Encóder de cuadratura y control de velocidad PID

1. Objetivo de la práctica

Conocer el funcionamiento de los encoders rotativos de cuadratura e implementarlos para generar el control de velocidad de los motores de corriente directa de un robot móvil diferencial.

2. Metas

Para la realización de la práctica se deben cumplir las siguientes metas:

- Adquirir la información del encoder para conocer la posición y dirección de giro del motor DC.
- Conectar y controlar los motores con la tarjeta de desarrollo ESP8266 utilizando el puente H L293D.
- Programar un controlador PID para regular la velocidad de los motores.
- Construir un robot móvil diferencial con dos ruedas fijas motorizadas con control de velocidad.

3. Antecedentes

En el campo de la robótica móvil suele requerirse un control y medición precisos del movimiento de las ruedas y actuadores. Esto es importante para lograr el movimiento deseado de los robots de manera correcta, incluso se utiliza para estimar la posición del sistema robótico mediante la odometría. Si bien existen motores avanzados y tarjetas de control que ya brindan estas funcionalidades, resulta importante comprender el principio básico de funcionamiento de estos sistemas para reconocer sus aplicaciones y limitaciones.

Para poder hacer un control de velocidad de un motor lo primero que se necesita es tener un medio que permita conocer la posición y velocidad a la que el motor gira, una forma común de lograr esto es mediante el uso de encoders rotativos. Los encoders rotativos son transductores que miden la posición angular de algún eje al girar, éstos utilizan un tipo de sensor para codificar, mediante señales eléctricas, la posición angular del eje; para esto se suelen utilizar sensores ópticos, electromecánicos o magnéticos.

Además, por su principio de funcionamiento se pueden identificar distintos tipos de encoders, por ejemplo:

- *Incrementales*: este tipo de encóder calcula la posición angular del eje al llevar la cuenta de los pasos que se han dado. Por ejemplo, se puede utilizar un disco ranurado que cambia de valor cada determinado ángulo, y se puede estimar cuánto ha girado el eje sumando los cambios que detecta el encóder.

  Dentro de este tipo también se pueden incluir los encoders de cuadratura como los que se utilizarán en esta práctica. La principal ventaja de los encoders de cuadratura es que proporcionan también información sobre la dirección de giro, esto se logra mediante el uso de

dos sensores cuyas señales se encuentran desfasadas 90°. De esta manera al observar el estado de las señales al cambiar se puede saber si el eje gira en dirección horaria o antihoraria.

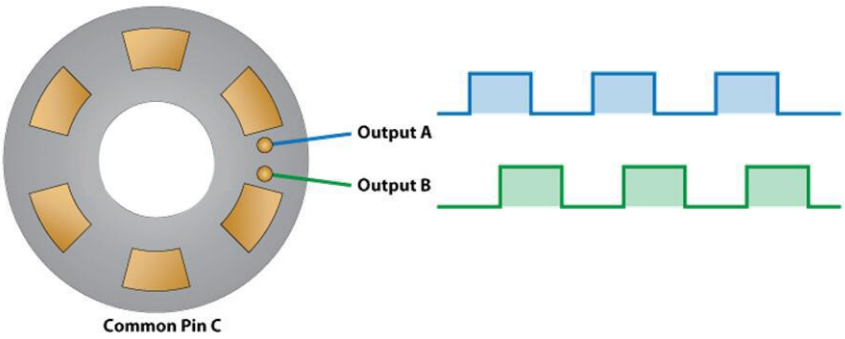


Figura 1. Representación de encóder incremental de cuadratura.

- *Absolutos*: este tipo de encoders no requiere llevar la cuenta de los pasos para conocer la posición angular actual. En este caso, a diferencia del anterior, todo el giro del eje esta codificado de forma única, de tal manera que basta con las señales actuales para conocer la posición absoluta del eje. Esto se puede lograr, por ejemplo, mediante el uso de discos ranurados con múltiples filas codificando cada posición de forma única. Una desventaja de este tipo de encoders es que requiere un mayor número de sensores y suele hacerse más compleja la codificación conforme se requiere mayor resolución.

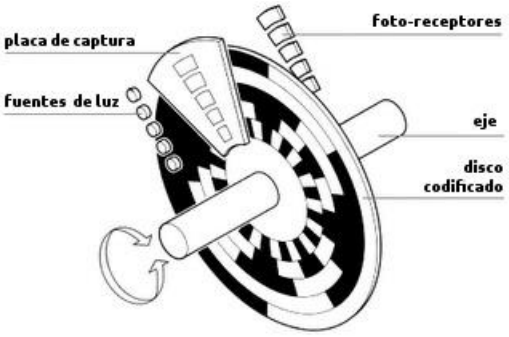


Figura 2. Representación de encóder absoluto.

4. Conocimientos previos

Los conocimientos necesarios para la realización de la práctica:

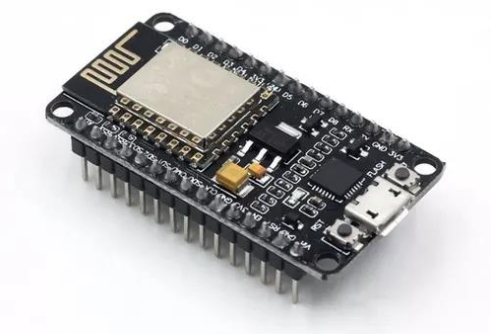
- Conocimientos básicos de computación y programación.
- Conocimientos básicos de electrónica.
- Conocimientos básicos de control.

5. Materiales y Equipo

Para la realización de la práctica es necesario contar con lo siguiente:

- Una computadora con Arduino IDE. (1)
- Tarjeta de desarrollo NodeMCU con Módulo WiFi ESP 8266. (2)
- Cable de conexión entre tarjeta y computadora. (3)
- 2 motorreductores DC de doble eje. (4)
- 2 encoders rotativos de cuadratura. (5)
- Puente H L293D. (6)
- Protoboard. (7)
- Alambre para conexiones. (8)
- Chasis para construcción del robot móvil. (9)
- Fuente de alimentación para motores / baterías.

(1)



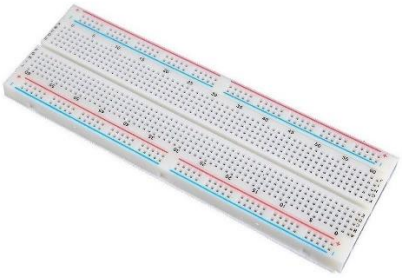
(2)

(3)

(4)

(5)

(6)



(7)

(8)

(9)

6. Preparativos previos y recomendaciones

- Es recomendable establecer una carpeta específica en la computadora para el desarrollo de la práctica.

7. Desarrollo de la práctica

a) Funcionamiento del encóder incremental de cuadratura

El primer paso de esta práctica consiste en instalar el encóder en el eje del motor, esto dependerá del tipo de motor y el tipo de encóder que se esté utilizando. Para el encóder recomendado se puede atornillar el disco ranurado de un lado del motor, del lado contrario de dónde se colocará la llanta. Por otra parte, será necesario ajustar la posición de los sensores ópticos para que detecten correctamente las ranuras del disco.

Recomendación: si se está utilizando el encóder sugerido en la práctica, se identificó que éste funciona mejor cuando el disco se encuentra más cerca de la parte inferior del sensor, además de tener una inclinación con respecto al disco ranurado, como se muestra en la figura 3. Adicionalmente, si no se está seguro de la colocación del encóder, se recomienda continuar con la práctica y hacer los ajustes necesarios cuando se estén obteniendo las mediciones de los sensores para verificar que detectan correctamente las ranuras del disco.

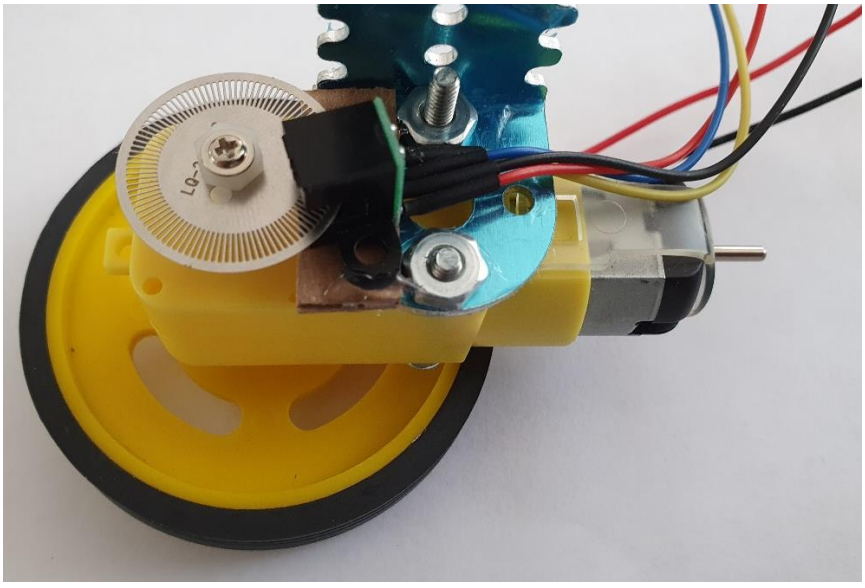


Figura 3. Instalación del encóder en el eje del motor.

Al tratarse de un encóder incremental de cuadratura se requiere de dos señales desfasadas 90° para determinar la posición angular y la dirección de giro del motor, a estas señales las llamaremos canal A y canal B. Con el cambio de estas dos señales, tanto de subida como de bajada, se puede constatar que el eje ha girado; para determinar la dirección del giro bastará con verificar el estado de la señal que no ha cambiado. Por ejemplo, si el eje gira en sentido horario (CW) se observa que, tras un cambio en la señal del canal A, el canal B se encuentra en el estado contrario al canal A. Por el contrario, si se realiza el giro del eje en sentido antihorario (CCW), tras un cambio en la señal del canal A, el canal B estará en el mismo estado que el canal A. Esto se puede visualizar mejor en la figura 4 donde se ejemplifica el cambio de las señales al hacer girar el eje en sentido horario y en sentido antihorario.

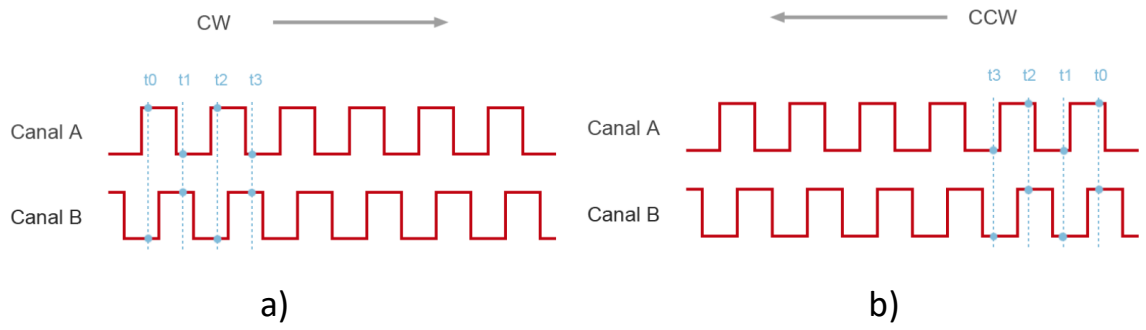


Figura 4. Señales del encóder de cuadratura para un giro a) en sentido horario y b) en sentido antihorario. [1]

También se identifica que para los cambios en la señal del canal B ocurre lo opuesto, en el caso de girar el eje en sentido horario, tras presentarse un cambio en el canal B, el canal A se encontrará en el mismo estado que el canal B; mientras que, al girar el eje en sentido antihorario, tras presentarse un cambio en la señal del canal B, el canal A se encontrará en el estado contrario al canal B.

Otra observación importante es que al considerar los 4 cambios en las señales se podrá tener una precisión de 4 veces el número de ranuras del disco, ya que cada una de las ranuras generará 4 cambios de las señales en tiempos distintos (flanco de subida y flanco de bajada para el canal A y para el canal B). Es por ello por lo que con el disco que tiene 100 ranuras podemos lograr una resolución en el encóder de 400 pasos por revolución.

Estas consideraciones se utilizarán posteriormente en la programación para poder medir la posición angular del motor y su sentido de giro.

El módulo de sensores ópticos del encóder utilizado tiene 4 cables, los cables negro y rojo son para tierra y voltaje respectivamente. Los dos cables restantes (azul y amarillo) corresponden a las señales de los canales A y B. Para la realización de esta práctica se conectaron las señales del encóder directamente a los pines D5 y D6 de la tarjeta NodeNCU ESP8266, tal como se muestra en la figura 5.

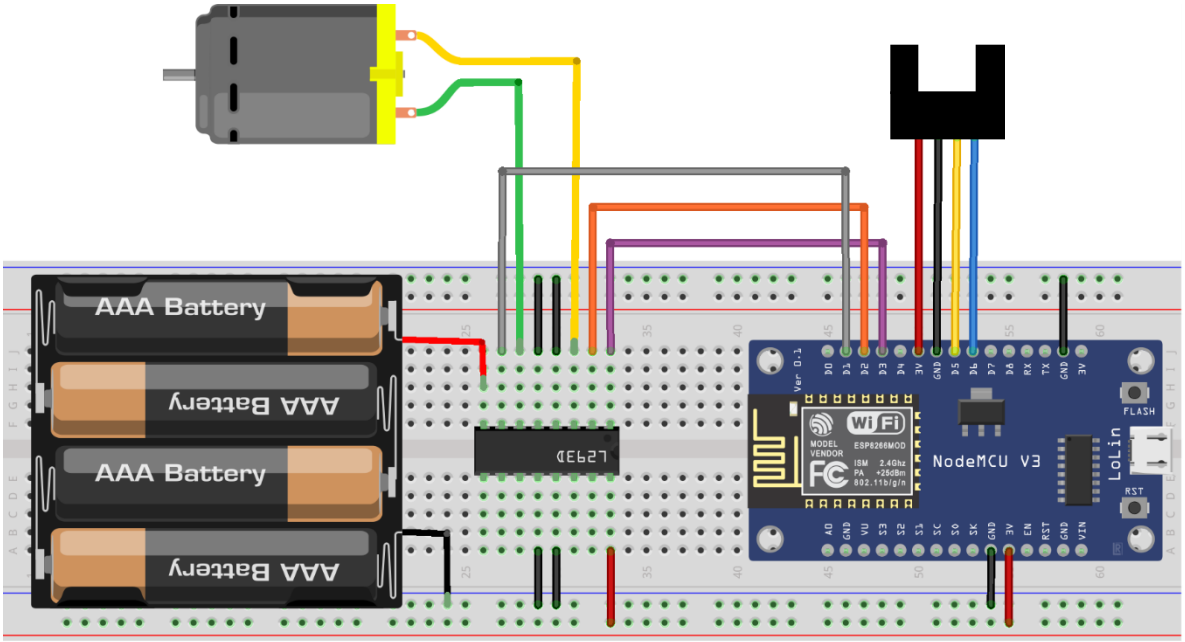


Figura 5. Esquema de conexiones eléctricas.

Para esta primera parte de la práctica basta con conectar el encóder a la tarjeta NodeMCU; las conexiones correspondientes al motor, el puente H y la alimentación se realizarán más adelante en el desarrollo de la práctica.

A continuación, se desarrollará un programa en Arduino IDE para comprobar el funcionamiento del encóder y verificar la detección de las ranuras de forma adecuada. Se propone el siguiente código.

Primero se inicializan las variables que se utilizarán en el código.

```
const int encoder_A = D5;
const int encoder_B = D6;
volatile int contador = 0;
int contador_ref = 0;
bool CW = true;
```

En la función *setup()* se configuran los pines que se utilizarán como entradas para las señales del encóder, en este caso se utiliza la configuración como *INPUT_PULLUP* ya que la conexión de este sensor requiere de la conexión de una resistencia en pullup que puede ser configurada mediante código para la tarjeta NodeMCU.

Además, se configura el uso de interrupciones para los pines en los que están conectadas las señales del encóder. Debido al tipo de funcionamiento, la mejor manera de manejar las señales es disparar interrupciones que se activan cada vez que cambia la señal, en lugar de dedicar el programa a estar verificando el estado de las señales en todo momento. Para configurar interrupciones se utiliza el comando:

attachInterrupt(digitalPinToInterrupt(encoder_B), cambioB, CHANGE);

Este comando recibe tres parámetros, el primero de ellos es el número del pin para el que se va a configurar la interrupción, el segundo corresponde al nombre de la función que se va a ejecutar cada vez que se presenta una

interrupción y, finalmente, el tercer parámetro indica el tipo de interrupción. El tipo de interrupción puede ser para flanco de subida (*RISING*), para flanco de bajada (*FALLING*) o para ambos flancos (*CHANGE*). Para esta aplicación utilizaremos ambos flancos de tal manera que podamos tener la máxima resolución del encóder como se explicó anteriormente.

Por último, se inicializa también aquí la comunicación serial para poder imprimir datos en el monitor serial.

```
void setup(){
   // Configuración de pines para el encoder de cuadratura
   pinMode(encoder_A, INPUT_PULLUP);
   pinMode(encoder_B, INPUT_PULLUP);
   attachInterrupt(digitalPinToInterrupt(encoder_A), cambioA, CHANGE);
   attachInterrupt(digitalPinToInterrupt(encoder_B), cambioB, CHANGE);

   Serial.begin(115200);
}
```

En la función *loop()* únicamente se programará la lógica para que se imprima el número de pasos del contador si hubo algún cambio, ya que toda la lógica de manejo de las señales del encóder se programará directamente en las funciones de las interrupciones.

```
void loop(){
   if (contador_ref != contador){
      contador_ref = contador;
      Serial.println(contador_ref);
   }

   delay(100);
}
```

Por último, se programan las funciones de las interrupciones para ambos pines. Para poder definir la función que se ejecutará con la interrupción se utiliza la siguiente estructura:

ICACHE_RAM_ATTR void nombre_funcion(){}

En donde se debe sustituir "*nombre_funcion*" por el nombre que se configuró al definir la interrupción y se incluye el código a ejecutar dentro de las llaves.

A continuación, se muestra una forma de programar las interrupciones para que sigan la lógica de los encoders de cuadratura. Lo que se hace es identificar el sentido de giro del motor, sabiendo cuál canal fue el que cambió y comparando los estados de las señales de ambos canales. Una vez identificado el sentido de giro se puede incrementar o decrementar un paso al contador según corresponda. Aquí se hace uso de la variable auxiliar *CW* para indicar si se trata de un giro en sentido horario (*CW* = true) o de un giro en sentido antihorario (*CW* = false).

```
ICACHE_RAM_ATTR void cambioA(){
    if (digitalRead(encoder_A) == digitalRead(encoder_B)){
       CW = true;
       contador++;
    }
    else{
       CW = false;
       contador--;
    }
}
```

```
ICACHE_RAM_ATTR void cambioB(){
    if (digitalRead(encoder_A) != digitalRead(encoder_B)){
       CW = true;
       contador++;
    }
    else{
       CW = false;
       contador--;
    }
}
```

Tras cargar el programa se podrá abrir el monitor serial para verificar los pasos que ha detectado el encóder, en este punto se podrá hacer girar la llanta manualmente y verificar que los pasos y el sentido de giro se detectan de manera correcta.

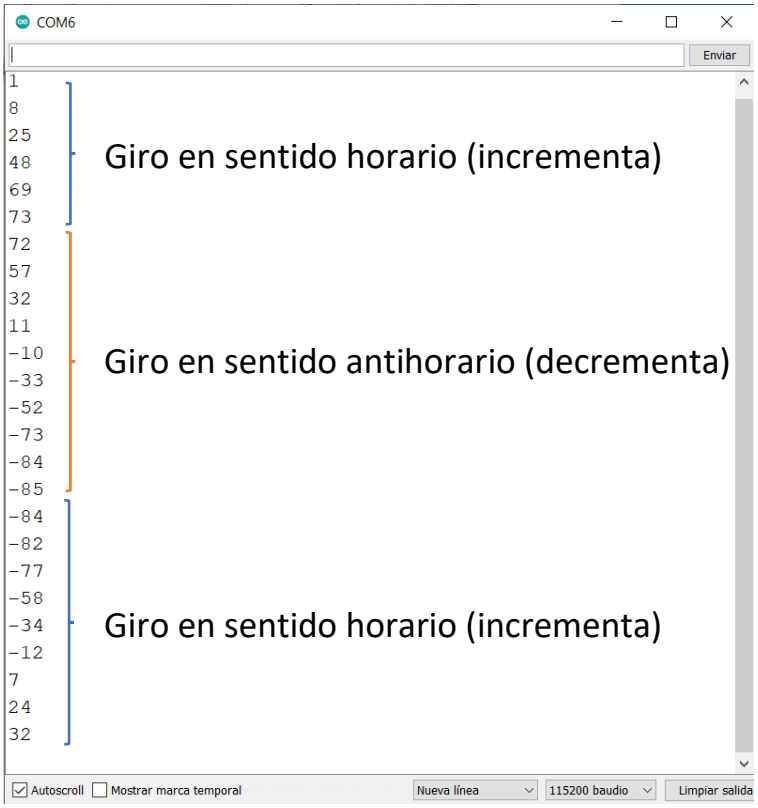


Figura 6. Verificación del funcionamiento del encóder.

Antes de continuar con la práctica:

- Instalar el encóder en el motor y hacer las conexiones correspondientes con la tarjeta NodeMCU ESP8266 (figura 5).
- Realizar la programación para detectar y visualizar los pasos del encóder. Verificar que tanto los pasos como el sentido de giro sean detectados de manera correcta.
- En caso de que no se detecten adecuadamente los pasos del encóder deberán hacerse los ajustes correspondientes, ya sea en la instalación física del encóder o en la programación de la tarjeta.

b) Control del motor DC con la tarjeta NodeMCU y el puente H L293D

Normalmente no es posible alimentar los motores directamente con el microcontrolador, ya que este no está diseñado para soportar grandes corrientes, sino únicamente para manejar la lógica del programa y las señales de control. Por ello, es necesario utilizar, además del microcontrolador, un controlador que permita alimentar a los motores correctamente.

Uno de los controladores más comúnmente utilizados para el manejo de motores pequeños de corriente directa es el puente H L293D. Este componente es muy económico y cuenta con 4 salidas que permiten controlar hasta dos motores de corriente directa en ambos sentidos de giro. De acuerdo con las especificaciones, este componente permite salidas de corriente continua de hasta 600 mA y soporta picos de corriente de hasta 1.2 A, mientras que el voltaje para las salidas puede ir desde 4.5 V hasta 36 V.

En la figura 7 se muestran los pines de este componente, donde las salidas se identifican con Y, las entradas de control con A, las señales de activación con EN, el voltaje lógico con VCC1 y la alimentación para los motores con VCC2.

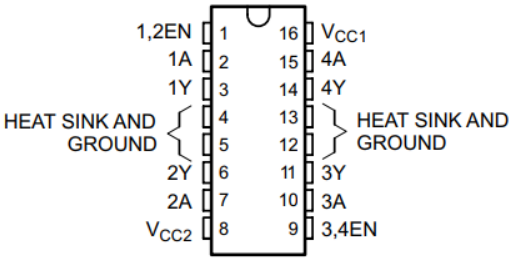


Figura 7. Configuración de pines del componente L293D. [2]

Para controlar dos motores DC con este componente deberán conectarse las terminales de un motor a las salidas 1Y y 2Y, y las terminales del otro motor a las salidas 3Y y 4Y. Para controlar la dirección de giro del motor se hará uso de las dos entradas de control correspondientes, ya que al poner la entrada 1A en HIGH y la entrada 2A en LOW, el motor girará en un sentido y al invertir ambas entradas se tendrá el giro en sentido contrario. Además, la velocidad de giro del motor podrá regularse introduciendo una señal PWM en los pines de activación EN. Esta configuración se muestra en la figura 8, así mismo, se puede hacer referencia a la figura 5 para las conexiones de uno de los motores.

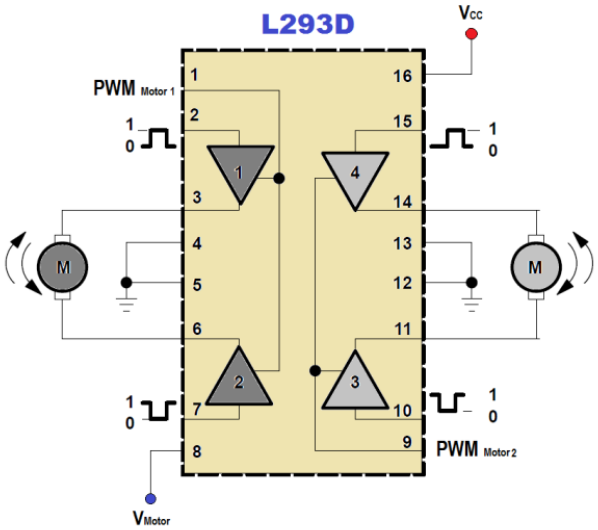


Figura 8. Configuración de L293D para control de dos motores DC. [3]

A continuación, se deberán realizar los ajustes en el programa para comprobar el funcionamiento del motor con el puente H. En el programa que se desarrolla a continuación se consideran las conexiones mostradas en la figura 5, además, se propone realizar las modificaciones sobre el programa anterior de tal manera que se vaya integrando la función del controlador junto con la lectura del encóder.

Una forma fácil de comprobar el funcionamiento del puente H consiste en desarrollar una función que permita leer un dato introducido mediante el monitor serial para, posteriormente, escribir este valor como PWM en la entrada de activación del controlador (EN). La función para leer el dato desde el monitor serial puede escribirse de la siguiente manera.

Primero se deben agregar las variables que se utilizarán para el almacenamiento de los datos leídos desde el monitor serial. Estas se agregan al principio del código en la sección dónde se declaran las variables.

```
// Variables para leer serial
const byte n = 32;
char datoRecibido[n];
int datoNumero = 0;
```

Al final del código se puede agregar la función que se utilizará para leer los datos desde el monitor serial. La lógica de esta función consiste en leer uno a uno los caracteres introducidos hasta el carácter que indica el final de línea (‘\n’).

Para verificar sí hay información disponible se utiliza la función:

Serial.available()

Para leer el siguiente carácter disponible se utiliza la función:

Serial.read()

Estos caracteres se almacenan primero en un arreglo de caracteres (*datoRecibido*) y posteriormente, una vez que se han terminado de leer todos los caracteres se realiza la conversión del dato recibido a tipo entero mediante la función *atoi()*.

```
void leerSerial() {
    static byte i = 0;
    char fin = '\n';
    char c;

    while (Serial.available() > 0) {
        c = Serial.read();
        if (c != fin) {
            datoRecibido[i] = c;
            i++;
            if(i >= n){
              i = n - 1;
            }
        }
        else {
            datoRecibido[i] = '\0';
            i = 0;
            datoNumero = atoi(datoRecibido);
        }
    }
}
```

Una vez que se ha definido esta función, bastará con llamar la función *leerSerial()* dentro de la función *loop()*, para que la variable *datoNumero* se actualice cada vez que haya un nuevo dato disponible desde el monitor serial.

Cabe mencionar que al estar utilizando el carácter de fin de línea (‘\n’) como señal del final del dato introducido, se deberá configurar también el monitor serial para que agregue este carácter cada vez que se envía un dato. Esto se puede configurar directamente en la barra inferior del monitor serial como se muestra en la figura 9.

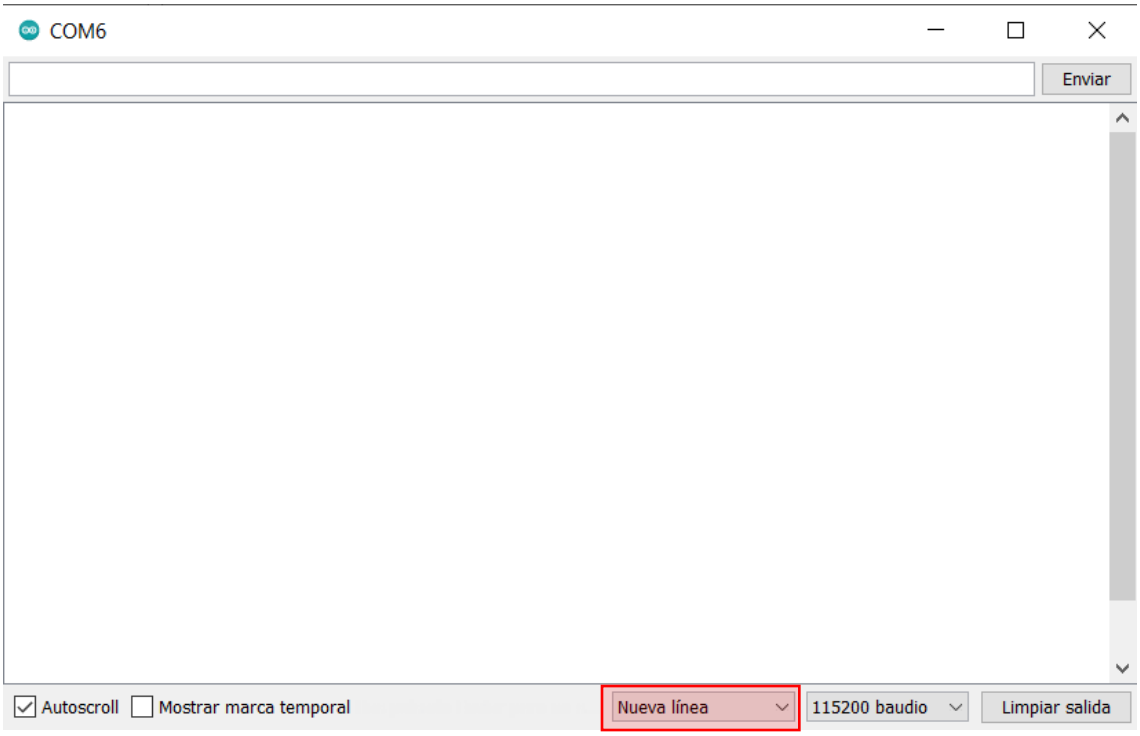


Figura 9. Configuración del monitor serial para agregar salto de línea.

Después de definir la función para leer el dato introducido desde el monitor serial deberá actualizarse el programa para enviar la señal PWM al puente H y controlar de esta manera la alimentación del motor. De igual manera es necesario declarar las variables que se utilizaran para el control del puente H al inicio del código, en este caso son las que se muestran a continuación.

```
// Variables puente H
const int driver_1A = D2;
const int driver_2A = D1;
const int driver_12EN = D3;
```

Se procede con la actualización de la función *setup()* para la configuración de los pines utilizados para el controlador.

```
void setup(){
   // Configuración de pines para el encoder de cuadratura
   pinMode(encoder_A, INPUT_PULLUP);
   pinMode(encoder_B, INPUT_PULLUP);
   attachInterrupt(digitalPinToInterrupt(encoder_A), cambioA, CHANGE);
   attachInterrupt(digitalPinToInterrupt(encoder_B), cambioB, CHANGE);

   // Configuración de pines para puente H
   pinMode(driver_1A, OUTPUT);
   pinMode(driver_2A, OUTPUT);
   pinMode(driver_12EN, OUTPUT);
   digitalWrite(driver_1A, LOW);
   digitalWrite(driver_2A, LOW);
   analogWrite(driver_12EN, 0);

   Serial.begin(115200);
}
```

Por último, se actualiza la función loop() para que se lea el dato introducido desde el monitor serial y, posteriormente, se configuren los pines conectados al puente H para determinar la dirección de giro (con las señales 1A y 2A) y la velocidad de giro (con la señal PWM del pin EN).

```
void loop(){
   if (contador_ref != contador){
      contador_ref = contador;
      Serial.println(contador_ref);
   }
   leerSerial();
   if(datoNumero >= 0){
     digitalWrite(driver_1A, LOW);
     digitalWrite(driver_2A, HIGH);
     analogWrite(driver_12EN, datoNumero);
   }
   else{
     digitalWrite(driver_1A, HIGH);
     digitalWrite(driver_2A, LOW);
     analogWrite(driver_12EN, -datoNumero);
   }
   delay(100);
}
```

Con estos cambios queda completo el código para probar el funcionamiento de los motores con el puente H. Para probarlo simplemente es necesario cargar el programa a la tarjeta NodeMCU, conectar la alimentación de los motores adecuadamente al controlador (figura 5) y abrir el monitor serial para el puerto COM correspondiente (figura 9). Para controlar el motor deberá escribirse un número entero entre -1023 y 1023 y dar clic en el botón de enviar.

Se debe verificar también que al introducir un valor positivo el motor debe girar en sentido horario, y al introducir un valor negativo el motor debe girar en sentido antihorario. Adicionalmente, es importante observar que debido a las características mecánicas del motor este puede tener un intervalo de valores bajos en el que la energía no es suficiente para hacer girar el motor, por lo que para observar el giro puede ser necesario introducir valores mayores. La velocidad de giro del motor dependerá del motor que se esté utilizando, el voltaje de la fuente que se administra y el PWM que se configura en el controlador, es por ello por lo que en la siguiente sección de la práctica se realizará un control de velocidad que permita obtener la velocidad deseada.

Antes de continuar con la práctica:

- Realizar las conexiones correspondientes del motor, del controlador y de la fuente de alimentación. (figura 5)
- Realizar la programación para controlar la velocidad del motor a partir del número introducido desde el monitor serial.
- Verificar que el motor funciona de manera correcta, ajustando la velocidad y la dirección de giro según el valor ingresado. En caso de que el sentido de giro no corresponda pueden intercambiarse los cables que conectan al motor, de tal manera que la dirección de giro coincida con la indicada y también con el sentido de los pasos que se miden con el encóder.

c) Controlador PID de velocidad del motor DC

Para lograr que el motor gire adecuadamente a la velocidad deseada es necesario implementar un controlador que haga el ajuste necesario en el PWM para lograr dicha velocidad. También, esto es necesario para compensar las posibles perturbaciones que se presentan en un ambiente real, por ejemplo, variaciones en el peso del robot móvil, variaciones en el voltaje de la fuente de alimentación, cambios en la fricción, el tipo de llanta y terreno, entre otras.

Uno de los tipos de controladores de lazo cerrado más comúnmente utilizados es el controlador PID. Este tipo de controlador utiliza la retroalimentación de las salidas y el valor deseado para calcular el error y, con base en ello, ejerce tres tipos de acciones de las que proviene el nombre: Proporcional, Integral y Derivativo. Cada una de estas tres componentes tiene un efecto distinto en el control del sistema, que en su conjunto logran obtener la salida y el desempeño deseado.

En forma simple se pueden describir los tres componentes del controlador PID de la siguiente manera:

- *Proporcional*: la variable de control se ajusta de manera proporcional al error, esto quiere decir, entre mayor es el error mayor será la acción de corrección y viceversa. En principio, esto funciona bien para disminuir el error a la salida del sistema, sin embargo, tiene el problema de que nunca se llega a eliminar por completo el error.
- *Integral*: la acción integral resulta proporcional al acumulado del error, es decir, a la integral del error. Esta componente ayuda a corregir el error en estado permanente de un controlador solamente proporcional, ya que, aunque la parte proporcional no llegue a corregirlo por completo, la parte integral acumulará el error y hará la corrección hasta que se logre eliminar. La principal desventaja es que puede incrementar las oscilaciones en la respuesta del sistema.

- *Derivativo*: la componente derivativa del controlador toma acción de acuerdo con la tendencia de cambio del error, es decir, de forma proporcional a la derivada del error. Esta componente resulta útil para mejorar el desempeño del controlador ya que, al considerar la derivada, la acción de corrección ya no depende únicamente del error actual (proporcional) o del acumulado de los errores pasados (integral), sino que también se considera la tendencia de cambio y se puede mejorar la reacción haciendo una estimación del comportamiento del error en el futuro. Una de las principales desventajas de esta componente es que resulta muy sensible al ruido en las mediciones.

Juntando estas tres componentes se puede obtener el modelo matemático de la señal de control PID como se muestra a continuación.

$$u = K_p e + K_i \int e\, dt + K_d \frac{de}{dt}$$

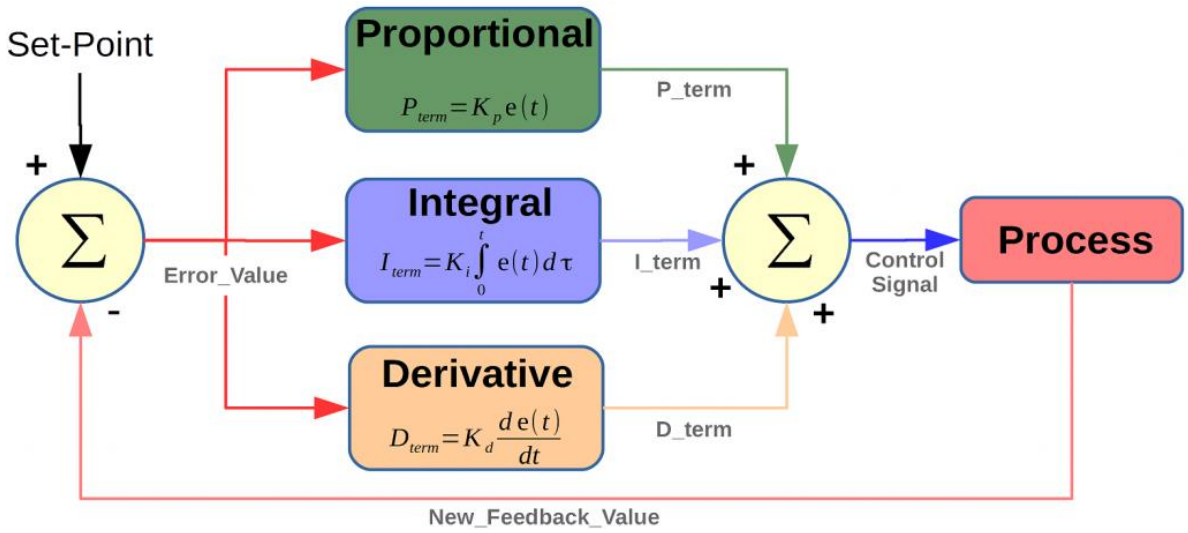


Figura 10. Diagrama de sistema con controlador PID. [4]

A continuación, se implementará este tipo de controlador en el programa de la tarjeta NodeMCU. Es importante mencionar que la ecuación anterior está en tiempo continuo, mientras que el microcontrolador funciona en realidad en tiempo discreto. Por ello, se utilizará la suma del error en lugar de la integral y el cambio del error en cierta unidad de tiempo en lugar de la derivada. Estas consideraciones pueden funcionar de manera correcta para tiempos de muestreo muy pequeños, para tiempos de muestreo grandes puede ser necesario aplicar la teoría de controladores en tiempo discreto que no será abordada en esta práctica. Para mayor referencia sobre los controladores PID, controladores en tiempo discreto y teoría de control se recomienda consultar libros de control como el de Norman Nise [5].

La primera modificación en el código corresponderá a añadir la declaración de las variables que se utilizarán para la implementación del controlador PID.

```
// Variables para el controlador PID
const int tiempoMuestreo = 25;
long tiempo = 0;
int error = 0;
float error_i = 0;
float error_d = 0;
int error_ant = 0;
float pwmControl = 0;
int pwmSalida = 0;
float kp = 100;
float ki = 100;
float kd = 1;
float u = 0;
float y = 0;
```

También deberá ser modificado el código dentro de la función *loop()*, de tal manera que se realice el ajuste en la señal PWM que se envía al puente H para controlar el motor según la ley de control del controlador PID propuesto.

```
void loop(){
   if (millis() > tiempo + tiempoMuestreo){
     error = datoNumero - contador;
     if(pwmSalida < 1023 && pwmSalida > -1023){
      error_i = error_i + float(error)/40;
     }
     error_d = float(error - error_ant)*40;

     pwmControl = kp * float(error) + ki * error_i + kd * error_d;

     u = float(datoNumero)/10;
     y = float(contador)/10;
     Serial.print(u);
     Serial.print("\t ");
     Serial.println(y);

     pwmSalida = int(pwmControl);
     if(datoNumero == 0 && contador == 0){
       pwmSalida = 0;
       pwmControl = 0;
       error_i = 0;
     }
     if(pwmSalida > 1023){pwmSalida = 1023;}
     if(pwmSalida < -1023){pwmSalida = -1023;}
     if(pwmSalida >= 0){
       digitalWrite(driver_1A, LOW);
       digitalWrite(driver_2A, HIGH);
       analogWrite(driver_12EN, pwmSalida);
     }
     else{
       digitalWrite(driver_1A, HIGH);
       digitalWrite(driver_2A, LOW);
       analogWrite(driver_12EN, -pwmSalida);
     }

     error_ant = error;
     contador = 0;
     tiempo = millis();
   }

   leerSerial();
}
```

Ahora se explicará brevemente la lógica del programa.

Lo primero es establecer el tiempo de muestreo, es decir cada cuánto tiempo se va a realizar la actualización de la variable de control. Este tiempo de muestreo también corresponderá al intervalo de tiempo utilizado para el cálculo de la integral y la derivada del error. Para lograr esto se hizo uso de la variable *tiempoMuestreo*, en la que se almacena el tiempo de muestreo en milisegundos; en este caso se utilizó un tiempo de muestreo de 25 milisegundos.

La condición:

```
if (millis() > tiempo + tiempoMuestreo)
```

es la encargada de checar que haya transcurrido el tiempo de muestreo desde la última ejecución antes de actualizar los cálculos del controlador PID. Al final de esta condicional if() se observa que se almacena el error anterior, que será utilizado para la componente derivativa, así como también se reinicia el contador y se almacena el tiempo de la última ejecución.

A continuación, se realizan los cálculos del error, la integral del error y la derivada del error.

El error se calcula simplemente como la velocidad deseada menos la velocidad medida con el encóder. En este caso, se programó de tal forma que el valor ingresado corresponda a las revoluciones por segundo multiplicadas por 10, es decir, al introducir un valor de 10 en el monitor serial se está indicando que se desea una velocidad de 1.0 revoluciones por segundo; de igual manera introducir un valor de -5 y 25 representa una velocidad de -0.5 rev/s y 2.5 rev/s. Ya que el encóder utilizado cuenta con una resolución de 400 pasos por vuelta, se observa que en 25 milisegundos el número de pasos deberá corresponder directamente con el número ingresado en el monitor serial, es decir, para 1 rev/s, el encóder deberá detectar 10 pasos por tiempo de muestreo.

Esto resulta conveniente ya que para estas condiciones el error se puede calcular directamente como el número ingresado en el monitor serial menos el número de pasos detectados por el encóder cada tiempo de muestreo.

```
error = datoNumero - contador;
```

Por otro lado, la integral del error se calcula sumando el error actual al acumulado del error, además, se divide entre 40 para obtener el acumulado en unidades de segundos, considerando el tiempo de muestreo de 25 milisegundos. También se utiliza una condición para verificar si ya se llegó al valor máximo de PWM, ya que en este caso se ha alcanzado la velocidad máxima del motor, y no debe seguir aumentándose el acumulado del error porque no podrá realizarse la corrección adicional.

```
if(pwmSalida < 1023 && pwmSalida > -1023){
    error_i = error_i + float(error)/40;
}
```

Para la componente derivativa se calcula el cambio del error por unidad de tiempo. Para esto se utiliza el valor almacenado del error anterior y se resta del error actual. Aquí también se realiza la corrección para tener las unidades en segundos al multiplicar por un factor de 40.

```
error_d = float(error - error_ant)*40;
```

Una vez que se tienen las 3 componentes, la señal de control (*pwmControl*), se calcula al sumar las tres componentes multiplicadas por sus respectivas constantes.

```
pwmControl = kp * float(error) + ki * error_i + kd * error_d;
```

Por último, se muestran los datos de la velocidad deseada y la velocidad actual en unidades de [rev/s] y se realizan los ajustes para sacar la señal PWM al pin que va conectado al puente H.

Para poder visualizar el funcionamiento del controlador se hará uso de otra herramienta incluida en el Arduino IDE que se llama *Serial Plotter*. Este puede abrirse una vez conectada la tarjeta y seleccionado el puerto COM, al igual que el monitor serial, desde la barra de menú /Herramientas/Serial Plotter, tal como se muestra en la figura 11.

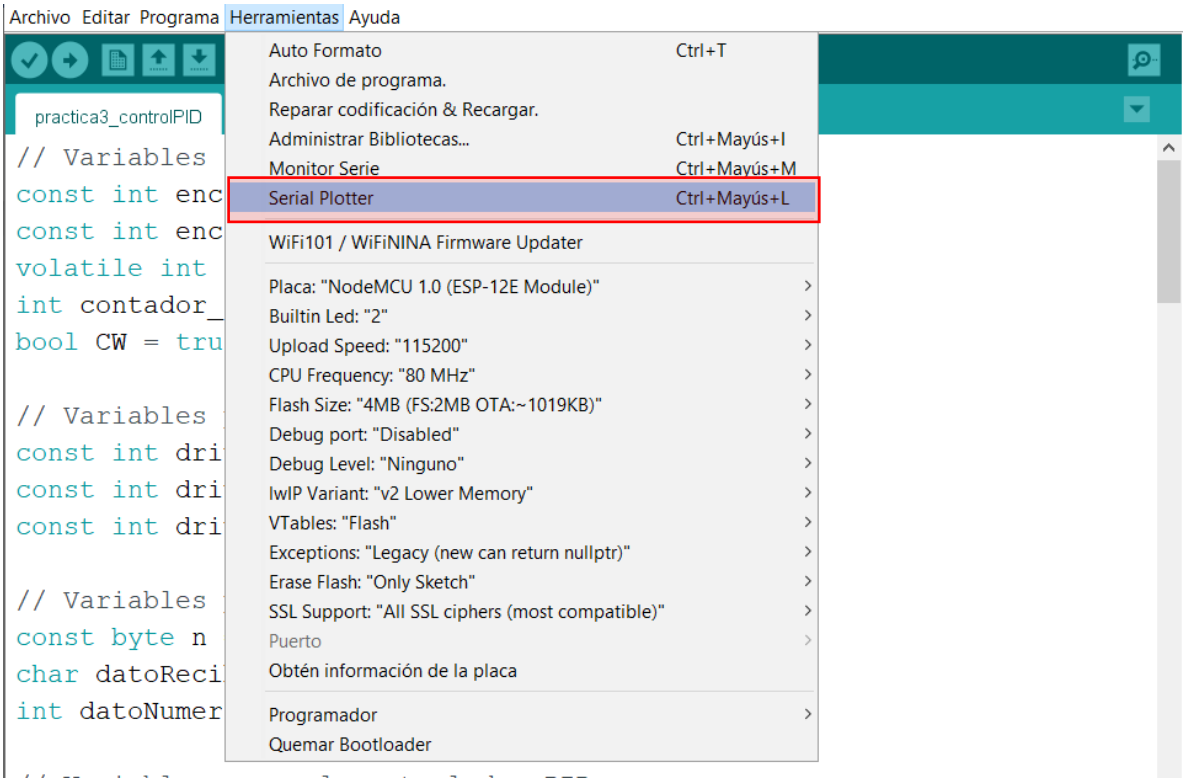


Figura 11. Opción para iniciar el Serial Plotter.

Tras abrir el Serial Plotter se deberá iniciar una nueva ventana en la que se grafican los datos que se imprimen desde el programa. Es importante recordar que, para la función que se implementó para leer los datos del monitor serial, es necesario seleccionar la opción de “Nueva línea” en la barra inferior de la ventana y configurar la velocidad de transmisión correspondiente, como se muestra en la figura 12.

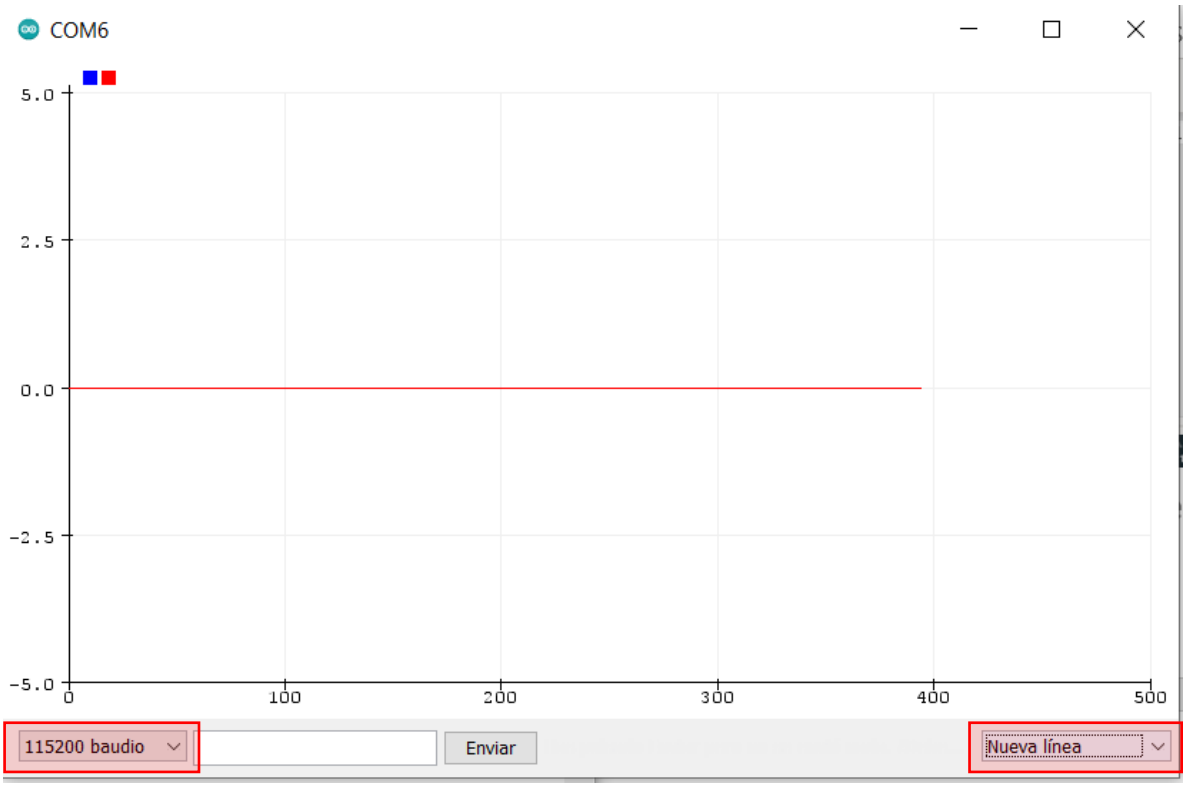
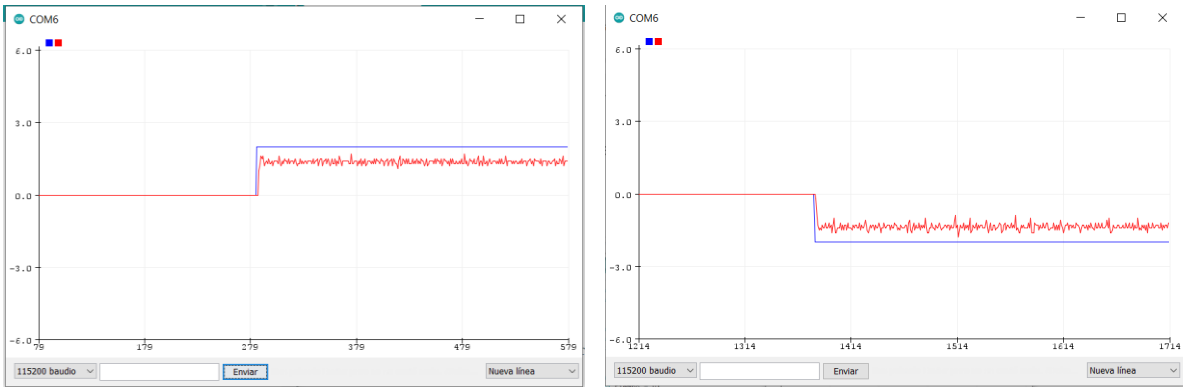


Figura 12. Ventana del Serial Plotter.

Desde la ventana del Serial Plotter también es posible enviar datos utilizando el recuadro correspondiente y el botón de “Enviar”. En la gráfica, la línea azul representa el valor de velocidad deseado y la línea roja representa la velocidad real del motor.

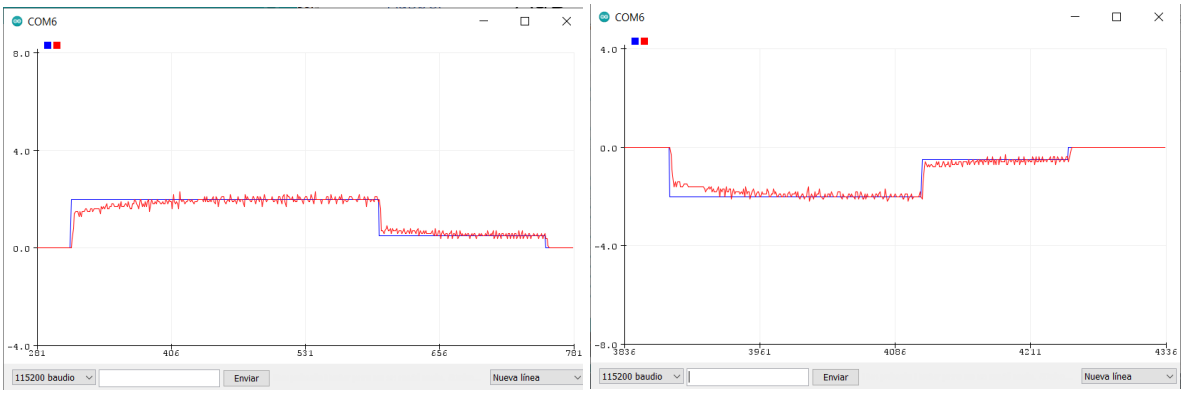
A continuación, se presentan los resultados obtenidos con el controlador. Primero se configuraron las constantes para que únicamente actuara la componente proporcional (kp=100, ki=0, kd=0). En esta prueba se aprecia que se mantiene un error en estado permanente. (Figura 13)



a) b)

Figura 13. Desempeño del controlador solo con componente proporcional a una velocidad de 2 rev/s, a) en sentido horario, b) en sentido antihorario.

Al introducir la componente integral se observa que es posible eliminar el error en estado permanente, por ejemplo, asignando los valores: kp=100, ki =100, kd =0.

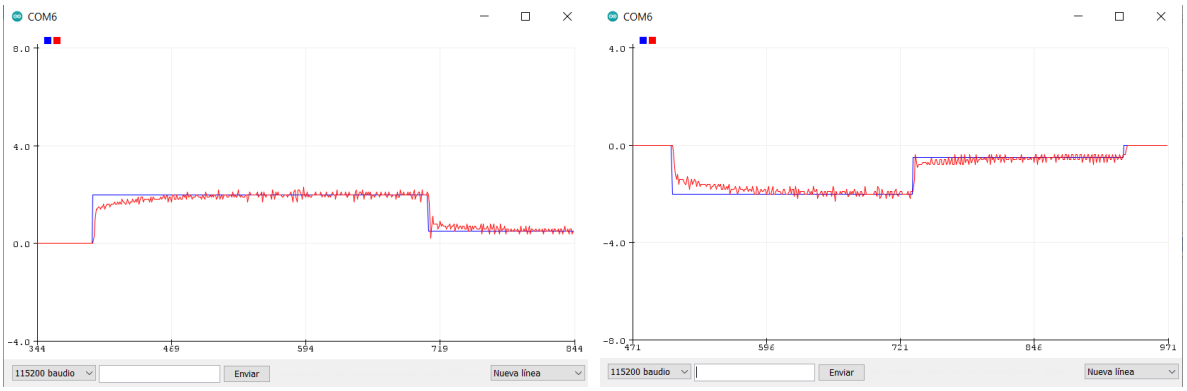


a) b)

Figura 14. Desempeño del controlador con componente proporcional e integral, a) en sentido horario, b) en sentido antihorario.

Por último, se muestra el comportamiento con el controlador PID completo con sus tres componentes: proporcional, integral y derivativa.

(kp=100, ki =100, kd =1)



a) b)

Figura 15. Desempeño del controlador con componentes proporcional, integral y derivativa, a) en sentido horario, b) en sentido antihorario.

Como último paso para poder definir correctamente el controlador PID, es necesario sintonizar el controlador, es decir, seleccionar los valores de las constantes que permitan obtener el desempeño deseado. Si bien es posible sintonizar el controlador de forma manual, existen metodologías que permiten sintonizar el controlador y determinar las constantes. Uno de los más conocidos es el método de Ziegler Nichols, el cual se encuentra bien explicado en la siguiente referencia:

https://controlautomaticoeducacion.com/control-realimentado/ziegler-nichols-sintonia-de-control-pid/

Realiza las siguientes actividades para concluir con la práctica:

- Realizar la programación del controlador PID para lograr el control de velocidad del motor DC.
- Verificar el funcionamiento del controlador y analizar el efecto que tienen cada una de las componentes del controlador PID.
- Realizar la sintonización del controlador PID para determinar los valores de las constantes que proporcionen un desempeño adecuado.
- Construir un robot móvil diferencial que cuente con dos ruedas fijas motorizadas con control de velocidad PID independiente. Para esto deberán añadirse las conexiones necesarias para incluir el segundo motor y encóder e integrarlo todo en el chasis del robot. También es necesario modificar el programa para que reciba dos datos que corresponderán a las velocidades deseadas de cada una de las ruedas del robot móvil.

8. Resultados

Como resultado de esta práctica se debe entregar el programa para el funcionamiento del robot móvil, así como un video en el que se explique el funcionamiento y se observe el control de velocidad PID en ambas llantas y en ambos sentidos de giro.

9. Conclusiones

En esta práctica se obtuvieron los conocimientos teóricos y prácticos necesarios para lograr la construcción y el funcionamiento de un robot móvil diferencial. También se abordó un elemento importante para la robótica móvil que es el control de velocidad de las ruedas. Así mismo, para poder implementar el control de velocidad se analizaron los conceptos básicos de los controladores PID y se realizó su implementación haciendo uso de un encóder rotativo de cuadratura y un puente H. Con esta base es posible comprender el funcionamiento básico de los robots móviles, además, estos conocimientos pueden ser extrapolados a sistemas y robots más complejos.

Bibliografía

[1] https://www.luisllamas.es/arduino-encoder-rotativo/

[2] Texas Instruments. Datasheet: L293x Quadruple Half-H Drivers.

[3] https://ardubasic.wordpress.com/2014/05/23/control-de-motores-de-cc-con-l293d/

[4] https://microcontrollerslab.com/pid-controller-implementation-using-arduino/

[5] Nise, Norman S. (2011). ***Control Systems Engineering***. (6th edition). John Wiley & Sons.