Proyecto Final - Teoría de Control

Pablo Bermeo
Telecomunicaciones
Universidad de Cuenca
Cuenca, Ecuador
pablo.bermeog@ucuenca.edu.ec

Sebastián Guazhima
Telecomunicaciones
Universidad de Cuenca
Cuenca, Ecuador
sebastian.guazhima@ucuenca.edu.ec

Tyrone Novillo
Telecomunicaciones
Universidad de Cuenca
Cuenca, Ecuador
tyrone.novillo@ucuenca.edu.ec

Carlos Calle
Telecomunicaciones
Universidad de Cuenca
Cuenca, Ecuador
carlos.calle@ucuenca.edu.ec

Freddy López
Telecomunicaciones
Universidad de Cuenca
Cuenca, Ecuador
freddy.lopez@ucuenca.edu.ec

***Abstract*—Este informe presenta el desarrollo e implementación de un controlador PID digital para un motor de CC utilizando la plataforma Arduino Mega. El sistema incorpora un EPC (Entrenador de Planta de Control), diseñado para la formación práctica en instrumentación y control. El encoder del EPC facilita la medición en tiempo real de la velocidad del motor, permitiendo ajustes precisos de control. El proceso de sintonización de los valores de ganancia del PID se realiza utilizando el método de última ganancia. Los resultados se presentan en una pantalla Nextion, proporcionando una visualización clara del controlador aplicado en el sistema.**

***Index Terms*—**

I. Introducción

En este informe de laboratorio, se presenta la implementación de un controlador PID (Proporcional-Integral-Derivativo) para regular la velocidad de un motor en una planta de entrenamiento. El controlador PID ha sido diseñado en su versión discreta y sintonizado cuidadosamente para optimizar el rendimiento del sistema.

El propósito de este experimento es demostrar la eficacia de un controlador PID en la regulación de la velocidad del motor, proporcionando una respuesta rápida y estable ante variaciones en la carga y perturbaciones. Además, se ha implementado una interfaz humano-máquina (HMI) utilizando una pantalla táctil NEXTION, que permite la visualización en tiempo real de las señales del sistema y el ajuste de los parámetros del controlador. Esta interfaz mejora significativamente la interacción del usuario con el sistema, facilitando el monitoreo y control de la planta.

Este informe describe los procedimientos necesarios para el diseño y sintonización del controlador PID discreto, así como los pasos para la integración con la planta de entrenamiento y la implementación de la HMI (Pantalla NEXTION). Además, se detallan también los resultados obtenidos y se discuten las observaciones realizadas del mismo. La finalidad es proporcionar una comprensión clara y completa del proceso de codificación implementado para control y su efectividad en el mantenimiento de la velocidad

II. Marco Teórico

A. PID Digital

Un controlador PID (Proporcional-Integral-Derivativo) digital es una versión discretizada del controlador PID continuo, adaptada para sistemas de control en tiempo discreto.

El algoritmo del controlador PID digital se basa en la transformación de las ecuaciones del controlador PID continuo al dominio discreto.

La ecuación de un controlador PID continuo está definida por:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

donde $u(t)$ es la señal de control, $e(t)$ es el error, y K_p, K_i, y K_d son las constantes de ganancia proporcional, integral y derivativa, respectivamente. Esta estructura se presenta en la figura 1. Para implementar este controlador en un sistema digital, es necesario discretizar estas componentes.

La discretización implica convertir las integrales y derivadas en formas que puedan ser calculadas en intervalos de tiempo discretos. Utilizando métodos de discretización, la integral y la derivada se pueden aproximar como:

$$\int_0^t e(\tau)d\tau \approx \sum_{i=0}^{k} e(i)T$$

$$\frac{de(t)}{dt} \approx \frac{e(k) - e(k-1)}{T}$$

donde T es el periodo de muestreo. Sustituyendo estas aproximaciones en la ecuación del PID continuo, se obtiene la ecuación de recurrencias para el PID discreto:

$$u(k) = u(k-1) + K_p(e(k) - e(k-1)) +$$
$$+ \frac{T}{T_i}e(k) + \frac{T_d}{T}(e(k) - 2e(k-1) + e(k-2))$$

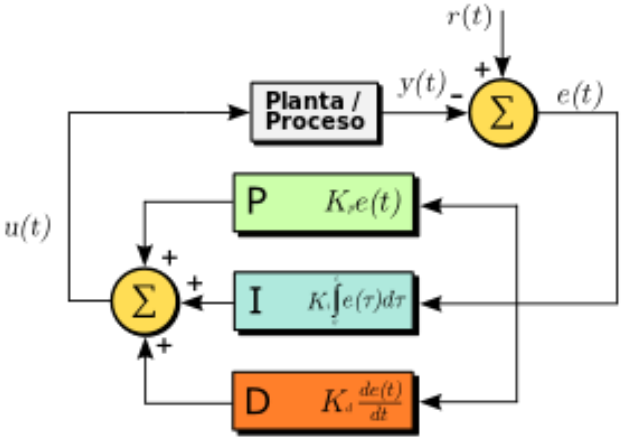

Fig. 1. Diagrama de bloques del control PID a lazo cerrado

Esta ecuación permite calcular la señal de control $u(k)$ en el instante k usando los valores de error actuales y anteriores, así como la señal de control previa.

B. Arduino Mega

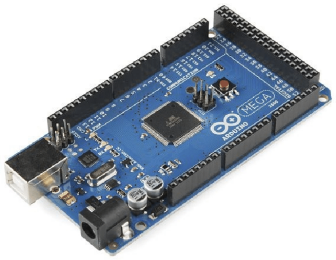

Fig. 2. Arduino Mega

El Arduino Mega es una tarjeta de aprendizaje (aunque se ha expandido a implementiaciones medianamente indrustriales) que puede servir como base para un prototipo. La tarjeta tiene incorporado un microprocesador Atmega2560 y es una mejora de los diseños de Arduino UNO y Arduino NANO. Esto, debido a su expansión en la cantidad de entradas analógicas y digitales.

Al igual que dispositivos anteriores es posbible compensar la falta de corriente ante cargas a través de una fuente externa que se implementa mediante una de sus entradas laterales. El arduino MEGA, al igual que sus antecesores, permite la incorporación de *shields* o módulo sobre su carcasa.

C. Nextion

El manejo de la pantalla Nextion se basa en la integración eficiente entre hardware y software para desarrollar interfaces de usuario interactivas y personalizadas. La pantalla Nextion es una solución HMI (Human-Machine Interface) que permite a los desarrolladores diseñar interfaces gráficas sin necesidad de profundos conocimientos en programación. Esta pantalla ofrece un entorno de desarrollo visual, el Nextion Editor, que facilita la creación y configuración de interfaces mediante herramientas intuitivas y elementos predefinidos como botones, cuadros de texto, imágenes y gráficos.

Una de las características más destacadas de la pantalla Nextion es su capacidad de comunicación serial. Esta comunicación se establece a través de puertos UART (Universal Asynchronous Receiver-Transmitter), permitiendo la transmisión de datos entre la pantalla y otros dispositivos, como microcontroladores, de manera eficiente y fiable. El protocolo de comunicación serial utilizado por la pantalla Nextion es simple y estructurado, permitiendo el envío y recepción de comandos y datos. Esta comunicación bidireccional es esencial para actualizar la interfaz en tiempo real y para que el microcontrolador pueda responder a las interacciones del usuario con la pantalla.

La configuración de la interfaz en la pantalla Nextion se realiza mediante el Nextion Editor, un software que permite a los desarrolladores diseñar y personalizar sus proyectos de forma visual. Dentro de este editor, los usuarios pueden arrastrar y soltar elementos gráficos, ajustar sus propiedades y definir eventos y acciones que se ejecutarán en respuesta a las interacciones del usuario. Además, el Nextion Editor genera el código necesario para que la pantalla interprete estas configuraciones y las ejecute. Este enfoque de desarrollo reduce significativamente la complejidad del diseño de interfaces gráficas, haciendo que el proceso sea accesible incluso para aquellos con poca experiencia en programación.

La implementación de la pantalla Nextion en proyectos de electrónica y automatización mejora significativamente la interacción entre el usuario y el sistema. Gracias a su comunicación serial robusta y su entorno de desarrollo amigable, las pantallas Nextion se han convertido en una herramienta valiosa para diseñar interfaces HMI eficientes y atractivas. Estas características permiten a los desarrolladores centrarse más en la funcionalidad y el diseño de la experiencia del usuario, asegurando que los dispositivos y sistemas sean más intuitivos y fáciles de usar.

D. EPC - Entrenador de Planta de Control

El Entrenador de Planta de Control, o EPC, es un equipo que incluye sistemas típicos de instrumentación y control. Esta diseñado para crear plantas reales para enseñanza y experimentación práctica.
Entre los diversos instrumentos incluye un motor de corriente continua (Motor DC), que es controlado por una señal de voltaje DC que puede variar entre *0 [V]* y *5 [V]*. La salida del encoder es una señal pulsante.

En la Figura 3 se observan los pines que se utilizarán para este proyecto. Se debe conectar una salida analógica del Arduino Mega a la entrada de control del Motor DC (*MOTOR DC IN*), y conectar la referencia *GND* de la tarjeta a la referencia *GND* del EPC. Se conecta la salida del encoder del Motor DC del EPC llamado *ENCODER MDC*.
La entrada del Motor DC será la señal PWM y la salida

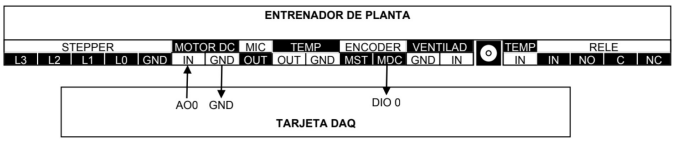

Fig. 3. Pines del EPC.

del encoder será la velocidad del motor. Ambas señales son digitales.

III. Desarrollo

A. Diagrama de instrumentación

Se desarrolló el diagrama de instrumentación del proyecto utilizando el software *Fritzing*. El resultado se presenta en la figura **??**. En este diagrama se pueden observar los elementos clave del sistema, incluyendo una pantalla, un Arduino Mega y un circuito para controlar la velocidad del motor. En este contexto, la EPC (Entrenador de Planta de Control) está representada por el motor, ya que este es el componente central del sistema a controlar.

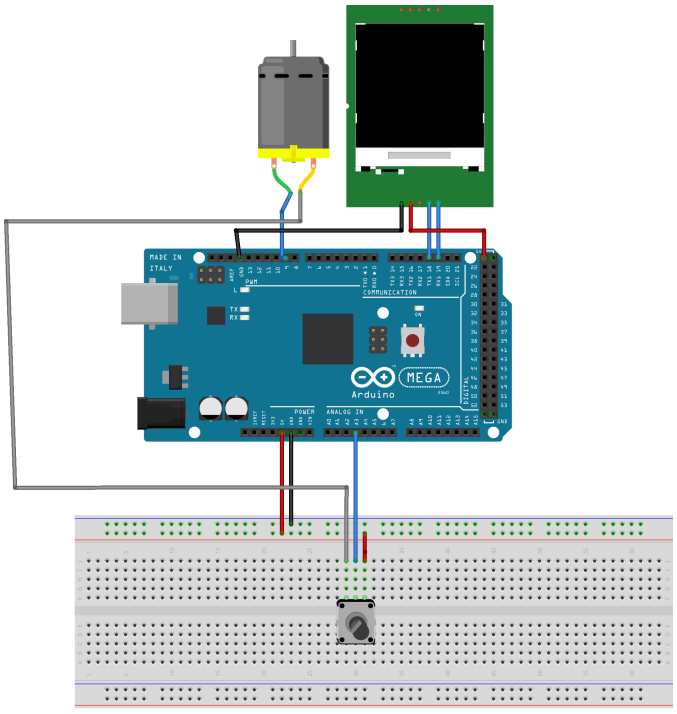

Fig. 4. Diagrama de instrumentación.

B. Codificación del Arduino Mega

La codificación del Arduino Mega se basó, principalmente, en dos secciones:

- Medición de las rpms del motor:
  Para este cometido se ha empleado el esquema presentado en la Figura 5. Note en el esquema que se ha planteado la recepción de pulsos mediante el puerno número 2 de

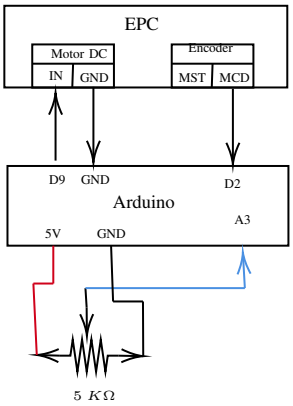

Fig. 5. Esquema general empleado para la codificación de la medición de velocidad y control Digital en el Arduino Mega.

Arduino. Este pin se ha de configurar como entrada para funciones de tipo *attach*. Esto, tal como se lo presenta a continuación

```
const int pin_InEncoder = 2;
attachInterrupt(digitalPinToInterrupt(...
    pin_InEncoder),
ContarPulsos, RISING );
```

De esta forma cuando el pin *pin_InEncoder* recibe un flanco de subida, ejecutará la función ContarPulsos. Esta última función tiene una defición simple y presentada a continuación:

```
void  ContarPulsos(){
pulsos = pulsos + 1;
}
```

Hasta este punto el pin en cuestión y la función presentada permiten tener un conteo contínuo de los pulsos provenientes del *Encoder* del EPC. En consecuencia, se procede a realizar un análisis de velocidad que considere una referencia de pulsos por vuelta en el motor y un tiempo de conteo de pulsos. De esta forma, será posible estimar su velocidad real.
En este sentido, se ha definido la constante T que refiere al periódo de muestreo o conteo. Las especificaciones del motor señalan que la cantidad de pulsos por vuelta es de 36. A partir de este valor es que se procede a realizar la siguiente relación:

$$\mathrm{V_{RPM}} = \frac{60(\text{Cantidad de pulsos contado en T})}{36T}$$

Esta última medición se ha de realizar cada T segundos, por lo que se ha definido una manera de cuantificar la cantidad de tiempo percibida por Arduino. Esto se ha realizado mediante la función `millis()` que lleva ese conteo. De tal manera que la instrucción:

```
unsigned long previousMillis = 0;
```

origina, al inicio de la codificación, una especie de cronómetro que se reiniciará cada que hayan pasado T segundos. El reinicio del contador, junto con la medición

de la velocidad en ese momento presente se codifica a través de una instrucción condicional:

```
if ( millis() - previousMillis >= T ) {
    float rpms = ( 60*pulsos )/( 36*T*0.001 );
    pulsos=0;
    previousMillis = millis();
}
```

A partir de todas las instrucciones anteriores, es posible medir la velocidad del motor. Como muestra de esto, considre la Figura 6.

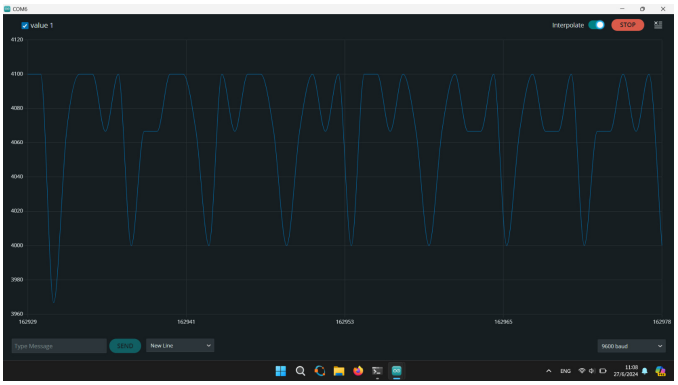

Fig. 6. Captura de la medición de velocidad. Observe que esto se se ha realizado incialmente a lazo abierto, de ahí las continuas oscilaciones y el hecho de que únicamente se esta presentando la velocidad proveniente del motor. El contexto de la imagen es que el motor fue sometido a los 5V máximos provenientes del Potenciometro. Esto originó una velocidad máxima entre 4000 y 5000 rpms.

La medición de velocidad permitió establecer un máximo al aplicar el voltaje de 5V del potenciometro. Este máximo se estimó en 4400 rpms.

- Implementación del control de velocidad a través de la Ecuación de recurrencias

  El objetivo de esta sección es la implementación de un controlador PID digital a partir de la ecuación de recurrencias. Dada la ecuación de recurrencias

$$u(k) = u(k-1) + k_c[e(k) - e(k-1) + \frac{T}{T_i}e(k)$$

$$+\frac{T_d}{T}(e(k) - 2e(k-1) + e(k-2))]$$

  Se puede ver que se requiere del error en tres instancias. Esto se ha de programar a través de un vector de tres elementos en Arduino. El valor *u* también requiere su programación como tipo *float* (aunque a la hora de enviarlo al motor, este último será de tipo entero); y finalmente se han de programar los valores para el control proporcional, derivativo e integral: k_p,k_i y k_d. Para la programación se los ha definido de la siguiente manera:

```
float e[3];
float u;
float Kp;
float Ti;
float Td;
```

Los valores para k_p,k_i y k_d, junto con los del error, se han inicializado en el apartado *setup()*:

```
Kp = 0.20;
Ti = 0.05;
Td = 0.1;
e[2] = 0; e[1] = 0; e[3] = 0;
```

Note que los valores del error, lógicamente, se han inicializado todos en cero y los valores de PID se han insertado con valores de prueba. Con estas variables es posible comenzar la codificación del PID digital, misma que se desarrollará luego de medir la velocidad. Es decir que cada T segundos se realizará una medición de la velocidad y un control de la misma. La lógica a implementar se describirá mediante los siguientes pasos:

1) Recorrer el vector de error para calcular uno nuevo y ser capaz de, en cada iteración, compararlo con el anterior: Para esto, se ha empleado una lógica simple en la que se desplaza *hacia la derecha* los valores del vector de error.

```
e[2] = e[1];
e[1]  = e[0];
e[0] = ref - rpms;
```

2) A partir de los nuevos valores de error es posible calcular la diferencia entre $U(K)$ y $U(K-1)$, es decir un $\Delta U = U(k) - U(K-1)$, cosa que es sencilla de ver, es el origen o equivale a la ecuación de recurrencias. En tanto se ha codificado:

```
int deltaU =
Kp * (e[0]-e[1] + 0.05/Ti * e[0]
+Td/0.05 *(e[0]-2*e[1]+e[2]));
```

3) Dado el ΔU, se procede a modificar la velocidad del motor, o la señal U enviáda hacia el mismo:

```
u += deltaU;
```

   En este sentido note una peculiaridad con la anterior sentencia: Dado que el motor únicamente podrá recibir un valor lógico de 0 a 255, u puede llegar a *desbordarse*, en tanto, se han programado límites en la codificación.

```
if (u>255)
    u= 255;
if (u<0)
    u = 0;
```

4) Llevar una nueva velocidad al motor: La nueva velocidad que llega al motor se presenta como un valor entero, debido a que el número enviádo al potenciometro está en PWM y se ubica entre 0 y 255 (dominio discreto).

```
analogWrite(PWM_out, int(u));
```

Dada esta codificación es posible realizar el control del motor mediante el PID digital.

C. Sintonización del controlador PID mediante el método de ganacia máxima

Para aplicar el método de máxima ganancia, se han realizado mínimas modificaciones sobre el código. Las modificaciones

se han dirigido, inicialmente al la eliminación de las constantes k_i y k_d; luego, se ha programado que el valor de potenciometro corresponda a la constante proporcional del PID, k_p y que sea capaz de proporcionar un valor entre 0 y 5. Además se ha ajustado una referencia fija de 3000 revoluciones, esto debido a que el objetivo acual es la visualización de oscilaciones sobre una referencia.

La programación de las modificaciones se presenta a continuación:

1) Ignorar las constantes k_i y k_d. Este modificación afecta únicamente a la forma de la ecuación de recurrencias, que ahora será unicamente proporcional. Esto se indica a través de la siguiente instrucción:

```
deltaU = Kp *  e[0];
U += deltaU;
```

2) La referencia se ha fijado en 3000 rpms al realizar

```
ref = 3000.0;
```

3) Permitir que el motor sea el elemento que altera el valor de k_p en un rango entre 0 y 5. Para esto, se procede a leer el valor de potenciometro (0 a 1023), convertirlo a PWM (0 a 255) y finalmente realizar una proporción a un rango entre 0 y 5. Esto último, empleando la expresión

$$k_p = \frac{5}{255}(\text{Potenciometro en PWM})$$

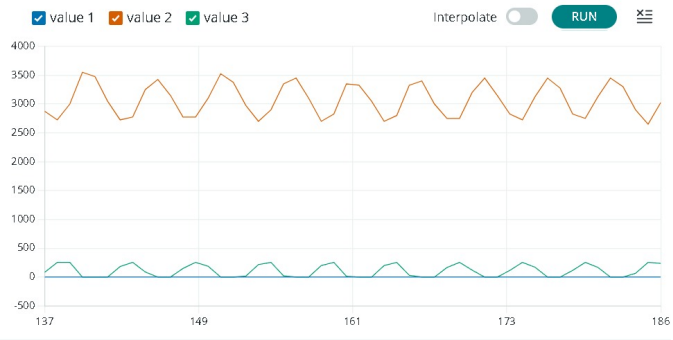

Fig. 7. Oscilación de la respuesta sobre la referencia.

Los resultados se presentan en la figura 7, donde se puede visualizar que efectivamente existe una valor de ganancia, para el cual se generan oscilaciones. El contexto de la Figura responde a un valor de k_p de 0.4250. Además, se requiere del periódo de oscilación. El periódo se ha obtenido a través de el conteo de datos de Arduino. Note que la gráfica informa el número de dato (Figuras 8 y 9). A partir de conocer que cada dato ocurre cada 50ms, se sabe que el periódo responde a la siguiente relación:

$$T_{\text{osilación}} = 50(\text{nro. de datos en un periódo}).$$

Los resultados del conteo se puede visualizar en las Figuras 8 y 9, se puede que el primer dato corresponde al número 172, y el siguiente al 177. En tanto el periódo de oscilación es de 250ms. No obstante se realizaron más pruebas con distintos valores en los que existía la oscilación.

La conclusión luego del análisis de varios valores fue una valor de ganancia máxima k_u de 0.42 y uno de periódo crítico de oscilación de 300ms.

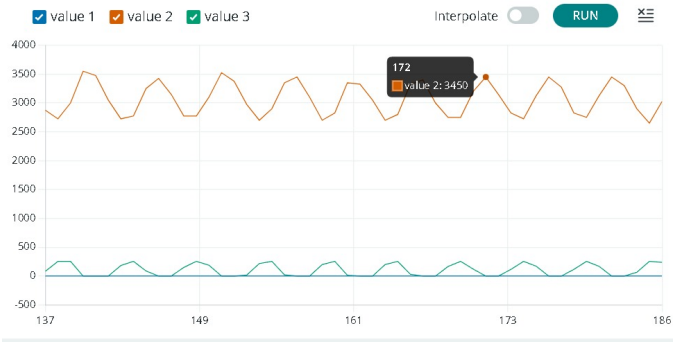

Fig. 8. Oscilación de la respuesta sobre la referencia.

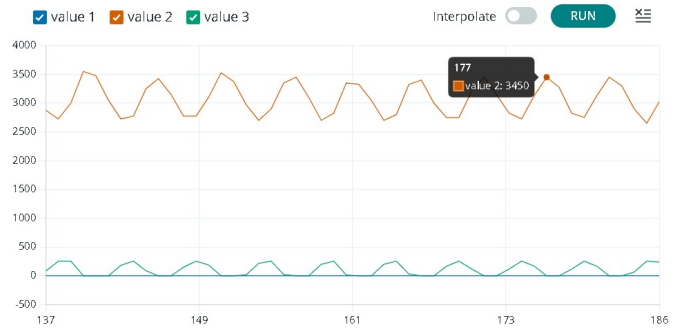

Fig. 9. Oscilación de la respuesta sobre la referencia.

Con estos datos es posible emplear las expresiones para calcular las distintas ganancias según el método de Ziegler Nichols. Esto resulta en los siguiente valores

$$k_p = \frac{k_u}{1.7} = 0.25$$
$$k_d = \frac{T_u}{2} = 0.15$$
$$k_i = \frac{T_u}{8} = 0.0375$$

La respuesta del sistema resultante se presenta se la sección de Análisis de resultados.

D. Codificación del HMI e interacción con Arduino

La comunicación mediante el Arduino con la pantalla Nextion se realiza de manera serial. En este caso, se utilizó la pantalla Nextion modelo Discovery NX4832F035_011 en disposición horizontal. La Figura 10 y 11 muestran las pantallas empleadas. La primera corresponde al ingreso de datos de parte del usuario que consiste en los siguientes elementos

con sus correspondientes nombres. Para la pantalla de ingreso de datos:

- Slider con nombre de objeto rpm
- Número entero variable con nombre de objeto kp
- Número entero variable con nombre de objeto ki
- Número entero variable con nombre de objeto kd
- Botón para el cambio de pantalla

Para la pantalla de gráfica de datos, se utilizaron los siguientes objetos:

- Plotter para graficar
- Botón para el cambio de pantalla
- Timer para graficar en tiempo real el valor de referencia y el valor real de revoluciones por minuto.
- Variable global con nombre rpm_real la cual tomará un valor enviado por la Arduino

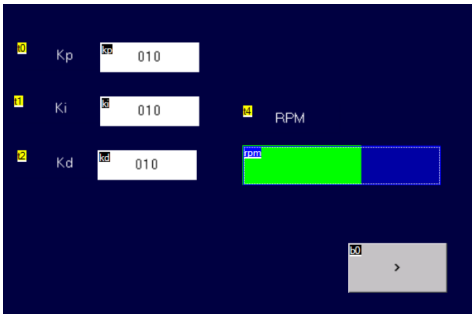

Fig. 10. Pantalla del ingreso de datos

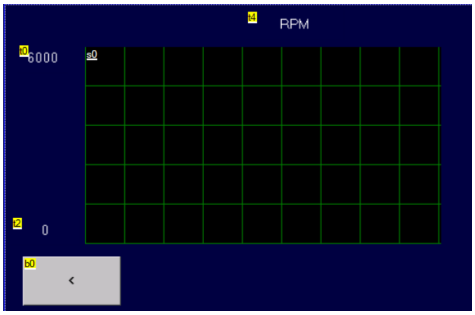

Fig. 11. Pantalla para mostrar datos

Cabe mencionar que no fue posible lograr la comunicación de la pantalla con controllino debido a que el dispositivo presentaba fallas en sus pines digitales. Una vez definidos.

Una vez definidos los objetos y sus valores, se deben configurar los comandos a ejecutarse al enviar y recibir datos en la Nextion. Para ello, en el botón de cambio de pantalla, se define el siguiente comando al ejecutarse el evento Touch press:

```
prints "b",1
print kp.val
print ki.val
print kd.val
print rpm.val
```

Lo que hace este comando es preparar en un mensaje que contiene en hexadecimal el comando `d` el cual corresponde a un envío de datos, seguido de los 4 valores a enviar como las ganancias del controlador y el valor en rpm a modificar. En realidad lo que manda la Nextion es un mensaje hexadecimal de 17 bytes. Cada 4 Bytes corresponde a una variable, tener en cuenta la posición de inicio de cada byte con la posición de la variable es importante para la recepción de datos desde la Arduino.

En la pantalla 2 se configurará un timer el cual constantemente graficará el valor de la variable global rpm de la pantalla anterior en el canal 0 y tomará el valor de la variable global definida en esta página la cual corresponde al valor de rpm real del motor enviado por el Arduino. El plotter graficará este valor cada 100ms. Los comandos dentro del timer son los siguientes:

```
add 5,0,page0.rpm.val*255/10
add 5,1,rpm_real.val
```

Se debe recalcar que el valor rpm de la página 1 es un valor que puede variar entre 0 y 10, por esta razón, se hace la conversión de 10 a 255 cuando se grafica en la pantalla puesto que el rango del plotter por defecto grafica valores entre 0 y 255. Esto permite regular la velocidad del motor en 10 velocidades diferentes.

Una vez definido en la Nextion el procesamiento de datos, se procederá a explicar la configuración de recepción y envío de datos en el Arduino Mega. Para ello, se debe inicializar el puerto de transmisión y recepción por el que enviará y recibirá datos de la Nextion, en este caso se trabajó con TX1 y RX1 del Arduino, por lo que los pines de TX y RX de la pantalla Nextion se deben conectar al RX1 y TX1 del Arduino respectivamente. A continuación, en el setup, se debe inicializar el puerto serial a una velocidad de transmisión de 9600 baudios. Luego, constantemente se pregunta si se reciben o no datos en el bucle loop de Arduino. Esto mediante el condicional:

```
if (Serial1.available() > 0){
    String Received = Serial1.readString();
    if (Received[0] == 'b'){
        Kp = float(Received[1]);
        Ti = float(Received[5]);
        Td = float(Received[9]);
        rpm = int(Received[13]);
    }
}
```

En este condicional, se obtiene el valor que se recibe por el pin RX1 y se guarda dicho valor en el String Received para separar dicho mensaje en las variables enviadas por la Nextion. Esto se traduce en dividir el mensaje de 17 bytes, incluyendo el comando de envío de datos de la Nextion `d` (Que se utiliza para reconocer que se están recibiendo datos en el condicional). Una vez reconocida la llegada de datos, se separa el mensaje simplemente haciendo un slicing y una conversión de hexadecimal a flotante y se asignan a las variables definidas

previamente en el preámbulo del código principal. Para enviar el dato de la velocidad real constantemente a la Nextion para posteriormente graficarla, se debe preparar de igual forma un mensaje hexadecimal con los comandos con los que trabaja la pantalla Nextion. Esto es:

```
Serial1.print("page1.rpm_real.val=");
Serial1.print(int(rpm_real));
Serial1.write(0xff);
Serial1.write(0xff);
Serial1.write(0xff);
```

En estas líneas, se establece el valor de la variable global rpm_real definida en la pagina 2 de gráfica con el comando .val. Siempre al enviar datos se debe seguir una cadena de FF FF FF que es lo que se envía al final del comando.

IV. Análisis de Resultados

A continuación, se muestran en las Figuras 12, 13 y 14, la implementación del proyecto entero, con la mayoría de las especificaciones completadas. Como se implementa el controlador de motor PID a la planta de entrenamiento mediante la placa de Arduino MEGA, además de la interfaz HMI,en la cual se puede observar el monitoreo la velocidad del motor controlada (gráfica amarilla) y como cambia en tiempo real la misma la recibir perturbaciones y estabilizarse por si misma. Así mismo, como fue sintonizado el PID discreto con buenos valores, pero al no variar correctamente entre el proporcional y el integral, entonces se observa como el sistema no es completamente estable pero siempre busca alinearse a la referencia. Por lo que se puede decir que le logró el objetivo del proyecto en lograr controlar mediante PID y su correcta sintonización, la velocidad de un motor. Cabe resaltar, que todos los aspectos involucrados para el sistema de control fueron abordados y especificados de manera detallada en las secciones anteriores

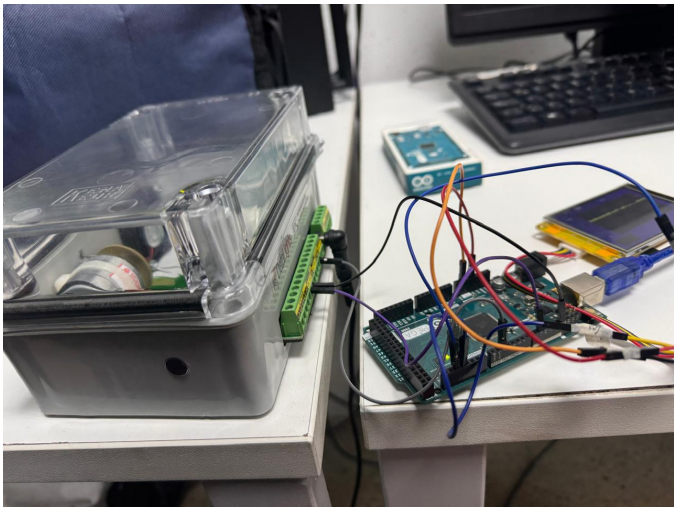

Fig. 12. conexión de entre la planta de entrenamiento, Arduino y la interfaz HMI (Nextion)

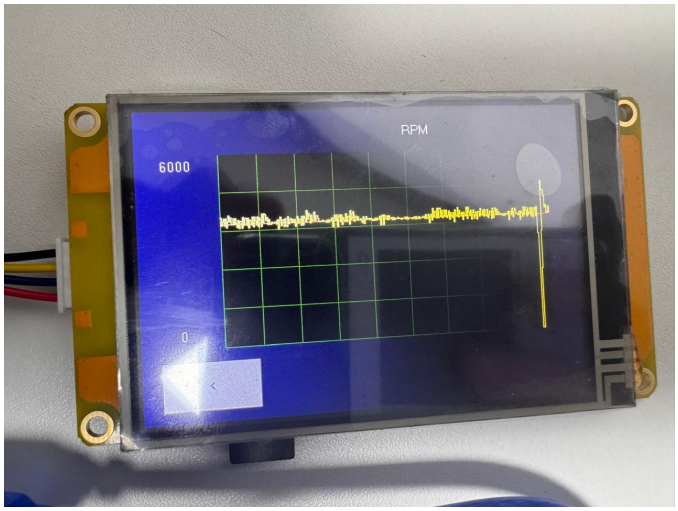

Fig. 13. Monitorio de señales de velocidad de motor (rmps) controlada por el PID, a través de la pantalla Nextion)

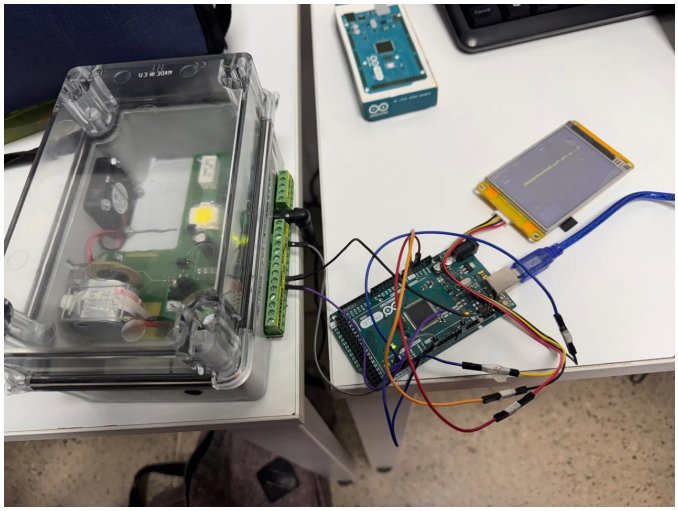

Fig. 14. Plano general de la interconexión entre el motor de la plata de entrenamiento, Arduino e interfaz HMI

V. Conclusiones

La sintonización del PID puede llegar a tener variaciones si se emplea el método de ganancia máxima. Esto, debido a que las diversas pruebas arrojaron que existe un valor mínimo que provoca la oscilación de sistema, más no un valor único. En este sentido, distintos valores podrían arrojar nuevos coeficientes para k_d,k_p y k_i. Además, en la red aparecen múltiples forma de, dada un valor de ganacia crítica y periódo crítico, calcular los demás coeficientes. A la final, luego de cualquier calculo, siempre ser requerirá de un ajuste fino.

La programación de la pantalla Nextion debe ser cuidadosa en lo referente al envío de información hacia arduino. Las página dentro de su programación envían información a base de *timers*; si los *timers* se ajustan para enviar información

cuando la página es visible (o ha sido seleccionada) el flujo de datos será finito e interrumpido. Esto generó problemas a la hora de recibir un valor de referencia a partir de la Nextion, pues la referencia se enviaba únicamente por un periódo corto de tiempo. Para volver a recibir información, era necesario volver a cargar la página.

El empleo de la pantalla Nextion tiene un costo grande relacionado a la resolución. Los datos que provienen de del motor estan en un rango de 0 a 4400, en cambio la Nextion los graficará en un rango de 0 a 255, esto es menos de una decima parte de los valores. Justamente esta pérdida de resolución provoca que las oscilaciones sean proco apreciables en comparación con la gráfica de la consola de Arduino.

Una limitante en la pantalla Nextion, reside en su incapacidad de enviar valores decimales. En este proyecto, los valores decimales son fundamentales, pues correspoden a los coeficientes del controlador PID; coeficientes que siempre están en valores decimales, ya que de otra forma generarían inestabilidad. En este sentido se debería proponer una solución que involucre el empleo de decimales únicamente en Arduino, de tal manera que la pantalla envía especificaciones del número decimal a partir de enteros (*e.g.*, lo que se realiza en la notación cientifica y el exponente del número 10).

Este proyecto se tenía pensado implementarlo en Controllino, un PLC de nivel industrial basado en la tecnología de software de código abierto Arduino, pero, durante el proceso ocurrieron varios inconvenientes.

Primeramente, se planteaba comunicar la salida digital del Controllino (esta sería la señal PWM) con la entrada del Motor DC, pero, las salidas digitales entregan hasta *12 [V]* y la entrada del EPC que se utilizaba, aceptaba máximo *5 [V]*. Por lo que, no era posible ingresar el número máximo para el PWM porque esto implicaría quemar el sensor del EPC, por el exceso de voltaje permitido.

Al momento de recolectar los pulsos que envíaba el motor, ocurrió otro contratiempo. Para que dichos pulsos sean procesados por el Controllino, se necesita que esa señal ingrese por una entrada PWM; pero el Controllino solo cuenta con entradas analógicas y salidas digitales. Para solucionar este problema, se debió ingresar esta señal por el pin *Interrupt 0* que se encuentra únicamente por el bus de datos *X1*.

Otro inconveniente ocurrió en ese elemento. Para el intercambio de información entre Controllino y Nextion, se necesita el pin *Tx*, *Rx*, *5V* y *GND* del bus de datos. Estos pines se encuentran en los dos buses, *X1* y *X2*. Pero, ninguno funcionaba.

Estas problemáticas ocasionaron que el proyecto sea implementado en Arduino Mega.

No se niega la eficiencia y potencia de Controllino, pero, debido a los otros elementos que no se complementaban entre si y con nuestro proyecto, se decidió cambiar al Arduino Mega.

REFERENCES

[1] Haber, A. (2019). PID Controller Discretization and Implementation in Arduino. https://aleksandarhaber.com/pid-controller-discretization-and-implementation-in-arduino/

[2] Microcontrollers Lab. (2019). PID Controller Implementation Using Arduino. https://microcontrollerslab.com/pid-controller-implementation-using-arduino/

VI. Anexos

A. *Enlace al código de Arduino en GitHub*

https://github.com/thyron001/Controlador-PID