Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

Grado en Ingeniería Electrónica y Automática

Control digital de luminosidad de una estancia usando Arduino

Autor:

Les Fuente, Rubén

Tutor(es):

García Ruiz, Francisco Javier

Ingeniería de Sistemas y Automática

Valladolid, 17 de Enero - 2024.

Contenido

Lista de Ilustraciones

1. Introducción

El control de la luz en tiempo real es una tecnología que permite ajustar la intensidad de la iluminación de forma dinámica y automática en función de diversas variables. Sus aplicaciones abarcan desde la eficiencia energética y la automatización residencial hasta la seguridad y el entretenimiento. Permite reducir el consumo de electricidad al adaptar la iluminación a la luz natural disponible o la actual en una estancia y a la presencia de personas, crea sistemas de iluminación inteligentes que se adaptan a las necesidades de los residentes, proporciona una iluminación óptima para mejorar la productividad y la comodidad en entornos comerciales y de oficina, garantiza que las áreas críticas estén bien iluminadas para una vigilancia efectiva, optimiza el crecimiento de los cultivos al controlar la iluminación artificial en la agricultura de invernadero, crea efectos de iluminación espectaculares en eventos y actuaciones en vivo, y ajusta automáticamente los faros de los vehículos para mejorar la seguridad y evitar deslumbramientos. En resumen, la regulación de luz en tiempo real mejora la eficiencia, el confort y la seguridad en una variedad de aplicaciones, al adaptar la iluminación de manera dinámica a las condiciones cambiantes.

1.1. Motivación para la realización del proyecto

La principal motivación de este proyecto es la adquisición de nuevos conocimientos en la plataforma de Arduino siendo uno de los microcontroladores más importantes del mundo utilizado para múltiples dispositivos y labores. A su vez este proyecto se enfocará a la docencia facilitando el aprendizaje de los controladores PID, efecto Windup y programación de un sistema.

De cara a futuro es importante conocer cómo trabajar con los microcontroladores y gracias al entorno que proporciona Arduino podemos identificar a la perfección el funcionamiento de nuestro sistema observando el valor numérico obtenido en cada situación y poder consultar una gráfica que el propio entorno nos muestra.

En este proyecto se abordan diversos ámbitos que han sido objeto de estudio a lo largo de la carrera. Se hace uso de la electrónica, empleando componentes como un LED, un sensor de iluminación y resistencias, además de llevar a cabo la interconexión de estos elementos con el microcontrolador de Arduino. Asimismo, se ha enfocado en el desarrollo del código y la configuración específica para el mencionado microcontrolador, lo que involucra habilidades de programación.

Este proyecto aporta también aspectos relacionados con el campo de la automatización, ya que se ha diseñado un controlador PID con el fin de garantizar un óptimo funcionamiento de nuestro sistema.

Otra de las motivaciones fundamentales para este proyecto radica en su relevancia como sistema de prácticas para la enseñanza en el ámbito de la automática. Específicamente, destaca por ser un entorno propicio para el desarrollo y estudio de controladores PID discretos, así como para la exploración detallada del fenómeno conocido como "Windup". Este proyecto contribuye significativamente a la formación académica en el campo de la ingeniería, proporcionando a los estudiantes una valiosa demostración práctica en el diseño y análisis de sistemas de control de los controladores y del efecto más común producido por la componente integral.

2. Objetivos

A continuación, se muestran los principales objetivos que se pretenden cumplir en el desarrollo de este proyecto:

1. **Diseñar y Construir el Sistema:** Desarrollar un sistema de control que regule la luminosidad dentro de un espacio cerrado (caja) utilizando Arduino como controlador digital.
2. **Efecto de las Perturbaciones:** Realizar un estudio sobre cómo las perturbaciones, como la luz exterior, afectan el rendimiento del sistema de control y cómo pueden controlarse.
3. **Analizar el Efecto WindUp:** Estudiar el fenómeno del efecto windup en el controlador y determinar cómo se puede prevenir o reducir, especialmente en situaciones de saturación y término integral.
4. **Implementar un Controlador PID Eficiente:** Diseñar y ajustar un controlador PID que sea eficiente en la regulación de la luminosidad, teniendo en cuenta la problemática que se presenta.
5. **Realizar Experimentos y Pruebas:** Llevar a cabo pruebas en el entorno deseado para evaluar el desempeño del sistema en diferentes situaciones.
6. **Optimizar el Sistema:** Refinar el sistema de control y realizar ajustes en los parámetros del controlador PID para lograr un control preciso y estable de la luminosidad.
7. **Documentar y Presentar los Resultados:** Realizar un informe detallado de cómo se ha logrado la solución del problema.
8. **Proponer Mejoras y Recomendaciones:** Identificar posibles mejoras en el sistema y ofrecer recomendaciones para futuras implementaciones.

2.1. Descripción del prototipo a desarrollar

El proyecto consta de un microcontrolador Arduino, al cual se le conectará un LED y un LDR. Esto permitirá la medición del nivel de iluminación del LED y la regulación automática basada en la información de luminosidad recibida por el LDR. Para asegurar que se controle exclusivamente la iluminación generada por el LED y no la luz externa, aunque se analizarán perturbaciones, todo el montaje se alojará en una caja cerrada.

Con el propósito de garantizar el rendimiento óptimo del sistema, se implementará un controlador PID que permitirá una regulación precisa en diversas condiciones. Además, se desarrollarán funciones específicas para garantizar un funcionamiento libre de ruido y la eficiencia del sistema.

La incorporación de un mecanismo de Anti-Windup evitará la acumulación de errores y garantizará un funcionamiento sin contratiempos provocados por los posibles casos de saturación del elemento actuador y la acción integral del PID.

Para implementar el controlador PID de manera efectiva, se realizará un cálculo teórico de los valores aplicables al sistema. Para verificar el funcionamiento correcto del controlador calculado teóricamente, se programarán las funciones correspondientes en MATLAB con el fin de confirmar los resultados obtenidos.

Una vez alcanzados los objetivos mencionados, se procederá a diseñar una PCB en Proteus para lograr un dispositivo compacto y cómodo, que pueda ser instalado en diversas ubicaciones con facilidad.

3. Sistemas de control

Un sistema de control se define como el conjunto de elementos que trabajan en conjunto para lograr un funcionamiento concreto.(*Sistema de control - Wikipedia, la enciclopedia libre*, s. f.)

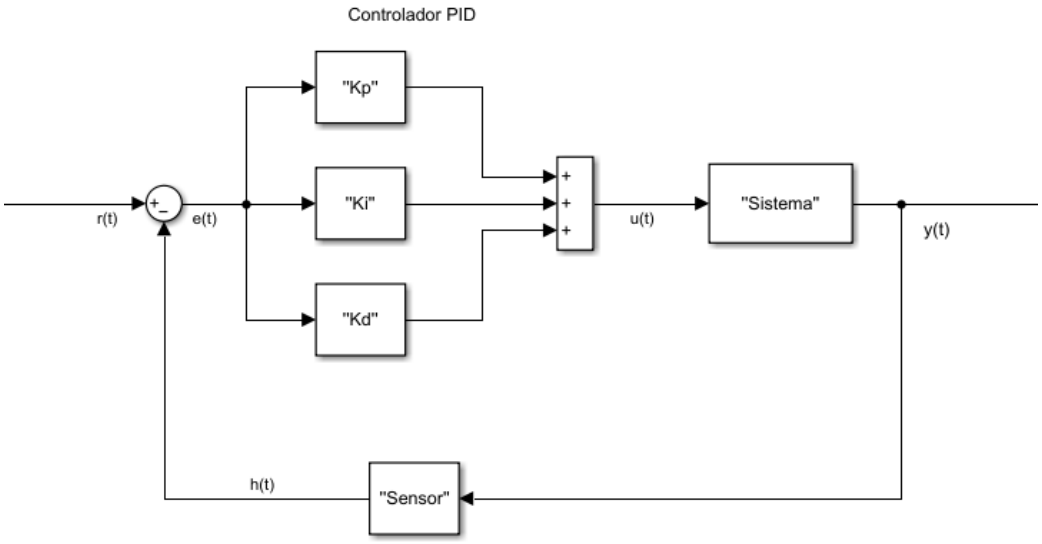

Ilustración 1. Diagrama Sistema de Control.

Un sistema de control está compuesto por diferentes elementos, los cuales son:

- **Sensor:** Dispositivos que detectan y convierten la información para ser tratada. En nuestro proyecto será el LDR, el cual mide el nivel de luz recibido.
- **Proceso o sistema:** Es el sistema que se está controlando. En este proyecto será la iluminación del LED.
- **Controlador:** Es el elemento que procesa la información y toma las decisiones oportunas para modificar el comportamiento del sistema y obtener el resultado buscado. Se crea un controlador PID en este proyecto con el objetivo de lograrlo.

- **Actuador:** El LED es el actuador en este sistema. Su brillo se ajusta en función de las decisiones tomadas por el controlador PID.
- **Señales de Entrada:** Representan las variables que afectan al sistema. Son las lecturas del LDR que indican la cantidad de luz medida.
- **Señales de Salida:** Son variables que se controlan y se ajustan. Es la intensidad luminosa del LED, que se ajusta para seguir las indicaciones del controlador PID.
- **Referencia (Setpoint):** Es el valor deseado por la variable controlada. El setpoint podría ser un valor deseado para la intensidad luminosa del LED. El controlador PID trabaja para que la salida del LED coincida con este valor de referencia.
- **Error:** El error se calcula como la diferencia entre la señal de referencia (setpoint) y la señal de salida del sistema (lectura actual del LDR o brillo actual del LED). El controlador utiliza el valor del error para tomar decisiones y ajustar el sistema.

(*¿Qué es un sistema de control? - AUTYCOM*, s. f.)

3.1. Controlador PID

3.1.1. Definición

Dispositivo de control que busca mantener una variable específica a un valor deseado. El controlador actúa sobre el error, el cual se define como la diferencia entre la referencia deseada por el usuario y la salida. Este controlador está constituido por la suma de tres controladores que pueden funcionar independientemente denominados Proporcional, indicado con una P; Integral, indicado con una I; y Derivativo, indicado con una D.

(▷ *Como Hacer un Control PID Arduino de Temperatura*, s. f.-a)

A continuación, se realizará una descripción detallada de cada componente.

3.1.2. Acción Proporcional

Como su nombre indica el controlador es proporcional al error, por lo tanto, el controlador actuará con más fuerza si el error aumenta. Su expresión es la siguiente.

$$P(t) = K_p \cdot e(t) \quad (1)$$

Siendo K_p la ganancia proporcional y $e(t)$ el error obtenido.

El error obtenido se calcula como la diferencia entre el valor de la salida y la consigna. El valor de K_p se calculará teóricamente en lazo abierto para más tarde ajustarlo al funcionamiento del sistema.

3.1.3. Acción Integral

Se utiliza para eliminar el error acumulado en el sistema y su expresión es la siguiente.

$$I(t) = K_i \cdot \int_0^t e(\tau)\, d\tau \text{ (2)}$$

Siendo K_i la constante integral y $e(\tau)$ es el error del cual se hace la integral en el tiempo para obtener el error acumulado.

La parte integral por lo tanto se calculará como el error obtenido más el valor integral en el instante anterior.

El valor de K_i se calculará teóricamente en lazo abierto para más tarde ajustarlo al funcionamiento del sistema.

3.1.4. Acción Derivativa

Este término anticipa y contrarresta las tendencias futuras del error, basándose en la velocidad con la que el error está cambiando. Este término mejor la respuesta transitoria del sistema.

$$D(t) = K_d \cdot \frac{de(t)}{dt} \text{ (3)}$$

Siendo K_d la constante proporcional que multiplica a la derivada del error. Dicho valor actuará solo en el caso de ser un error constante, de lo contrario únicamente actuará la acción proporcional e integral.

El cálculo de $\frac{de(t)}{dt}$ se obtiene mediante la diferencia entre el error obtenido menos el error obtenido en el instante anterior.

3.1.5. Setpoint y Error

El Setpoint o también llamado Consigna (SP o r(t)), es el valor que se quiere alcanzar o a la que se pretende que el sistema se mantenga. Puede ser tanto una posición, una velocidad, una temperatura, una intensidad de luz o incluso un simple valor numérico que se quiere mantener constante. Generalmente se pretende obtener un valor constante pero el valor del Setpoint podría ser variable.

El Error (e(t)) es la diferencia entre el Setpoint y la variable que se está controlando.

Por lo tanto, el error que se está produciendo actualmente se puede calcular como la diferencia entre el Setpoint (SP o r(t)) y el valor actual en el proceso o variable controlada (PV o y(t)).

$$e(t) = r(t) - y(t) \text{ (4)}$$

El objetivo del PID es reducir el error a un valor lo más próximo a cero y por ello se tiene que conseguir que la variable controlada se aproxime lo más posible al valor deseado Setpoint. El controlador PID utiliza sus componentes para ajustar el valor de la salida y conseguir el menor error posible.

(*Proportional–integral–derivative controller - Wikipedia*, s. f.)

3.1.6. Modelo matemático completo PID.

La combinación de estos tres términos se denomina PID y permite un ajuste de la salida del sistema para tener control sobre esta y reducir el error.

La ley de control PID en continuo se representa con una ecuación la cual se muestra a continuación.

$$y(t) = K_p \cdot e(t) + K_i \int_0^t e(t)\,dt + K_d \cdot \frac{de}{dt} \quad (5)$$

La función de transferencia del controlador se escribe de la siguiente manera.

$$PID = \frac{U(s)}{E(s)} = K_p \left(1 + \frac{1}{T_i s} + T_d s\right) \quad (6)$$

Un aumento de la constante proporcional por lo tanto provocará una respuesta más rápida, pero provocando mayor oscilación lo cual se relaciona con la inestabilidad. Por mucho que se aumente dicha constante, es imposible eliminar el error estacionario.

Si se pretende eliminar el error estacionario se introduce la constante integral. A mayor valor, más se reducirá el valor del error, pero aumentando la oscilación y a su vez la inestabilidad.

Por último, añadir la constante derivativa mejora el comportamiento general, pero se puede provocar un funcionamiento extraño y gran sensibilidad al ruido junto con grandes cambios bruscos en el error.

3.1.7. Método de sintonización del controlador PID.

La sintonización de un controlador PID es un proceso mediante el cual nos permite ajustar los parámetros del controlador para obtener el mejor rendimiento del sistema y lograr una respuesta rápida y lo más estable posible.

A continuación, se describen los dos métodos más comunes dependiendo de las situaciones que se encuentran.

3.1.7.1. Ziegler y Nichols

Lazo abierto

Este método se realiza en lazo abierto, aunque también podría realizarse en lazo cerrado. Se crea una entrada de tipo escalón y a partir de la respuesta del sistema se pueden obtener los parámetros del controlador PID.

Para poder aplicar este sistema la salida debe tener una forma como la que se muestra en la siguiente ilustración número 2.

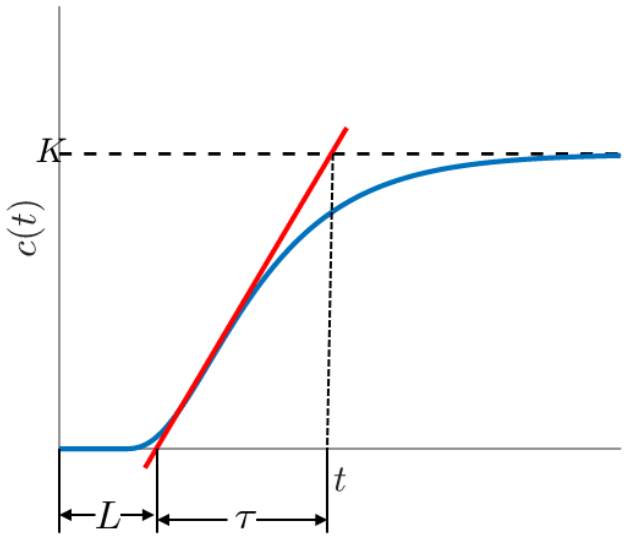

Ilustración 2. Forma de la función que debe tener la salida del sistema.

Debe tener forma de S, sino no se va a poder utilizar dicho método. El sistema debe tener un retardo en el tiempo y una constante de tiempo, dicha función de transferencia se representa de la siguiente manera.

$$G_p(s) = \frac{C(s)}{U(s)} = \frac{Ke^{-Ls}}{\tau s+1} \quad (7)$$

A partir de la información proporcionada por la función de transferencia, se pueden elegir los valores de K_p, T_i y T_d utilizando la tabla de sintonía de Ziegler y Nichols.

Controlador	K_p	T_i	T_d
P	$\frac{T}{KL}$	∞	0
PI	$0.9\frac{T}{KL}$	$\frac{T}{0.3}$	0
PID	$1.2\frac{T}{KL}$	$2L$	0.5L

Ilustración 3. Tabla Ziegler-Nichols lazo abierto.

Para obtener la función de transferencia en lazo abierto controlada se multiplica el controlador PID (5) con la función de transferencia en lazo abierto (6).

Desarrollando se pueden despejar las diferentes componentes que forman el controlador PID que se busca.

Lazo cerrado

Para realizarlo en lazo cerrado no es necesario retirar el controlador PID. Para este método los valores del PID asignados tienen que ser bajos o incluso nulos.

Se asigna un valor del setpoint al deseado y se comienza a incrementar el valor de K_p hasta que el sistema presente oscilaciones sostenidas, es decir, oscilaciones continuas y estables.

En el caso de no presentar oscilaciones sostenidas, entonces este método no se podrá aplicar.

Observando el valor de la K crítica (K_c) cuando se encuentra en el límite de la estabilidad junto con el periodo crítico (P_c), y procediendo de la misma manera que en lazo abierto, podremos obtener los valores a partir de la tabla de Ziegler y Nichols.

El periodo crítico puede calcularse de la siguiente manera.

$$P_c = \frac{2\pi}{\omega_c} \quad (8)$$

Controlador	K_p	T_i	T_d
P	$0.5K_c$	∞	0
PI	$0.45K_c$	$\frac{P_c}{1.2}$	0
PID	$0.6K_c$	$0.5P_c$	$0.125P_c$

Ilustración 4. Tabla Ziegler-Nichols lazo cerrado.

(▷ *Todo sobre Ziegler Nichols - Sintonia de Control PID*, s. f.)

3.1.7.2. Control PI Asignación de Polos

Este concepto de sintonización se basa en la ubicación de los polos del sistema en posiciones del plano complejo para lograr una salida deseada en lazo cerrado.

(▷ *Como Hacer un Control PID Arduino de Temperatura*, s. f.-b)

El objetivo principal es manipular el sistema forzando que los polos del sistema se ubiquen en posiciones predeterminadas ajustando los diferentes parámetros del controlador.

Existen dos posibles casos, que el sistema sea de primer orden, o de segundo y se tratarán de una manera similar. El sistema que se va a tratar va a ser de primer grado o de segundo grado el cual se puede aproximar a uno de primer orden. Por lo tanto, la explicación de este sistema se va a centrar en el caso de un sistema de primer orden.

Se plantea una función de transferencia de primer grado que cuenta con una representación como la siguiente.

$$G(s) = \frac{k_p}{\tau s + 1} \quad (9)$$

Donde k_p es la ganancia estática del proceso y τ es la constante de tiempo del proceso.

Por otro lado, el controlador PID tiene una forma como la que se representa a continuación. Es importante conocer las dos funciones de transferencia porque son las que van a formar la función de transferencia final que dará la respuesta a nuestro sistema controlado.

$$PID = C(s) = k_c \frac{\tau_i s+1}{\tau_i s} \text{ (10)}$$

Donde k_c es la ganancia proporcional del controlador y τ_i es el tiempo integral de la acción integral del controlador.

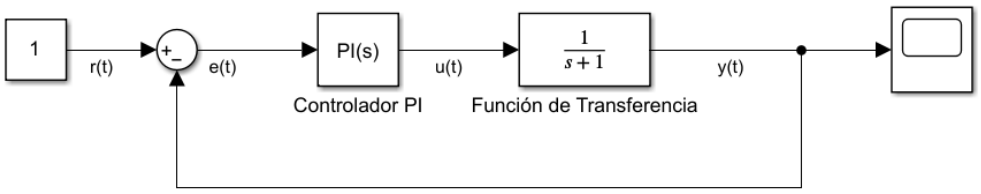

Ilustración 5. Diagrama de un sistema en lazo cerrado.

Siguiendo la forma que tiene el sistema de la planta en lazo cerrado como en la ilustración anterior, cerramos el lazo y tendremos una función de transferencia Y(s) (10).

$$Y(s) = \frac{C(s)G(s)}{1+C(s)G(s)} \text{ (11)}$$

Desarrollando la ecuación anterior, y sustituyendo las ecuaciones anteriores, obtenemos la siguiente ecuación resultante.

$$Y(s) = \frac{\frac{k_c k_p}{\tau_i \tau}(\tau_i s+1)}{s^2+\frac{1}{\tau}(1+k_c k_p)s+\frac{k_c k_p}{\tau_i \tau}} \text{ (12)}$$

El desarrollo anterior muestra el denominador de la función de transferencia en lazo cerrado, y el denominador en concreto define la ecuación característica, define la dinámica del sistema.

En los sistemas de segundo orden, es conocido que la función de transferencia correspondiente es la siguiente.

$$F(s) = \frac{K{w_n}^2}{s^2+2\delta w_n+{w_n}^2} \text{ (13)}$$

Donde δ es el factor de amortiguamiento y w_n es la frecuencia natural.

Por lo tanto, conociendo dos denominadores característicos del sistema de segundo orden, se igualarán los de Y(s) y F(s) para conseguir obtener los valores del controlador PID que se buscan.

Necesitaremos los valores del factor de amortiguamiento δ y de la frecuencia natural w_n. Para ello propondremos un factor de amortiguamiento crítico o sobreamortiguado, esto quiere decir que sea igual o mayor que la unidad puesto que se podrá ajustar mejor al sistema que se quiere tratar en este trabajo y la frecuencia natural se ajustará entre 0 y 1. Más adelante se mostrarán los resultados obtenidos.

(*CONTROLADOR PI - Asignación de Polos [FÁCIL - Aprende]*, s. f.)

3.2. Efecto Windup y Anti-Windup

3.2.1. Efecto Windup

Debido al uso de un controlador con acción integral, como el que se implementa en este proyecto, puede ocurrir un mal comportamiento del sistema debido a la acumulación de error integral que se produce en los cálculos de la componente integral.

El funcionamiento no deseado comienza cuando el sistema ya ha llegado a saturación, es decir, que la salida ha alcanzado ya el valor máximo o mínimo en el sistema, pero la acción integral continúa acumulando valor en el error y comienza a aumentar hasta un valor muy grande que cuando el sistema intenta compensarlo, tarda un tiempo tan grande que se produce un retardo en el funcionamiento del sistema.

Para evitar esta acumulación y que el sistema funcione correctamente se tiene que incorporar un mecanismo de control llamado Anti-Windup.

3.2.2. Anti-Windup

Este método es un mecanismo que pretende evitar que la acción integral del sistema continue creciendo cuando el sistema ya ha llegado a la saturación.

Este mecanismo se utiliza para evitar retardos en el funcionamiento del sistema.

Debido a que el proyecto que se lleva a cabo se realiza en Arduino, el método de implementación del Anti-Windup se va a realizar mediante código de programación.

Para implementar este mecanismo simplemente dejamos de permitir la adición del error integral acumulado cuando se ha llegado a los límites del valor de entrada, en nuestro caso, el máximo de iluminación del LED.

```
%PID
u(k) = u(k-1) + q0*e(k) + q1*e(k-1) + q2*e(k-2);
%Anti-Windup
 if (u(k) >= umax)
    u(k) = umax;
 elseif (u(k) <= umin)
     u(k) = umin;
 end
```

El método anterior es el más eficiente para implementar en nuestro proyecto y código, pero, existen otras técnicas como por ejemplo el incremento de control, en este caso si el actuador se satura, el programa no suma el incremento de control.

```
%PID
deltaU(k) =  q0*e(k) + q1*e(k-1) + q2*e(k-2);
%Anti-Windup
if (u(k-1)+deltaU(k) >= umax || u(k-1)+deltaU(k) <= umin)
   deltaU(k) = 0;
end
%Acción de Control
u(k) = u(k-1)+deltaU(k);
```

Este efecto es muy común en sistemas controlados y se va a mostrar gráficamente en el apartado número cinco donde se demuestra experimental cual es el efecto que se produce y cómo se ha resuelto el problema correctamente.

(▷ *Anti Windup en un Control PID - [Detallado]*, s. f.)

Por último, se puede aplicar el método de descarga de la acción integral. Este método se desarrolla en un entorno de diagrama de bloques como puede ser Simulink donde se realiza una descarga de la acción integradora.

3.3. Filtrado exponencial o Suavizado

El filtrado exponencial o suavizado exponencial consiste en obtener un valor filtrado a partir de una medición utilizando la siguiente expresión. El objetivo es eliminar el gran ruido que se genera cuando el sistema tenga que realizar un gran número de cambios en su referencia y por lo tanto en la salida en poco tiempo.

$$A_n = \alpha M + (1 - \alpha) A_{n-1} \quad (14)$$

Donde A_n es el valor filtrado y A_{n-1}es el valor filtrado anterior. M es el valor muestreado de la señal a filtrar, y α es un factor entre 0 y 1.

El factor de suavizado (α) es un parámetro importante que controla la rapidez con la que el valor suavizado responde a las nuevas lecturas. Un valor pequeño de α proporciona un suavizado más lento y elimina el ruido de alta frecuencia, pero también puede retrasar la respuesta a cambios en los datos. Un valor grande de α proporciona un suavizado más rápido, pero es menos eficaz en la eliminación del ruido de alta frecuencia.

(*Filtro paso bajo y paso alto exponencial (EMA) en Arduino*, s. f.-a)

4. Material necesario y utilizado

4.1. Arduino Uno

Arduino Uno es una plataforma de desarrollo hardware de código abierto, es decir, cualquier persona puede modificar y compartir la tecnología desarrollada, la cual se utiliza para la creación de diferentes proyectos y prototipos haciendo uso de su microcontrolador.

(*Arduino Uno - Wikipedia*, s. f.)

Su microcontrolador ATmega328p es de 8 bits creado por Atmel pertenece a la serie de microcontroladores megaAVR basado en RISC.

Un microcontrolador es un dispositivo integrado, o circuito integrado programable que cuenta con un procesador (unidad central de procesamiento), memoria y periféricos de entrada y salida.

El principal objetivo de un microcontrolador es reducir el coste económico y reducir el consumo de energía, por ese motivo existen diversos microcontroladores destinados a cubrir diferentes necesidades. Estos microchips pueden variar desde los 4 u 8 bits hasta los 32 o 64 bits.

(*Microcontrolador - Wikipedia, la enciclopedia libre*, s. f.)

Los microcontroladores más utilizados son:

- **AVR:** Utilizado en Arduino Uno siendo ATMega328p el microcontrolador. Cuenta con una velocidad de entre 8 MHz y 16 MHz.
- **PIC:** Los más utilizados son el PIC12C508/509 que es un encapsulado reducido de 8 pines utilizado para pequeños diseños o el PIC16F88 que cuenta con oscilador interno, USART, SSP, A/D, etc. La velocidad puede variar desde muy pocos MHz utilizados por el primero o más de 100 MHz.

  (*Microcontrolador PIC - Wikipedia, la enciclopedia libre*, s. f.)
- **ARM Cortex-M:** Los más utilizados son los STM32 fabricados por STMicroelectronics. Las velocidades de estos varían desde los 24 MHz hasta los 480 MHz.

  (*STMicroelectronics - Wikipedia, la enciclopedia libre*, s. f.)
- **ESP:** El más conocido es el ESP32 fabricado por Espressif Systems. La velocidad puede llegar hasta los 200 MHz.

  (*ESP32 - Wikipedia, la enciclopedia libre*, s. f.)

A parte de la importancia que tiene la velocidad de procesamiento es muy importante tener en cuenta diferentes factores como puede ser la cantidad de pines, tamaño de memoria, compatibilidades de periféricos, etc.

Las aplicaciones más comunes para las que se aplican los microcontroladores son las siguientes.

- **Electrónica de hogar:** Utilizado en dispositivos como son los electrodomésticos, frigoríficos, lavavajillas, lavadoras, etc.
- **Automoción:** Se utilizan para controlar cualquier sistema electrónico en un vehículo.
- **Sistemas embebidos:** Aparecen en dispositivos aplicados a la medicina, electrodomésticos, industria, fábricas, dispositivos electrónicos, etc.
- **Robótica:** Múltiples dispositivos para el control del movimiento, sensores para la percepción del entorno, etc.

4.1.1. Especificaciones técnicas

A continuación, se va a crear una tabla recogiendo las especificaciones principales del microcontrolador utilizado en este proyecto. El microcontrolador que incorpora el Arduino Uno se denomina ATmega328 y cuenta con las siguientes especificaciones.

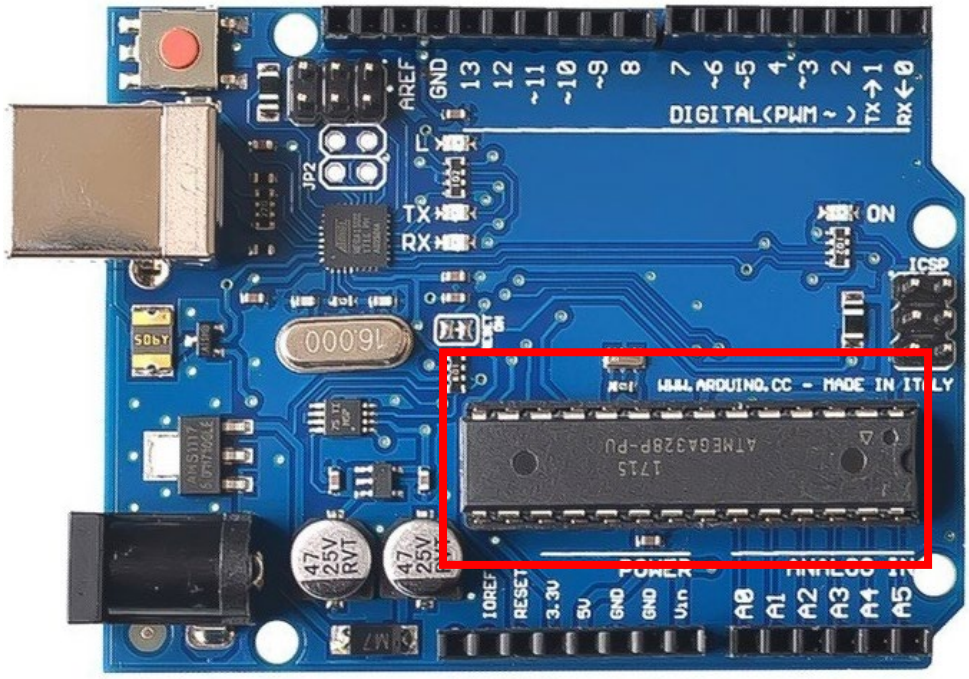

Ilustración 6. Microcontrolador ATmega328 en placa de Arduino UNO.

Velocidad de reloj	16 MHz
Tamaño de memoria Flash	32 Kb (0.5 utilizados por el bootloader)
Tamaño de memoria SRAM	2 Kb
Tamaño de memoria EEPROM	1 Kb
Número de pines digitales	14 (6 de ellos utilizados para salidas PWM)
Número de pines analógicos	6
Número de Timers	2 de 8 bits y 1 de 16 bits
Número de USART	1
Número de SPI	1
Número de I2C	1
Voltaje de alimentación	5V

Ilustración 7. Tabla de características de ATmega328.

Además, el Arduino Uno cuenta con el ATmega16U2 el cual se programa con un firmware específico que su principal función es ser convertidor USB a UART para la comunicación entre el IDE y Arduino. Sus especificaciones técnicas principales son las siguientes.

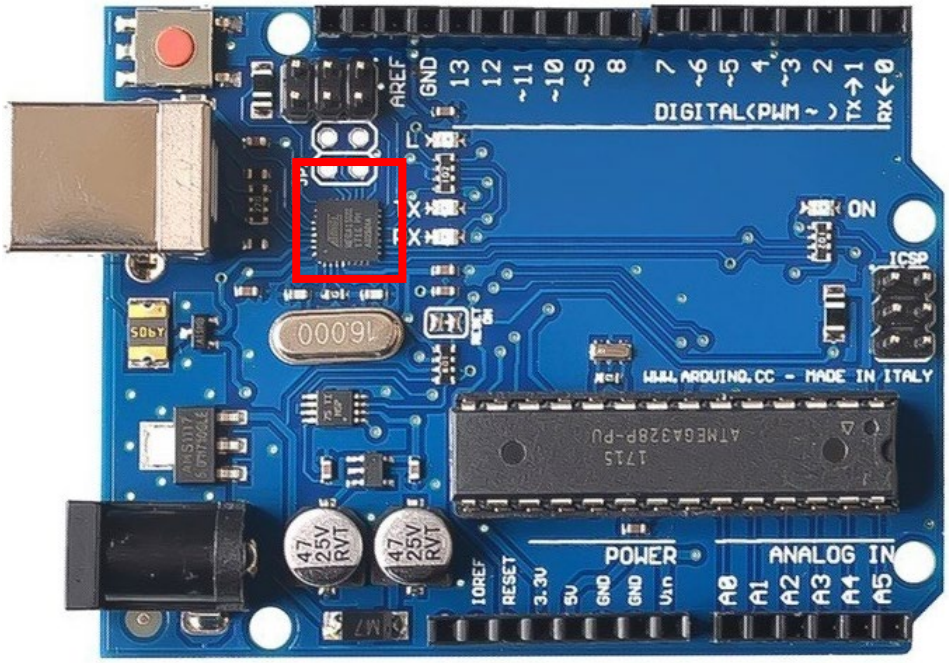

Ilustración 8. Microcontrolador ATmega16U2 en placa de Arduino UNO.

Arquitectura	8 bit.
Tamaño de memoria Flash	16 Kb
Tamaño de memoria SRAM	512 bytes
Tamaño de memoria EEPROM	512 bytes
Número de pines	32 de entrada y salida
Voltaje de operación	2.7 a 5.5 V
Velocidad de reloj	8 MHz
Tecnología	RISC

Ilustración 9. Tabla de características deATmega16U2.

4.1.2. Nociones básicas sobre el uso de Arduino Uno y su IDE

Arduino Uno se alimenta mediante un cable USB o desde una fuente de alimentación entre 6 y 18 voltios. Arduino cuenta con un regulador de voltaje para evitar que el circuito se dañe.

Los componentes que se quieren utilizar se conectar a unos pines mediante cables para poder trabajar más tarde con ellos. Además, cuenta con los siguientes pines de importancia:

- **Reset:** El microcontrolador reinicia el sistema si ocurre algún fallo.
- **5V y 3.3V:** Estas clavijas sirven de alimentación para los diferentes componentes.
- **GND:** Conexión a tierra.
- **Vin:** Se utiliza para conectar la placa con una fuente externa de entre 6 y 12 V.

Arduino cuenta con unos pines analógicos (A0 a A5) capaces de leer una señal analógica o bien transformarla en digital.

(*Arduino: Guía completa para principiantes y expertos | Aprende ya*, s. f.)

Como ayuda para el usuario Arduino cuenta con un LED que se ilumina para indicar que la placa está funcionando correctamente. La conexión UART también utiliza un LED para indicar que se está realizando un intercambio de datos.

La programación de Arduino se realiza mediante un entorno o programa llamado “IDE Arduino” (Integrated Development Environment).

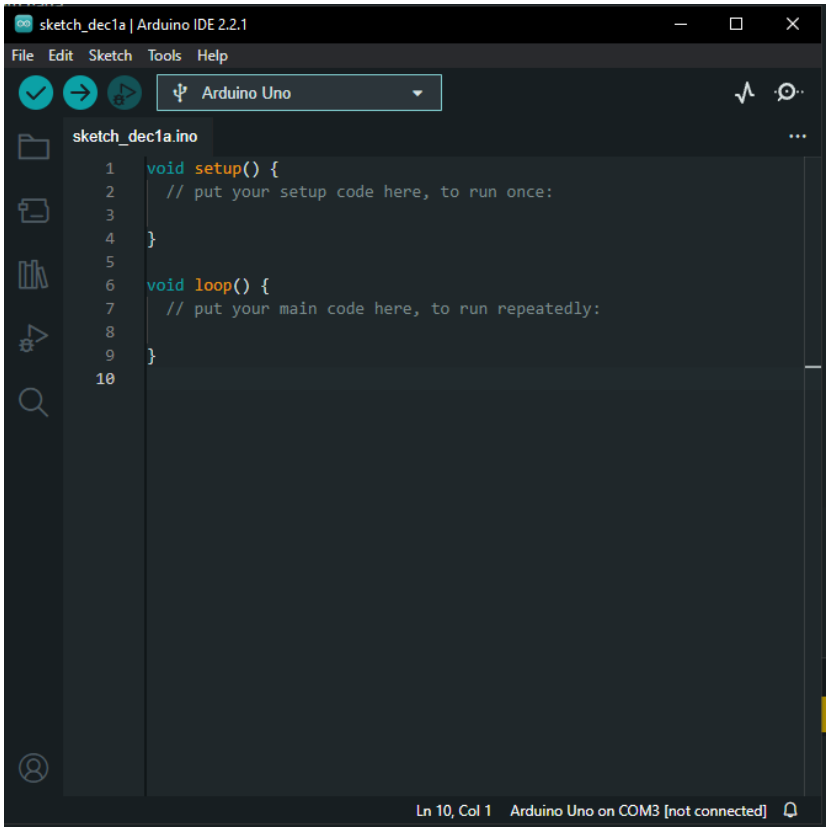

Ilustración 10. Ventana primer programa en Arduino IDE.

(*Tutorial Arduino: IDE Arduino | OpenWebinars*, s. f.)

Para conseguir este programa, se accede en la página oficial de Arduino para descargar el entorno. Con el programa en nuestro poder se puede seleccionar cualquier ejemplo pulsando en File/Examples/ ahí se encuentra uno de los ejemplos de programas más famosos, "Blink" utilizado en diversos microcontroladores.

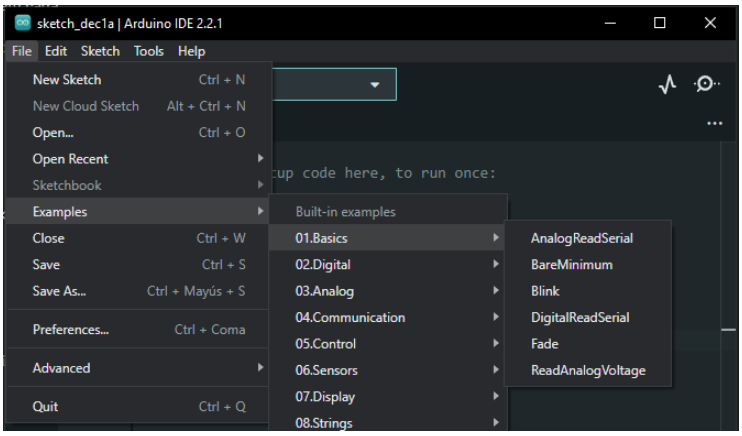

Ilustración 11. Ejemplos de programas en Arduino IDE.

Para conectar Arduino al ordenador es necesario un cable USB tipo B.

Ilustración 12. Cable USB de tipo B.

Con Arduino conectado al PC podremos observar cómo el LED se ilumina indicando dicha conexión.

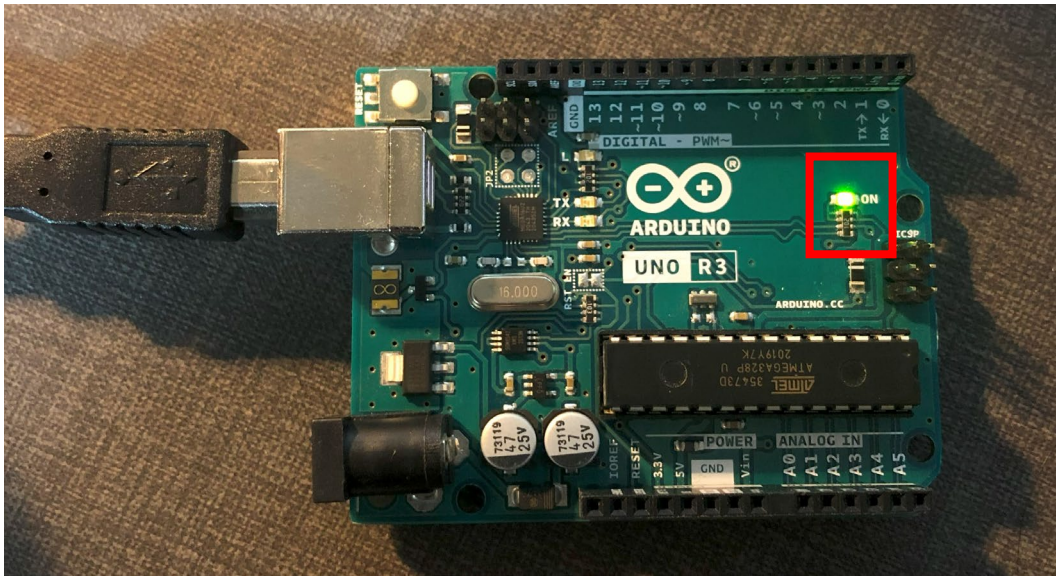

Ilustración 13. LED iluminado indicando conexión en Arduino UNO.

En el IDE seleccionaremos en Tools/Board/ el modelo de Arduino correspondiente, verificando que el puerto serie de la placa coincide con el mostrado por el sistema operativo de nuestro PC.

A continuación, pulsando sobre el botón “Upload”, los LEDs correspondientes a la UART comenzarán a parpadear y una vez finalice la subida del programa a nuestro microcontrolador aparecerá un mensaje “Done uploading” si no existe ningún error de compilación.

Una vez finalizado el proceso, comenzará un LED de color naranja indicando que Arduino tiene cargado el programa y funcionando correctamente.

El lenguaje de programación de Arduino se basa en C y C++ y tiene dos secciones perfectamente divididas. Como se puede ver en la ilustración 10 existe una parte denominada “void setup()” y “void loop()”.

- **Setup().** Esta función se ejecuta una sola vez al iniciarse el programa. Generalmente se utiliza para realizar las configuraciones iniciales como pueden ser la inicialización de los pines de los componentes, la inicialización de la frecuencia de la comunicación serial o cualquier configuración que tenga que inicializarse cuando se ejecuta el programa.
- **Loop().** Esta función se ejecuta una vez ha finalizado setup() y se ejecuta de forma continua en un bucle infinito. Por lo tanto, sirve para que se ejecute repetidamente el conjunto de instrucciones que se quieren realizar para lograr el objetivo propuesto.

Algunas de las funciones más comunes de la librería que incorpora Arduino en su entorno son las siguientes.

- **pinMode(pin, mode):** Configura el modo de un pin como INPUT u OUTPUT indicando exactamente el número de pin.
- **digitalWrite(pin, value):** Establece el valor de un pin, indicando el número correspondiente, como HIGH o LOW en valor digital.
- **digitalRead(pin):** Lee el valor digital de un pin, si es HIGH o LOW.
- **analogWrite(pin, value):** Escribe un valor analógico en un pin PWM (modulación por ancho de pulso).
- **analogRead(pin):** Lee el valor analógico de un pin.
- **delay(ms):** Realiza una pausa en la ejecución del programa durante el tiempo especificado en milisegundos.
- **millis():** Devuelve el valor de milisegundos desde que el programa comenzó a ejecutarse.
- **Serial.begin(baudrate):** Inicia la comunicación serial con una velocidad de baudios especificada. El más común es utilizar 9600.
- **Serial.print(data):** Imprime datos en el puerto serie. Suele utilizarse para mostrar por pantalla los resultados obtenidos.
- **Serial.println(data):** Imprime datos seguidos de un salto de línea en el puerto serie.
- **Random (min, max):** Genera un número aleatorio en el rango especificado.
- **Constrain(valor, límite inferior, límite superior):** Función que sirve para limitar una variable entre dos valores fijos.

(*Arduino Reference - Arduino Reference*, s. f.)

Arduino IDE cuenta con funciones integradas para el análisis y la visualización de datos como son el Serial Monitor y el Serial Plotter:

- **Serial Monitor:** Se utiliza para monitorear la comunicación entre la placa Arduino y el computador conectado. Permite ver mensajes, datos, y cualquier depuración que se envía desde la placa de Arduino.

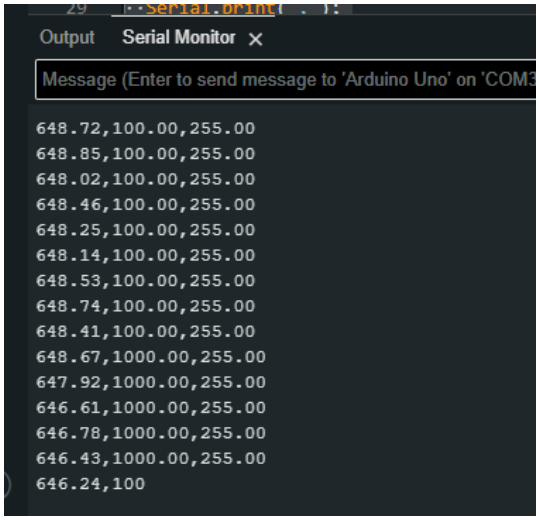

Ilustración 14. Serial Monitor de Arduino IDE.

- **Serial Plotter:** Esta herramienta sirve para visualizar gráficamente los datos enviados desde Arduino a lo largo del tiempo.

Ilustración 15. Serial Plotter de Arduino IDE.

4.2. Diodo LED

Un diodo emisor de luz es un dispositivo formado por un material semiconductor que emite luz cuando circula corriente eléctrica a través de él.

(*Led - Wikipedia, la enciclopedia libre*, s. f.)

Es un diodo de unión PN (Unión de dos cristales, generalmente Silicio y Germanio) que cuando se aplica la tensión adecuada en los terminales, los electrones se introducen en los huecos de la unión PN liberando una energía en forma de fotones. Los LEDs se utilizan en multitud de aplicaciones como, por ejemplo, iluminación, señalización, pantallas, etc.

La gran fortaleza de este componente es la larga vida de funcionamiento y su eficiencia energética.

Para este proyecto se ha escogido un LED de color blanco y sus características técnicas son las siguientes.

- Tensión Directa (Vf): La cantidad de voltaje necesario para que el LED comience a conducir corriente tiene un rango desde los 1.8 V hasta los 3.8 V. En concreto un LED blanco será aproximadamente de 3.2 V.
- Corriente Directa (If): Es la corriente que circula por el LED cuando el dispositivo está funcionando. Circula una cantidad de 20 mA. Para LEDs de alto brillo podrían necesitar 30 mA.
- Flujo Luminoso (mcd): En el caso de un LED blanco cuenta con más lúmenes que el resto de los colores, en este caso el valor típico es de 12000 mcd.
- Temperatura de operación (°C): Va desde los -25 °C hasta los 80 °C.

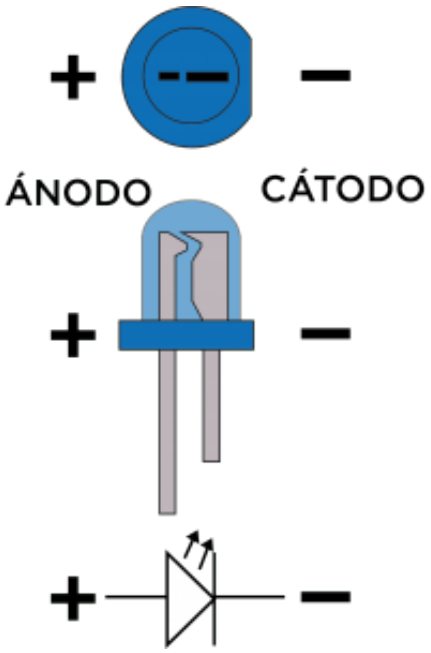

Ilustración 16. Identificación de Ánodo y Cátodo en diodo LED.

Un LED cuenta con dos patillas, la larga se denomina ánodo y la negativa cátodo. La corriente circula desde el ánodo hasta el cátodo, en el caso de un LED blanco tiene que circular una cantidad de 30 mA. Por lo tanto, para realizar la construcción de un circuito con un LED siempre tendremos que lograr la circulación de 20 mA, con una caída de tensión de entre 1.8 y 3.8 V o en concreto el LED blanco de 3.2 V aproximadamente. Para lograr esto se deben añadir resistencias para no dañar el LED.

4.3. Fotorresistencia (LDR)

Del inglés "Light-dependent resistor" es un componente eléctrico que cuenta con una resistencia que varía, generalmente disminuye su valor, en función de la intensidad de luz incidente sobre él. El comportamiento de este dispositivo se basa en el efecto fotoeléctrico, donde la energía de los fotones de luz libera electrones en el semiconductor y de esta manera afecta a la resistencia del dispositivo.

(*Fotorresistor - Wikipedia, la enciclopedia libre*, s. f.)

Las principales características técnicas son las siguientes:

- **Resistencia en oscuridad:** Cuando no existe ninguna luz incidente la resistencia tiene un valor de un megaohmio (MΩ).
- **Resistencia en luz:** Cuando existe una luz incidente se tiene un valor de 400 Ω cuando recibe 1000 lux y un valor de 9 kΩ cuando la luz incidente es de 10 lux.
- **Valor analógico:** El valor de lectura con el que cuenta este dispositivo es entre 0 y 1023. Este valor es el que se utilizará para trabajar con él.

Existen dos formas diferentes de conectar un LDR para obtener un resultado diferente.

- **Mayor luz, mayor voltaje**. Conectando la fotorresistencia al nodo positivo de nuestra fuente de voltaje provoca que, al incidir mayor luz, la diferencia de potencial será menor entre la fuente de voltaje y el pin Vout y por lo tanto recibiremos un valor de lectura mayor. Esto quiere decir que cuando haya mucha iluminación el valor analógico leído será mayor que si hay poca iluminación.

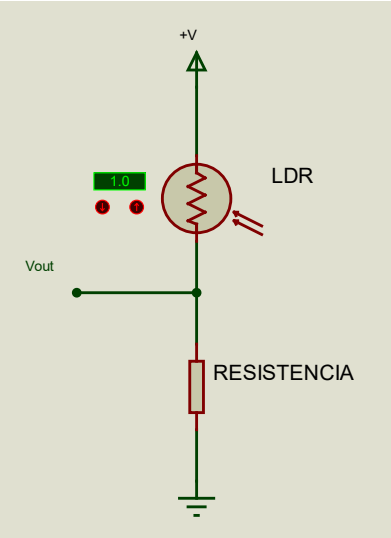

Ilustración 17. Divisor de tensión mayor luz, mayor voltaje.

- **Mayor luz, menor voltaje.** Configurando el circuito de manera opuesta, al incidir mayor luz, la diferencia de potencial será mayor entre la fuente de voltaje y el pin Vout y por lo tanto recibiremos un valor de lectura menor. Esto quiere decir que cuando haya poca iluminación el valor analógico leído será mayor que si hay poca iluminación.

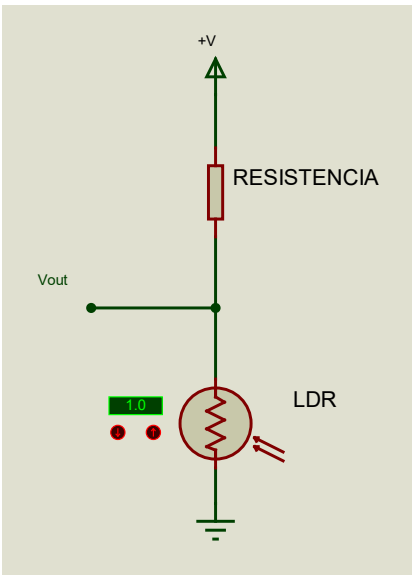

Ilustración 18. Divisor de tensión mayor luz, menor voltaje.

Sus principales ventajas es que son muy fáciles de utilizar, con un bajo costo y alta sensibilidad. Por el contrario, cuenta con desventajas como puede ser una respuesta espectral estrecha, esto quiere decir que puede ser más sensible a una banda en concreto de longitudes de onda de luz en lugar de tener una sensibilidad a un rango grande de frecuencias de luz. Se puede producir el efecto de histéresis, lo cual puede retardar el funcionamiento del dispositivo.

(*Fotoresistor, LDR o Fotoresistencia — MecatrónicaLATAM*, s. f.)

4.4. Resistencia

Es un dispositivo utilizado para introducir una resistencia eléctrica en un circuito eléctrico. Su principal objetivo es oponerse al paso de la corriente a través de él.

(*Conoce los tipos de resistencias electrónicas | Aprende Institute*, s. f.)

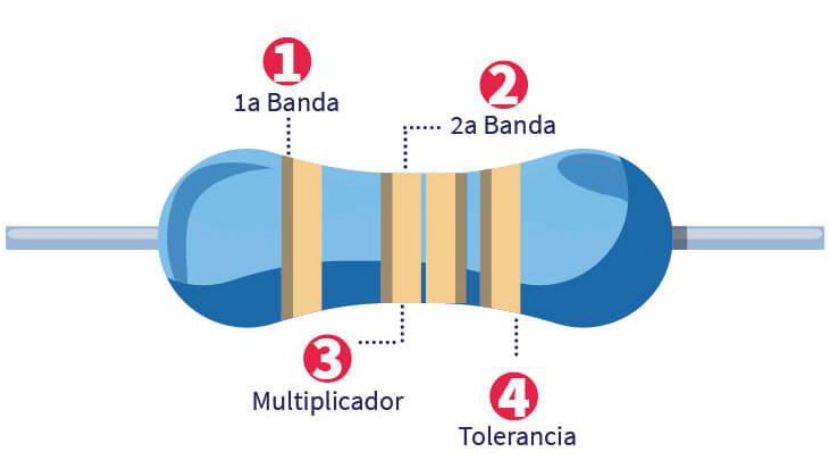

Ilustración 19. Definición de las bandas de una resistencia.

Como se puede observar en la ilustración, las resistencias cuentan con unas bandas con las cuales se puede conocer el valor de cada una de ellas consultando una tabla. La primera banda representa primer el valor del componente, la segunda indica el segundo valor del componente, la tercera banda sirve como valor multiplicador y por último la cuarta banda indica la tolerancia del valor en un porcentaje.

El valor final se halla uniendo los dígitos de las primeras bandas, multiplicando por el valor de la tercera y añadiendo una tolerancia a modo de porcentaje.

Algunas resistencias pueden tener una banda opcional que indica el coeficiente de temperatura.

En este proyecto se utilizan para proteger un LED y limitar la corriente que circula por él con el fin de evitar daños sobre el dispositivo. A su vez se utiliza como divisor de tensión cuando se conecta el LDR para conseguir una referencia y poder adaptar los niveles de voltaje y ajustar señales analógicas.

4.5. Printed Circuit Board, Placa de circuito impreso (PCB)

La Placa de Circuito Impreso (PCB) proporciona un componente físico para la interconexión y montaje de diversos elementos electrónicos. Fundamentada en la eficacia y funcionalidad, su diseño requiere una cuidadosa consideración de diversos criterios que aseguren la optimización de la distribución, enrutamiento de señales, disipación de calor y otros aspectos clave.

(*Printed circuit board - Wikipedia*, s. f.)

En primer lugar, la distribución eficiente de componentes es de gran importancia. La disposición estratégica busca minimizar la longitud de las conexiones eléctricas, para reducir inductancias y capacitancias que puedan provocar funcionamientos malignos para conseguir un rendimiento mayor y evitar posibles interferencias.

Es de gran importancia la planificación de cómo enrutarlas señales, no cruzar rutas si no es necesario, etc. Es muy importante intentar realizar giros curvos, que no sean de 90 grados puesto que puede lograrse un mal funcionamiento de la circulación de las rutas.

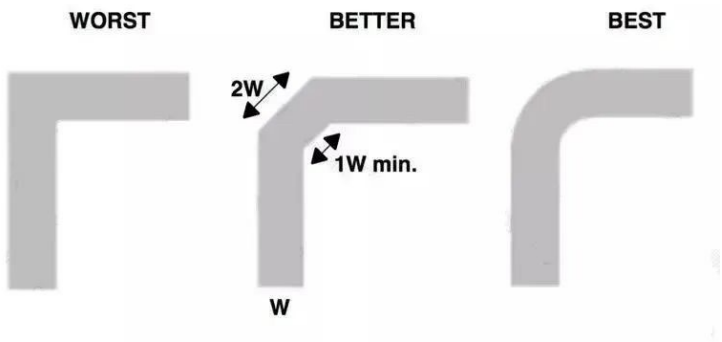

Ilustración 20. Posibilidades de rutado en PCB.

(*¿Por qué las trazas de PCB no pueden ir en un ángulo recto de 90 grados? - Conocimiento - St.Qin Cross Boarder E-Commerce Co., Ltd*, s. f.)

Además, es importante tener un ancho de rutas mínimo o recomendado para el correcto funcionamiento del circuito. Para corrientes de hasta 1 A, se podría considerar un ancho de 10 mils (0.254 mm) a 20 mils (0.508 mm). Para corrientes entre 1 A y 5 A, se podría considerar un ancho de 20 mils a 40 mils (1.016 mm).

La separación de señales, especialmente de alta y baja frecuencia, es esencial para prevenir interferencias electromagnéticas. La colocación de aquellos componentes que puedan generar calor con cobre adicional para optimizar la transferencia del calor es también muy importante.

Antes de realizar la fabricación de la placa, se deben realizar verificaciones de reglas eléctricas (ERC) u de reglas de diseño (DRC) para garantizar el correcto funcionamiento.

Estas reglas se van a tener en cuenta para realizar el diseño de la placa de circuito impreso más adelante.

(*5 reglas de diseño de PCB importantes que debes saber | Altium*, s. f.)

5. Desarrollo del proyecto.

5.1. Planteamiento del proyecto y problemática

El principal objetivo de este proyecto es lograr desarrollar un sistema que permita controlar la luminosidad de un espacio cerrado mediante un dispositivo LDR y un LED. Se realizará un estudio del comportamiento del sistema, construyendo un controlador PID que permita controlar dicha iluminación evitando que se produzca el efecto Windup debido a la saturación que puede provocar el término integral.

Para el desarrollo de este proyecto se utiliza una placa de Arduino donde se desarrollará el circuito donde se encontrará el LDR encargado de recibir el nivel de iluminación recibido por el LED que estará instalado próximo a él. El programa se va a desarrollar en el entorno de Arduino IDE y será cargado en el microcontrolador para alcanzar el objetivo introduciendo un controlador PID para el control preciso de iluminación.

Todo este proyecto se construye en una caja cerrada al completo.

5.2. Resolución del problema

Este proyecto está formado principalmente por un Arduino, un LED y un LDR. Tenemos un sistema en el que la entrada será la iluminación que tiene el LED con un rango de 0 a 255 en valor analógico que será procesada y tratada por el controlador PID para lograr el valor indicado por la consigna o setpoint, que es el valor que se desea conseguir y obtener el valor de la variable de salida que será el valor de la regulación recibida por el LDR con un rango analógico de 0 a 1023.

Para la construcción del controlador se ha hecho un análisis previo teórico a través de los resultados obtenidos con un sistema en lazo abierto y tras métodos de sintonización se han calculado las componentes que forman el PID para más tarde realizar un ajuste preciso incorporado en el sistema real.

A la hora de comprobar los resultados, al introducir una respuesta rápida el sistema genera ruido y se ha incorporado un suavizado para evitar un funcionamiento errático.

Para poder realizar diferentes medidas, pruebas o diferentes fines se ha realizado la posibilidad de generar una onda cuadrada para que la consiga realice cambios cada intervalo de tiempo.

5.2.1. Montaje y cálculos teóricos

En primer lugar, se va a calcular el valor de la resistencia necesaria para que el LED funcione de manera correcta.

Ilustración 21. Diodo LED blanco.

Como se ha mencionado en el punto 4.2. de este documento, el LED cuenta con una corriente de funcionamiento aproximadamente de 30 mA y una diferencia de tensión entre sus terminales de 3.2 V. El voltaje de alimentación será de 5 V proporcionados por la placa de Arduino por lo que aplicando la ley de Ohm, la cual reconoce que el voltaje es igual a la resistencia multiplicada por la corriente que circula por ella, y la ley de tensiones de Kirchhoff, la cual dice que en un lazo cerrado la suma de todas las diferencias de tensión es igual a la tensión total suministrada, podremos realizar las siguientes operaciones.

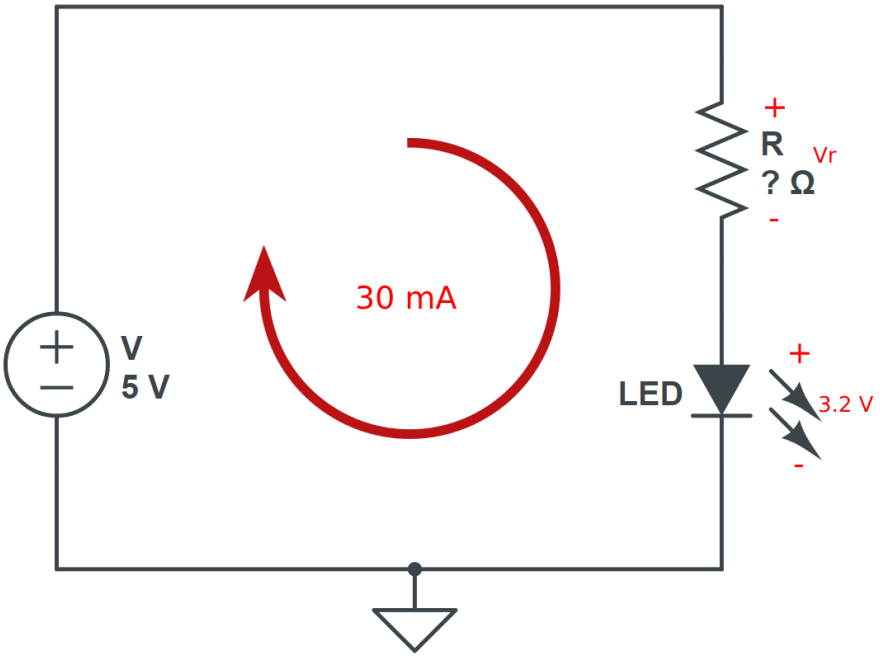

Ilustración 22. Cálculo y descripción del circuito.

Resolvemos la malla indicada en la ilustración anterior

$$R * I_{LED} + V_{LED} = V_F \text{ (15)}$$

Donde R es el valor en Ohm de la resistencia, I_{LED} es la intensidad que circula por la malla, V_{LED} es la caída de tensión que existe en el LED, el cual se aproxima a 3.2 V para nuestro LED, y V_F es el valor de voltaje de la fuente, 5 Voltios en este caso.

$$R = \frac{5\ V - 3.2\ V}{30 * 10^{-3}} = 90\ \Omega$$

A continuación, buscaremos una resistencia equivalente convencional. El valor que más se aproxima es de 220 Ω. Podemos colocar dos resistencias de este mismo valor para obtener una resistencia equivalente más próxima.

$$\frac{1}{R_{eq}} = \frac{1}{220} + \frac{1}{220}$$

$$R_{eq} = 110\ \Omega$$

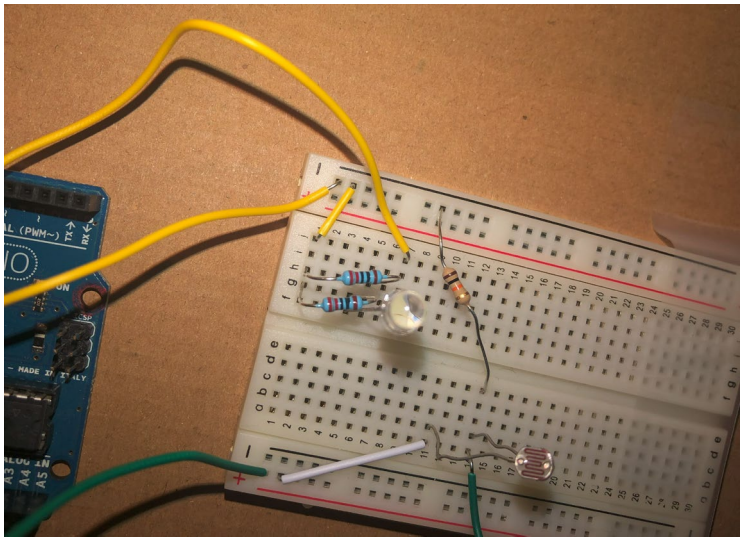

Ilustración 23. Resistencias colocadas en paralelo.

En la imagen anterior se puede observar cómo aparecen conectadas dos resistencias en paralelo para lograr el resultado buscado. A continuación, observamos experimentalmente los resultados obtenidos.

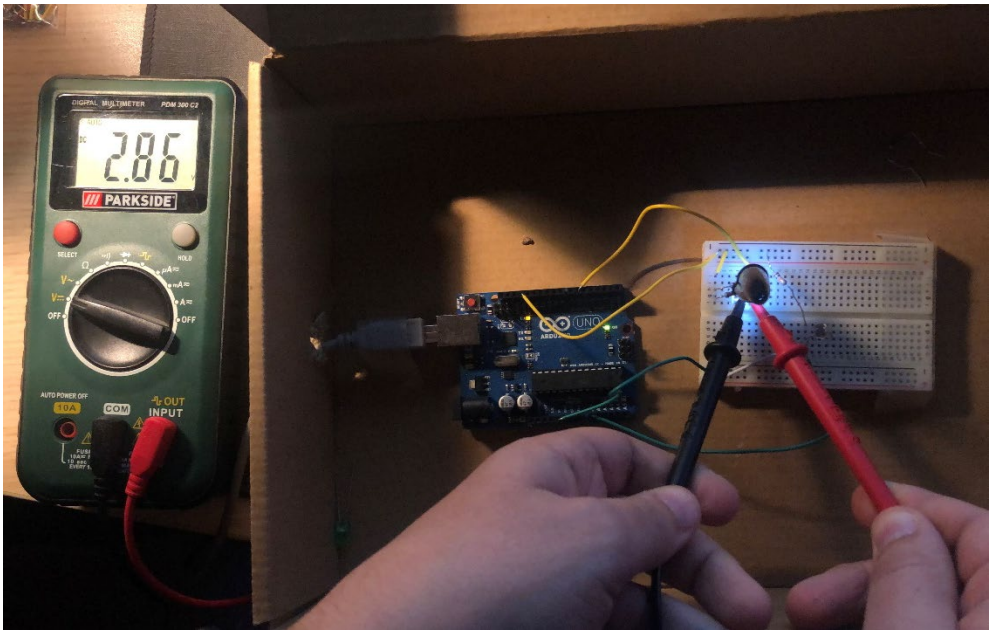

Ilustración 24. Valor obtenido de voltaje entre los terminales del LED.

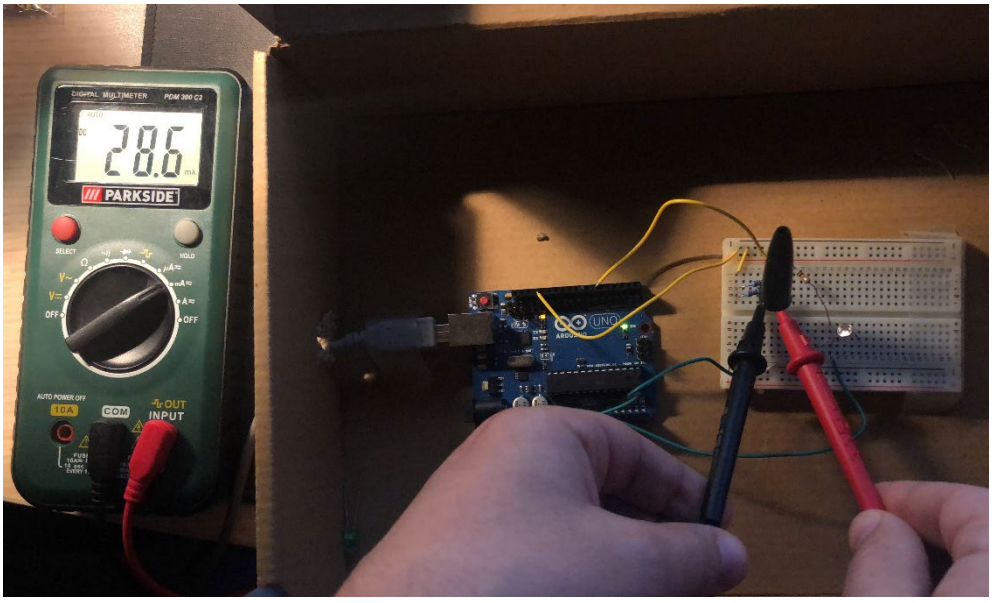

Ilustración 25. Valor de corriente que circula por los terminales del LED.

Obtenemos un valor de caída de tensión en el LED de 2.86 Voltios y circula un valor de intensidad de 28.6 mA. Valores muy próximos a los calculados teóricamente adaptados a las resistencias normalizadas disponibles.

En segundo lugar, debemos construir el circuito para colocar el sensor de luz. Generalmente se utiliza una resistencia de valor de 10 kΩ para construir el divisor de tensión. Este valor es suficientemente alto para limitar la corriente que circula p0or el divisor, pero lo suficientemente bajo para conseguir una sensibilidad útil en el LDR.

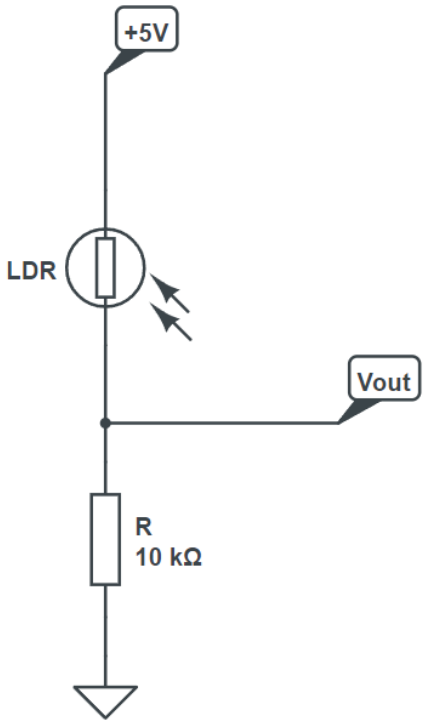

Ilustración 26. Divisor de tensión utilizado.

Asignar un valor fijo de resistencia en el divisor de tensión nos permite obtener un valor variable del voltaje que obtenemos por el nodo Vout lo cual es útil para tener un control sobre los resultados recibidos por el pin analógico del Arduino.

5.2.2. Diseño y fabricación de la PCB

Comprobado el funcionamiento del sistema utilizando una protoboard y el Arduino conectado con cables, se va a proceder al diseño de una placa que va a ir incorporada sobre la propia placa de Arduino.

Para ello se ha utilizado un programa de diseño y simulación electrónica muy utilizado para sistemas y circuitos llamado “Proteus”. Este programa nos permite realizar las interconexiones del circuito eléctrico y a su vez realizar el rutado de la placa de diseño impreso en la misma plataforma.

En la pestaña de Esquema Electrónico se va a diseñar el circuito añadiendo componentes utilizando la ventana “Devices” como se muestra en la siguiente ilustración.

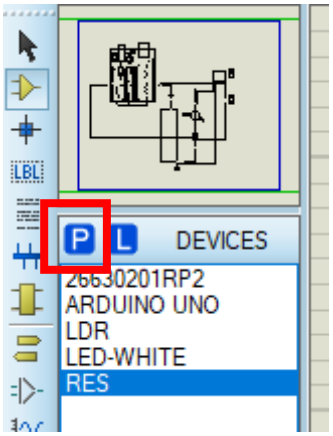

Ilustración 27. Selector de componentes en Proteus.

Pulsando en la P de “Pick Devices” tenemos una lista completa de todos los componentes con los que cuenta Proteus divididos en categorías.

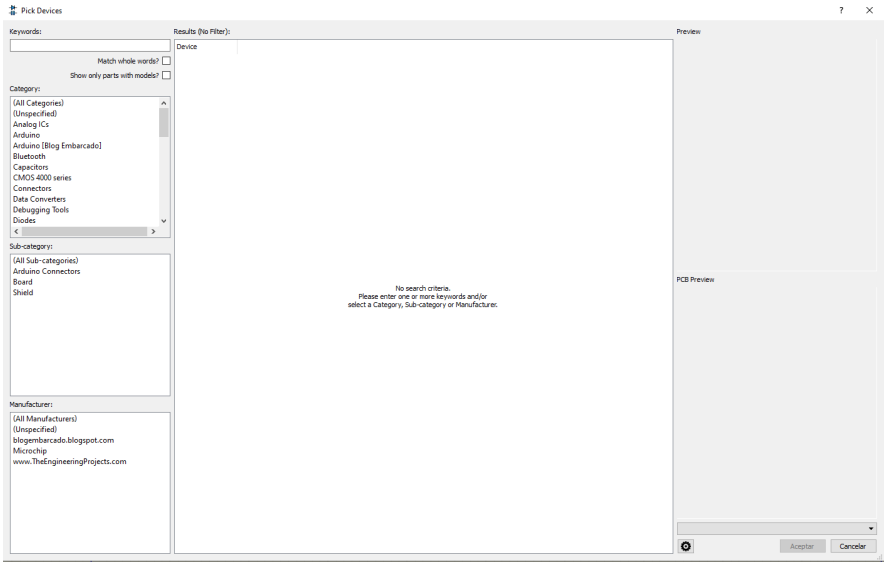

Ilustración 28. . Selector de componentes y subcarpetas en Proteus.

Para realizar nuestro diseño, necesitaremos colocar la placa de Arduino Uno, un LED, un LDR, resistencias y bornes de tornillo para poder aportar tensión y tierra y también para tener los pines utilizados seleccionados y más tarde poder añadirlos en el diseño de la PCB.

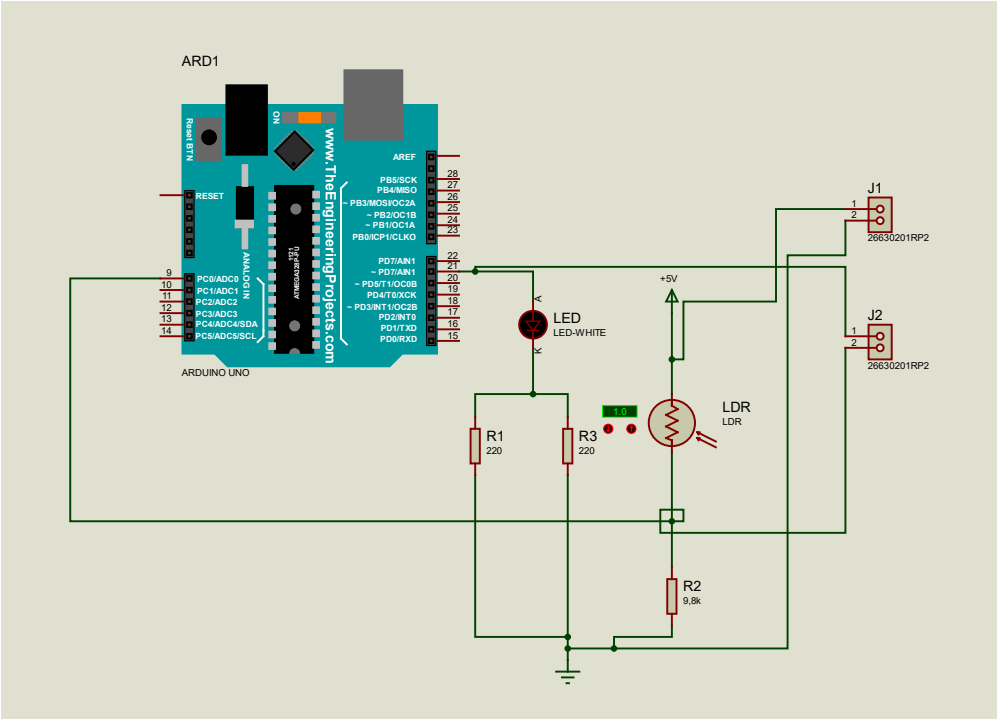

Ilustración 29. Circuito esquemático construido en Proteus.

Realizando las conexiones oportunas obtenemos el circuito resultado anterior, tal y como se ha construido con anterioridad.

A continuación, podremos pasar a la pestaña del diseño de PCB pulsando en el botón que se muestra en la ilustración 30.

(*(186) HOW TO MAKE A PCB (PRINTED CIRCUIT BOARD) | PROTEUS SOFTWARE | D&R TUTORIALES - YouTube*, s. f.)

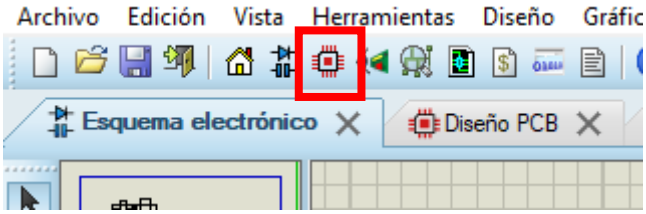

Ilustración 30. Botón de la pestaña de diseño de PCB.

Para conseguir un proyecto que sea cómodo de utilizar y eficiente en su trabajo se ha decidido construir una placa que se sitúe encima del Arduino la cual esté conectada directamente a los pines digitales y analógicos con los que cuenta el Arduino. De esta manera el proyecto se va a resumir en una placa de Arduino con el circuito que se pretende utilizar sobre ella.

Para conseguir las medidas exactas del Arduino y hacer coincidir correctamente los pines de nuestro diseño, se ha importado una huella a escala 1:1 para asignarla al componente seleccionado de la pestaña de esquema electrónico y poder trabajar con ella en la pestaña de diseño de PCB.

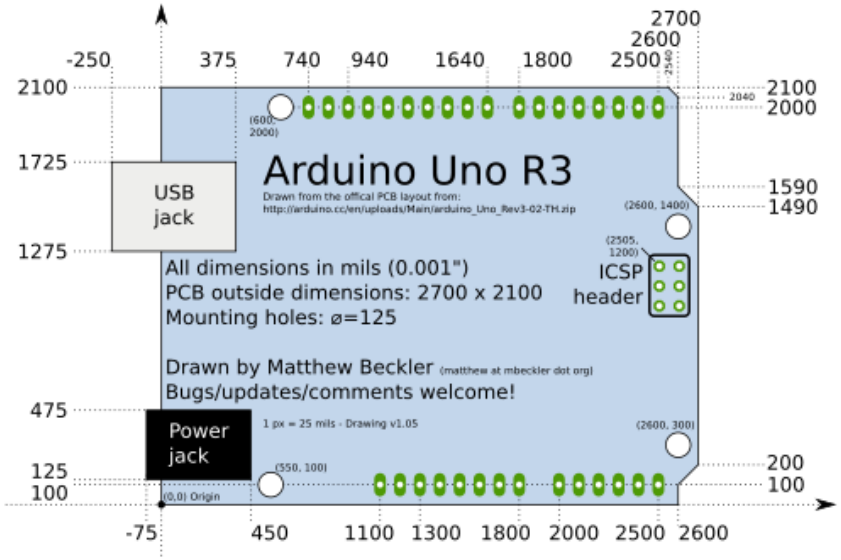

Ilustración 31. Dimensiones placa Arduino UNO.

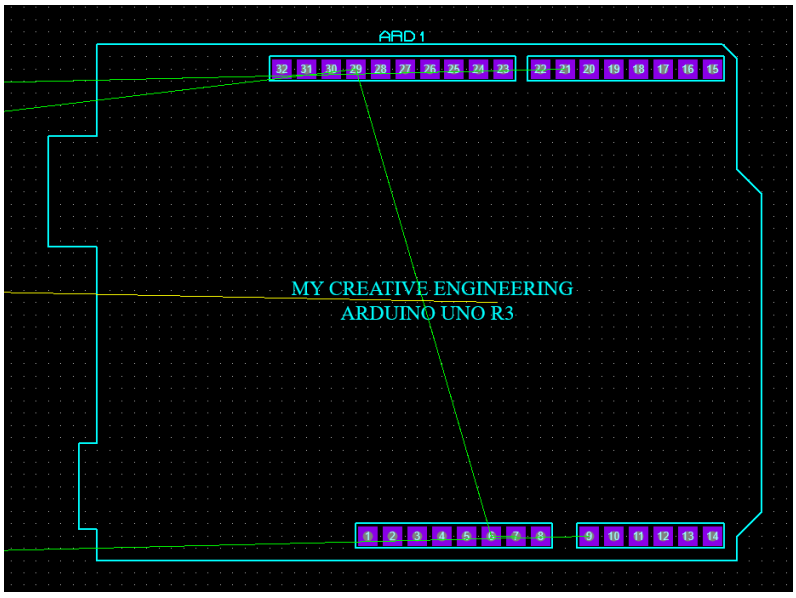

Ilustración 32. Huella en Diseño de PCB de Arduino UNO.

(*Nice drawings of the Arduino UNO and Mega 2560 | Arduino Blog*, s. f.)

El LED y el LDR no cuenta con componente esquemático para incorporar en la pestaña de PCB por lo que hay que proceder de la manera mostrada en la siguiente ilustración.

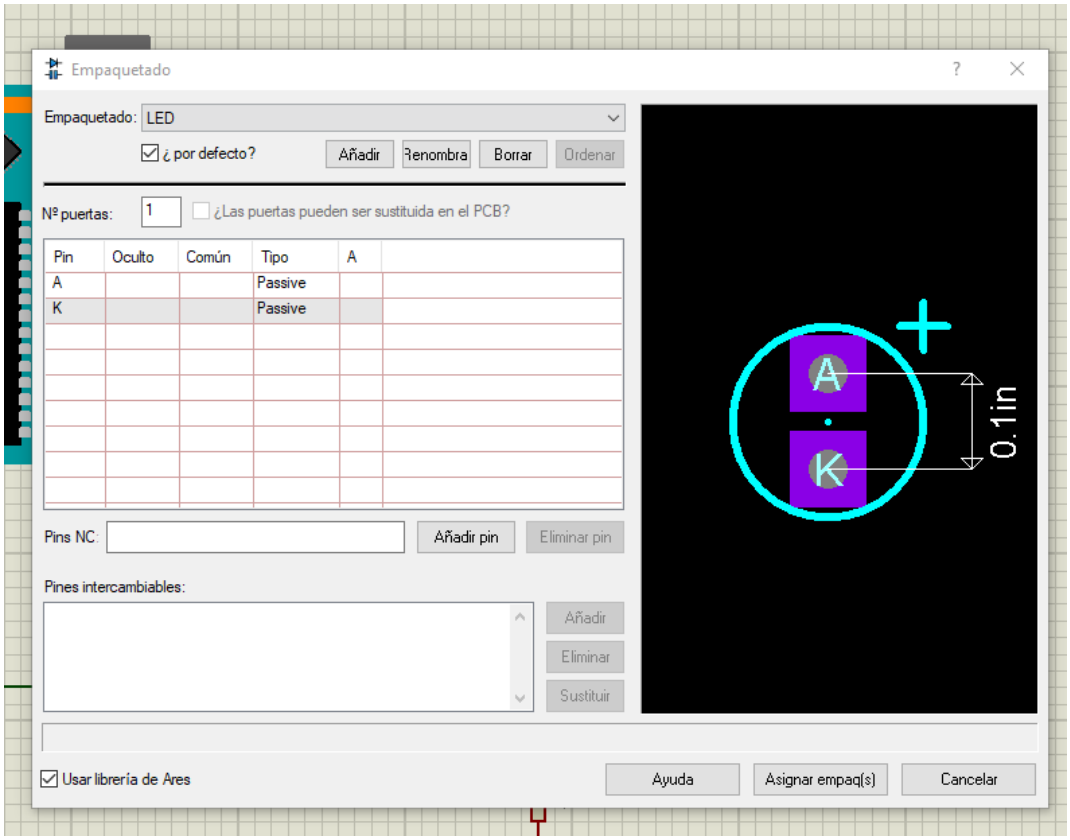

Ilustración 33. Configuración de los terminales de la huella del LED.

(*(186) How to Add Arduino UNO Footprint PCB Package on Proteus 8 | R1 | R2 | R3 - YouTube*, s. f.)

Pulsando botón derecho en el LED del esquemático y herramientas de empaquetado se abre la ventana anterior, pulsamos añadir y buscamos "LED" seleccionándolo se generará el esquemático. Asignamos "A" a "A" y "K" a "K" para que no de error y realizamos lo mismo para el LDR.

Utilizamos un tamaño de pista de 40 mils como se indicaba en el apartado 4.5. puesto que es el recomendado para el circuito que tenemos. Además, la manera de eficiente de realizar los giros es utilizando curvas por lo que será la manera de realizar el rutado.

(*(186) Como hacer Pistas Curvas en PROTEUS - YouTube*, s. f.)

Colocando todos los componentes en el entorno obtenemos la siguiente distribución y diseño.

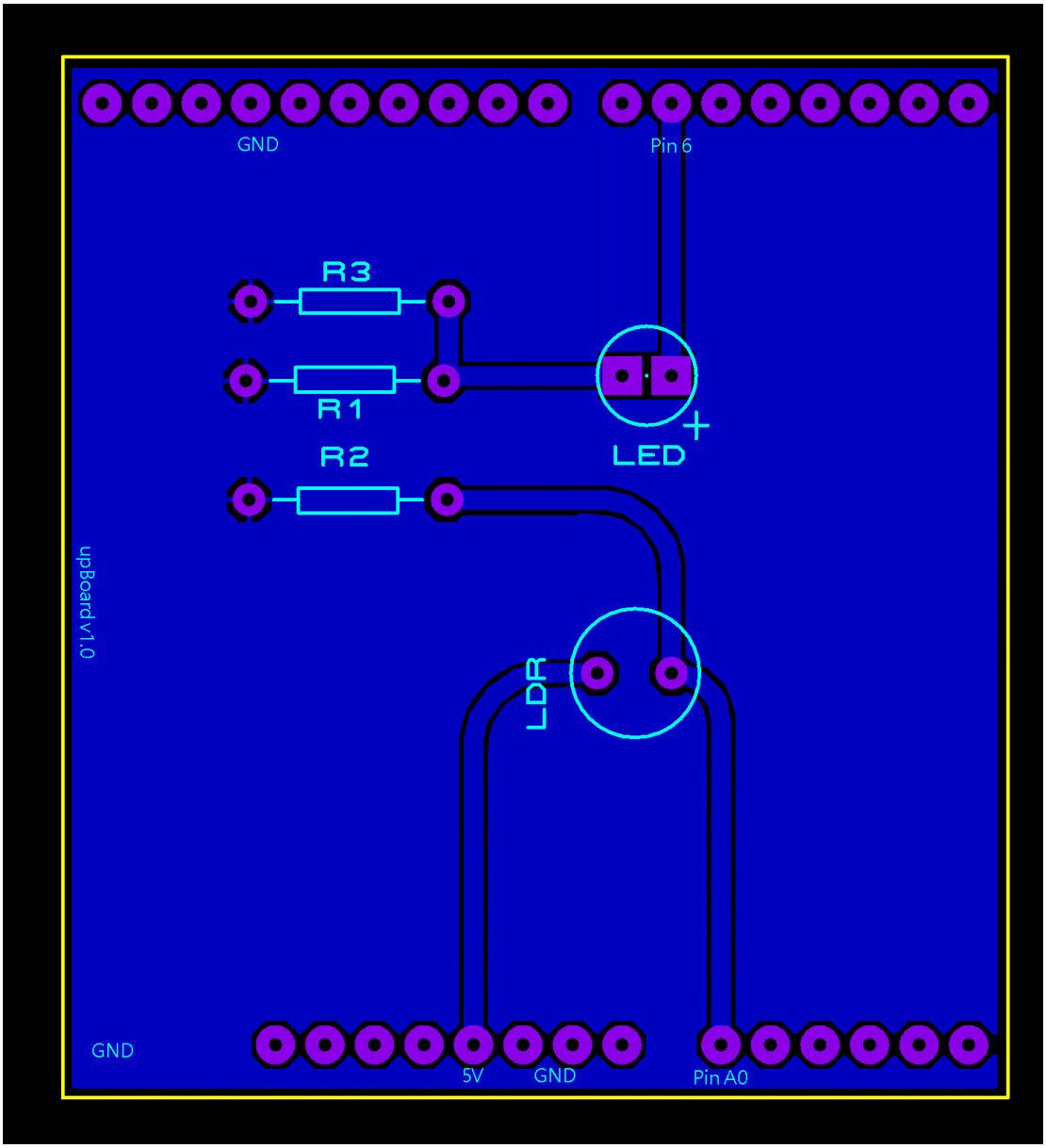

Ilustración 34. Diseño final del circuito en PCB.

Asignamos las etiquetas correspondientes a los pines de cada componente. También se ha creado un plano de masas para la eficiencia del circuito.

Proteus cuenta con una herramienta para poder visualizar en tres dimensiones como se visualizaría la placa cuando se fabrique. Se adjuntan diferentes ilustraciones para mostrar dicho resultado.

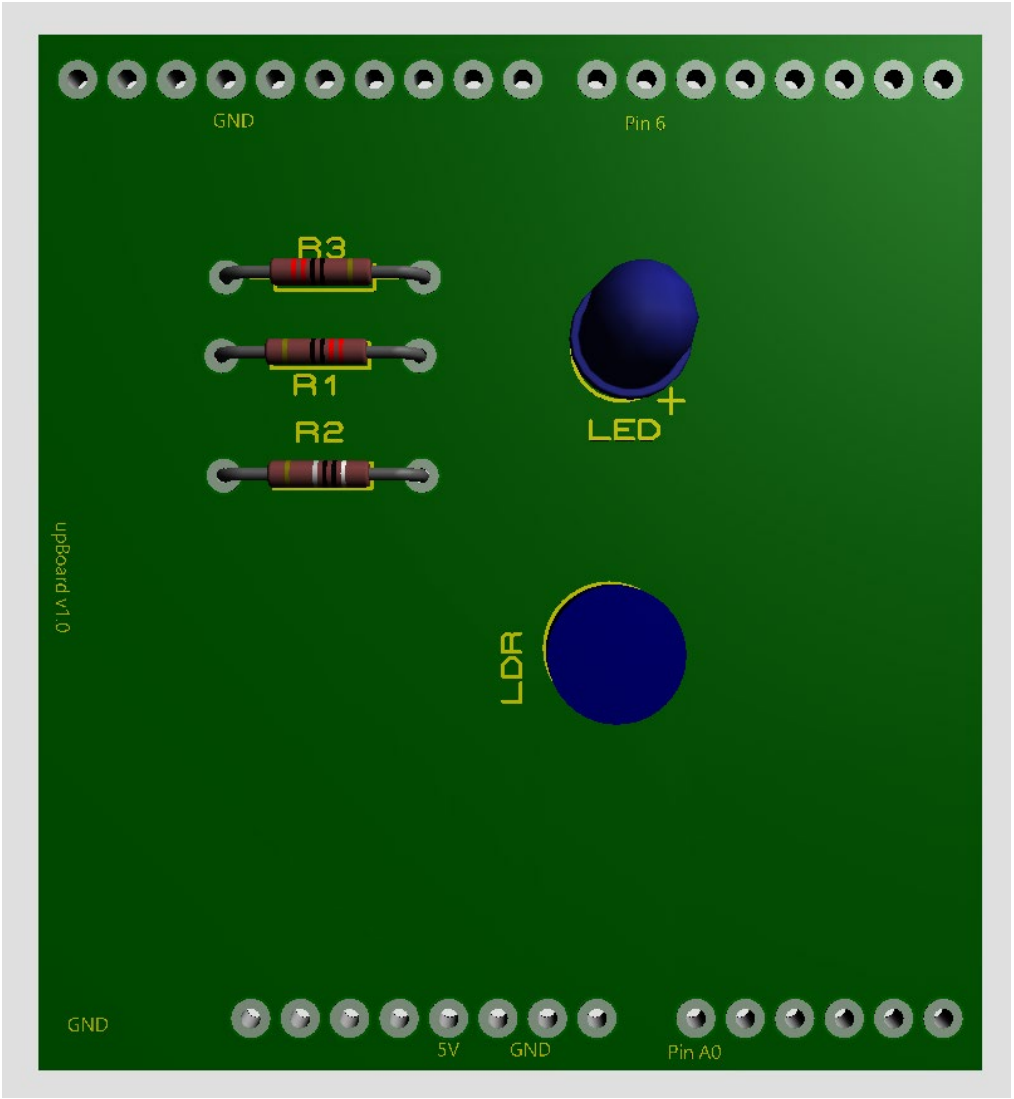

Ilustración 35. Resultado final de la cara "top" de la PCB en 3D.

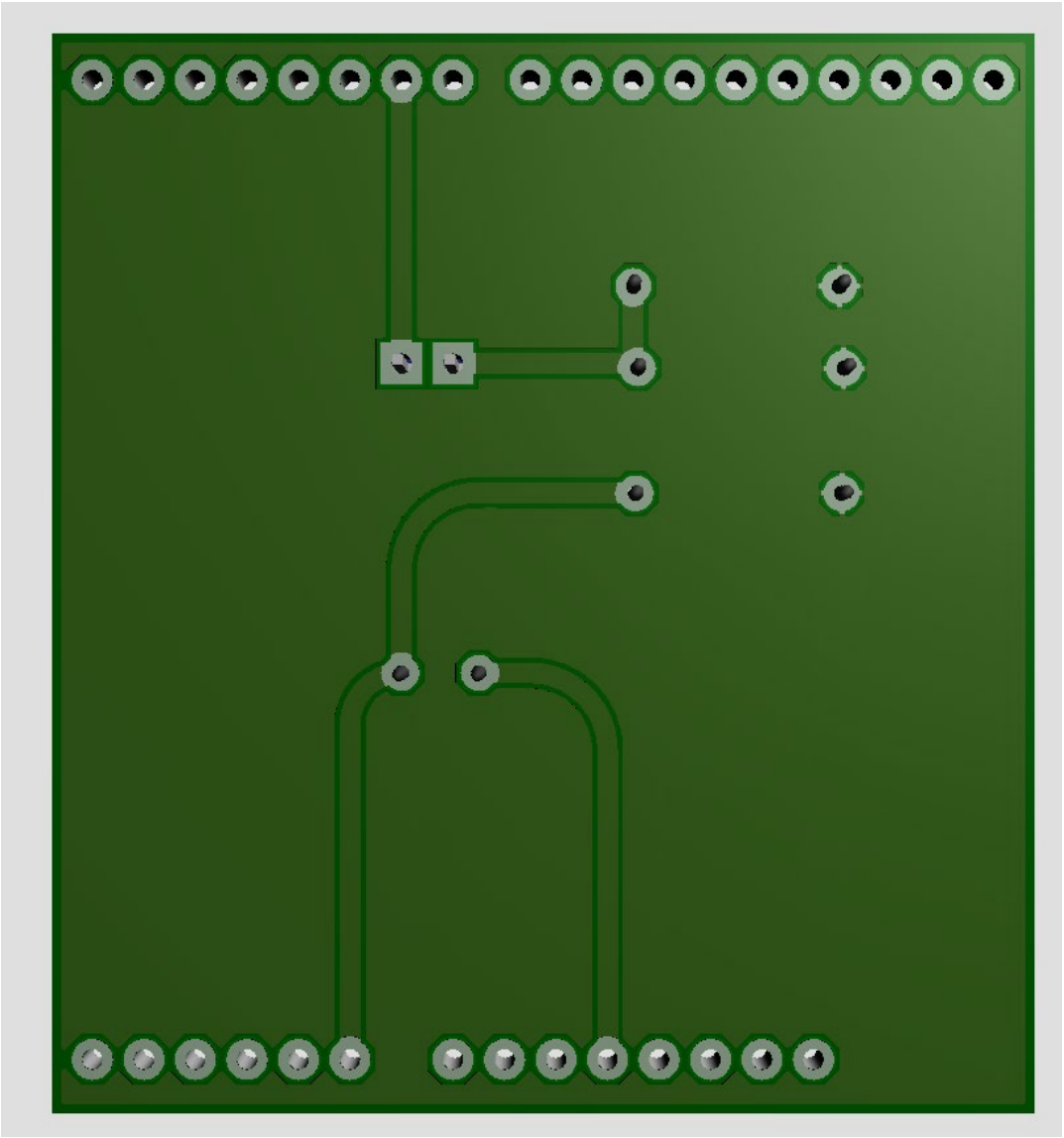

Ilustración 36. Resultado final de la cara "back" de la PCB en 3D.

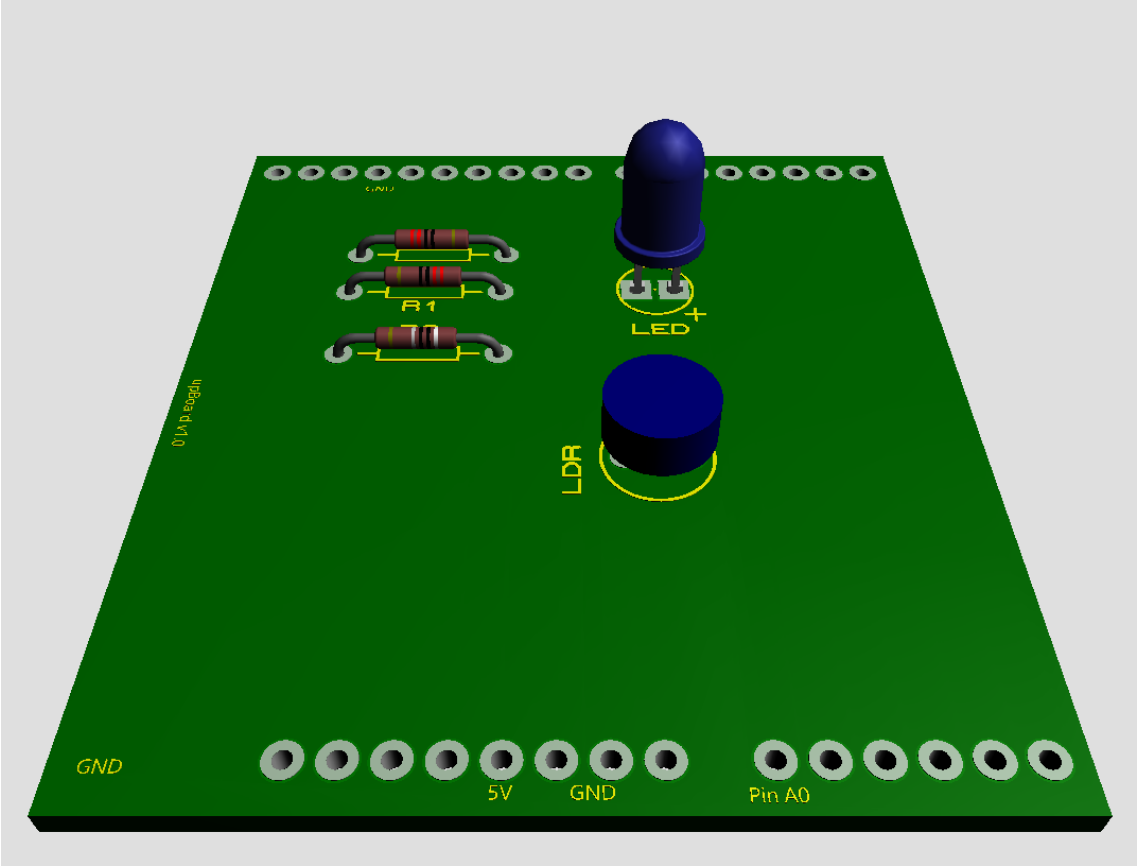

Ilustración 37. Resultado final de la PCB en 3D.

De esta manera se habrá concluido el diseño de la placa que se utilizará para colocar encima del Arduino.

(*ArduinoUnoFP.pdsprj - Google Drive*, s. f.)

Para enviar a fabricación es necesario exportar un archivo llamado Gerber el cual contiene toda la información para el diseño de la placa. Describen las capas individuales de la PCB, las ubicaciones de las pistas, los pads, orificios, etc. Para ello en el programa pulsaremos en “Salida” y después en “Generar un archivo Gerber” como se indica a continuación.

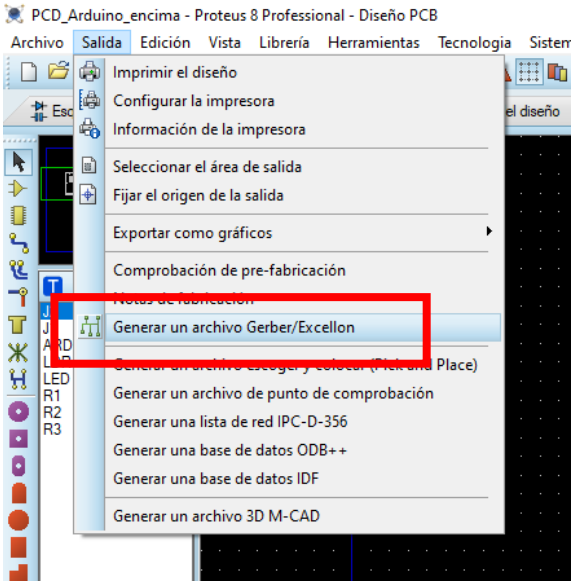

Ilustración 38. Selección para generar archivos Gerber en Proteus.

El programa informa sobre las configuraciones necesarias para construir los archivos Gerber y con todo configurado se enviará a su fabricación para conseguirla físicamente.

5.2.3. Desarrollo del programa

En este apartado se realiza una explicación detallada del funcionamiento del programa para lograr el objetivo indicando las funcionalidades incorporadas.

En primer lugar, se han configurado los pines siguiendo las conexiones realizadas en la placa de Arduino.

```
// Define pins
const int ldrPin = A0;  // LDR conectado al pin A0
const int ledPin = 6;   // LED conectado al pin 6

void setup() {
  pinMode(ledPin, OUTPUT); //Inicializamos el LED como salida
  analogWrite(ledPin, 0); //Comenzamos con el LED apagado
  pinMode(ldrPin, INPUT); //Inicializamos el LDR como entrada
}
```

Ilustración 39. Conexiones del LED y el LDR con sus pines.

En las líneas anteriores indicamos que el LDR se conecta al pin analógico denominado A0 y el LED se conecta al pin digital 6. Dentro de la configuración void setup() indicamos que el LED es un dispositivo de salida inicializado en 0 y el LDR es un dispositivo de entrada.

A continuación, y para lograr una comunicación serie con la UART necesitamos asignar una velocidad de transmisión de datos medida en baudios.

```
Serial.begin(9600); //Para la comunicacion serial
```

Ilustración 40. Velocidad de transmisión con la UART.

Con este código podremos realizar depuraciones del programa y poder mostrar por pantalla los valores de las variables y los gráficos que se pretendan analizar.

Realizamos un pequeño programa para realizar la medida del lector LDR para obtener los valores reales medidos de nuestro circuito construido.

Medida_de_valores_LDR.ino

```
1   void setup() {
2     // put your setup code here, to run once:
3     pinMode(A0,INPUT);
4     Serial.begin(9600);
5   }
6
7   void loop() {
8     // put your main code here, to run repeatedly:
9
10    int lectura_LDR = analogRead(A0);
11
12    Serial.println(lectura_LDR);
13    delay(1000);
14  }
15
```

Ilustración 41. Programa para obtención de valores del LDR.

Utilizamos la salida por pantalla para observar los resultados obtenidos por el sensor de luz con máxima luz incidiendo y mínima para comprobar el correcto funcionamiento.

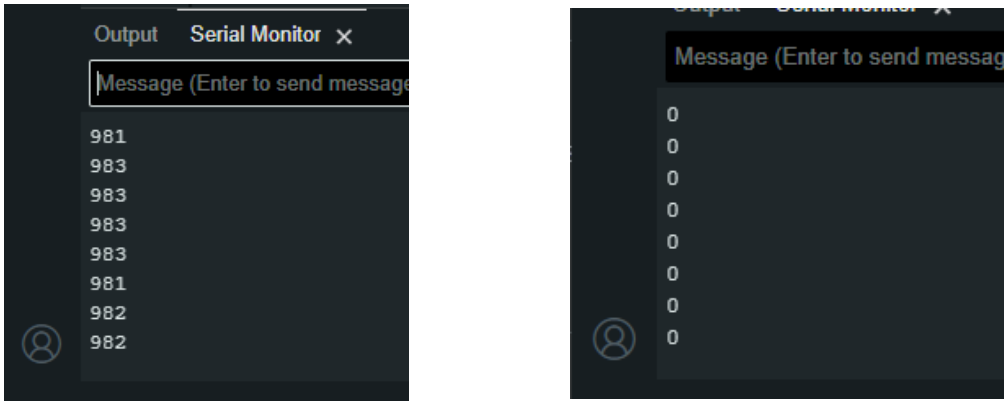

Ilustración 42. Valores obtenidos por el LDR en Arduino IDE.

Obtenemos un resultado de 983 cuando el LDR está al descubierto y 0 cuando el LDR está tapado completamente por lo que el dispositivo está funcionando a la perfección.

Lo siguiente que introduciremos en nuestro programa es la declaración del controlador PID. Para realizar la programación de este controlador seguiremos la teoría explicada en el apartado 3.1 de este documento. El controlador PID estará formado principalmente por las componentes proporcional e integral ya que el sistema es demasiado sencillo para aplicar una componente derivativa en primera instancia.

En el apartado 3.1.6.2. se indica el método de sintonización que aplicaremos a continuación para obtener los valores de las componentes que más tarde se ajustarán observando el resultado experimental.

Para comenzar necesitamos obtener la respuesta de nuestro sistema en lazo abierto, para ello convertimos el valor leído por el LDR a un valor de rango 0 a 255 que mide el LED para obtener la respuesta de nuestro sistema sin ningún tipo de control.

Con los pines del LED y LDR declarados, convertiremos el valor de salida del sensor en un valor adaptado al LED. Para este cálculo es suficiente con restringir el valor que se obtiene de luz del LDR al rango admitido por el LED de 0 a 255 puesto que interesa la velocidad con la que reacciona el sistema en lazo abierto.

Como se explicó en el apartado 3.3. El filtrado exponencial o suavizado exponencial consiste en obtener un valor filtrado a partir de una medición mediante la aplicación de la siguiente expresión:

$$A_n = \alpha M + (1 - \alpha) A_{n-1} \quad (14)$$

Donde A_n es el valor filtrado y A_{n-1}es el valor filtrado anterior. M es el valor muestreado de la señal a filtrar (nuevalectura), y α es un factor entre 0 y 1.

Para realizar esta operación en Arduino creamos una variable que se va a utilizar para suavizar "nuevaLectura" correspondiente a M. También se aplica un factor de suavizado acorde a lo explicado en el apartado 3.3.

```
double nuevaLectura = analogRead(ldrPin);
double factorSuavizado = 0.45;  // Factor de suavizado
```

Ilustración 43. Declaración de M y el factor de suavizado.

Aplicamos la expresión (14) anterior de la siguiente manera.

```
salida = (1 - factorSuavizado) * salida + factorSuavizado * nuevaLectura;
```

Ilustración 44. Expresión del suavizado en el código de Arduino.

Donde “salida” será la variable suavizada. En función de la lectura que se está realizando en el momento de ejecución (nuevaLectura) y el valor suavizado anterior (La propia variable salida leída anteriormente) se logrará el resultado que saldrá por pantalla.

```
void loop() {
    double factorSuavizado = 0.45;  // Factor de suavizado
    double nuevaLectura = analogRead(ldrPin); //Lectura del LDR

    salida = (1 - factorSuavizado) * salida + factorSuavizado * nuevaLectura;
```

Ilustración 45. Código completo del suavizado.

(*Arduino Filtro pasa bajo Media Móvil Exponencial | Eliminar ruido - YouTube*, s. f.; *Filtro paso bajo y paso alto exponencial (EMA) en Arduino*, s. f.-b)

A continuación, se mostrará el código utilizado para obtener la respuesta del sistema en lazo abierto para el futuro cálculo de las componentes del controlador a incorporar en el sistema en lazo cerrado.

```
  entrada = salida; //Salida sin control directamente relacionada con la entrada

  // Convertimos el valor a un rango de 0 a 255 para el PWM
  entrada = constrain(entrada, 0, 255);

  // Set the LED brightness
  analogWrite(ledPin, entrada);

  //Salida por pantalla y graficos
  Serial.print(entrada); //value 1
  Serial.print(",");
  Serial.println(salida); //value 2

  // Ponemos un retardo ajustado a la planta
  delay(100);
}
```

Ilustración 46. Código utilizado para el sistema en lazo abierto.

Mediante la función “constrain()” explicada en el apartado 4.1.2. limitaremos el valor al admitido por el LED, escribiremos en el LED dicho valor utilizando “analogwrite()” y mediante un grupo de Serial.print() podremos observar en el Serial Plotter el resultado de la respuesta en lazo abierto.

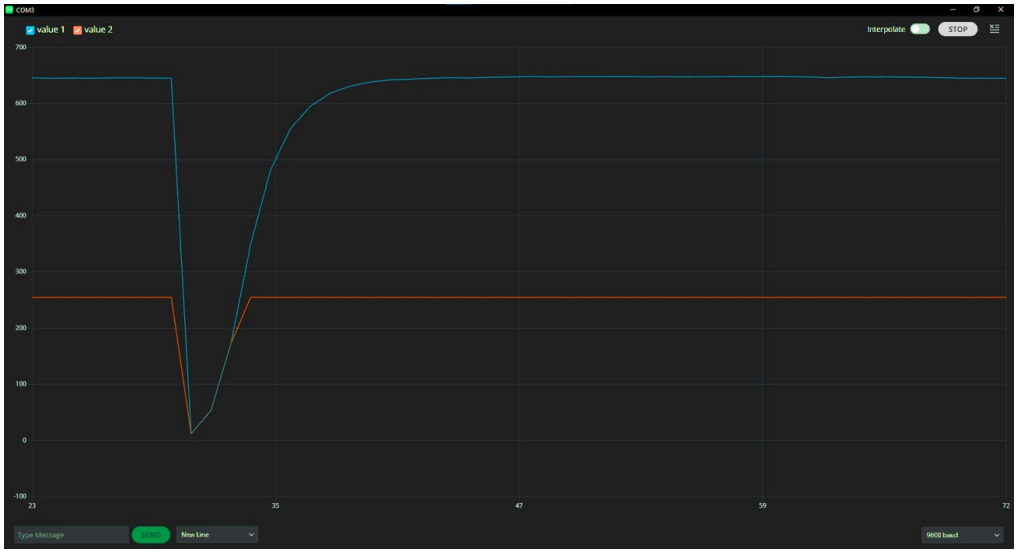

Ilustración 47. Respuesta del sistema en lazo abierto.

A continuación, tomaremos los valores cuando se realiza el inicio del sistema y los valores cuando se ha estabilizado para calcular la función de transferencia en lazo abierto siguiendo el método de sintonización mediante el método de asignación de polos.

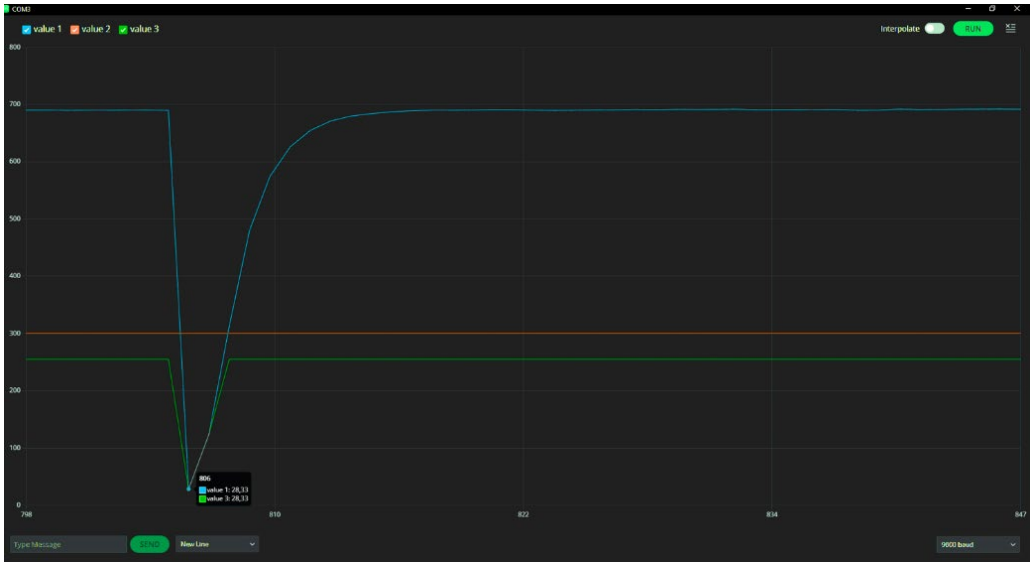

Ilustración 48. Valor de la salida en el inicio del sistema en lazo abierto.

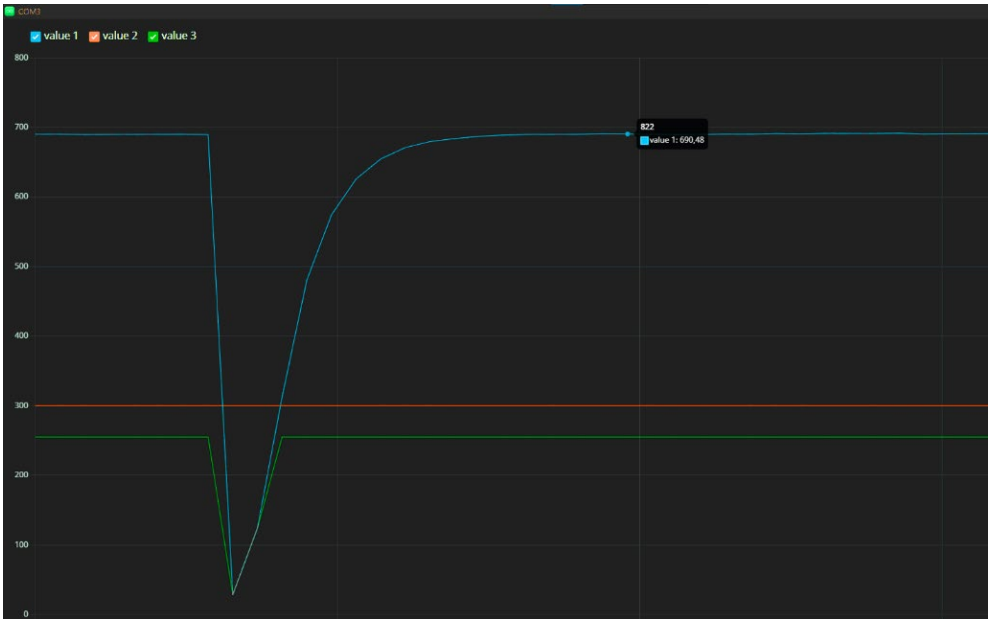

Ilustración 49. Valor de la salida cuando se ha estabilizado el sistema en lazo abierto.

La función de transferencia no presenta ningún retardo por lo que su forma será de la siguiente manera:

$$G(s) = \frac{K}{\tau s+1} \quad (8)$$

A partir de los valores observados en el gráfico anterior podemos resumir los valores en la siguiente ilustración donde aparece de color verde la señal del LED y de color azul la señal del LDR.

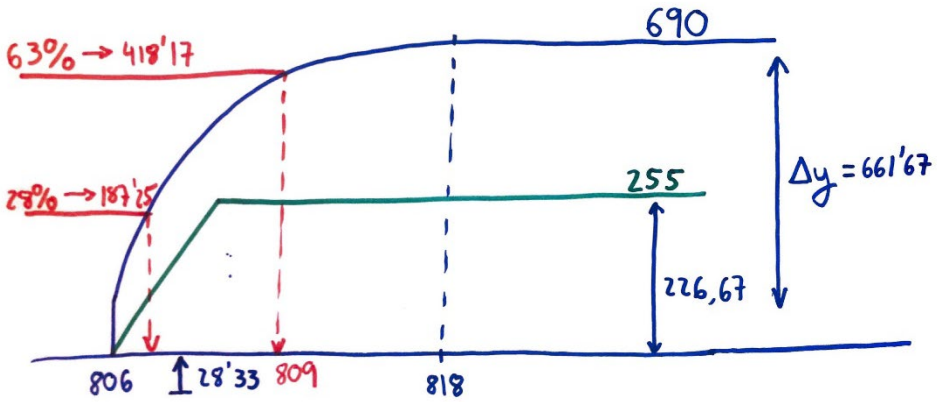

Ilustración 50. Datos obtenidos del sistema en lazo abierto.

Para obtener la constante de tiempo τ necesitaremos obtener los tiempos en un 63% y un 28%.

$t1 \Rightarrow 0.283(28.33 - 690) = -187.25 + 690 = 502.7 \Rightarrow t1 = 809\, ms$

$t2 \Rightarrow 0.63(28.33 - 690) = -412.22 + 690 = 277.78 \Rightarrow t1 = 807\, ms$

$$\tau = 1.5(t1 - t2) = 3\, ms$$

La constante K referida a la ganancia del sistema se calcula conociendo la variación en la entrada ΔU y la variación de salida ΔY.

$$K = \frac{\Delta Y}{\Delta U} = 2.919$$

Por lo tanto, obtenemos la función de transferencia en lazo abierto.

$$G(s) = \frac{K}{\tau s + 1} = \frac{2.919}{3s + 1}$$

Cerramos el lazo con un controlador PI quedando el diagrama que se muestra a continuación.

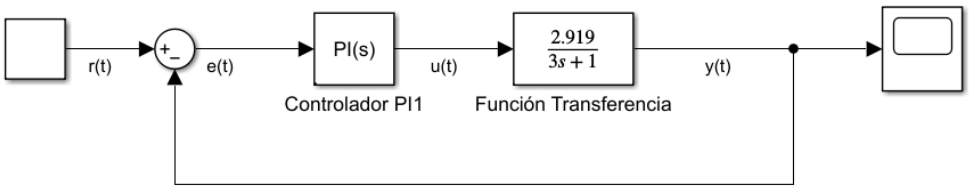

Ilustración 51. Diagrama del sistema obtenido en lazo cerrado.

El controlador PI tiene una componente proporcional e integral construyéndose de la siguiente manera.

$$PI = C(s) = Kp + \frac{Ki}{s} \quad (16)$$

Cerrando el lazo y desarrollando.

$$Gc(s) = \frac{C(s)G(s)}{1 + C(s)G(s)} \quad (17)$$

$$Gc(s) = \frac{\frac{2.919}{3s + 1}(Kp + \frac{Ki}{s})}{1 + \frac{2.919}{3s + 1}(Kp + \frac{Ki}{s})}$$

$$Gc(s) = \frac{2.919(Kps + Ki)}{s^2 + \frac{2.919Kp + 1}{3}s + \frac{2.919}{3}Ki}$$

La función de transferencia que representa un sistema de segundo grado, como se explicaba con anterioridad en el apartado 3.1.6.2. es la siguiente

$$F(s) = \frac{K{w_n}^2}{s^2+2\delta w_n+{w_n}^2} \text{ (18)}$$

Donde δ es el factor de amortiguamiento y w_n es la frecuencia natural.

Los denominadores de las anteriores ecuaciones tienen la misma distribución por lo que se pueden igualar los términos de s y el término independiente.

$$\frac{2.919Kp + 1}{3} = 2\delta w_n$$

$$\frac{2.919}{3}Ki = {w_n}^2$$

El factor de amortiguamiento y la frecuencia natural se van a asignar para obtener un resultado de las constantes del controlador. Estableciendo δ=1 y w_n= 0.2 y despejando de las ecuaciones anteriores obtenemos los siguientes valores.

$$\boldsymbol{Kp = 0.0685}$$

$$\boldsymbol{Ki = 0.0411}$$

Estas constantes son las que se incorporará en el controlador PI. Para comprobar que se ha realizado el cálculo de dichos valores correctamente, se va a realizar un sistema en Matlab Simulink para observar el comportamiento del sistema.

Calculo_PI.m

```
% Variables de la funcion de transferencia
K = 2.919;
tao = 3;

% Función de transferencia en lazo abierto
G_La = tf([K], [tao, 1])

% Parámetros del controlador PI
Kp = 0.0685;
Ki = 0.0411;

% Controlador PI
PI = tf([Kp, Ki], [1, 0])

% Transfer en lazo cerrado
G_Lc = feedback(G_La * PI, 1)

%Grafica ante una entrada escalon
figure;
step (G_Lc)
title('Respuesta del Sistema ante entrada escalón');
xlabel('Tiempo (ms)');
ylabel('Amplitud');
grid on;

% Cambiar el nombre del eje X
xlabel('Tiempo (ms)');
```

Ilustración 52. Código matlab para obtener la respuesta en lazo cerrado.

Construimos este sencillo código en Matlab donde se establecen el valor de las constantes de la ganancia del sistema y de la constante de tiempo para más tarde formar la función del sistema de primer orden en lazo abierto. Lo siguiente es crear el controlador PI con el que se cierra el lazo del sistema y por ello se tienen que asignar los valores de las constantes proporcional e integral calculados anteriormente. Por último, escribimos las líneas que permiten ver la respuesta del sistema ante una entrada escalón.

```
G_Lc =

      0.2 s + 0.12
  --------------------
  3 s^2 + 1.2 s + 0.12
```

Ilustración 53. Función de transferencia obtenida en matlab en lazo cerrado.

Una vez ejecutado el programa observamos la función obtenida en lazo cerrado y una ventana emergente donde aparece la respuesta del sistema.

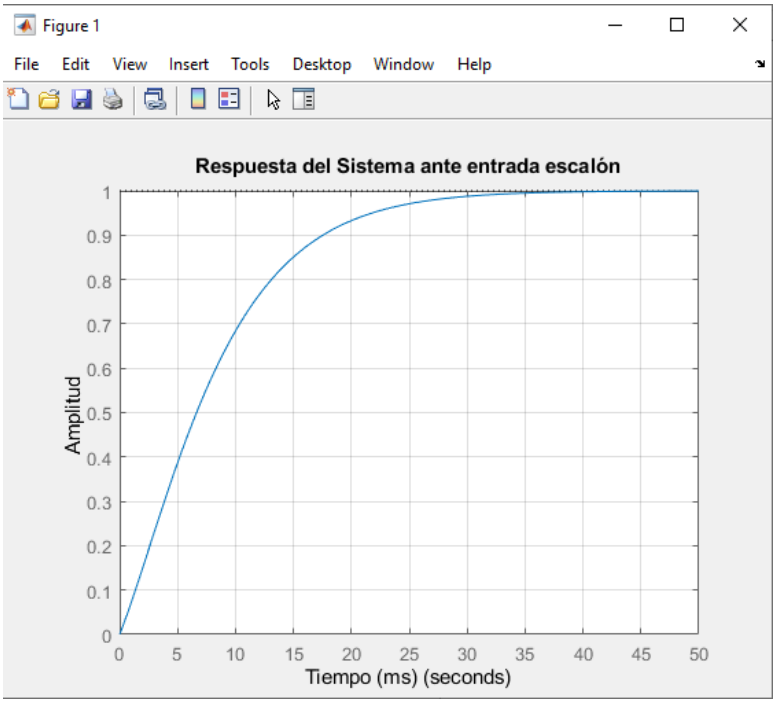

Ilustración 54. Respuesta del sistema ante una entrada escalón.

El resultado de la respuesta del sistema es rápido y no muestra sobreoscilación ni comportamientos que puedan ser problemáticos para el proyecto al que vamos a incorporar dicho controlador PI.

A continuación, incorporamos la programación del sistema en lazo cerrado junto con el controlador PI.

Al inicio del programa declaramos las variables para el controlador que vamos a incorporar las cuales se ajustarán dependiendo del resultado del sistema experimental.

```
// Definicion de constantes del controlador PID
double Kp = 0.0685;    // Componente proporcional
double Ki = 0.0411;    // Componente integral
double Kd = 0;    // Componente derivativa
```

Ilustración 55. Declaración de las componentes del controlador PID.

Se necesita un valor de consiga o setpoint. El setpoint se corresponde con la lectura deseada del LDR y se va a asignar un valor de, por ejemplo, 300, ya que el LDR cuenta con un rango de 0 a 1023. También se inicializan las variables necesarias para el cálculo de la entrada, salida y componentes del controlador, así como los valores del error actual y anterior.

```
// Define variables
double setpoint = 300.0;  // Lectura deseada para el LDR, se ajusta dependiendo el sistema
double entrada = 0; //Valor del LED
double salida = 0; //Valor medido por el LDR
double proporcional = 0; //Valor accion proporcional
double comp_integral = 0; //Valor accion integral
double integral = 0; //Valor componente integral
double comp_derivativo = 0; // Valor accion derivativa
double derivativo = 0; //Valor constante derivativa
double error = 0; // Valor del error actual
double lastError = 0; //Valor del error anterior
```

Ilustración 56. Inicialización de las variables utilizadas.

Con estas variables inicializadas, se realiza la programación del controlador PID siguiendo la base teórica explicada en el apartado 3.1. a partir del suavizado.

Se ha creado una variable booleana denominada “suavizado” para poder activar y desactivarlo como se puede observar a continuación. A su vez también se utiliza una variable booleana denominada “anti” para activar y desactivar el Anti-Windup.

```
if (suavizado == true) {
  double factorSuavizado = 0.45;  // Factor de suavizado
  double nuevaLectura = analogRead(ldrPin);

  salida = (1 - factorSuavizado) * salida + factorSuavizado * nuevaLectura;
} else {
  salida = analogRead(ldrPin);  //Lectura del LDR
}

// Calculo del error
error = setpoint - salida;

// Calculo del valor P proporcional
proporcional = Kp * error;

if (anti == true)
{
  if(entrada > 0 && entrada < 255)
  {
      integral += error;
  }else{}
}else
{
  integral += error;
}
//Calculo de la componente I integral
comp_integral = Ki * integral;
//Calculo de la componente D derivativa
derivativo = error - lastError;
comp_derivativo = Kd * derivativo;

// Calculamos la entrada, la ecuacion del PID
entrada = proporcional + comp_integral + comp_derivativo; //u(k) teorico

// Convertimos el valor a un rango de 0 a 255 para el PWM
entrada = constrain(entrada, 0, 255);

// Entrada al LED
analogWrite(ledPin, entrada);

lastError = error; //En el siguiente bucle el error actual es el anterior
```

Ilustración 57. Implementación del controlador PID en Arduino.

En las líneas anteriores se puede observar cómo se implementa el Anti-Windup. Se evalúa el valor de la entrada, correspondiente al valor del LED, el cual varía entre 0 y 255 por lo que, si el valor de la entrada está comprendido entre esos valores, se incrementa el valor de la componente integral, y por el contrario si el valor está saturado, para evitar que se produzca Windup, no se incrementará el valor integral.

Pueden encontrarse situaciones en las que el setpoint no sea constante durante toda la sesión, es decir, podría variar en intervalos de tiempo. Se ha añadido la posibilidad de realizar una onda cuadrada que varíe entre dos valores. Esto también ayudará a comprobar el funcionamiento correcto del sistema consiguiendo el valor deseado en todo momento.

Se ha construido la función Onda_cuadrada() fuera del loop.

```
void Onda_cuadrada(){
  unsigned long actual_time = millis();
  unsigned long tiempo_demorado = actual_time - tiempo_inicial;

  /*Vemos en que modulo de 0 al valor del periodo nos
  encontramos para poder establecer la onda en alto o bajo*/
  if((tiempo_demorado % periodo) < (periodo/2))
  {
    setpoint = 1000;
  }else
  {
    setpoint = 100;
  }
}
```

Ilustración 58. Implementación de la función de Onda Cuadrada en Arduino.

Para construir dicha función hay que controlar el tiempo que transcurre y establecer un instante concreto en el tiempo. Para conseguir establecer un intervalo concreto de tiempo utilizamos "%" que calcula el módulo. Si por ejemplo contamos con 2000 milisegundos de periodo, el resto va a variar entre 0 y 2000. En otras palabras, el tiempo total se subdivide en fascículos de 2000 en 2000 milisegundos. Por lo tanto, comparando dicho valor utilizando un condicional con la mitad del periodo asignado, en este caso sería un valor de 1000, podremos ver cuando el valor se encuentra por encima o por debajo de 1000 repartiendo el tiempo en partes iguales.

En void loop() se llama a esta función con una variable booleana para poder activar o desactivar la onda cuadrada estableciendo la variable a true o false.

```
void loop() {
  if (Onda_Si == true)  //True = Activada
  {
    Onda_cuadrada();
  }
```

Ilustración 59. Llamada a la función de Onda Cuadrada si se cumple la condición booleana.

Al implementar esta onda cuadrada podemos comprobar cómo la lectura del LDR genera mucho ruido y el funcionamiento del LED es irregular. Gracias al suavizado explicado anteriormente, este ruido desaparece consiguiendo un funcionamiento perfecto del sistema.

5.3. Pruebas y verificación de los resultados

Los objetivos propuestos al inicio de este proyecto son el diseño de un sistema que regule la luminosidad en un entorno cerrado, previamente se ha construido el Arduino con la PCB sobre él para tener un aparato reducido en tamaño y eficiente. Analizar el efecto de las perturbaciones y el efecto Windup que se realizará a continuación. También se va a realizar experimentos y pruebas comprobando la correcta eficiencia del PID creado y configurado. Así como ajustar los valores de las componentes del PID para que el sistema funcione de la manera más eficiente posible.

A continuación, se muestran las diferentes pruebas y posibles configuraciones creadas en este proyecto.

Durante todo el apartado presente los colores de las gráficas representarán los siguientes valores:

- Value 1 (Azul): Salida, valor del LDR.
- Value 2 (Naranja): Setpoint.
- Value 3 (Verde): Valor integral (Para el Windup).
- Value 4 (Amarillo): Entrada, valor del LED.

5.3.1. Ajuste experimental a partir del controlador calculado teóricamente

Según lo calculado teóricamente hemos obtenido unos valores del PI de Kp de 0.0685 y Ki de 0.0411, y vamos a comprobar el funcionamiento del sistema real con estas componentes.

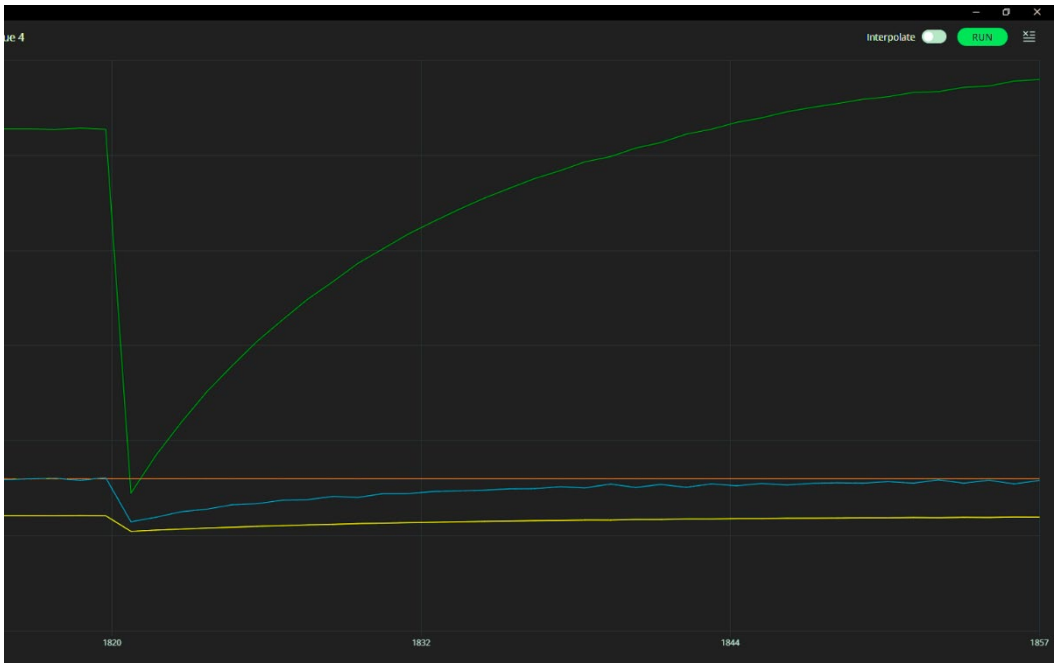

Ilustración 60. Resultado del sistema con los valores calculados.

Se observa una respuesta algo lenta de la respuesta del sistema y se pretende alcanzar el valor de setpoint con mayor velocidad por lo que ajustamos los valores incrementando los valores de las componentes.

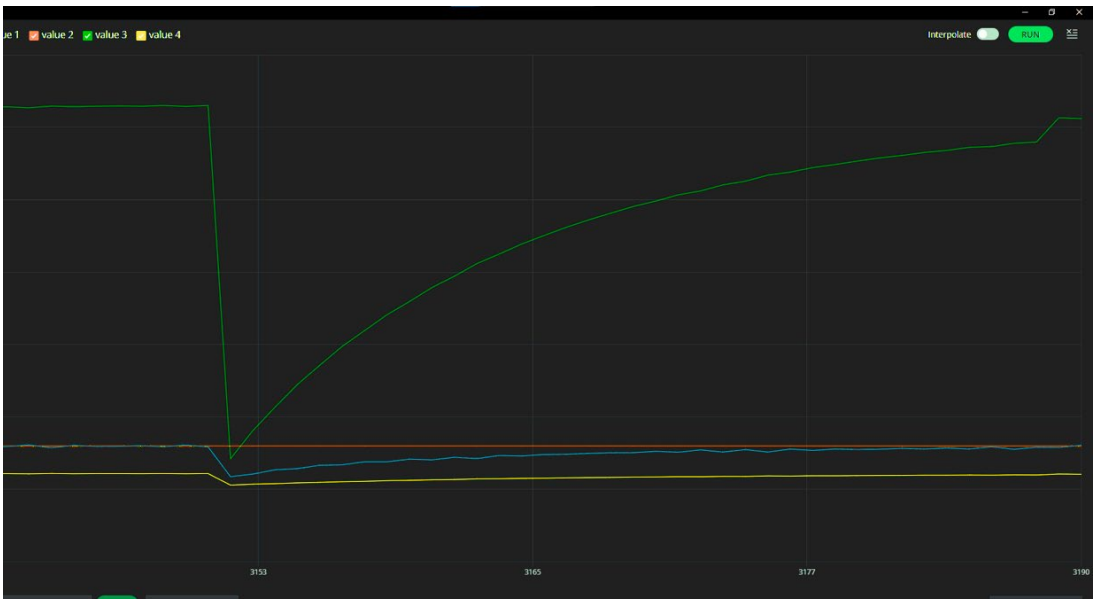

Ilustración 61. Resultado del sistema con el valor proporcional asignado a 0.1

Aumentar la ganancia proporcional a 0.1 no mejora la rapidez a la hora de alcanzar el valor deseado.

Por otro lado, incrementar el valor de la ganancia integral a 0.1 aumenta la rapidez del sistema, pero introduciendo bastante ruido, el cual no interesa en nuestro sistema porque va a provocar un funcionamiento del LED con saltos en la luz.

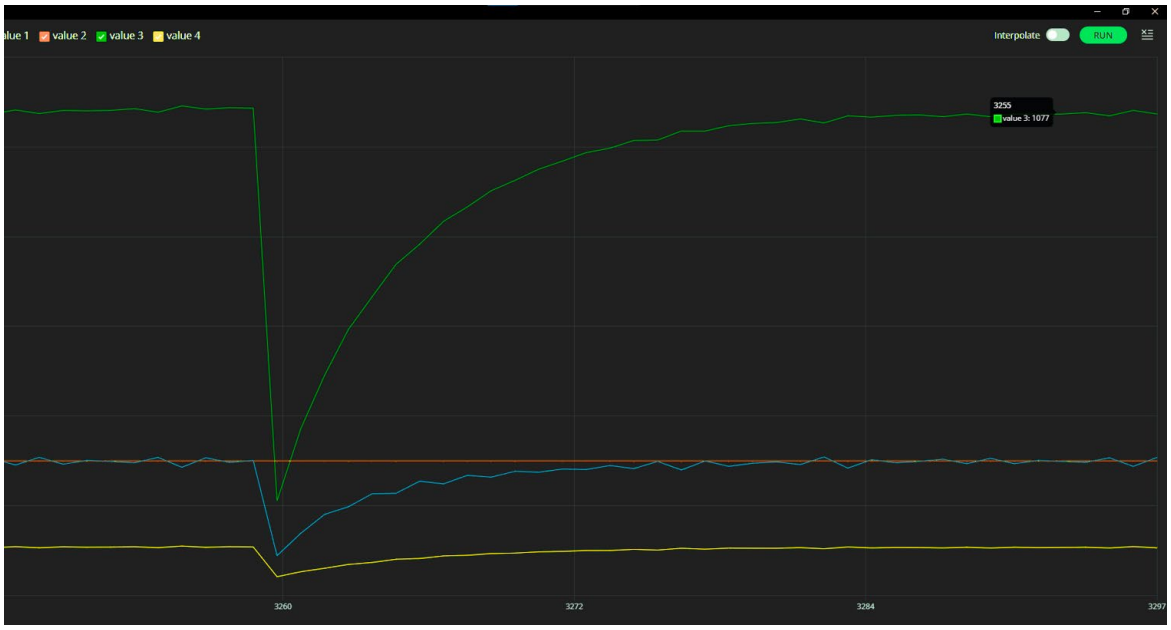

Ilustración 62. Resultado del sistema con 0.1 de valor de la componente integral.

Por lo tanto, el valor experimental obtenido para que el funcionamiento del sistema sea ideal es asignar tanto a la ganancia proporcional como la integral un valor de 0.05, muy similar al obtenido teóricamente. Se muestra la respuesta del sistema en la siguiente ilustración donde la luz al aumentar su brillo es lineal sin dar saltos.

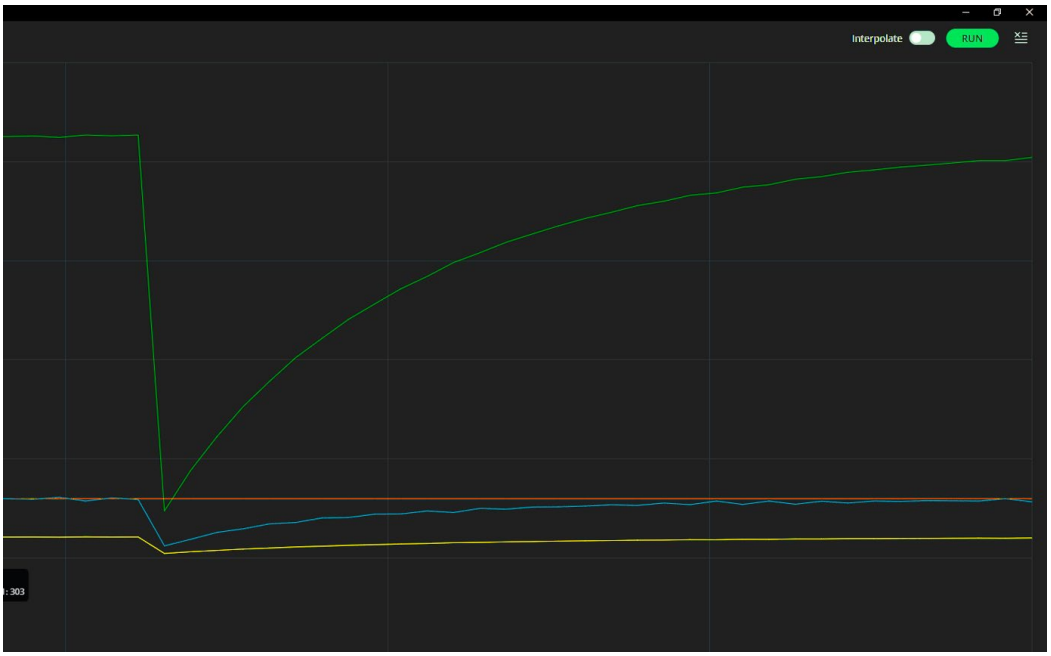

Ilustración 63. Resultado del sistema con los valores ideales Kp = 0.05 y Ki=0.05.

Los valores de Kp = 0.05 y Ki=0.05 son los ideales para nuestro sistema como se puede comprobar observando la respuesta del sistema en la ilustración anterior. A continuación, comprobaremos el funcionamiento de las funcionalidades incorporadas en el programa.

5.3.2. Suavizado

La respuesta del sistema puede contar con ruido o es posible encontrarse con un valor deseado muy grande que el sistema pueda realizar un incremento de luz de manera irregular. Para solucionar esto se ha introducido el suavizado. Activando la variable booleana que controla el suavizado denominada “suavizado” que se coloca en true.

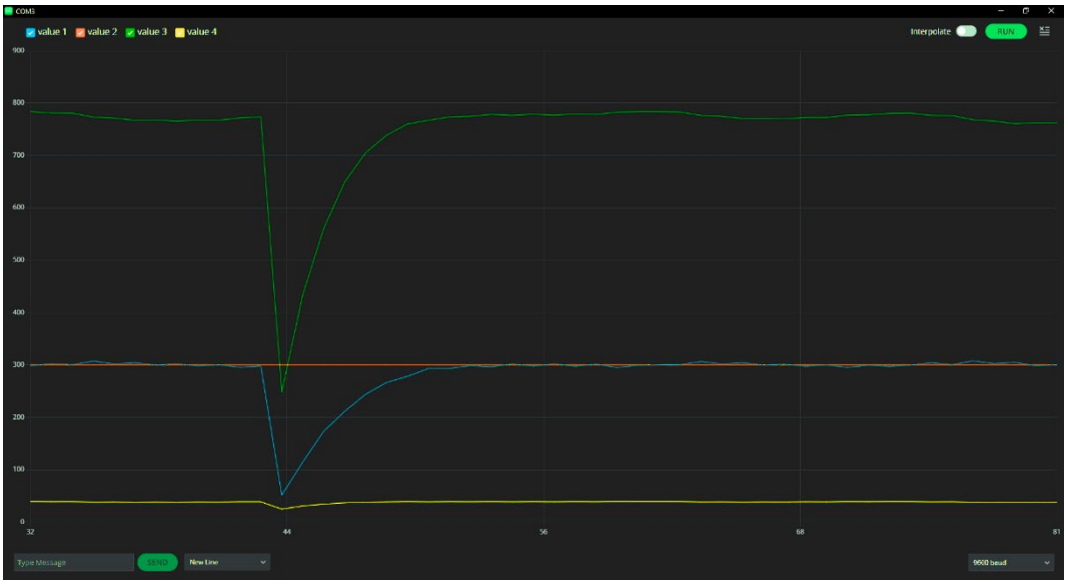

Ilustración 64. Respuesta del sistema con suavizado activado.

Se ha podido comprobar cómo el resultado es lineal sin variaciones ni escalones en el incremento del valor. A continuación, se va a mostrar el funcionamiento de la Onda cuadrada donde también se comprobará la diferencia entre contar con el suavizado y no tenerlo activado.

5.3.3. Onda cuadrada

En múltiples ocasiones tiene una situación en la que el valor deseado no sea siempre el mismo y por lo tanto varíe entre valores. Por ese motivo en este programa se puede activar una onda cuadrada para el valor deseado que varíe entre dos puntos, dos valores asignados por el usuario.

Como ejemplo se ha construido una onda que cambie de valor cada dos segundos desde un valor de 100 hasta 600. El funcionamiento sin suavizado es el que se muestra en la siguiente ilustración.

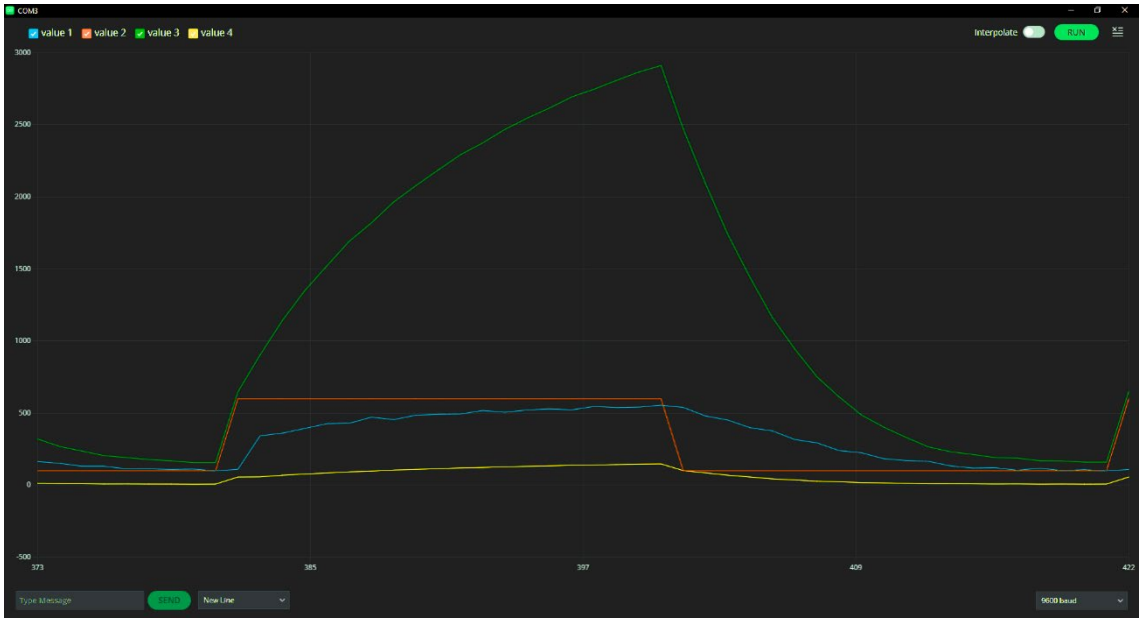

Ilustración 65. Resultado del sistema con Onda cuadrada sin suavizado.

El sistema se adapta correctamente al valor deseado asignado, pero se observa pequeños escalones en el incremento y decremento del valor de la salida. Esto es perjudicial para el funcionamiento del LED puesto que provoca intermitencias. A continuación, activamos el suavizado obteniendo el funcionamiento que se muestra en la ilustración siguiente.

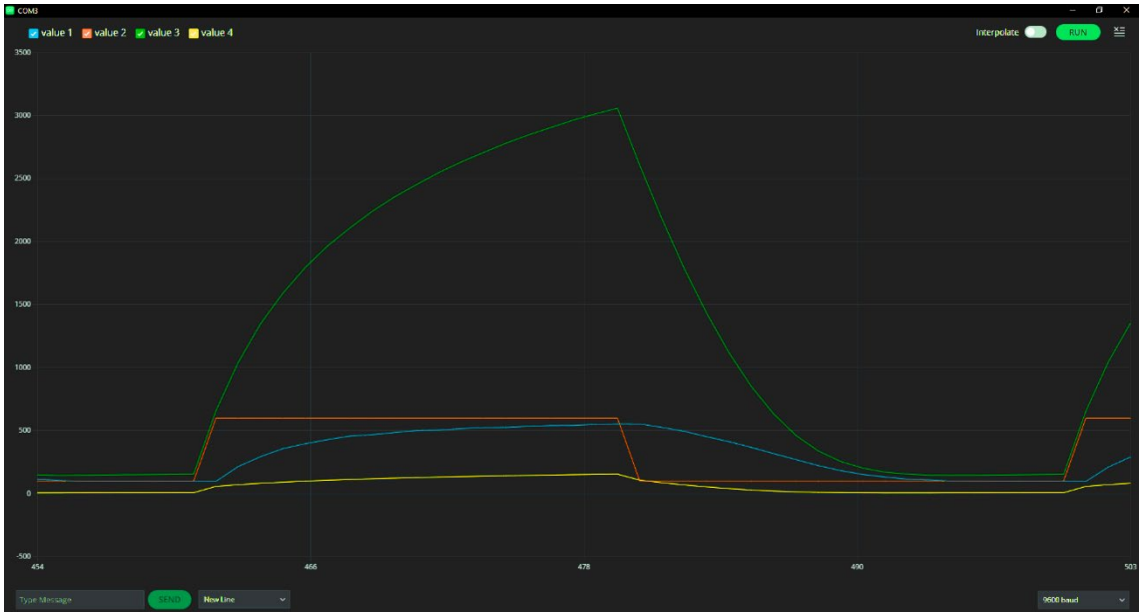

Ilustración 66. Resultado del sistema con Onda cuadrada con suavizado

El funcionamiento del sistema no tiene ningún escalón ni irregularidad por lo que el aumento y decremento de la luz es suave y correcto.

5.3.4. Anti-Windup

En este apartado se va a comprobar realizar el análisis del funcionamiento del Anti-Windup, para ello se mostrará el resultado del sistema sin aplicar el Anti-Windup con una onda cuadrada y sin onda cuadrada y después se compararán los resultados obtenidos aplicando el Anti-Windup al sistema.

Colocaremos la onda cuadrada en un intervalo de valores de entre 1000 y 100 para que el sistema sature y se pueda observar el funcionamiento.

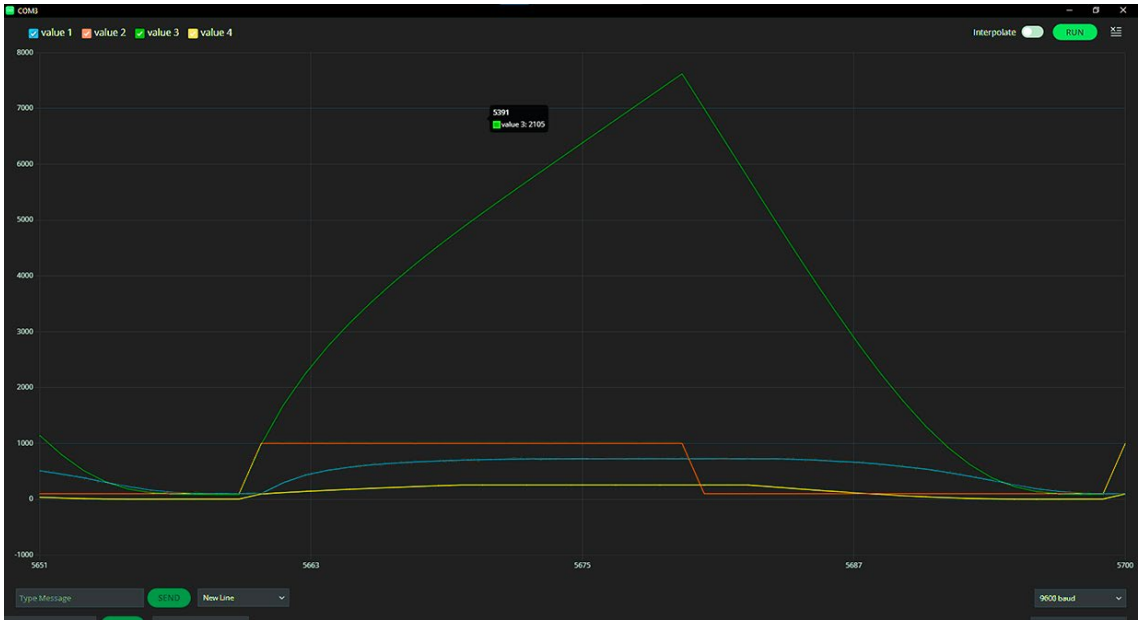

Ilustración 67. Respuesta del sistema con Onda Cuadrada sin Anti-Windup

En la ilustración anterior el sistema se satura y el valor integral sube hasta un valor de 7600 el cual tiene que recuperar en el siguiente ciclo de la onda cuadrada y debido al valor acumulado no llega a tiempo y provoca un funcionamiento incorrecto del sistema.

Colocando un valor de consigna o referencia de 1000 para que el sistema sature comprobaremos la respuesta del sistema cuando no se aplica la onda cuadrada.

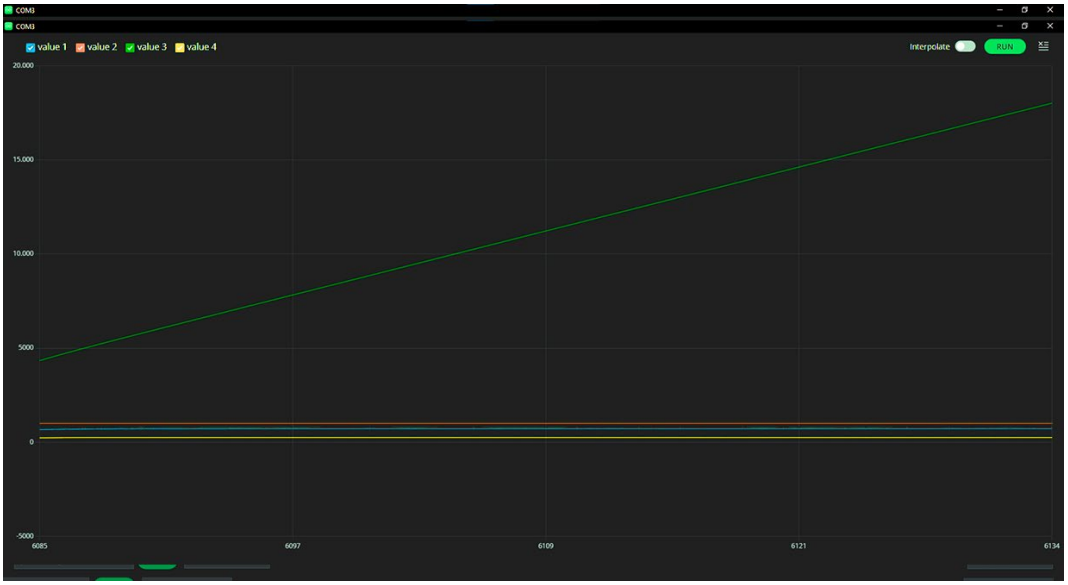

Ilustración 68. Respuesta del sistema sin Onda Cuadrada sin Anti-Windup.

El sistema acumula un valor integral constante cuando se ha saturado. Esto es un problema si existiese cualquier perturbación externa que provocase un valor inferior de la iluminación del LED puesto que el sistema tiene que recuperar el valor acumulado durante un gran intervalo de tiempo.

En este instante se activa el Anti-Windup y comparamos los resultados obtenidos colocando la referencia en los mismos valores para la onda cuadrada y a su vez cuando no se ha aplicado.

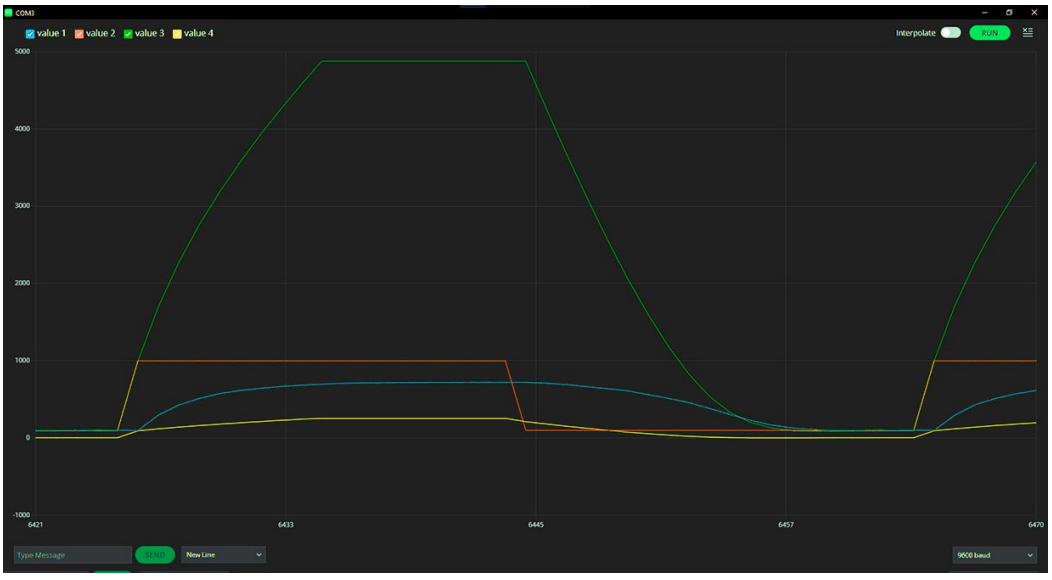

Ilustración 69. Respuesta del sistema con Onda Cuadrada con Anti-Windup

Se puede apreciar a la perfección la limitación que provoca el Anti-Windup en la ilustración anterior cuando el sistema ha saturado y el sistema recupera el valor de inmediato cuando la onda cuadrada cambia su valor de 1000 a 100. El valor integral deja de acumular valor cuando se detecta la saturación del sistema obteniendo un funcionamiento correcto.

Ahora observamos el resultado cuando no existe onda cuadrada y se alcanza la saturación del sistema en la ilustración que se muestra a continuación.

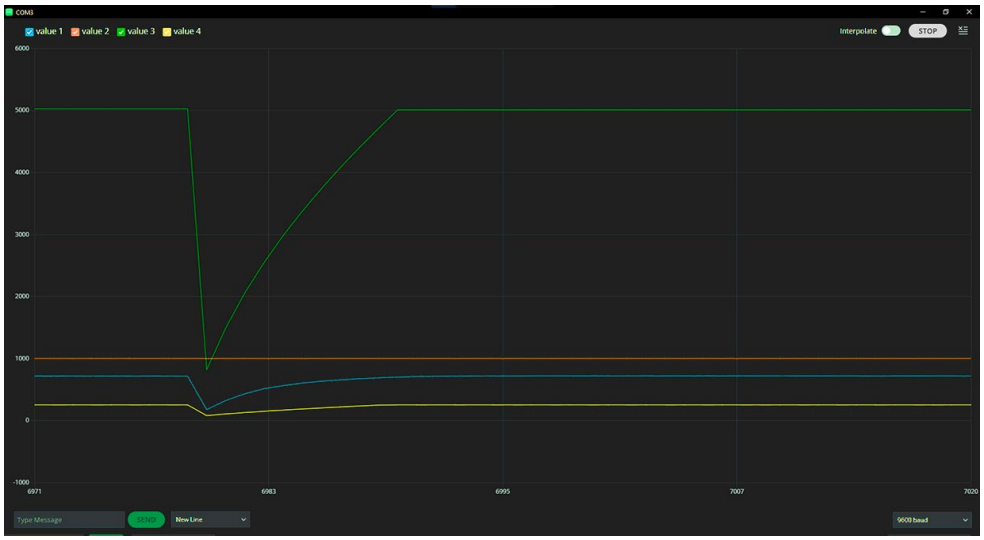

Ilustración 70. Respuesta del sistema sin Onda Cuadrada con Anti-Windup.

Cuando el sistema ha alcanzado la saturación el valor integral deja de acumular valor y si existiese cualquier perturbación el sistema recuperaría el valor correspondiente de inmediato para que el funcionamiento del sistema sea perfecto.

En el siguiente apartado se realizarán algunas comprobaciones cuando se aplica una perturbación exterior como puede ser la luz que entra cuando la tapa de la caja se abre.

5.3.5. Perturbaciones externas (Luz exterior)

En ocasiones se pueden producir perturbaciones producidas por la luz externa. En este caso vamos a observar cómo es la respuesta del sistema cuando se recibe luz exterior mientras el sistema se autorregula con una onda cuadrada.

Ilustración 71. Respuesta del sistema con perturbaciones externas con onda cuadrada.

El funcionamiento del sistema se ajusta ante variaciones en la luz externa, adaptándose a la variación del setpoint a su vez. A continuación, desactivamos la Onda Cuadrada para observar cómo varía la respuesta del sistema.

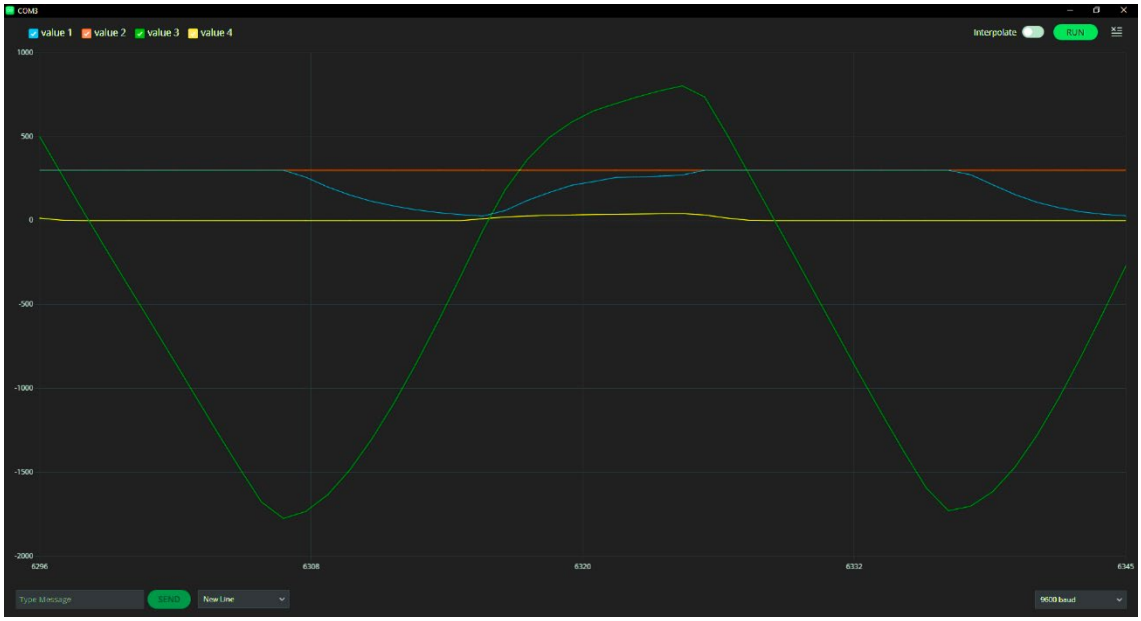

Ilustración 72. Respuesta del sistema con perturbaciones externas sin onda cuadrada.

Cuando se ilumina el sensor de luz, el valor se convierte en 0 pero la variable integral sigue acumulando error aun estando la entrada saturada. Esto provoca el efecto Windup que se comentaba a lo largo del proyecto, es decir, el valor del error integral incrementa y cuando la salida tiene que alcanzar el valor deseado se tiene que esperar un tiempo hasta que se recupera todo el error acumulado en el intervalo de tiempo que la entrada ha estado saturada.

Este efecto se soluciona como se ha comentado a lo largo de este proyecto. Si activamos el Anti-Windup, obtenemos el siguiente resultado.

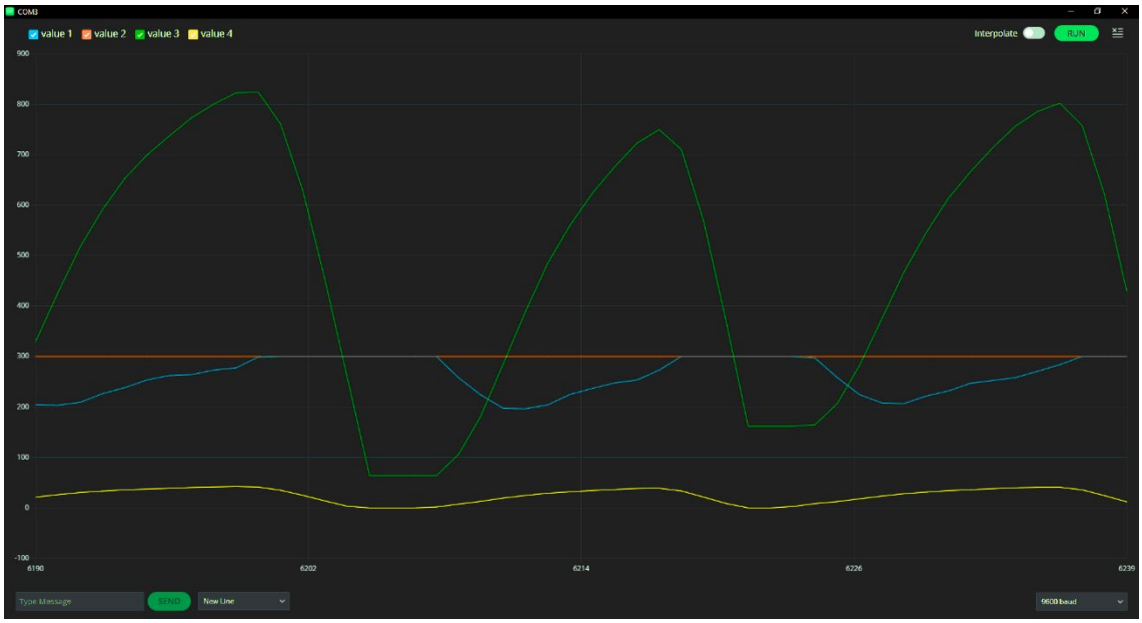

Ilustración 73. Respuesta del sistema con perturbaciones externas sin onda cuadrada con Anti-Windup.

El valor de la variable integral (color verde) deja de acumular error cuando la variable de la entrada está saturada, de esta manera la respuesta de la salida es inmediata y el funcionamiento es el deseado.

Ilustración 74. Respuesta del sistema con perturbaciones externas con onda cuadrada con Anti-Windup.

Al inicio de la ilustración anterior se ha iluminado el sensor con luz externa provocando un aumento en la recepción, el controlador PI compensa esa iluminación para lograr el valor deseado de nuevo sin sufrir efecto Windup.

Podemos concluir las pruebas de funcionamiento con un resultado correcto, deseado y buscado con el programa planteado para su resolución tras realizar las comprobaciones anteriores oportunas.

6. Conclusiones y Mejoras posibles.

En el apartado número 2 de este documento se proporcionaban los objetivos que se pretendían cumplir a lo largo de este proyecto.

En primer lugar, se ha realizado un diseño y un sistema que regule la luminosidad de una estancia utilizando Arduino en el que se incorporase un controlador PID eficiente. Para lograr un controlador PID eficiente se realizó un análisis teórico a partir de la respuesta en lazo abierto del sistema para más tarde optimizar el sistema observando el comportamiento de la salida del sistema cerrando el lazo con el controlador PI obtenido teóricamente. Se realizó un ajuste de las componentes del controlador hasta lograr un funcionamiento lo más correcto posible.

Para continuar se ha realizado un análisis completo del efecto Windup que se produce cuando el sistema llega a saturación consiguiendo solventar el problema introduciendo un Anti-Windup para que el sistema no acumule error y el sistema se comporte de manera correcta.

En tercer lugar, se ha realizado un análisis experimental de los resultados obtenidos a partir del programa y el sistema construido en este trabajo mostrando los resultados, los efectos que produce el efecto Windup como también el producido por las perturbaciones exteriores de la luz exterior.

Para terminar, se ha documentado el progreso del trabajo y los métodos de resolución de los problemas encontrados junto con la presentación de los resultados obtenidos en cada caso.

Este proyecto está enfocado para utilizarse en la docencia y poder mostrar el efecto que se produce cuando el sistema satura y aparece el efecto Windup. Se ha diseñado para brindar la posibilidad de modificar tanto los valores de la consigna, como los valores del controlador PID como la ejecución del programa y sus funciones, proporcionando así una comprensión integral del funcionamiento del sistema y de los diversos aspectos inherentes a la aplicación del controlador PID.

A su vez se ha realizado un proyecto de diseño de una placa de circuito impreso muy sencillo de instalar en la placa de Arduino que se construye de manera compacta para que resista movimientos y posibles golpes en las aulas, evitando tener cables.

Algunos de los aspectos que podrían mejorar este proyecto es que utilizando Arduino el funcionamiento del sistema en tiempo real podría ser más eficiente ya que este microcontrolador no permite una medida precisa del tiempo y la velocidad es limitada si se pretendiese realizar un proyecto más complejo.

También se produce un funcionamiento lento cuando se activa el Anti-Windup con onda cuadrada y el sistema satura, se debería conseguir una recuperación del valor del LDR inmediata para que el LED funcionase a la velocidad coherente.

En conclusión, se ha realizado un trabajo completo tanto en diseño como en programa obteniendo un funcionamiento perfecto, aunque con ligeras posibles mejoras, del sistema a partir del objetivo planteado al inicio del proyecto.

7. Bibliografía.

▷ *Anti Windup en un Control PID - [Detallado]*. (s. f.). Recuperado 26 de noviembre de 2023, de https://controlautomaticoeducacion.com/control-realimentado/anti-windup-en-un-control-pid/#Anti_Windup_en_Codigo

▷ *Como Hacer un Control PID Arduino de Temperatura*. (s. f.-a). Recuperado 12 de noviembre de 2023, de https://controlautomaticoeducacion.com/arduino/control-pid-de-temperatura-con-arduino/#Que_es_el_controlador_PID

▷ *Como Hacer un Control PID Arduino de Temperatura*. (s. f.-b). Recuperado 21 de noviembre de 2023, de https://controlautomaticoeducacion.com/arduino/control-pid-de-temperatura-con-arduino/#Control_PID_de_Temperatura_con_Arduino_sin_Libreria

▷ *Todo sobre Ziegler Nichols - Sintonia de Control PID*. (s. f.). Recuperado 21 de noviembre de 2023, de https://controlautomaticoeducacion.com/control-realimentado/ziegler-nichols-sintonia-de-control-pid/#Sintonia_de_Controladores_PID_usando_Ziegler-Nichols

5 reglas de diseño de PCB importantes que debes saber | Altium. (s. f.). Recuperado 29 de noviembre de 2023, de https://resources.altium.com/es/p/pcb-layout-guidelines

(186) Como hacer Pistas Curvas en PROTEUS - YouTube. (s. f.). Recuperado 30 de noviembre de 2023, de https://www.youtube.com/watch?v=NCQfGew-Rqo

(186) How to Add Arduino UNO Footprint PCB Package on Proteus 8 | R1 | R2 | R3 - YouTube. (s. f.). Recuperado 28 de noviembre de 2023, de https://www.youtube.com/watch?v=pM3QlAgiROM

(186) HOW TO MAKE A PCB (PRINTED CIRCUIT BOARD) | PROTEUS SOFTWARE | D&R TUTORIALES - YouTube. (s. f.). Recuperado 27 de diciembre de 2023, de https://www.youtube.com/watch?v=78GVsMHJnPg

Arduino Filtro pasa bajo Media Móvil Exponencial | Eliminar ruido - YouTube. (s. f.). Recuperado 26 de noviembre de 2023, de https://www.youtube.com/watch?v=QGDG5v_UnIk

Arduino: Guía completa para principiantes y expertos | Aprende ya. (s. f.). Recuperado 30 de noviembre de 2023, de https://www.cursosaula21.com/arduino-todo-lo-que-necesitas-saber/

Arduino Reference - Arduino Reference. (s. f.). Recuperado 3 de diciembre de 2023, de https://www.arduino.cc/reference/en/

Arduino Uno - Wikipedia. (s. f.). Recuperado 27 de noviembre de 2023, de https://en.wikipedia.org/wiki/Arduino_Uno

ArduinoUnoFP.pdsprj - Google Drive. (s. f.). Recuperado 27 de diciembre de 2023, de https://drive.google.com/file/d/1ZmVmnkk4Rc2AOfGa5PdfyZ0IwkfcdIND/view?pli=1

Conoce los tipos de resistencias electrónicas | Aprende Institute. (s. f.). Recuperado 3 de diciembre de 2023, de https://aprende.com/blog/oficios/instalaciones-electricas/tipos-de-resistencias-electronicas/

CONTROLADOR PI - Asignación de Polos [FÁCIL - Aprende]. (s. f.). Recuperado 24 de noviembre de 2023, de https://controlautomaticoeducacion.com/control-realimentado/controlador-pi-por-asignacion-de-polos/

ESP32 - Wikipedia, la enciclopedia libre. (s. f.). Recuperado 27 de noviembre de 2023, de https://es.wikipedia.org/wiki/ESP32

Filtro paso bajo y paso alto exponencial (EMA) en Arduino. (s. f.-a). Recuperado 26 de noviembre de 2023, de https://www.luisllamas.es/arduino-paso-bajo-exponencial/

Fotoresistor, LDR o Fotoresistencia — MecatrónicaLATAM. (s. f.). Recuperado 3 de diciembre de 2023, de https://www.mecatronicalatam.com/es/tutoriales/sensores/sensor-de-luz/ldr/

Fotorresistor - Wikipedia, la enciclopedia libre. (s. f.). Recuperado 2 de diciembre de 2023, de https://es.wikipedia.org/wiki/Fotorresistor

Led - Wikipedia, la enciclopedia libre. (s. f.). Recuperado 2 de diciembre de 2023, de https://es.wikipedia.org/wiki/Led

Microcontrolador - Wikipedia, la enciclopedia libre. (s. f.). Recuperado 27 de noviembre de 2023, de https://es.wikipedia.org/wiki/Microcontrolador

Microcontrolador PIC - Wikipedia, la enciclopedia libre. (s. f.). Recuperado 27 de noviembre de 2023, de https://es.wikipedia.org/wiki/Microcontrolador_PIC

Nice drawings of the Arduino UNO and Mega 2560 | Arduino Blog. (s. f.). Recuperado 27 de diciembre de 2023, de https://blog.arduino.cc/2011/01/05/nice-drawings-of-the-arduino-uno-and-mega-2560/

¿Por qué las trazas de PCB no pueden ir en un ángulo recto de 90 grados? - Conocimiento - St.Qin Cross Boarder E-Commerce Co., Ltd. (s. f.). Recuperado 30 de noviembre de 2023, de https://es.jhd-material.com/info/why-can-t-the-pcb-traces-go-at-a-right-angle-o-70485034.html

Printed circuit board - Wikipedia. (s. f.). Recuperado 26 de diciembre de 2023, de https://en.wikipedia.org/wiki/Printed_circuit_board

Proportional–integral–derivative controller - Wikipedia. (s. f.). Recuperado 25 de diciembre de 2023, de https://en.wikipedia.org/wiki/Proportional%E2%80%93integral%E2%80%93derivative_controller

¿Qué es un sistema de control? - AUTYCOM. (s. f.). Recuperado 10 de noviembre de 2023, de https://www.autycom.com/que-es-un-sistema-de-control/

Sistema de control - Wikipedia, la enciclopedia libre. (s. f.). Recuperado 10 de noviembre de 2023, de https://es.wikipedia.org/wiki/Sistema_de_control

STMicroelectronics - Wikipedia, la enciclopedia libre. (s. f.). Recuperado 27 de noviembre de 2023, de https://es.wikipedia.org/wiki/STMicroelectronics

Tutorial Arduino: IDE Arduino | OpenWebinars. (s. f.). Recuperado 1 de diciembre de 2023, de https://openwebinars.net/blog/tutorial-arduino-ide-arduino/

8. Anexos

8.1. Código Arduino completo

```
/* ----------CONTROL LUMINOSIDAD----------*/
/*
1. Seleccionar true o false en las booleanas para activar opciones
2. Seleccionar el valor del PID en Kp, Ki y Kd
3. Elegir valor de la consigna en setpoint
4. Elegir valor del setpoint en la onda cuadrdad en la funcion void Onda_cuadrada()
----------*/

// Definicion de constantes del controlador PID
double Kp = 0.05;    // Componente proporcional
double Ki = 0.05;    // Componente integral
double Kd = 0;    // Componente derivativa

// Define variables
double setpoint = 300.0;  // Lectura deseada para el LDR, se ajusta dependiendo el sistema
double entrada = 0; //Valor del LED
double salida = 0; //Valor medido por el LDR
double proporcional = 0; //Valor accion proporcional
double comp_integral = 0; //Valor accion integral
double integral = 0; //Valor componente integral
double comp_derivativo = 0; // Valor accion derivativa
double derivativo = 0; //Valor constante derivativa
double error = 0; // Valor del error actual
double lastError = 0; //Valor del error anterior

//Variables Booleanas para activar y desactivar secciones
bool Onda_Si = false; //Si el valor es true, se ejecuta una onda cuadrada en la funcion Onda_cuadrada se encuentran los parametros
bool suavizado = true; //Ejecutar un suavizado en el ruido que se genera en la lectura del LDR, true para aplicarlo, false para desactivarlo
bool anti = true; //Variable booleana que activa y desactiva el Anti-Windup
/***True = Activado, False = desactivado***/

//Variables para Onda Cuadrada.
unsigned long tiempo_inicial = 0;
double tiempo_muestreo = 0.9;
unsigned long periodo = 4000; //Segundos entre valores de la onda

void Onda_cuadrada(){
  unsigned long actual_time = millis();
```

```
39.   unsigned long tiempo_demorado = actual_time - tiempo_inicial;
40.
41.   if((tiempo_demorado % periodo) < (periodo/2)) //Vemos en que
    modulo de 0 al valor del periodo nos encontramos para poder
    establecer la onda en alto o bajo
42.   {
43.     setpoint = 6000;
44.   }else
45.   {
46.     setpoint = 100;
47.   }
48. }
49.
50. // Define pins
51. const int ldrPin = A0;  // LDR conectado al pin A0
52. const int ledPin = 6;   // LED conectado al pin 6
53.
54. void setup() {
55.   pinMode(ledPin, OUTPUT); //Inicializamos el LED como salida en el
    Arduino
56.   analogWrite(ledPin, 0); //Comenzamos con el LED apagado
57.   pinMode(ldrPin, INPUT); //Inicializamos el LDR como entrada en el
    Arduino
58.   Serial.begin(9600); //Para la comunicacion serial
59.
60.   tiempo_inicial = millis(); //Para el control del tiempo
61. }
62.
63. void loop() {
64.
65.   //Contamos el tiempo que se tarda en realizar cada bucle
66.   double tiempo_inicio = millis();
67.   if (Onda_Si == true)  //True = Activada
68.   {
69.     Onda_cuadrada();
70.   }
71.   //Suavizado
72.   if (suavizado == true) {
73.     double factorSuavizado = 0.45;  // Factor de suavizado
74.     double nuevaLectura = analogRead(ldrPin);
75.
76.     salida = (1 - factorSuavizado) * salida + factorSuavizado *
    nuevaLectura;
77.   } else {
78.     salida = analogRead(ldrPin);  //Lectura del LDR
79.   }
80.
```

```
81.  // Calculo del error
82.  error = setpoint - salida;
83.
84.  // Calculo del valor P proporcional
85.  proporcional = Kp * error;
86.
87.  //Anti Wind-up
88.  if (anti == true)
89.  {
90.    if(entrada > 0 && entrada < 255)
91.    {
92.        integral += error;
93.    }else{}
94.  }else
95.  {
96.    integral += error;
97.  }
98.
99.  //Calculo de la componente I integral
100.       comp_integral = Ki * integral;
101.
102.       //Calculo de la componente D derivativa
103.       derivativo = error - lastError;
104.       comp_derivativo = Kd * derivativo;
105.
106.       // Calculamos la entrada, la ecuacion del PID
107.       entrada = proporcional + comp_integral + comp_derivativo;
   //u(k) teorico
108.
109.       // Convertimos el valor a un rango de 0 a 255 para el PWM
110.       entrada = constrain(entrada, 0, 255);
111.
112.       // Entrada al LED
113.       analogWrite(ledPin, entrada);
114.
115.       lastError = error; //En el siguiente bucle el error actual
   es el anterior
116.
117.       //Salida por pantalla y graficos
118.       Serial.print(salida); //value 1
119.       Serial.print(",");
120.       Serial.print(setpoint); //value 2
121.       Serial.print(",");
122.       Serial.print(integral); //3
123.       Serial.print(",");
124.       Serial.println(entrada); //value 4
125.
```

```
126.         // Ponemos un retardo ajustado a la planta
127.         delay(100);
128.       }
```