vect){
    /*
    *   El código  de esta ISR es el cambio más notorio que vamos haciendo según el experimento que estemos realizando.
    */
}


 /* -------------- main --------------- */

int main(void)
{
    cli();
    configADC();
    configPWM();
    configTimer1();
    iniPWM();
    RS232_Init(rs232inputbuf,MYRS232BUFSIZE,MYRS232ENDCHAR,MYRS232BAUDS);
    sei();

    //Usamos el Pin 7 y el Pin 8 para controlar en el driver de motor los pines IN3 e IN4 respectivamente, con esto controlamos el sentido de giro del motor.
    DDRD |= (0x01<<DDD7); //Pin 7 de salida
    DDRB |= (0x01<<DDB0); //Pin 8 de salida

    /*Encendemos o apagamos los pines 7 y 8 para controlar el sentido de giro.

        7 - 8
        0 - 0 -> Apagado
        1 - 0 -> Izquierda
        0 - 1 -> Izquierda
        1 - 1 -> CORTOCIRCUITO
```

```
152
153     char cREF[10];
154     char cADC[16];
155     char cError[16];
156     //char cU[16];
157     char cKp[16];
158
159     //Mandamos el valor de referencia por el puerto serie una vez.
160     snprintf(cREF, sizeof(cREF), "%d", REFERENCIA);
161     snprintf(rs232sentcommand, MYRS232SENDSIZE, "%s\r\n", cREF);
162     RS232_Send(rs232sentcommand, 0);
163
164     dtostrf(Kp, 1, 2, cKp);
165     //snprintf(cREF, sizeof(cREF), "%f", REFERENCIA);
166     snprintf(rs232sentcommand, MYRS232SENDSIZE, "%s\r\n", cKp);
167     RS232_Send(rs232sentcommand, 0);
168
169     /* Replace with your application code */
170     while (1)
171     {
172         if(listosEnviar) {
173
174             /*Enviar luz (ADC), error y control (signo) */
175
176             for(volatile uint8_t i = 0; i<TAM; i++)
177             {
178                 snprintf(cADC, sizeof(cADC), "%d", vec_adc[i]);
179                 snprintf(cError, sizeof(cError), "%d", vec_error[i]);
180                 //snprintf(cU, sizeof(cU), "%d", vec_U[i]);
181
182
183                 snprintf(rs232sentcommand, MYRS232SENDSIZE, "%s %s\r\n", cADC, cError/*, cU*/);
184                 RS232_Send(rs232sentcommand, 0);
185             }
186
187             //Ya se han enviado los datos
188             listosEnviar = 0;
189             //OCR0B = 20;       //Reanudamos el motor
190         }
191
192     }
193
194     return 0;
195 }
```

Figura 61. Software desarrollado durante el proyecto Arduino.

En la Figura 61 mostramos el código fuente que se ha usado para programar el Arduino. Concretamente este es el código del controlador P y se ha eliminado el código de la ISR del *timer 1,* con el objetivo de que se vea todo aquello que se ha ido realizando en los experimentos y no se había mencionado anteriormente en la memoria.

```
serial = serialport("COM5", 38400);

%Establecer el tiempo de espera (timeout) del puerto serie
%serial.Timeout = 150;

% Leer la referencia una vez
linea = readline(serial);
referencia = sscanf(linea, '%d');

linea = readline(serial);
Kp = sscanf(linea, '%f');

%Abrir el fichero para almacenar los datos
fichero = fopen('ControlP.txt', 'w');

%Numero de datos que quiero leer
num_lecturas = 3000;

while num_lecturas > 0
    linea = readline(serial);
    resul = sscanf(linea, '%d');
    %Guardar los pulsos en el archivo correspondiente
    fprintf(fichero, '%d %d\n', resul);
    %Reducir el contador de lecturas
    num_lecturas = num_lecturas - 1;
end

% Cerrar el archivo y el puerto serie
fclose(fichero);
clear serial;
```

Figura 62. Script para guardar datos en un fichero.

Otro fragmento de código que ha sido reutilizado y adaptado entre todos los experimentos que se han ido realizando durante el proyecto es el que se muestra en la Figura 62. Este *script* es el que nos permite ejecutar nuestro código del Arduino e ir almacenando los datos que se mandan por el puerto serie en ficheros o bien, algunos datos muy concretos, en variables para posteriormente poder usarlas más adelante.

Bibliografía

[1] Comparasolar, «Comparasolar,» [En línea]. Available: https://comparasolar.com/blog/solar-tracker-como-funciona/. [Último acceso: 25 07 2024].

[2] LotsA, «instructables,» [En línea]. Available: https://www.instructables.com/Agile-Eye-Solar-Tracker/. [Último acceso: 25 07 2024].

[3] DemetrisEng, «projecthub.arduino,» [En línea]. Available: https://projecthub.arduino.cc/DemetrisEng/solar-tracker-35w-with-dc-motors-de5a85. [Último acceso: 25 07 2024].

[4] Arduino, «Arduino Uno,» [En línea]. Available: https://store.arduino.cc/products/arduino-uno-rev3. [Último acceso: 22 06 2024].

[5] Microchip, «Atmega328p,» [En línea]. Available: https://www.microchip.com/en-us/product/ATMEGA328P. [Último acceso: 22 06 2024].

[6] «iotspace,» [En línea]. Available: https://iotspace.dev/arduino-uno-pinout-und-uebersicht/. [Último acceso: 06 09 2024].

[7] Microchip, «Datasheet Atmega328p,» [En línea]. Available: https://lc.cx/x9zV86. [Último acceso: 22 06 2024].

[8] AVR library, «AVR library,» [En línea]. Available: https://onlinedocs.microchip.com/pr/GUID-317042D4-BCCE-4065-BB05-AC4312DBC2C4-en-US-2/index.html. [Último acceso: 22 06 2024].

[9] Microchip, «Microchip,» [En línea]. Available: https://www.microchip.com/. [Último acceso: 06 09 2024].

[10] Microchip, «Microchip Studio,» [En línea]. Available: https://www.microchip.com/en-us/tools-resources/develop/microchip-studio. [Último acceso: 23 06 2024].

[11] B. S. Dean, «AVR-Dude,» [En línea]. Available: https://github.com/avrdudes/avrdude/. [Último acceso: 23 06 2024].

[12] Microchip, «Getting Started Microchip Studio,» [En línea]. Available: https://onlinedocs.microchip.com/pr/GUID-ECD8A826-B1DA-44FC-BE0B-5A53418A47BD-en-US-12/index.html?GUID-00257F02-E33C-40C3-B324-83DBCC05EC30. [Último acceso: 23 06 2024].

[13] Matlab, «Matlab,» [En línea]. Available: https://es.mathworks.com/help/index.html?s_tid=CRUX_lftnav. [Último acceso: 23 06 2024].

[14] Matlab, «Control System Toolbox,» [En línea]. Available: https://es.mathworks.com/help/control/index.html?s_tid=CRUX_lftnav. [Último acceso: 23 06 2024].

[15] Matlab, «System Identificacion Toolbox,» [En línea]. Available: https://es.mathworks.com/help/ident/. [Último acceso: 23 06 2024].

[16] Matlab, «rltool,» [En línea]. Available: https://es.mathworks.com/help/control/ug/root-locus-design.html. [Último acceso: 23 06 2024].

[17] L. Llamas, «luisllamas.es,» [En línea]. Available: https://www.luisllamas.es/entradas-analogicas-en-arduino/. [Último acceso: 08 08 2024].

[18] DigiKey, «DigiKey Placa Solar,» [En línea]. Available: https://www.digikey.es/es/products/detail/panasonic-bsg/AM-5904CAR-DGK-T/2165197. [Último acceso: 27 06 2024].

[19] Panasonic, «Datasheet Placa Solar,» [En línea]. Available: https://panasonic.net/electricworks/amorton/assets/pdf/Brochures_Amorton_E_2.pdf. [Último acceso: 27 06 2024].

[20] Polulu, «Datasheet Motor DC,» [En línea]. Available: https://www.pololu.com/file/0J1736/pololu-37d-metal-gearmotors-rev-1-2.pdf. [Último acceso: 28 06 2024].

[21] DigiKey, «DigiKey L298N,» [En línea]. Available: https://www.digikey.es/es/products/detail/stmicroelectronics/L298N/585918. [Último acceso: 28 06 2024].

[22] Bricogeek, «Bricogeek Polea Motor,» [En línea]. Available: https://tienda.bricogeek.com/impresion-3d-accesorios/1212-polea-gt2-20-dientes-5mm.html?search_query=correa&results=4. [Último acceso: 28 06 2024].

[23] BricoGeek, «BricoGeek Polea Placas,» [En línea]. Available: https://tienda.bricogeek.com/impresion-3d-accesorios/1207-polea-loca-gt2-20-dientes-3mm.html?search_query=correa&results=4. [Último acceso: 28 06 2024].

[24] BricoGeek, «BricoGeek Correa,» [En línea]. Available: https://tienda.bricogeek.com/impresion-3d-accesorios/1213-correa-dentada-gt2-

6mm-5-metros.html?search_query=correa&results=4. [Último acceso: 28 06 2024].

[25] Bricogeek, «Bricogeek Anillo,» [En línea]. Available: https://tienda.bricogeek.com/cables/437-anillo-de-cables-deslizante-22mm.html. [Último acceso: 28 06 2024].

[26] MathWorks, «MathWorks Solar Cell,» [En línea]. Available: https://es.mathworks.com/help/sps/ref/solarcell.html. [Último acceso: 08 08 2024].

[27] J. A. F. Madrigal, «C library for RS232 communications in the ATmega328P,» [En línea]. Available: https://babel.isa.uma.es/jafma/index.php/2015/06/29/c-library-for-rs232-communications-in-the-atmega328p/. [Último acceso: 08 08 2024].

[28] math.stackexchange, «math stackexchange Regresion lineal,» [En línea]. Available: https://math.stackexchange.com/questions/3297060/linear-regression-without-intercept-formula-for-slope. [Último acceso: 01 07 2024].

[29] Matlab, «Matlab Regresion Lineal,» [En línea]. Available: https://es.mathworks.com/matlabcentral/answers/230107-how-to-force-the-intercept-of-a-regression-line-to-zero. [Último acceso: 01 07 2024].

[30] BricoGeek, «BricoGeek Motor DC,» [En línea]. Available: https://tienda.bricogeek.com/motores-dc/1443-motor-con-reductora-1501-con-encoder.html?search_query=encode&results=27. [Último acceso: 28 06 2024].

[31] Mikroe, «Mikroe Proceso Compilacion,» [En línea]. Available: https://www.mikroe.com/ebooks/microcontroladores-pic-programacion-en-c-con-ejemplos/caracteristicas-principales-del-mikroc. [Último acceso: 23 06 2024].

[32] Polulu, «Polulu Motor DC,» [En línea]. Available: https://www.pololu.com/product/2828. [Último acceso: 28 06 2024].

[33] «Diagrama de bloques,» [En línea]. Available: https://ocw.ehu.eus/file.php/83/cap9_html/cap915x.png. [Último acceso: 23 06 2024].

[34] laboratoriogluon, «Driver L298N,» [En línea]. Available: https://www.laboratoriogluon.com/motores-dc-y-arduino/. [Último acceso: 28 06 2024].

[35] rwb, «ermicro,» [En línea]. Available: https://www.ermicro.com/blog/?p=1971. [Último acceso: 10 08 2024].