

LABORATORIO DE CONTROL



GRUPO N°3

TÍTULO: SEGUIDOR SOLAR (SOLAR TRACKER)

INTEGRANTES PRESENTES EL DÍA QUE SE REALIZÓ

Sanchez

Samuel

	FECHAS	FIRMA Y ACLARACIÓN DEL DOCENTE
REALIZADO EL	26/11/2024	
CORREGIDO		
APROBADO		

INDICACIONES PARA LAS CORRECCIONES:



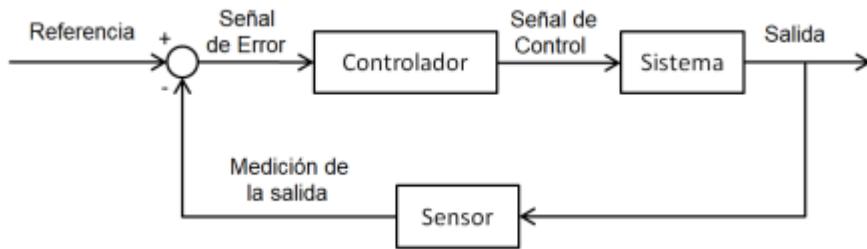
ÍNDICE.....	1
OBJETIVOS.....	2
INTRODUCCIÓN.....	2
ARQUITECTURA DEL SISTEMA.....	3
● REFERENCIA Y SALIDA:.....	3
● SISTEMA.....	3
○ Motores paso a paso (STEPPER):.....	3
○ Módulo CNC A4988:.....	4
● CONTROLADOR.....	5
● PLANTA:.....	5
● SENSOR.....	6
○ Módulo LDR.....	6
IMPLEMENTACIÓN DEL SISTEMA.....	7
● Hardware:.....	7
○ Lista de materiales utilizados.....	7
○ Lista de conexiones eléctricas al Arduino:.....	7
● Piezas adicionales fabricadas:.....	7
● Software:.....	8
● Programas de prueba:.....	8
● Seguidor sin PI:.....	8
● Seguidor con P:.....	8
● Seguidor con PI:.....	8
● Seguidor con PID con constantes ajustables por MONITOR SERIAL:.....	8
● Seguidor con PID (velocidad como variable de control):.....	9
PRUEBAS Y RESULTADOS.....	9
● Prueba de Sensores.....	9
● Prueba de Motores Paso a Paso.....	9
● Prueba de Integración y de control P, y PI, PID.....	9
CONCLUSIONES.....	11
REFERENCIAS.....	11

OBJETIVOS

- Producir un acercamiento a la implementación de sistemas de control con microcontroladores.
- Relacionar los conceptos incorporados a lo largo de la materia con aplicaciones concretas.
- Comprender los ámbitos de aplicación de los sistemas de control continuos.
- Adquirir habilidad en el manejo de sensores y actuadores de distinto tipo.

INTRODUCCIÓN

Este proyecto se enfoca en diseñar e implementar un sistema de control para un seguidor solar que utilice un lazo de control cerrado y permita ajustar la posición del panel solar en tiempo real. El sistema debe garantizar la estabilidad, la precisión en el seguimiento y la capacidad de ajustar parámetros del controlador.

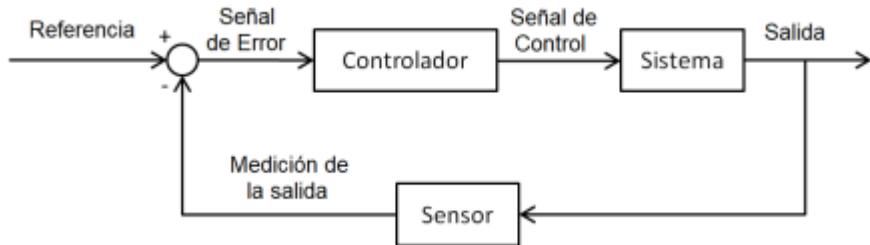


La importancia de las energías renovables ha impulsado el desarrollo de tecnologías que maximizan la eficiencia en la captación de recursos como la luz solar. Los seguidores solares son dispositivos mecánicos que ajustan la orientación de un panel solar para optimizar la captación de energía, alineándolo con la posición del sol en el cielo.



El seguimiento preciso del sol mejora significativamente la eficiencia energética de los paneles solares, lo que hace que este proyecto tenga aplicaciones prácticas y gran relevancia en la transición hacia sistemas de energía sostenible. Además, el diseño e implementación del sistema de control permite aplicar conceptos fundamentales de la teoría de control en un entorno práctico, facilitando el aprendizaje y el desarrollo de habilidades técnicas esenciales.

ARQUITECTURA DEL SISTEMA



El sistema estará basado en la estructura clásica de un lazo cerrado de control, que incluye:

- **REFERENCIA Y SALIDA:**

En nuestro sistema consideramos la referencia como la intensidad lumínica determinada entre Este y Oeste y entre Norte y Sur, y la salida será la intensidad leída a determinada posición. Por lo que buscamos que el sistema se acomode hasta llegar que el error de las intensidades medidas sea nulo.

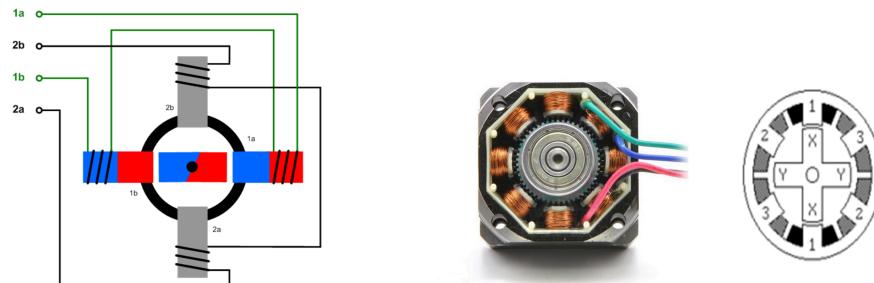
- **SISTEMA**

Al observar el sistema notamos que está compuesto, por una parte, mecánica, la cual está compuesta por varios elementos como ejes de transmisión, un sistema de correa y un sistema de engranaje, y, por otra parte, eléctrica, en la que se hace uso de un motor paso a paso junto a su driver de control, el módulo para Arduino A4988.

- **Motores paso a paso (STEPPER):**

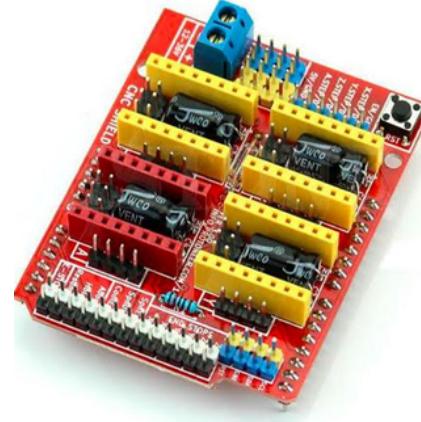
Un motor paso a paso (Step Motor o simplemente Stepper) es un tipo de motor eléctrico diseñado para convertir pulsos eléctricos en movimientos discretos y controlados. Cada pulso aplicado al motor equivale a un movimiento angular fijo conocido como paso, lo que lo hace ideal para aplicaciones que requieren alta precisión y control de posición.

Estos funcionan mediante un conjunto de bobinas organizadas en fases. Al energizar las bobinas en una secuencia específica, se genera un campo magnético rotativo que atrae los polos del rotor. Esto hace que el motor gire en pasos discretos. El ángulo de cada paso depende del número de polos en el rotor y el número de fases del motor.

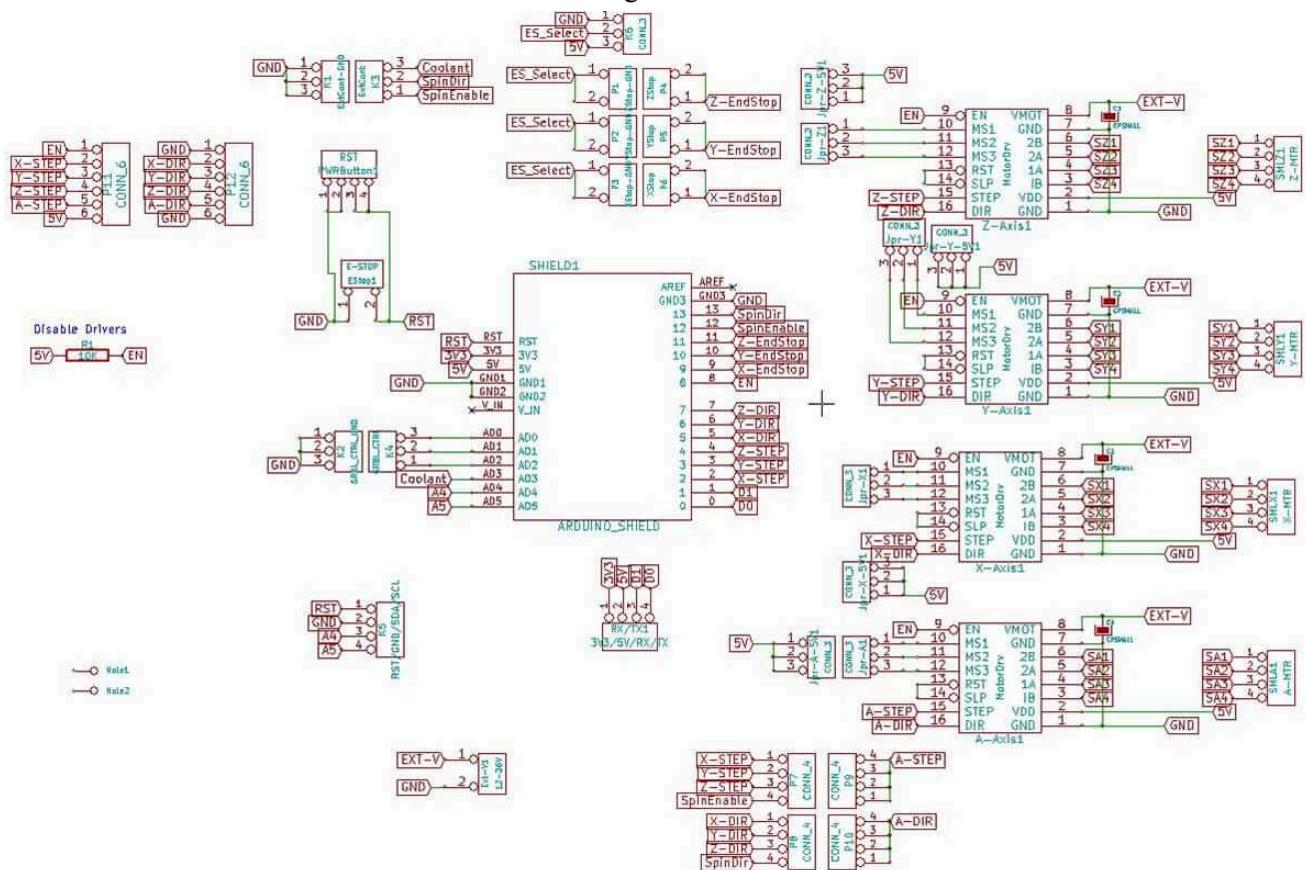


- **Módulo CNC A4988:**

Es un controlador de motores paso a paso que traduce señales de entrada (pulsos) en movimientos precisos del motor. Funciona con el principio de microstepping, lo que permite un control suave y preciso del motor al dividir los pasos completos en fracciones.



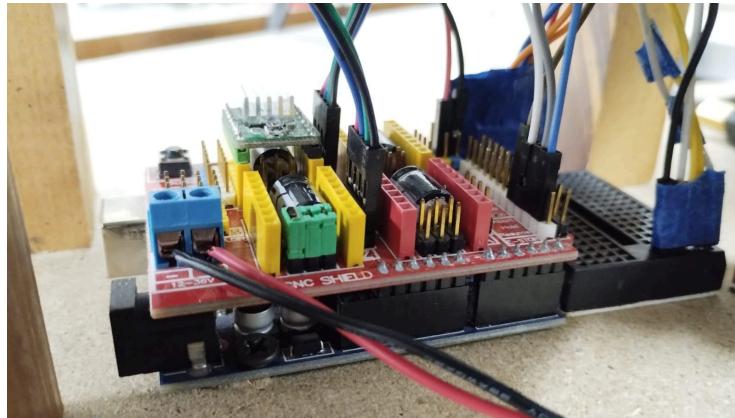
Siendo sus conexiones internas las siguientes:



En el cual se puede apreciar que si necesitamos usarlo para controlar los motores paso a paso debemos usar los pines 2 hasta 7 para controlar la dirección y paso del motor, además observamos que debemos poner el pin **ENABLE** en un estado bajo (0 V) y por ultimo, para utilizar las salidas analógicas (A0 a A3) debemos usar los pines **Coolant**, **Abort**, **Hold**, y **Resume**, más adelante hablaremos de los sensores usados.

- **CONTROLADOR**

Utilizaremos arduino como controlador, el cual es una plataforma de hardware y software de código abierto diseñada para facilitar la creación de proyectos de electrónica y sistemas embebidos. Su principal ventaja radica en su facilidad de uso, versatilidad y amplia comunidad de usuarios, lo que lo convierte en una herramienta ideal tanto para principiantes como para profesionales.

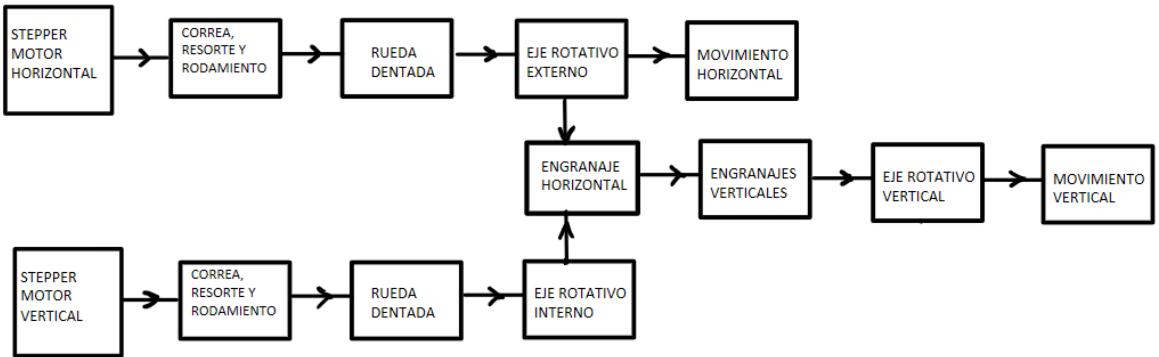


- **PLANTA:**

Si inspeccionamos el sistema, podemos observar que a partir de los motores paso a paso se encuentra un sistema de poleas, en las cuales se encuentran unos rodamientos y unos resortes para mantenerla tensada, conectada a una rueda y luego a un eje de transmisión INTERNO y EXTERNO, los cuales están conectados por a una serie de engranajes y un eje vertical que permiten el giro y orientación del equipo.



Representado en un diagrama sería visto de la siguiente manera.



• SENSOR

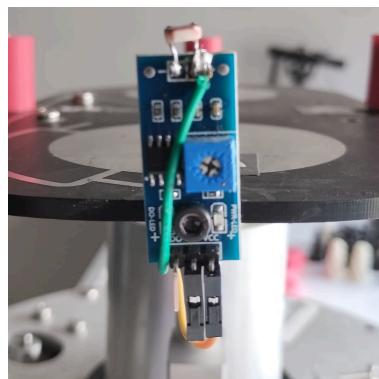
Para seleccionar el sensor apropiado para nuestro sistema, es fundamental que este sea capaz de medir con precisión la intensidad lumínica irradiada. Esto nos permitirá identificar variaciones en la iluminación y ajustar la orientación del sistema de manera efectiva.

Para lograr una corrección óptima en la posición, se emplearán cuatro sensores LDR (resistencias dependientes de luz), los cuales estarán dispuestos de forma estratégica en diferentes puntos del sistema. Cada sensor detectará la intensidad lumínica en su entorno inmediato, proporcionando datos sobre la distribución de luz. Con esta configuración, es posible determinar la dirección en la que el sistema debe orientarse para optimizar su rendimiento.

- **Módulo LDR**

Un módulo LDR (Light Dependent Resistor) es un sensor diseñado para medir la intensidad lumínica. El componente principal es una fotorresistencia que varía su resistencia eléctrica en función de la cantidad de luz que incide sobre su superficie.

En un sistema Arduino, el módulo LDR permite realizar mediciones analógicas, generando un valor proporcional a la cantidad de luz detectada. Estos datos son utilizados comúnmente en aplicaciones como seguidores solares, sistemas de iluminación automatizada y dispositivos de monitoreo ambiental.



*Nota: Para optimizar el diseño, se conectó un cable al LDR.

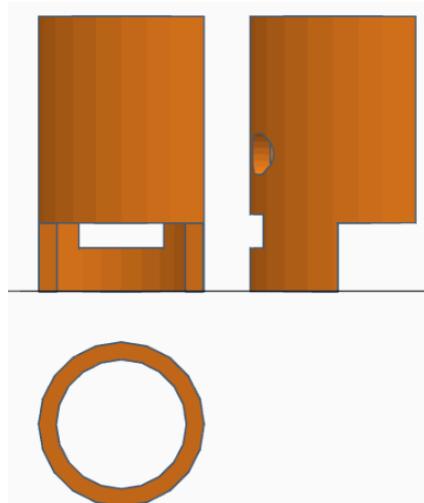
IMPLEMENTACIÓN DEL SISTEMA

- **Hardware:**

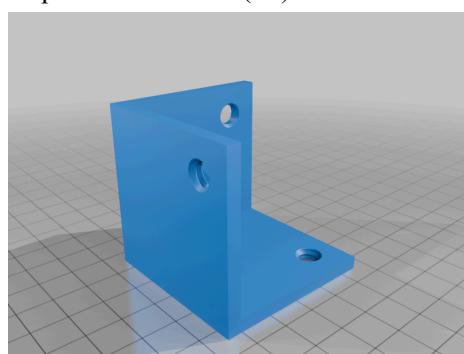
- **Lista de materiales utilizados**
 - 1 Arduino UNO.
 - 1 Módulo A4988.
 - 4 Módulos sensor “LDR”.
 - 2 motores “PAP”.
 - Cable UTP.
 - Fuente de alimentación de 12V fija.
- **Lista de conexiones eléctricas al Arduino:**
 - LDR W - A0 (Abort)
 - LDR S - A1 (Hold)
 - LDR N - A2 (Resume)
 - LDR E - A3 (Coolant)
 - Step X - 2
 - Step Z - 4
 - Dir X - 5
 - Dir Z - 7
 - Enable - 8

- **Piezas adicionales fabricadas:**

- Encapsulado para sensores LDR (x4):



- Esquinas cuadradas (x4):





- **Software:**

Todos los programas generados se encuentran en el siguiente enlace: (TC 2024) Trabajo integrador
https://drive.google.com/drive/folders/1MWLOZYS5MP7zwh9b_oK074x0MlalyIVTa?usp=drive_link

- **Programas de prueba:**

Consiste en una serie de programas en los cuales se puso a prueba la planta original, permitiendo entender como funcionan los elementos de la misma, como los motores, los ldrs y como funciona la librería AccelStepper. Sin embargo, esta librería tuvo la desventaja que no se puede ajustar la velocidad a medida que avanza el seguidor.

- **Seguidor sin PI:**

Con los conocimientos obtenidos en el programa de prueba, ensayamos el sistema con un programa en el cual comparamos la intensidad recibida entre los sensores, y realizamos pequeños pasos fijos de antemano para orientar el seguidor frente a un margen de 5 unidades entre las comparaciones de los sensores. Resultando en un movimiento lento junto a un sistema muy ruidoso e inestable, ya que en ciertas ocasiones comenzaba a moverse descontroladamente.

- **Seguidor con P:**

Reescribimos el código anterior y aplicamos los conocimientos de errores y controladores vistos en la materia. Luego de definir el error como la diferencia entre dos sensores y con la ayuda de una constante Kp calculamos la cantidad de pasos que los motores deben realizar. Esto mejoró el movimiento del sistema, resultando en un mejor seguimiento que mejoraba al variar la ganancia Kp, pero una variación excesiva produce movimientos no deseados en el seguimiento.

- **Seguidor con PI:**

Tomamos como referencia el código anterior y le implementamos un control integrativo. Definimos la ganancia integrativa Ki, junto a las variables de tiempo y modificamos el cálculo de pasos. Luego de ajustar ambas constantes observamos que el sistema se orienta mucho mejor en comparación con el controlador anterior. Aunque el seguimiento era correcto, resultaba que tardaba demasiado tiempo en orientarse, de hecho, no era posible que a mayor error los motores se muevan más rápido, dejando inútil el uso de la librería.

- **Seguidor con PID con constantes ajustables por MONITOR SERIAL:**

Descartamos los programas anteriores junto a la librería Accelstepper, y rescribimos el programa de manera que los motores puedan moverse cada ciertos pulsos de determinados tiempos. Además, implementamos un suavizado mediante un promedio en la lectura en los ldrs, y mediante el monitor Serial mostramos los errores medidos los cuales son graficados en el Serial Plotter y podemos ingresar las constantes del mismo las cuales pueden ser ajustadas mediante los siguientes comandos.

- + **p(valor):** ajusta la constante kp.
- + **i(valor):** ajusta la constante ki.
- + **d(valor):** ajusta la constante kd.

Resultando en un programa con gran capacidad para supervisar como se comporta el sistema y visualizar como el error se comporta frente a distintos valores de ganancias.



- **Seguidor con PID (velocidad como variable de control)**

Descartamos los pasos como señal de control del sistema y lo reemplazamos por un ajuste de velocidad que debe utilizar. Este cambio de paradigma fue vital, ya que el seguimiento en los programas anteriores poseía problemas de sobre-pasos, esto era debido a que ante un error relativamente grande resultaba en un número grande de pasos. Por lo tanto, se tomó dos periodos de tiempo en el que el motor pudiera moverse a una velocidad rápida al igual que una lenta. Luego, se utilizó el controlador para que la velocidad del sistema se encuentre entre los periodos de tiempo obtenidos.

PRUEBAS Y RESULTADOS

- **Prueba de Sensores**

Al inicio del proyecto, se verificaron los sensores LDR instalados en el dispositivo. Durante la prueba inicial, se observó que uno de los sensores no estaba midiendo correctamente la intensidad lumínica. Después de revisar las conexiones de los sensores con el controlador, se descubrió que uno de los cables estaba mal conectado. El cable utilizado era un cable DuPont que estaba conectado entre sí, lo que provocaba una mala lectura del sensor. Al reemplazar los cables defectuosos por conexiones más confiables, el problema se resolvió y los sensores comenzaron a funcionar correctamente.

- **Prueba de Motores Paso a Paso**

Una vez los sensores se encontraban en buenas condiciones, realizamos una prueba con los motores paso a paso, a lo cual observamos inicialmente que uno de ellos no se podía mover. Cuando se cambió el driver del motor fallido en el módulo instalado se resolvió este problema, sin embargo, al realizar la prueba vimos que cuando el dispositivo se movía horizontalmente también lo hacía verticalmente debido a la conexión mecánica que había entre los ejes rotativos. Esto provocó movimientos no deseados y un comportamiento errático, lo que llevó a ajustar los como se efectúan los movimientos en el programa.

- **Prueba de Integración y de control P, y PI, PID**

Tras experimentar con los componentes, realizamos una prueba inicial de seguidor solar sin aplicar un control PI, lo que resultó en un código difícil de entender. Luego, descubrimos la librería **AccelStepper**, que facilitó el control de los motores paso a paso.

Implementamos un control proporcional agregando una constante **K_p**, ajustándola según la respuesta de la planta y mejorando el movimiento con una notable reducción del error. Después, incorporamos un control integral, con la constante **K_i** y pequeños tiempos de muestra, buscando nuevos valores de **K_p** y **K_i** para reducir aún más el error. Sin embargo, la librería no permitía ajustes en tiempo real, lo que causaba movimientos lentos.

Finalmente, implementamos un controlador **PID** completo, abandonando la librería **AccelStepper** y reescribiendo el programa para mover los motores con pulsos y tiempo. Se suavizaron las lecturas de los LDR promediando varias mediciones, y mediante el Monitor Serial y el Serial Plotter, ajustamos las constantes del PID en tiempo real, logrando un control más flexible y preciso del seguidor solar.



PROGRAMA “SUNTRACKER PID VELOCIDAD”:

```
// ----- CONSTANTES -----  
  
// PINES DE MOTOR PAP  
  
#define PIN_ENABLE 8  
  
#define PAP_DIR_X 5  
  
#define PAP_STEP_X 2  
  
#define PAP_DIR_Z 7  
  
#define PAP_STEP_Z 4  
  
// PINES DE LOS SENSORES LDR  
  
#define ldr_W A0  
  
#define ldr_S A1  
  
#define ldr_N A2  
  
#define ldr_E A3  
  
// Tiempo para mostrar la informacion en el serial  
  
#define INTERVALO_MOSTRAR 500 // Intervalo de 500 ms  
  
unsigned long tiempo_inicio = 0;  
  
// PUNTO MUERTO DEL SISTEMA  
  
#define ERROR_MINIMO_X 2  
  
#define ERROR_MINIMO_Z 2  
  
// PARA EL AJUSTE DE VELOCIDAD  
  
#define ERROR_MAX 1023  
  
// VARS DE LECTURA DE LOS SENSORES LDR  
  
int valor_ldr_W = 0, valor_ldr_S = 0, valor_ldr_N = 0, valor_ldr_E = 0;  
  
// CONSTANTES CONTROLADOR PID  
  
float Kp = 235; //Ajuste de ganancia proporcional ()
```



```
float Ki = 0.01; //Ajuste de ganancia integral      ()

float Kd = 0.001; //Ajuste de ganancia derivativa    ()

// ERRORES DEL CONTROLADOR

float senial_control_horizontal = 0, senial_control_vertical = 0;

float error_horizontal = 0, error_vertical = 0;

float integral_error_horizontal = 0, integral_error_vertical = 0;

float derivada_error_horizontal = 0, derivada_error_vertical = 0; // No lo
acumulamos, lo usamos para mostrar cuanto fue el error en el loop, luego
vuelve a 0

float error_horizontal_previo = 0, error_vertical_previo = 0;

float tiempo_anterior = 0; // CALCULO DE dt

// Límites en velocidad minima y maxima.

#define T_VEL_MIN 250

#define T_VEL_MAX 750

int tiempo_pulso_vertical = T_VEL_MIN;

int tiempo_pulso_horizontal = T_VEL_MIN;

String direccion_horizontal = "Sin movimiento"; // Dirección del movimiento
horizontal

String direccion_vertical = "Sin movimiento"; // Dirección del movimiento
vertical

#define NUM_LECTURAS 3

#define PASOS 10

// ----- SETUP -----

void setup() {

  // Habilitar motores
```



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

pinMode(PAP_DIR_X, OUTPUT);
pinMode(PAP_DIR_Z, OUTPUT);

pinMode(PAP_STEP_X, OUTPUT);
pinMode(PAP_STEP_Z, OUTPUT);

// Iniciar monitor serie
Serial.begin(115200);

// Comentamos por si queremos realizar SOLO la grafica de los errores.

iniciarSeguidor();

tiempo_inicio = millis();

}

// ----- LOOP -----
void loop() {

leerEntradaSerial();

// Calcular errores
calcularErrores();

// Transformamos el error mediante un controlador P
controladorPID();

// Mostramos los valores de los ldr, los errores, los pasos, las constantes
y la direccion.

// mostrarDatos(); // Comentamos por si queremos realizar la grafica de
los errores.

// Mostramos solo los errores para ver la grafico en el plotter
plotErrores();
```



```
// Indicamos los movimientos a realizar  
  
orientarSeguidor();  
  
}  
  
// ----- FUNCIONES -----  
  
// Leemos el serial para recibir el valor de Kp, Ki, y Kd  
  
void leerEntradaSerial() {  
  
    if (Serial.available()) {  
  
        String input = Serial.readStringUntil('\n'); // Leer hasta el salto de  
        linea  
  
        ajustarConstantes(input); // Llamar a la función que  
        procesa el comando  
  
    }  
  
}  
  
// Funcion para iniciar el seguidor solar.  
  
void iniciarSeguidor() {  
  
    Serial.println("-----  
---\n");  
  
    Serial.println("Iniciando seguidor solar con control PID.");  
  
    mostrarConstantesControlador(6);  
  
    Serial.println("\nIngresa 'P(nuevo_valor)' para cambiar Kp.");  
  
    Serial.println("\nIngresa 'I(nuevo_valor)' para cambiar Ki.");  
  
    Serial.print("\nIngresa 'D(nuevo_valor)' para cambiar Kd.");  
  
    // Mostramos como se cambian las cts por el monitor serial  
  
    for (int i = 0; i < 5; i++) {  
  
        delay(INTERVALO_MOSTRAR);  
    }  
}
```



```
    Serial.print(".");
}

Serial.println(".");

}

// Funcion para procesar el grafico del error (SERIALPLOTTER)

void plotErrores() {

    Serial.print("Error_Horizontal:");
    Serial.print(error_horizontal);      // Error horizontal
    Serial.print(",");
    Serial.print("Error_Vertical:");
    Serial.print(error_vertical);        // Error vertical
    Serial.print(",");
    Serial.print("Max:");
    Serial.print(700);                  // Valor maximo
    Serial.print(",");
    Serial.print("Min:");
    Serial.println(-700);               // Valor minimo
}

// Función para procesar y cambiar Kp o Ki si es válido

void ajustarConstantes(String comando) {

    char tipo = comando.charAt(0);    // Obtener el primer carácter ('P' o 'I')

    // Verificar si el comando tiene la estructura correcta

    if (comando.startsWith("P(") || comando.startsWith("I(") ||
comando.startsWith("D(") || comando.startsWith("p(") ||
comando.startsWith("i(") || comando.startsWith("d(")) {

        if (comando.endsWith(")")) {

```



```
String valorStr = comando.substring(2, comando.length() - 1); //  
Extraer el valor entre paréntesis  
  
float nuevoValor = valorStr.toFloat(); //  
Convertir a número  
  
// Verificar si el valor es válido  
  
if (nuevoValor >= 0) {  
  
    switch (tipo) {  
  
        case 'p': // Cambiar Kp  
  
        case 'P':  
  
            Kp = nuevoValor;  
  
            Serial.print("\nNuevo valor de Kp: ");  
  
            Serial.println(Kp);  
  
            break;  
  
        case 'i': // Cambiar Ki  
  
        case 'I':  
  
            Ki = nuevoValor;  
  
            Serial.print("\nNuevo valor de Ki: ");  
  
            Serial.println(Ki);  
  
            break;  
  
        case 'd': // Cambiar Kd  
  
        case 'D':  
  
            Kd = nuevoValor;  
  
            Serial.print("\nNuevo valor de Kd: ");  
  
            Serial.println(Kd);  
  
            break;  
  
        default: // No debería entrar aquí  
    }  
}
```



```

        Serial.println("ERROR inesperado.");

        break;

    }

} else {

    Serial.println("ERROR: el valor debe ser mayor o igual a 0.");

}

} else {

    Serial.println("ERROR: falta el cierre del paréntesis.");

}

} else {

    Serial.println("Comando no reconocido. Usa 'P(valor)' o 'I(valor)'.");
}

}

// Realizamos N cantidad de lecturas analogicas

int leerSuavizado(int pin) {

    long suma = 0;

    for (int i = 0; i < NUM_LECTURAS; i++) {

        suma += analogRead(pin);

        delayMicroseconds(5); // Pequenia espera entre lecturas

    }

    return suma / NUM_LECTURAS;
}

// Calculamos el error entre los ejes

void calcularErrores() {

    valor_ldr_W = leerSuavizado(ldr_W);
}

```



```
valor_ldr_S = leerSuavizado(ldr_S);

valor_ldr_N = leerSuavizado(ldr_N);

valor_ldr_E = leerSuavizado(ldr_E);

error_horizontal = valor_ldr_E - valor_ldr_W; // Error entre Este y Oeste

error_vertical     = valor_ldr_N - valor_ldr_S; // Error entre Norte y Sur

}

// Ajustamos la velocidad de acorde al error que exista entre sus ejes

void ajustarVelocidad(int error, int *tiempo_pulso) {

    *tiempo_pulso = map(abs(error), 0, ERROR_MAX, T_VEL_MAX, T_VEL_MIN);

    *tiempo_pulso = constrain(*tiempo_pulso, T_VEL_MIN, T_VEL_MAX);

}

// Transformamos el error prop, int y derivativamente a la velocidad del motor

void controladorPID() {

    float tiempo_actual = millis();

    float dt = (tiempo_actual - tiempo_anterior) / 1000.0; // Convertir a segundos

    // Acumular el error integral

    integral_error_horizontal += error_horizontal * dt;

    integral_error_vertical     += error_vertical * dt;

    derivada_error_horizontal = 0;

    derivada_error_vertical   = 0;

    if (dt > 0) {

        derivada_error_horizontal = (error_horizontal - error_horizontal_previo) / dt;

        derivada_error_vertical   = (error_vertical - error_vertical_previo) / dt;
    }
}
```



```

}

// Calcular pasos proporcionales e integrales al error

senial_control_horizontal = Kp * error_horizontal + Ki *
integral_error_horizontal + Kd * derivada_error_horizontal;

senial_control_vertical = Kp * error_vertical + Ki *
integral_error_vertical + Kd * derivada_error_vertical;

ajustarVelocidad(senial_control_vertical, &tiempo_pulso_vertical);

ajustarVelocidad(senial_control_horizontal, &tiempo_pulso_horizontal);

error_horizontal_previo = error_horizontal;

error_vertical_previo = error_vertical;

tiempo_anterior = tiempo_actual;

}

// Movemos el seguidor y indicamos en que direccion se movio.

void orientarSeguidor() {

    // Zona muerta

    if( abs(error_horizontal) <= ERROR_MINIMO_X && abs(error_vertical) <=
ERROR_MINIMO_Z) digitalWrite(PIN_ENABLE, HIGH);

    else digitalWrite(PIN_ENABLE, LOW);

    // Movimiento horizontal

    if (error_horizontal < 0) {

        direccion_horizontal = "Izquierda";

        moverHorizontal("Izquierda");

    } else if (error_horizontal > 0) {

        direccion_horizontal = "Derecha";

        moverHorizontal("Derecha");
}

```



```

} else {

    direccion_horizontal = "Sin movimiento";

}

// Movimiento vertical

if (error_vertical > 0) {

    direccion_vertical = "Abajo";

    moverVertical("Abajo");

} else if (error_vertical < 0) {

    direccion_vertical = "Arriba";

    moverVertical("Arriba");

} else {

    direccion_vertical = "Sin movimiento";

}

}

// Función para mover un motor paso a paso

void moverVertical(String direccion) {

    if (direccion == "Abajo") {

        digitalWrite(PAP_DIR_Z, LOW);

    } else if (direccion == "Arriba"){

        digitalWrite(PAP_DIR_Z, HIGH);

    }

    int paso = 0;

    while(paso < PASOS) {

        digitalWrite(PAP_STEP_Z, HIGH);           // Pulso alto

        delayMicroseconds(tiempo_pulso_vertical); // Ajustar para velocidad
}

```



```

digitalWrite(PAP_STEP_Z, LOW); // Pulso bajo

delayMicroseconds(tiempo_pulso_vertical);

paso++;

}

}

// Función para los motor paso a paso, giro puramente Horizontal

void moverHorizontal(String direccion) {

if (direccion == "Izquierda") {

digitalWrite(PAP_DIR_X, LOW);

digitalWrite(PAP_DIR_Z, LOW);

} else if (direccion == "Derecha") {

digitalWrite(PAP_DIR_X, HIGH);

digitalWrite(PAP_DIR_Z, HIGH);

}

}

int paso = 0;

while(paso < PASOS) {

digitalWrite(PAP_STEP_Z, HIGH);

digitalWrite(PAP_STEP_X, HIGH);

delayMicroseconds(tiempo_pulso_horizontal); // Ajustar para velocidad

digitalWrite(PAP_STEP_Z, LOW);

digitalWrite(PAP_STEP_X, LOW);

delayMicroseconds(tiempo_pulso_horizontal);

paso++;

}

```



```
    }

}

// Mostramos todos los datos en cada INTERVALO

void mostrarDatos() {

    // Cuando la diferencia de marcas temporales supere el intervalo, muestra
    los datos

    if (millis() - tiempo_inicio >= INTERVALO_MOSTRAR) {

        tiempo_inicio = millis();

        Serial.println("\n-----");
        mostrarLDR();
        mostrarErrores();
        mostrarDireccion();
        mostrarTiemposDePulso();
        mostrarConstantesControlador(6);
    }
}

// Mostrar las constantes Kp, Ki, Kd

void mostrarConstantesControlador(int cifras) {

    Serial.println("\n== Constantes controlador PID ==");

    Serial.print("Kp: ");
    Serial.print(Kp, cifras);

    Serial.print(" | Ki: ");
    Serial.print(Ki, cifras);

    Serial.print(" | Kd: ");
}
```



```
Serial.println(Kd, cifras);

}

// Mostrar el pulso que usa en cada direccion

void mostrarTiemposDePulso() {

    Serial.println("\n== Tiempos de Pulso Actuales ==");

    Serial.print("Tiempo Pulso Vertical: ");

    Serial.print(tiempo_pulso_vertical);

    Serial.println(" ms");

    Serial.print("Tiempo Pulso Horizontal: ");

    Serial.print(tiempo_pulso_horizontal);

    Serial.println(" ms");

}

// Mostrar los LDRS en el monitor serial

void mostrarLDR() {

    Serial.println("\n== Sensores ==");

    Serial.print("LDR_W: ");

    Serial.println(valor_ldr_W);

    Serial.print("LDR_E: ");

    Serial.println(valor_ldr_E);

    Serial.print("LDR_S: ");

    Serial.println(valor_ldr_S);

    Serial.print("LDR_N: ");

    Serial.println(valor_ldr_N);

}

// Mostrar los errores en el monitor serial
```



```
void mostrarErrores() {  
  
    Serial.println("\n== Errores entre sensores ==");  
  
    Serial.print("Error Horizontal: ");  
  
    Serial.print(error_horizontal);  
  
    Serial.print(" | Error Vertical: ");  
  
    Serial.println(error_vertical);  
  
    Serial.println("\n== Errores integrativos ==");  
  
    Serial.print("Error Integral Horizontal: ");  
  
    Serial.print(integral_error_horizontal);  
  
    Serial.print(" | Error Integral Vertical: ");  
  
    Serial.println(integral_error_vertical);  
  
    Serial.println("\n== Errores derivativos ==");  
  
    Serial.print("Error derivativo Horizontal: ");  
  
    Serial.print(derivada_error_horizontal);  
  
    Serial.print(" | Error derivativo Vertical: ");  
  
    Serial.println(derivada_error_vertical);  
  
}  
  
// Mostrar la dirección del seguidor  
  
void mostrarDireccion() {  
  
    Serial.println("\n== Direcciones ==");  
  
    Serial.print("Horizontal: ");  
  
    Serial.println(direccion_horizontal);  
  
    Serial.print("Vertical: ");  
  
    Serial.println(direccion_vertical);  
  
}
```

CONCLUSIONES

El desarrollo del proyecto de seguidor solar permitió aplicar de manera práctica los conocimientos adquiridos sobre control de sistemas y electrónica, enfrentando desafíos que nos brindaron valiosas lecciones. A lo largo del proceso, se implementaron controles proporcionales e integrales que mejoraron la precisión del dispositivo, permitiéndonos ajustar y optimizar el movimiento del seguidor solar para una mayor eficiencia en la captación de luz.



El uso de un control proporcional (P) y el ajuste posterior con control integral (PI) mejoraron el desempeño del seguidor solar, reduciendo el error en la posición de los motores. Si bien el control P permitió un seguimiento más preciso, la integración del control I ayudó a corregir errores acumulados. Sin embargo, el sistema aún no respondía rápidamente a grandes errores.

La implementación de un control PID ajustable en tiempo real mediante el Monitor Serial permitió mejorar la estabilidad y respuesta del sistema al ajustar las constantes K_p, K_i y K_d. Este enfoque brindó la flexibilidad necesaria para optimizar el rendimiento.

En conclusión, este proyecto consolidó nuestros conocimientos y habilidades en programación, control de sistemas y electrónica, y nos preparó para enfrentar futuros desafíos más complejos en el área de la automatización y la robótica.

REFERENCIAS

- **CNC shield for A4988 -**
<https://rajguruelectronics.com/Product/434/CNC%20shield%20for%20A4988.pdf>
- **CNC-3axis-shield -**
<https://www.handsontec.com/dataspecs/cnc-3axis-shield.pdf>
- **Light Sensing Module - LDR [4589]: Sunrom Electronics -**
<https://www.sunrom.com/p/light-sensing-module-ldr>
- **CNC Shield V3 Para Arduino Uno - UNIT Electronics-**
<https://uelectronics.com/producto/cnc-shield-v3-para-arduino-uno/>
- **AccelStepper: library for Arduino -**
[AccelStepper: AccelStepper library for Arduino](https://www.arduino.cc/en/Tutorial/AccelStepper)
- **Serial plotter:** docs.arduino.cc/software/ide-v2/tutorials/ide-v2-serial-plotter/