INFORME ACADÉMICO

Título

“Sistema de Control de Distancia”

Autores
DE ROSSO, Nicolas Agustin
MAIOLO, Joaquín
RODRIGUEZ ECHANIZ, Mora
PILONI, Ignacio

Profesores
SAN MARTIN, Hernan Dario
FUSARIO, Jorge Ruben

Asignatura
Teoría de Control - K4053

BUENOS AIRES - ARGENTINA
19/11/2024

Índice

Introducción

En un mundo donde la automatización y la robótica desempeñan un papel fundamental en la industria y la vida cotidiana, el desarrollo de sistemas precisos y eficientes de control es esencial. Este proyecto se centra en el diseño e implementación de un sistema de control de distancia utilizando Arduino como plataforma principal, junto con un motor de corriente continua, un puente H y un sensor ultrasónico.

Arduino, por su versatilidad y facilidad de uso, permite integrar y controlar múltiples componentes electrónicos de manera eficiente. En este caso, se emplea para gestionar el motor y el sensor, implementando un algoritmo de control que ajusta la posición de un objeto móvil para mantenerlo a una distancia específica de una referencia.

Durante el desarrollo del informe, se analizará cómo varía el comportamiento del sistema en relación con la forma en que el objeto se aproxima a la referencia establecida, ya sea de manera lineal, exponencial o mediante una combinación de ambas. Este análisis permitirá comprender mejor las dinámicas del sistema y su respuesta a distintas configuraciones de control.

El informe incluye los fundamentos teóricos de los componentes principales, el diseño del circuito, la implementación del software en Arduino y los resultados obtenidos al probar el sistema en diferentes escenarios. Además, se presentan las limitaciones del proyecto, así como posibles mejoras futuras, destacando la relevancia de Arduino como herramienta clave para la innovación en proyectos tecnológicos.

Objetivos

El objetivo principal de este proyecto es aplicar los conocimientos adquiridos en clase, como así también relacionar los conceptos teóricos con aplicaciones prácticas, para desarrollar un modelo funcional de un sistema de control en lazo cerrado. Utilizando una plataforma basada en Arduino, se integran un sensor ultrasónico, un puente H, un motor de corriente continua y un objeto móvil (simulado en este caso con un cabezal de impresora) para implementar un mecanismo que permita posicionar el objeto a una distancia específica X (en centímetros), definida como parámetro de entrada a través del puerto serial de la consola.

Es importante destacar que la velocidad con la que el objeto se acerca a la distancia deseada debe ser lo más rápido posible hasta llegar a un error de 5 centímetros, para luego aplicar un sistema de control integral proporcional (PI) hasta acercarse al objetivo.

Este proyecto busca no solo validar los conceptos teóricos, sino también demostrar su implementación práctica en un sistema electromecánico real.

Además del objetivo principal, se busca alcanzar varios objetivos secundarios que complementan nuestro aprendizaje como estudiantes de Ingeniería en Sistemas de Información, quienes, a pesar de no contar con un conocimiento previo profundo sobre hardware, tenemos la oportunidad de ampliar nuestras competencias mediante:

- Comprensión de los sistemas de control continuo.
- Familiarización con dispositivos de hardware.
- Desarrollo de habilidades prácticas en el manejo de sensores y actuadores.
- Consolidación del aprendizaje teórico mediante la práctica.
- Introducción a la interacción hardware-software.

Marco Teórico

Motores de corriente contínua

Un motor de corriente continua (DC) es un dispositivo electromecánico que convierte energía eléctrica en energía mecánica mediante la interacción entre un campo magnético y una corriente eléctrica. Es ampliamente utilizado por su capacidad de controlar velocidad y dirección de manera eficiente.

Imagen 1 - Motor de corriente continua (3)

En este proyecto, el motor de corriente continua es el actuador que permite mover el objeto controlado para ajustar su posición y alcanzar la distancia objetivo.

Sin embargo, este motor presenta la siguiente limitación: actúa en una sola dirección. Por lo que, para lograr el objetivo propuesto, debimos utilizar un puente H en conjunto para modificar esto y lograr así que el motor actúe en ambos sentidos.

Sensores de distancia

Los sensores de distancia, como los sensores ultrasónicos, miden la distancia entre el sensor y un objeto mediante el tiempo que tarda un pulso ultrasónico en viajar hasta el objeto y regresar.

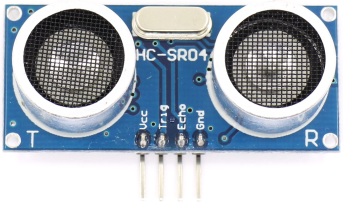

Imagen 2 - Sensor de distancia ultrasónico HC-SR04 (4)

En este trabajo, se utilizó un sensor de distancia ultrasónico HC-SR04 para medir en tiempo real la distancia del objeto al punto de referencia, permitiendo realizar los ajustes necesarios en el sistema de control.

La arquitectura de este sensor es su construcción de 2 piezoeléctricos en forma de cilindro: Un transmisor (emisor o transductor TRIG) y un receptor (transductor ECHO). El transmisor emite señales ultrasónicas de alta frecuencia (40 kHz), la cual es detectada por el receptor cuando la señal rebota en cualquier objeto que se encuentra delante, es así cómo podemos calcular la distancia entre el sensor y el objeto, mediante el tiempo que tarda en recibir la señal ultrasónica.

Debemos tener en cuenta que el valor que obtenemos es el tiempo entre la emisión del pulso y la recepción del eco, por lo que para obtener la distancia en centímetros debemos utilizar la siguiente fórmula:

$$Distancia\ (cm) = \frac{Tiempo\ (us)\ x\ 0.0343\ (vel\ sonido)}{2}$$

Nota: la división por 2 se debe a que el pulso recorre ida y vuelta.

Puente H

Un Puente H es un circuito electrónico que permite controlar la dirección de un motor de corriente continua (CC). Se llama así porque la disposición de los interruptores (o transistores) forma un esquema similar a la letra "H". Este diseño facilita cambiar el sentido de giro del motor al modificar la polaridad de la corriente que recibe.

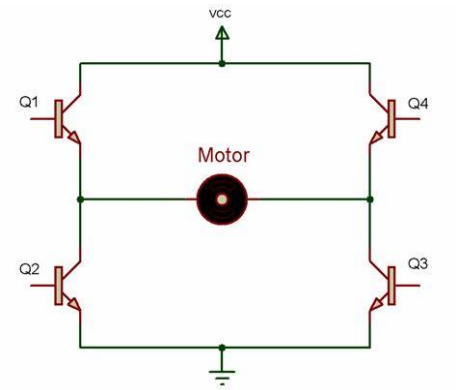

Imagen 3 - Puente H (5)

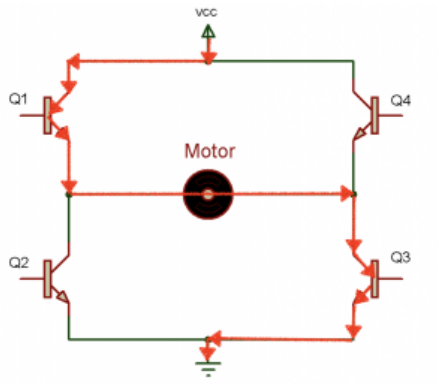

Imagen 4 - Puente H activando transistores Q1 y Q3 (5)

El Puente H tiene cuatro interruptores o transistores (llamados típicamente Q1, Q2, Q3, y Q4), organizados en pares diagonales. Dependiendo de qué transistores estén activados, la corriente fluye en una dirección u otra a través del motor.

- *Sentido de giro 1:* Activar Q1 y Q3 (como se observa en la imagen 4); la corriente fluye del terminal positivo al negativo del motor.
- *Sentido de giro 2:* Activar Q2 y Q4; la corriente fluye en dirección opuesta.

Placa de Pruebas

La placa de pruebas es un elemento usado para conectar componentes electrónicos sin necesidad de soldarlos. Su diseño facilita la experimentación y el desarrollo de prototipos.

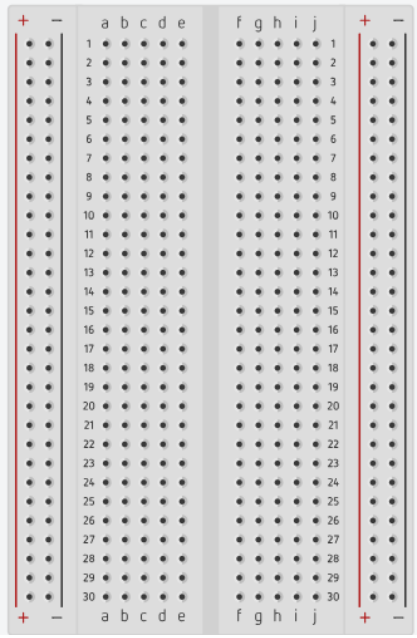

Imagen 5 - Placa de pruebas pequeña (Tinkercad)

La estructura básica es la siguiente:

1. Filas horizontales (pistas centrales):
    - Estas filas están divididas en dos secciones con un espacio en el medio (habitualmente para conectar circuitos integrados).
    - Cada sección de una fila está interconectada internamente. Por ejemplo, todos los agujeros de la fila A1 a E1 están conectados, y lo mismo ocurre para F1 a J1.
2. Columnas verticales:
    - En sus laterales tiene carriles dedicados a Vcc (positivo) y GND (tierra). Las tiras están marcadas con líneas rojas (+) y negras o azules (-) que recorren todo lo largo de la placa y facilitan la conexión de energía.
3. División central:
    - El espacio en el centro separa las filas y facilita la conexión de circuitos integrados (ICs), donde cada lado del chip queda conectado a filas separadas.

PWM

PWM (Pulse Width Modulation, o Modulación por Ancho de Pulso) es una técnica de modulación utilizada para controlar la cantidad de energía entregada a una carga mediante una señal digital de tipo cuadrada.

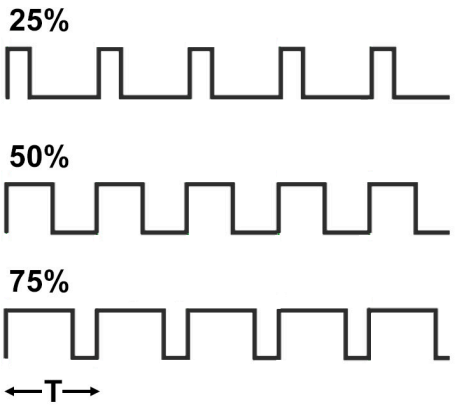

Imagen 6 - PWM y su variación de energía (6)

En lugar de proporcionar una corriente constante, PWM varía el tiempo (ancho de pulso) durante el cual la señal está encendida (ON) y apagada (OFF). Esta alternancia ocurre a una frecuencia tan alta que los componentes controlados no perciben los cortes de la señal.

Es importante destacar que la amplitud (A) y el periodo (T) se mantienen constantes.

La frecuencia de la señal PWM es la cantidad de ciclos completos que se generan por segundo, medida en hertzios (Hz). Esta frecuencia, combinada con el ciclo de trabajo, determina la cantidad de energía entregada al dispositivo controlado.

En este trabajo, fue de suma importancia el uso de PWM debido a que nos permitió modificar la velocidad con la que el objeto se acercaba a la distancia deseada.

Control Proporcional Integral (PI)

El control proporcional integral (PI) es una estrategia de control utilizada en sistemas para regular variables de proceso, como temperatura, velocidad, presión o posición. Este tipo de controlador combina dos acciones fundamentales: el control proporcional (P) y el control integral (I), lo que le permite corregir errores y alcanzar una estabilidad más precisa en el sistema (7).

La ecuación para obtener la señal de control en este tipo de control es la siguiente:

$$w(t) = Kp.e(t) + Ki\int e(t).dt$$

Donde:

- *w(t):* es la señal de control generada
- *Kp:* es la ganancia proporcional, un valor arbitrario.
- *Ki:* es la ganancia integral, otro valor arbitrario.
- *e(t):* es el error del sistema, calculado como referencia entre el valor deseado y el medido.

Para implementar esto en Arduino, tratamos al tiempo como intervalos definidos (Δt), la integral se puede aproximar mediante una suma acumulativa, se la define como:

$$w(n) = Kp.e(n) + Ki.\sum_{i=0}^{n} e[i].\Delta t$$

Esta será, entonces, la fórmula que utilizaremos en nuestro informe para definir la velocidad con la que el objeto se acerca al sensor cuando este se encuentre a menos de 5 centímetros.

Diseño del Sistema

Diagrama de bloques del circuito de control de lazo cerrado

El sistema de control de lazo cerrado que desarrollamos funciona mediante el principio de retroalimentación, donde el sistema ajusta continuamente su comportamiento (la velocidad del motor) en función de la diferencia entre la distancia requerida y la distancia real (error). Por lo que el diagrama de bloques sería el siguiente:

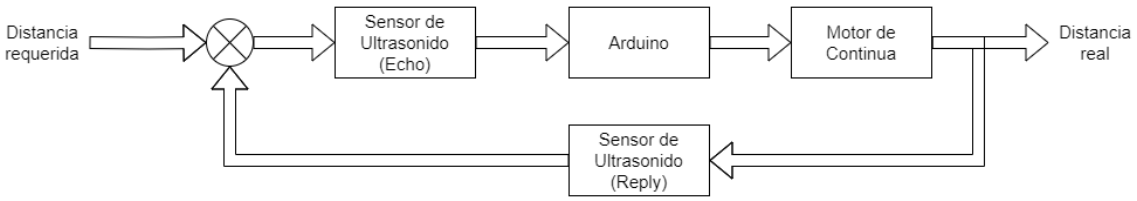

Imagen 7 - Diagrama de bloques

Para entenderlo mejor, explicaremos cada paso:

1. Distancia requerida: este valor es el punto de partida del sistema, ya que es lo que queremos obtener.
2. Sensor ultrasónico (Echo): el sensor ultrasónico mide la distancia al objeto en tiempo real y envía dicho valor al Arduino.
3. Arduino: es el encargado de darle lógica al valor obtenido y en base al mismo indicar al motor de continua hacia qué dirección moverse y con qué velocidad. Una vez obtenido el valor de entrada, obtiene el error en conjunto con la distancia deseada pasada por puerto serial y en base al mismo manda una señal al motor de continua mediante el puente H.
4. Motor de continua: si el error es positivo (el objeto está más lejos de la distancia deseada), el sistema aumenta la velocidad del motor en una dirección para acercar el objeto; si el error es negativo (el objeto está demasiado cerca de la distancia deseada), el sistema cambia de dirección.
5. Retroalimentación (Reply del sensor ultrasónico): el sensor vuelve a medir la distancia, y el Arduino sigue recibiendo estas mediciones y ajusta la salida del motor. Este ciclo continúa hasta estar en la posición deseada (o en una posición dentro del intervalo de tolerancia o error aceptado para el análisis).

Diseño realizado en Tinkercad

Para lograr el objetivo de este trabajo, primero definimos los elementos que necesitamos para lograrlo, estos fueron:

- Un Arduino UNO.
- Un sensor de ultrasonido HC-SR04.
- Un motor de corriente continua.
- Una fuente.

- Un puente H.
- Una placa de pruebas.
- Cables de tipo macho-macho.

Teniendo en cuenta los ejemplos vistos en clase y, trabajando con la plataforma web Tinkercad (8), logramos armar el circuito que puede apreciarse en la siguiente imagen:

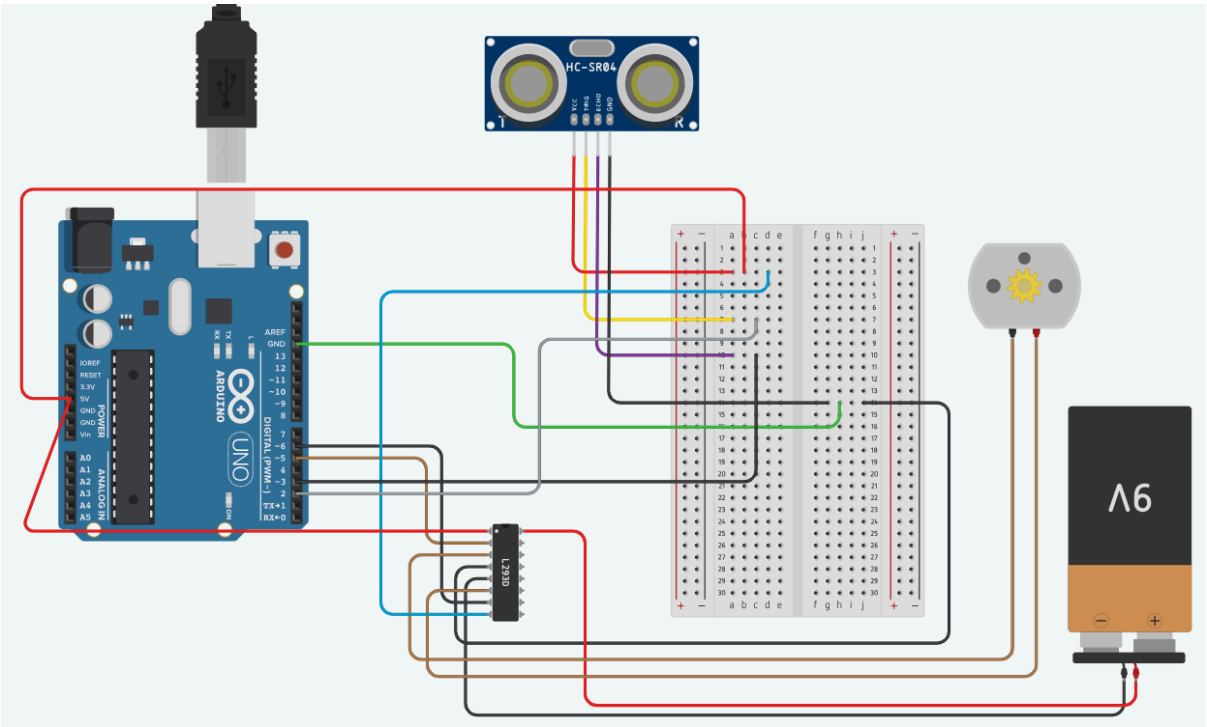

Imagen 8 - Diseño del sistema implementado en Tinkercad (9)

Para dar más contexto a la imagen, indicaremos las conexiones y su justificación, comenzando por las del puente H:

I. En lugar de la fuente, se utilizó una pila de 9V, la misma conectada al puente H (del *positivo* de la pila a la *potencia 1*, el cable de color rojo; y el *negativo* de la pila a la *tierra*, de color negro). Esto proporciona el voltaje necesario para alimentar al motor conectado al puente H.

II. Las *salidas 1 y 2* del puente H se conectan a las *terminales 1 y 2* del motor de contínua (cables marrones). Estas conexiones permiten que el puente H controle la dirección y la velocidad del motor.

III. La *entrada 1* del puente H se conecta al *pin 5* de Arduino mediante el cable marrón, y la *entrada 2* conectada al *pin 6* mediante el cable negro. Combinando las señales de los pines 5 y 6, Arduino puede decidir si el motor avanza, retrocede o se detiene.

IV. La *potencia 2* del puente H se conecta a la placa de pruebas en la posición *d3*, mediante el cable celeste. Esta conexión está vinculada a la lógica interna del puente H, proporcionando alimentación para las señales de control.

V. Además, tenemos una conexión de otra entrada de *tierra* del puente H hacia la posición *j14* de la placa de pruebas, mediante el cable negro. Esto asegura que la

tierra del puente H esté conectada a una tierra común compartida con todos los componentes del sistema, incluido Arduino.

VI. Finalmente, en el puente H, tenemos la conexión mediante cable rojo entre *Activar 1 y 2* y los *5v* de la placa de Arduino. Esto se hace para habilitar y energizar la lógica interna del puente H.

De las conexiones del sensor de ultrasonido, podemos decir que:

I. La *potencia* del sensor se conecta al pin *a3* de la placa de pruebas, mediante el cable rojo. Además, dentro de la misma fila, en *b3*, tenemos una conexión hacia los *5v* de la placa de Arduino, mediante otro cable de color rojo. Sin esta conexión, el sensor no funcionaría.

II. Luego, el *desencadenador* se conecta al pin *a7* de la placa mediante el cable amarillo. Como en la misma columna de la placa existe otra conexión desde *c7* hacia el pin *2* de Arduino mediante el cable gris, esto nos indica que dicho pin será el encargado de generar el pulso ultrasónico inicial que permite medir la distancia.

III. Mediante un cable violeta existe la conexión entre *echo* del sensor y *a10* que, en conjunto con otro cable de color negro colocado desde *c10* hacia el pin *3* de Arduino, permiten obtener la distancia. Esta conexión trabaja en conjunto con la anterior y son indispensables para el correcto funcionamiento del sistema.

IV. Por último, contiene una conexión a tierra hacia el pin *g14* de la placa mediante un cable negro. Esta fila es la encargada de la conexión a tierra, gracias al pin *h14* que va hacia uno de los *GND* de Arduino mediante el cable verde.

Estas conexiones establecen la base para el funcionamiento del sistema en su componente de hardware. Sin embargo, es fundamental desarrollar un código que permita al Arduino interpretar y gestionar dichas conexiones, garantizando así la integración y operatividad del sistema completo.

Nota: Si se desea observar el funcionamiento de dicho sistema en la plataforma de Tinkercad, se puede lograr haciendo clic aquí.

Implementación

IDE Arduino

Como se mencionó en la etapa anterior, para lograr el funcionamiento adecuado del sistema, fue necesario implementar el código en el lenguaje de programación utilizado por Arduino, basado en C/C++. Para ello, se utilizó el **IDE de Arduino** (10), que facilita la creación, carga y depuración del código en la placa Arduino UNO. Este entorno de desarrollo simplifica el proceso de programación mediante una interfaz gráfica amigable, permitiendo que el usuario escriba código en un formato estructurado y luego lo cargue directamente al microcontrolador.

El IDE de Arduino es un entorno de desarrollo integrado que incluye un editor de texto donde se escribe el código, un compilador para verificar que no haya errores y un cargador de programas que transfiere el código compilado al Arduino a través del puerto USB. Además, el IDE incluye varias bibliotecas predefinidas que facilitan la programación de tareas comunes como el control de sensores, motores y otros periféricos.

Este entorno también incluye una consola de salida que permite al usuario ver mensajes de depuración y errores durante la ejecución del programa. Gracias a la interfaz del IDE, fue posible cargar el programa en la placa Arduino de manera rápida y sencilla, asegurando que los sensores y actuadores conectados funcionaran correctamente según la lógica definida en el código.

Código del sistema

El código utilizado puede observarse de una mejor manera en el siguiente enlace de github. Definiremos paso a paso el código del sistema para que sea comprensible:

I. Primero, definimos valores estáticos utilizando ***#define***, por ejemplo, la velocidad máxima en *PWM* (que es 255), la distancia máxima del sensor al objeto posible (30 centímetros por cuestiones físicas), el pin de *TRIG* del sensor ultrasónico (indicado en el pin 2) y el pin de *ECHO* del mismo sensor (indicado en el pin 3):

```
#define TRIG_PIN 2
#define ECHO_PIN 3
#define MAX_DIST 30
#define MAX_VEL 255
```

II. Luego, indicamos que los pines 5 y 6 se utilizarán para controlar las direcciones que puede tomar el motor.
Además, guardamos en una variable la distancia que deseamos recibir por puerto serial, como así también la velocidad que tendrá el motor de continua en los últimos 5 centímetros de error (que debe ir disminuyendo paulatinamente).

Como últimas variables, tendremos las velocidades del motor en una u otra dirección, junto con un *booleano* que nos ayudará a que el sistema no comience si no se ingresó el valor de referencia por el puerto serial:

```
int motorPin1 = 5;
int motorPin2 = 6;

int distanciaRecibida = 0;

float velocidadAjustada = 0;
int velocidadMotor1 = 0;
int velocidadMotor2 = 0;

bool configurado = false;
```

III. Para concluir con las variables de configuración, definiremos las que necesitamos para obtener la velocidad en función del control proporcional integral (PI):

```
float errorAcumulado = 0;
float tiempoAnterior = 0;
float Kp = 8;
float Ki = 2;
```

Como indicamos anteriormente, los valores de Kp y Ki son arbitrarios, nos decidimos por estos valores para hacer la prueba, aunque en la presentación final podremos variarlos.

IV. Después de las configuraciones iniciales, tenemos los dos métodos principales, *setup()* y *loop()*. El setup es un método que se corre una sola vez al iniciar la simulación, y el loop, como su nombre lo indica, es el iterador que correrá siempre.
Dentro del setup, colocaremos entonces, el número de puerto Serial para poder comunicarnos mediante la terminal (elegimos una velocidad de transmisión de 9600 baudios), y el tipo de PIN que tendrá el Arduino en cada conexión (motorPin1, motorPin2, trigPin, echoPin):

```
void setup() {
  Serial.begin(9600);
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);

  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
}
```

Marcamos a los pines del motor (1 y 2) como OUTPUT. De este modo, le informamos al Arduino que a estos pines debe enviarles señales para activarlos.

También, al TRIG_PIN lo indicamos como OUTPUT porque es el pin que le pedirá al sensor que emita la señal para poder medir la distancia.
Finalmente, marcamos al ECHO_PIN como INPUT porque es el pin que recibirá el eco de la señal que el sensor envió. Dicho eco vuelve al sensor luego de haber chocado con un objeto.

V. En el método loop() tendremos la lógica principal del programa, comenzando por un condicional que impedirá el comienzo de la iteración si no se recibe nada por parámetro:

```
if (!configurado) {
    Serial.read();
    distanciaRecibida = -1;
    while (distanciaRecibida <= 0 || distanciaRecibida > MAX_DIST)
    {
        Serial.println("Ingrese la distancia objetivo (valor
        referencia): ");
        while (!Serial.available());
        String input = Serial.readStringUntil('\n');
        distanciaRecibida = input.toInt();
        if (distanciaRecibida <= 0 || distanciaRecibida > MAX_DIST)
        {
        Serial.println("Error: valor de distancia no válido.");
        }
     }
     configurado = true;
}
```

VI. Una vez indicado el valor de referencia, comienza la iteración gracias al comando *configurado = true*, esto permite salir del condicional y continuar con este código:

```
digitalWrite(TRIG_PIN, LOW);
delayMicroseconds(2);
digitalWrite(TRIG_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW);

long duration = pulseIn(ECHO_PIN, HIGH);
float distanciaActual = duration * 0.034 / 2;

float error = distanciaRecibida - distanciaActual;
```

Con estos comandos, estamos indicando al sensor ultrasónico que emita una señal y luego obtenga la duración con el método *pulseIn()*, dicho valor lo guardamos en la variable *duration*; en otra variable denominada *distanciaActual* calculamos la

distancia (en centímetros) en la que se encuentra el objeto del sensor. Gracias a ello, estamos en condiciones de calcular el error restando la distancia recibida por puerto serial de la actual.

VII. Para obtener el delta T, obtenemos primero el tiempo actual y realizamos la resta del actual con el anterior, para luego dividirlo por 1000 y, de esta manera, convertirlo en segundos. Luego, generamos el error acumulado, que será la suma del valor del error por el delta tiempo definido.

Finalmente, la velocidad ajustada es la variable que pasaremos al pin indicado del motor (a uno o al otro, dependiendo si está demasiado cerca o demasiado lejos).
En la prueba notamos que el objeto, al aplicarle una potencia menor a 150, no se movía, esto se debe al **torque de arranque (zona muerta)** del mismo (torque mínimo necesario para vencer la fricción, resistencia y cualquier otra fuerza que se oponga al movimiento inicial del motor o dispositivo), es por ello que se indica dicho 150 en la velocidad ajustada. Luego, utilizamos la ecuación definida anteriormente para calcular con qué potencia el objeto debe acercarse al sensor. Agregamos también un condicional que verifique que la velocidad que le pasaremos al motor no supere los 255 de señal PWM (debido a que es el 100% de potencia).

```
unsigned long tiempoActual = millis();
float deltaTiempo = (tiempoActual - tiempoAnterior) / 1000;
tiempoAnterior = tiempoActual

errorAcumulado += abs(error) * deltaTiempo;

velocidadAjustada = Kp * abs(error) + Ki * errorAcumulado;

if (velocidadAjustada > MAX_VEL) {
   velocidadAjustada = MAX_VEL;
}
```

VIII. Este fragmento verifica que si el error (o lo que le falta al objeto para llegar a la distancia deseada), se encuentra a más de 5 cm de distancia, se aplique la velocidad máxima del motor. Cuando se llega a una distancia menor que 5 centímetros, se disminuye paulatinamente la velocidad, hasta llegar a un valor aceptable de la distancia deseada (en base a la tolerancia indicada de 3 mm). Al llegar a dicha posición, coloca al booleano como *false*, para que se pueda agregar otra distancia sin la necesidad de apagar la conexión.

```
float tolerancia = 0.3;

if (error > 5) {
   velocidadMotor1 = MAX_VEL;
   velocidadMotor2 = 0;
```

```
} else if (error < -5) {
   velocidadMotor1 = 0;
   velocidadMotor2 = MAX_VEL;
}

if (error > 0 && error <= 5) {
   velocidadMotor1 = velocidadAjustada;
   velocidadMotor2 = 0;
} else if (error <= 0 && error >= -5) {
   velocidadMotor1 = 0;
   velocidadMotor2 = velocidadAjustada;
}

if (abs(error) <= tolerancia) {
   velocidadMotor1 = 0;
   velocidadMotor2 = 0;
   errorAcumulado = 0;
   configurado = false;
}
```

IX. Una vez definidas las velocidades del motor, se las enviamos mediante el método *analogWrite()*, indicando a qué pin y con qué ancho de pulso irán. Finalmente, mostraremos por consola los valores que tomarán las variables, para luego observarlas en el gráfico:

```
analogWrite(motorPin1, velocidadMotor1);
analogWrite(motorPin2, velocidadMotor2);

Serial.print(distanciaActual);
Serial.print("\t");

Serial.print(velocidadAjustada);
Serial.print("\t");

Serial.println(abs(error));
```

X. Por último, generamos un condicional que nos ayudará a que, cuando el objeto se acerque correctamente al objetivo, utilice el flag configurado para dar como finalizada dicha simulación y, de este modo, volver a solicitar una distancia de referencia. Esto nos permitirá realizar varias simulaciones sin la necesidad de reiniciar el código:

```
if (!configurado) {
    Serial.println("Aproximación finalizada.");
    loop();
}
```

Relaciones

Estado Estacionario: Con este código realizado, logramos eliminar el error en estado estacionario debido a que, una vez que el objeto llega a la distancia indicada, el sistema se detiene.
Si siguiera en su loop, el objeto debería mantenerse estable, siempre y cuando ningún objeto se interponga entre el sensor y el objeto.

Orden del Sistema: El orden del sistema se relaciona con la cantidad de integradores presentes en la función de transferencia del modelo. Al tener un control integral, modifica la naturaleza del sistema, haciéndolo más complejo pero también más capaz de eliminar el error estacionario.

Nota: Al elevar el orden, el sistema puede responder de forma más precisa a cambios en la referencia y corregir errores persistentes (12).

Estabilidad: Queda demostrado en las pruebas que el objeto se acerca al objetivo y queda estable gracias a lo anteriormente nombrado. Además, los controles de velocidad mínima y máxima permiten que la estabilidad del sistema sea segura.

Pruebas y Resultados

Una vez conectado el Arduino a la PC y ejecutado el código, estamos en condiciones de probar el funcionamiento del sistema y, de este modo, determinar si lo realizado cumple lo solicitado o no. Estas pruebas se llevaron a cabo con el objetivo de validar el funcionamiento del sistema en condiciones reales, verificar su capacidad para mantener la distancia objetivo, y analizar la eficacia de las etapas de control implementadas.

Se realizaron pruebas tanto de la primera versión (de manera presencial, el día martes 12/11/2024) como en la versión final (las que se encuentran en este informe), abordando aspectos clave como la precisión en el mantenimiento de la distancia objetivo y la respuesta del motor ante cambios en la distancia de referencia. Con azul se detalla la distancia, en naranja la potencia y de color verde el error.

A continuación, se presentan las imágenes de las distintas pruebas:

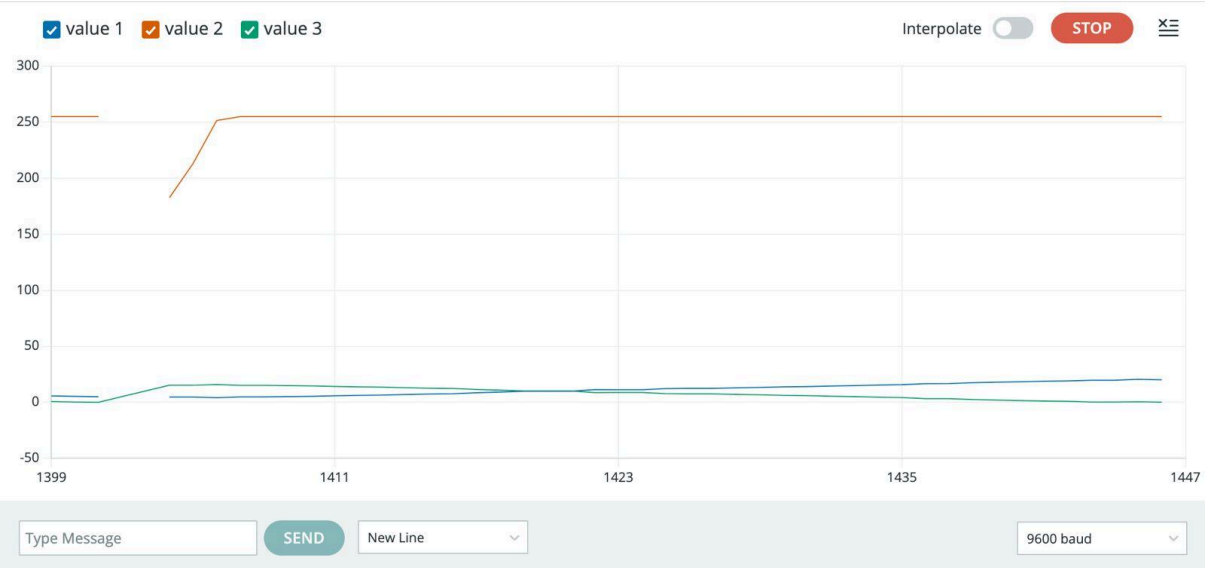

Imagen 9 - Prueba A realizada con una medición de 20 cm.

Imagen 10 - datos en consola de Prueba A

En la primera prueba realizada, se puede observar gracias a la imagen 10, que el objeto se acerca hacia el objetivo correctamente, reduciendo el error y a la máxima velocidad debido a que la cuenta de control proporcional integral dió un valor alto (mayor a 255).

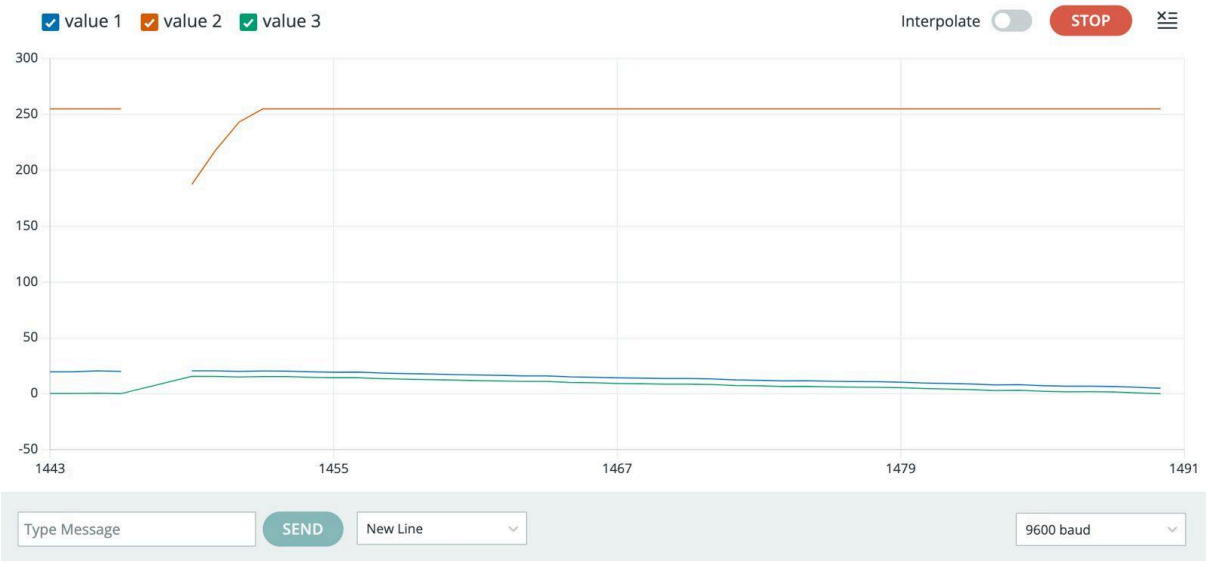

Imagen 11 - gráfico de Prueba B

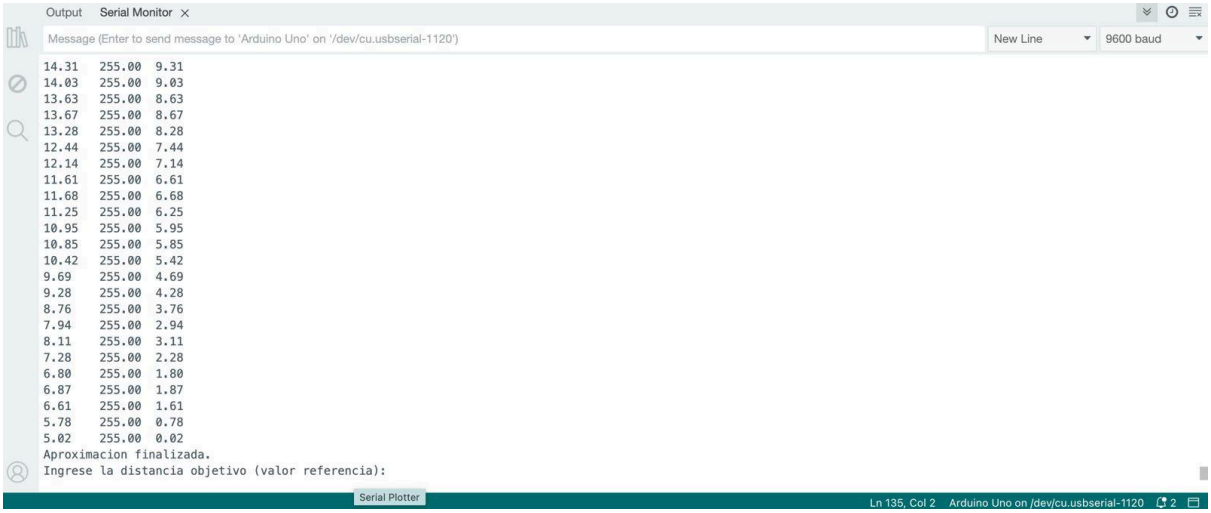

Imagen 12 - datos en consola de Prueba B

En esta segunda prueba, realizamos una medición de 20 centímetros hacia 5 centímetros, observando que en la tercera columna el error se reduce hasta casi cero, específicamente, 0.02 centímetros.

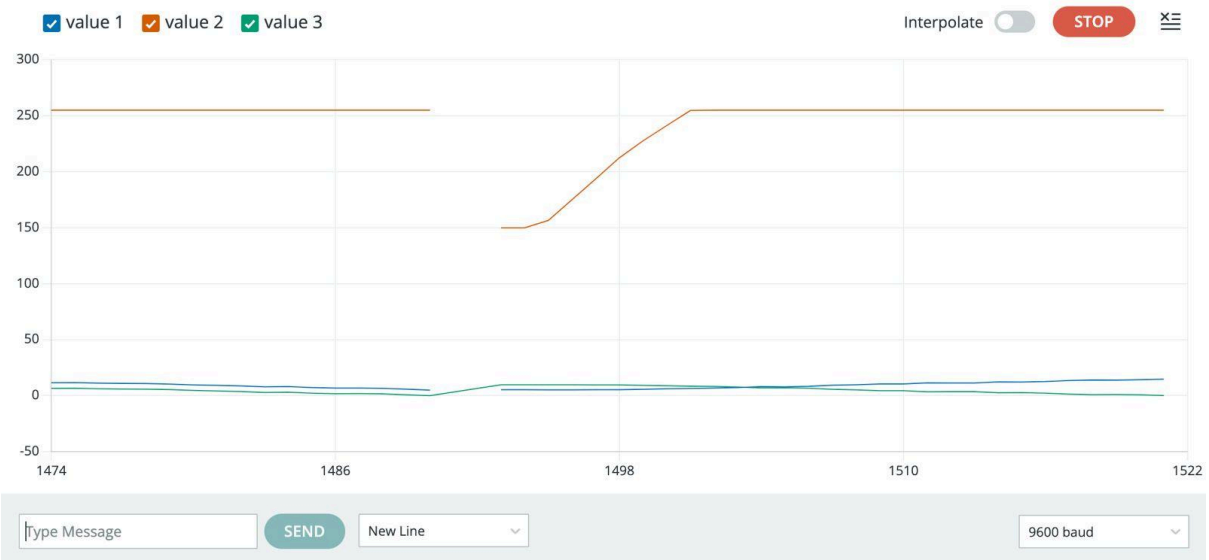

Imagen 13 - gráfico de Prueba C

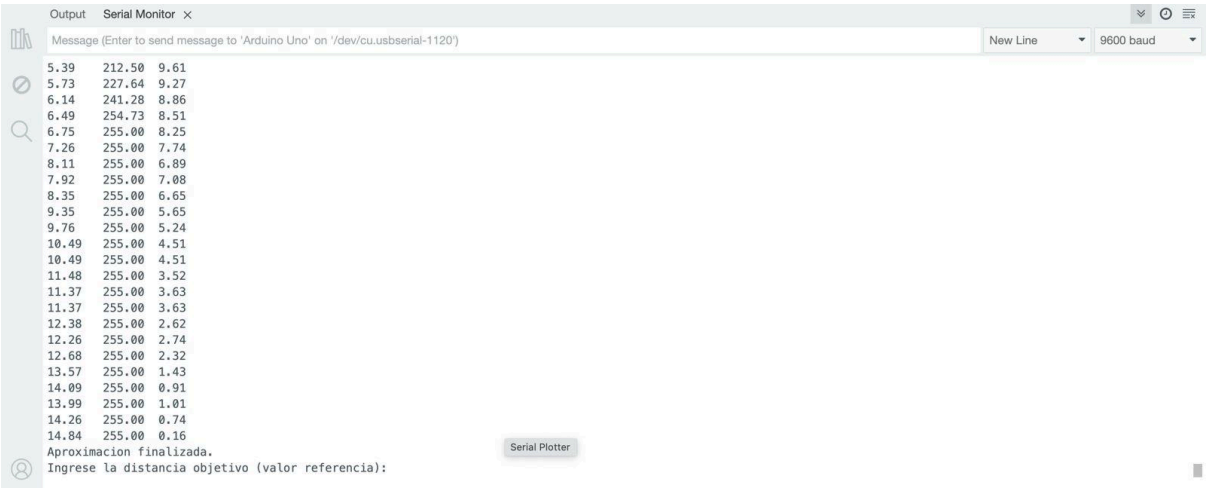

Imagen 14 - datos en consola de Prueba C

Como última prueba de este informe, y como continuación de la prueba anterior, se puede apreciar que la misma se realizó de 5 a 15 centímetros. Con un error de 0.16 centímetros.
Es de suma importancia notar que al principio de la imagen 13 se observa la gráfica de la prueba B y no de la C, pero luego se puede notar esta última.

Estos resultados nos brindan información sobre el rendimiento del sistema, su sensibilidad a variaciones externas, y la efectividad de las mejoras realizadas.

Nota: las imágenes presentadas en esta prueba fueron realizadas con una versión anterior a la final, es por esto que la potencia del motor se encuentra constantemente a 255 (100%). Para observar el correcto funcionamiento, se puede probar el sistema en la plataforma de Tinkercad.

Conclusión

El desarrollo del sistema de control basado en Arduino representó un desafío para nosotros como estudiantes de Ingeniería en Sistemas de Información, por el hecho de no tener conocimientos previos en el desarrollo de circuitos.

En su etapa primaria, el sistema se diseñó para ajustar la posición de un objeto en función de una distancia objetivo pasada por puerto, como así también la velocidad a la cual se deseaba acercar, empleando un sensor ultrasónico y un puente H para controlar la dirección y velocidad del motor. Si bien esta versión inicial cumplió con los requisitos básicos, se deseó mejorar esto haciendo que el objeto se acerque a la máxima velocidad, reduciendo la misma paulatinamente a medida que se acerca.

Para lograr lo anteriormente mencionado, se optó por ajustar la velocidad en función de la distancia restante (o error). De este modo, el sistema logró lo solicitado. Finalmente, en esta entrega, se solicitó que el sistema se acerque a la máxima velocidad hasta un error de 5 centímetros, para luego disminuir la velocidad a medida que se acerca al objetivo.

Esto pudo cumplirse gracias a una lógica implementada que corroboraba que el error, si era mayor a 5 centímetros, siga a una velocidad máxima, caso contrario reduzca utilizando el algoritmo de la entrega anterior.

Durante el desarrollo, enfrentamos varios retos, entre ellos:

- **Torque de arranque:** Como se nombró en etapas anteriores, este problema lo experimentamos al enviar una potencia que no alcanzaba para superar la **Zona Muerta** (11) y notamos que el motor emitía una especie de ruido pero no lograba mover el objeto. Esto logramos solucionarlo a *prueba y error* detectando cuál era la potencia mínima necesaria del motor para romper esa barrera.
- **Exceso de velocidad al llegar al objetivo:** Lo experimentamos cuando la velocidad era tan alta que el sensor medía una distancia y, como el motor en ese instante seguía en funcionamiento, sobrepasaba el cálculo y no lograba estabilizarse en el objetivo.
- **Agregado de una tolerancia:** Al llegar a la distancia objetivo, detectamos que el sistema no frenaba en ningún momento debido a que el sensor no lograba medir la distancia exacta que se le había solicitado, esto lo solucionamos agregando una tolerancia que permitió detener el motor en una aproximación al objetivo.
- **Funcionamiento del Puente H**: Al utilizar un Puente H, los posibles estados lógicos de los transistores que lo conforman son tres: **High-Low** , **Low-High** o **Low-Low** . Esto se debe a que el estado **High-High** (activación simultánea de ambos transistores de una misma rama vertical) genera un cortocircuito directo entre la fuente de alimentación y tierra, lo que puede provocar daños graves en los componentes electrónicos del sistema, incluido el controlador. o la computadora que lo opera. Este tipo de cortocircuito no solo compromete la integridad del puente H, sino que también puede resultar en un sobrecalentamiento y fallas críticas en el sistema.

El sistema final logró alcanzar la distancia objetivo con precisión y estabilidad, demostrando ser adecuado para aplicaciones prácticas como vehículos autónomos, brazos robóticos, asistentes de estacionamiento, puertas automáticas, entre otros. Además, la integración de gráficas en tiempo real proporcionó una herramienta útil para evaluar el desempeño del sistema en base a distintos parámetros de entrada.

Para más contenido, contamos con un archivo .ppt (con la versión de la primera entrega) que se puede observar haciendo clic aquí.

Referencias

1. Maquinas Electricas y Electromecanicas - S. A. Nasar
2. Controlling a DC Motor with Motor Shield Rev3
3. Motor de corriente continua - MP100 series - AMER - síncrono / 12 V / IP20
4. Sensor De Distancia Ultrassônico Hcsr04 Arduino Pic Robótica | MercadoLivre
5. https://www.automatizacionparatodos.com/puente-h-arduino/
6. 北岛夜话
7. CONCEPTOS BÁSICOS DE CONTROL
8. Tinkercad - https://www.tinkercad.com/
9. Diseño del sistema implementado en Tinkercad - Circuit design Circuito TP Teoria de Control - Tinkercad
10. Software | Arduino
11. Este concepto fue visto en clase donde el Profesor Hernan San Martin nos explicó que se trata de un rango de valores de entrada con el que cuentan la mayoría de los dispositivos para el cual el sistema no genera una respuesta.
12. Ogata, K. (2010). *Ingeniería de Control Moderna*