Trabajo Práctico N3

Motor DC y Controlador PI

Estudiante:
Scala, Tobias 55391

Profesores:
Perfumo, Lucas Alberto
Fortunatti, Nelson Ariel

Materia:
Mecatrónica Aplicada

Facultad:
Instituto Tecnológico de Buenos Aires, ITBA

Introducción

El control de un motor es muy importante en la industria para que cierta aplicación complete una determinada tarea de forma satisfactoria. Algunas aplicaciones requieren una determinada velocidad constante, otras de mucha potencia requeren un arranque suave a fin de disminuir la corriente. Cada aplicación requiere que el motor sea controlado de tal forma que cumpla ciertos requisitos y sincronización.

En el presente trabajo se diseñará un controlador discreto para un motor DC del fabricante "Pololu" para una aplicación que requiera un arranque suave y velocidad constante (Figura 1). A pesar de que el motor provienen de dicho fabricante, su modelo es totalmente desconocido. Por lo que, antes de diseñar el controlador, es necesario realizar una identificación del sistema.

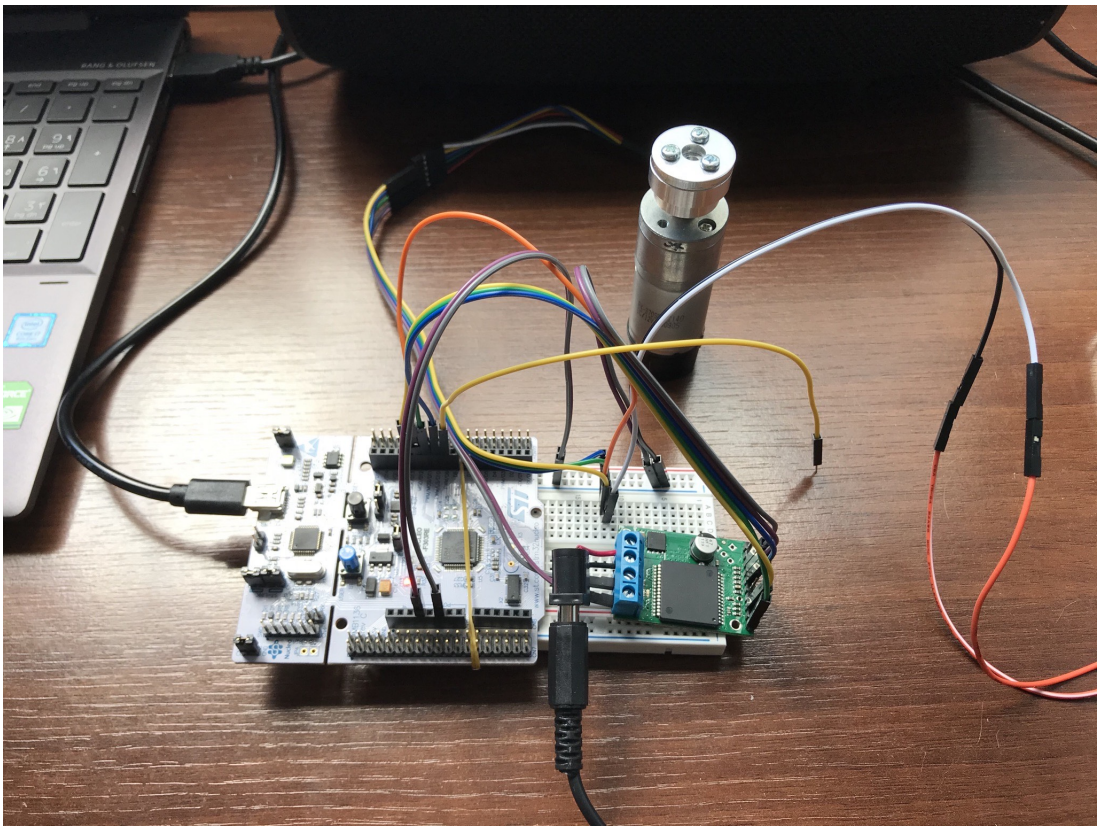

Figure 1: Conexión del Motor DC al MCU.

1 Identificación de Sistemas

El motor DC que se tiene a disposición (cuyo fabricante es "Pololu") es desconocido. Es decir, no se conocen parametros como:

- Máxima velocidad RPM.
- Máxima corriente admisible.
- Relación de la caja reductora.

Los únicos parámetros conocidos son:

- Tensión máxima de alimentación (12 V).

- Ciclos por revolución (48 CPR)
- Relación de la caja reductora.

1.1 Relación de la Caja Reductora

El proceso de identificación de sistemas inicia midiendo la salida del encoder del motor (usando un osciloscopio) mientras el motor es alimentado con una señal PWM de 50% de duty cycle ($V_{rms} = 6$) (Figura 2).

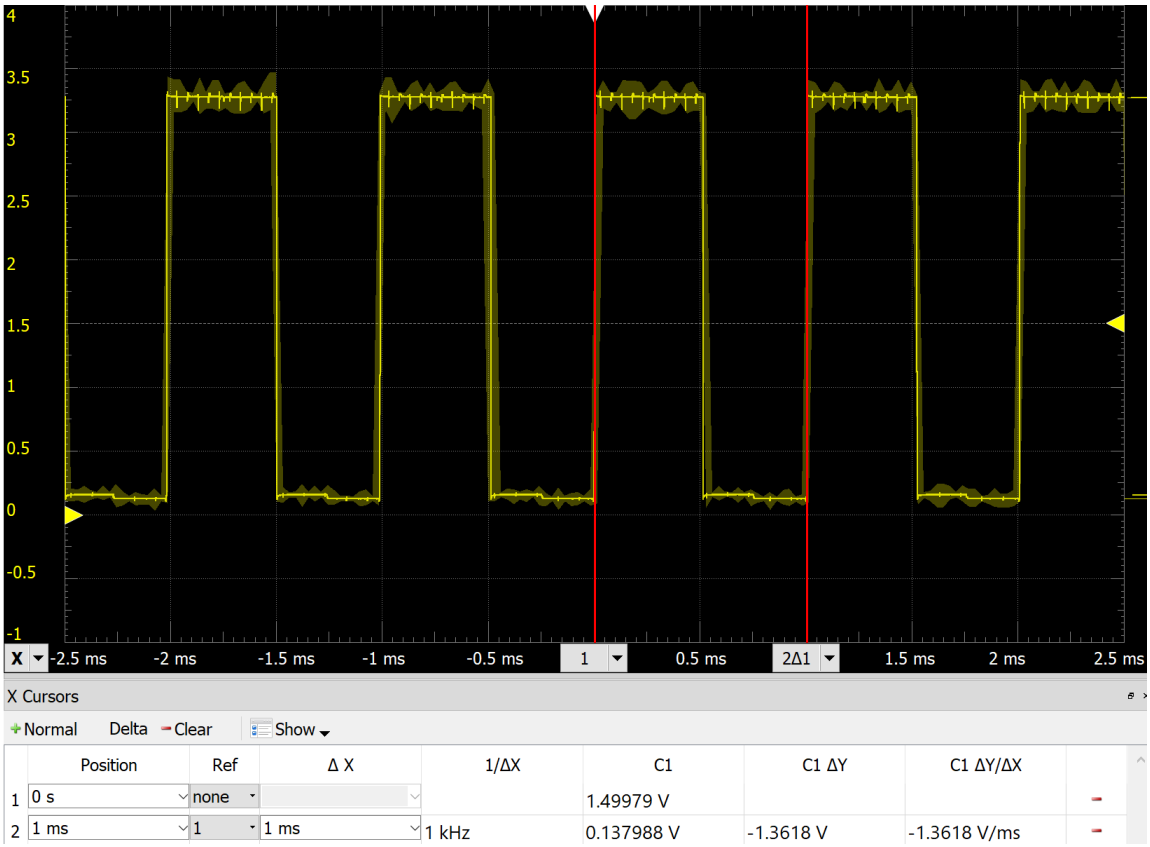

Figure 2: Señal de Salida del Encoder del Motor DC.

La señal de salida del encoder tiene un periodo de 1 ms. Ya que el motor tiene 48 CPR, entonces se puede calcular la velocidad del motor en RPM.

$$n_{RPM} = \frac{60}{48 \cdot 0.001} = 1250$$

Pero, al medir el tiempo en el cual la rueda (a la salida del motor) daba una vuela, se obtuvo un valor aproximado de 0.4 segundos (150 RPM). Esto es debido a la caja reductora del motor. Con los valores obtenidos es posible calcular la relación de la caja reductora.

$$n_i = \frac{150}{1250} \cdot n_o = \frac{1}{8.333} \cdot n_o \rightarrow 8.333 : 1$$

Lamentablemente no se ha encontrado un modelo existente de motores "Pololu" con esta realición (Catálogo Pololu). La relación más cercana a la obtenida es 9.7:1. Como no ha sido posible encontrar el modelo del motor para conocer sus parámetros, se prosiguió con la identificación del sistema con los datos experiementales.

1.2 Medición del Sistema a Lazo Abierto

Para poder obterner el modelo matemático del motor a lazo abierto, se optó por medir la respuesta al escalón del mismo.
Para poder medir la respuesta al escalón, se midieron los valores del encoder con un tiempo de sample de 3 ms. Para poder asegurarse que el tiempo de sample sea el mencionado, se usó un test pin el cual realiza un cambio de estado (toggle) cada vez que se vuelve a medir el próximo valor (Figura 3).

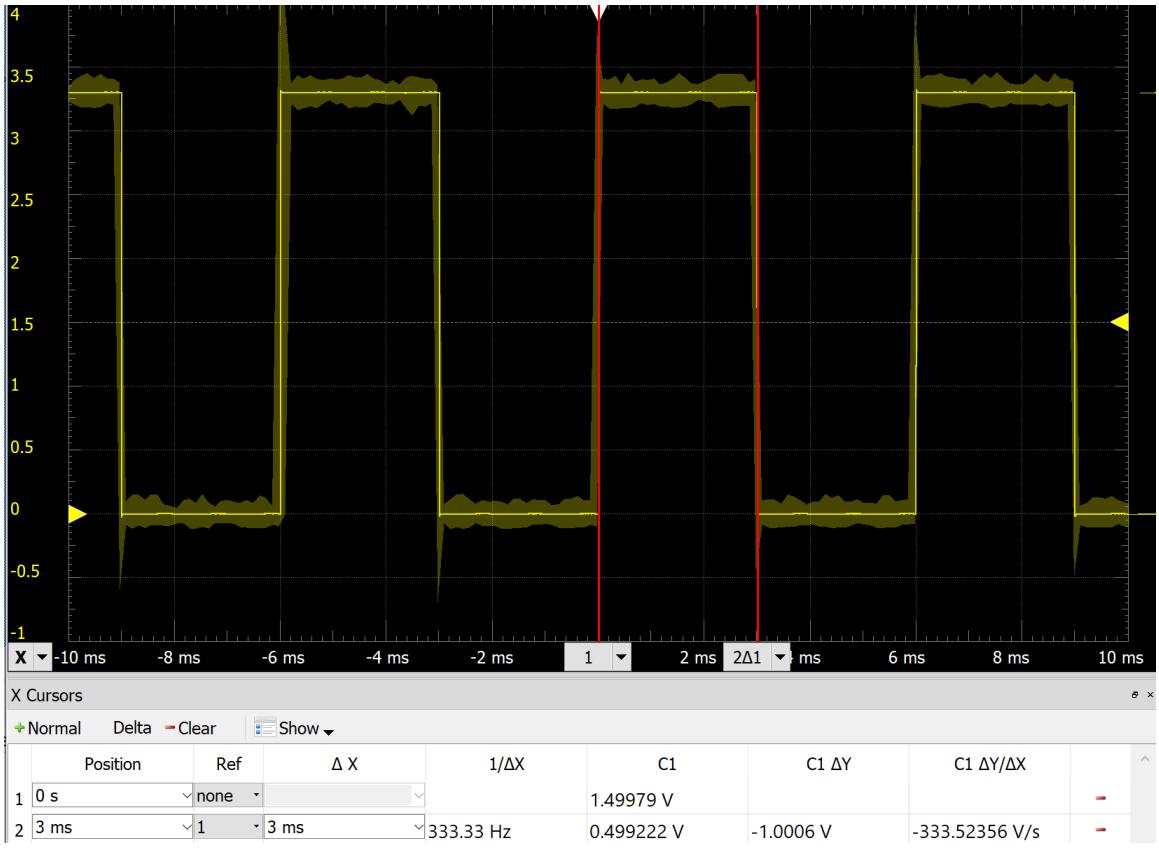

Figure 3: Señal del test pin. Para asegurarse que el tiempo de sample sea exacto.

Los valores medidos del encoder se transmiten a la PC y mediante un programa en Python se grafican los valores obtenidos (Apéndice A) (Figura 4).

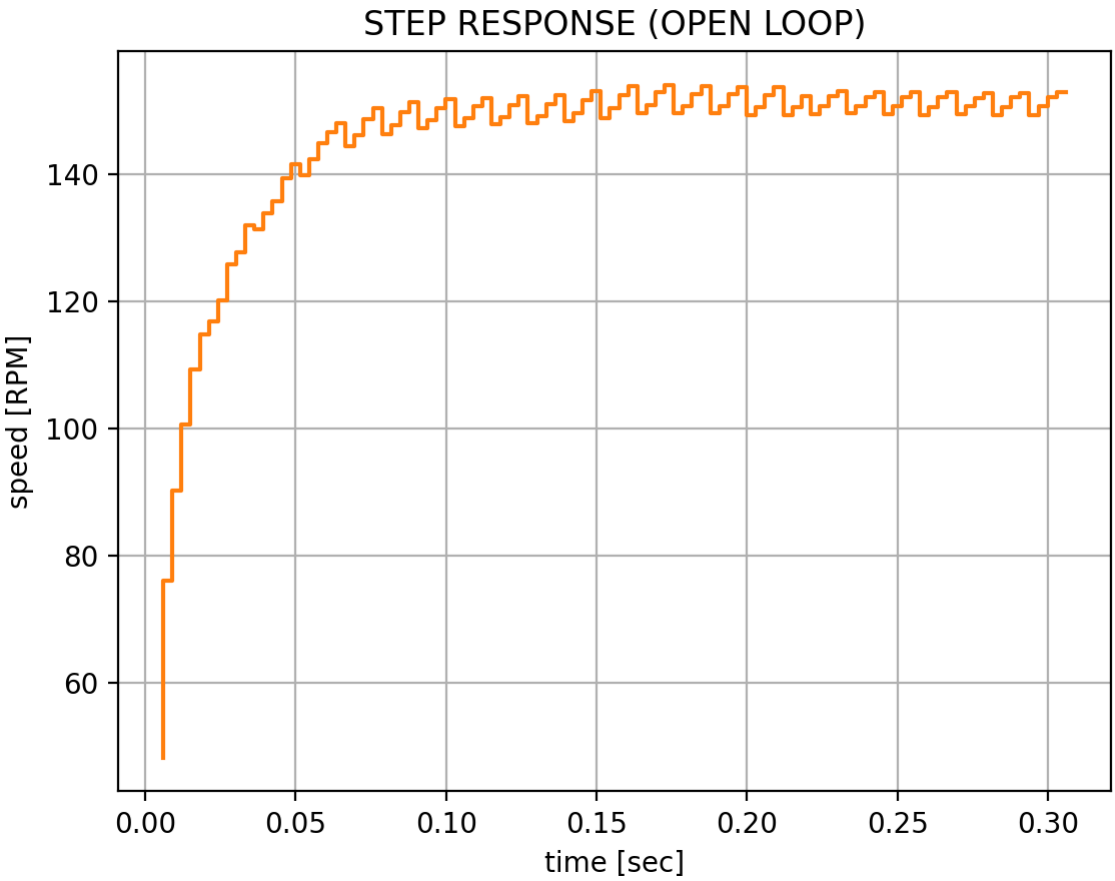

Figure 4: Respuesta al escalón del sistema a lazo abierto (medido: naranja)

Se puede ver a simple vista que, a pesar de que los motores DC son de segundo orden, la respuesta al escalón es característico de un sistema de primer orden. Teniendo dicha respuesta se hicieron los cálculos correspondientes para obtener la transferencia del motor (Identificación de Sistemas).

$$P(s) = \frac{25}{0.019 \cdot s + 1}$$

La ganancia de 25 es debido a que cuando la acción de control es una señal PWM de 50% de duty cycle (equivalente a 6V en valor eficaz), entonces la salida en RPM es $6 \cdot 25 = 150$. A continuación se muestra el diagrama de Bode del sistema (Figura 5).

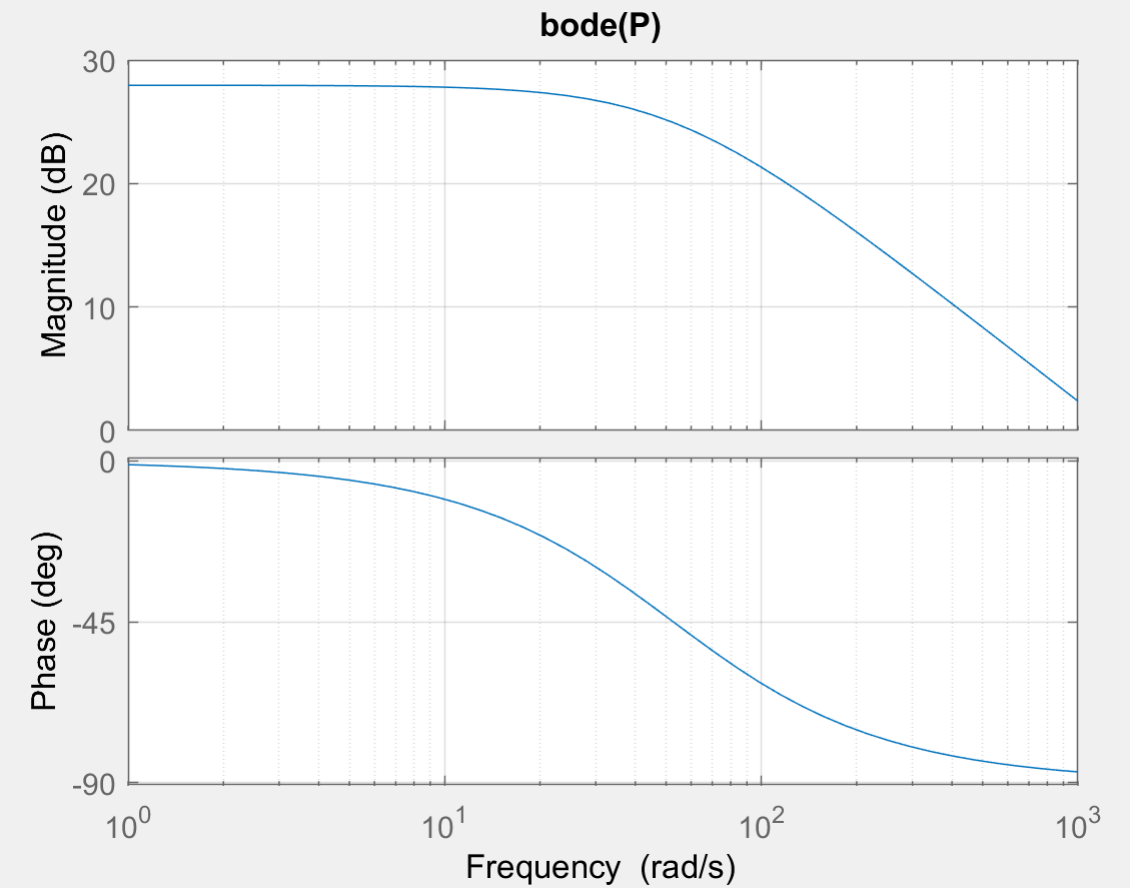

Figure 5: Diagrama de Bode del modelo identificado del motor.

A continuación se comparan las respuesta al escalón de los sistemas real (medido) y modelado (Figuras 6 y 7).

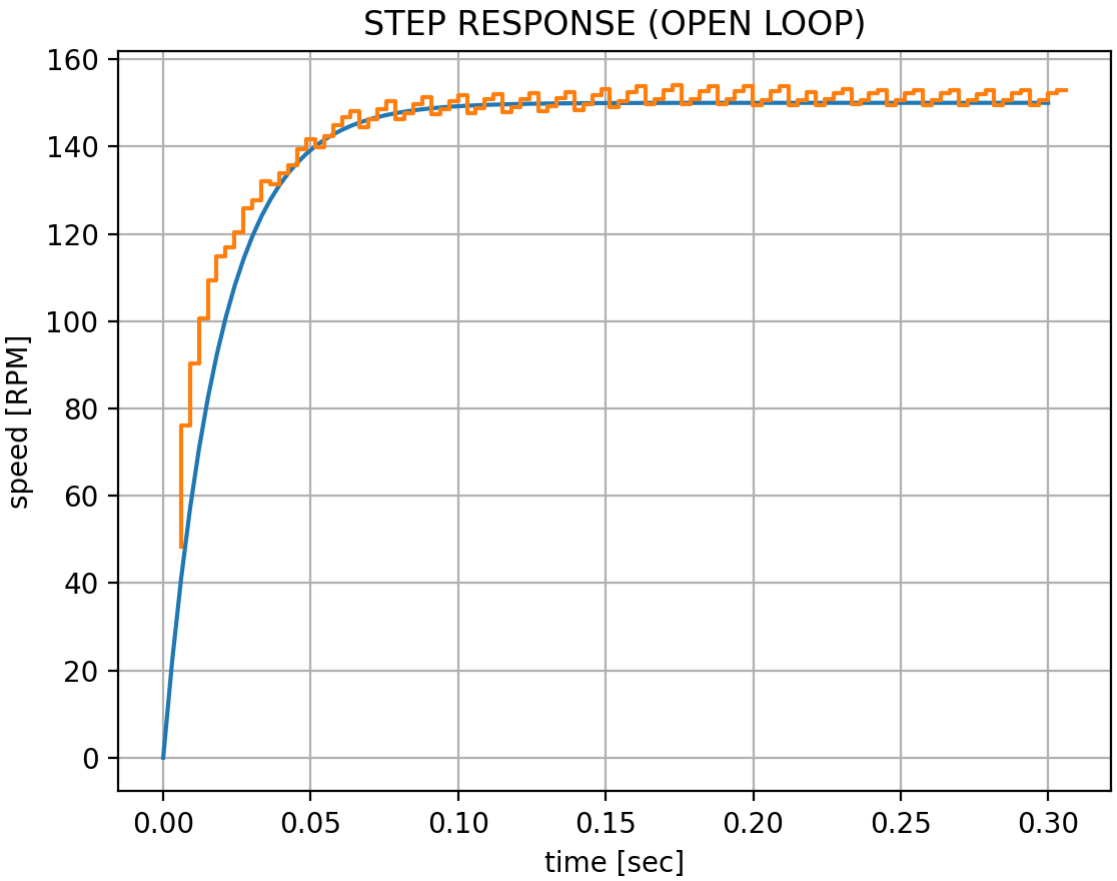

Figure 6: Comparación de las respuesta al escalón de los sistemas real (medido: naranja) y modelado (en tiempo continuo: azul)

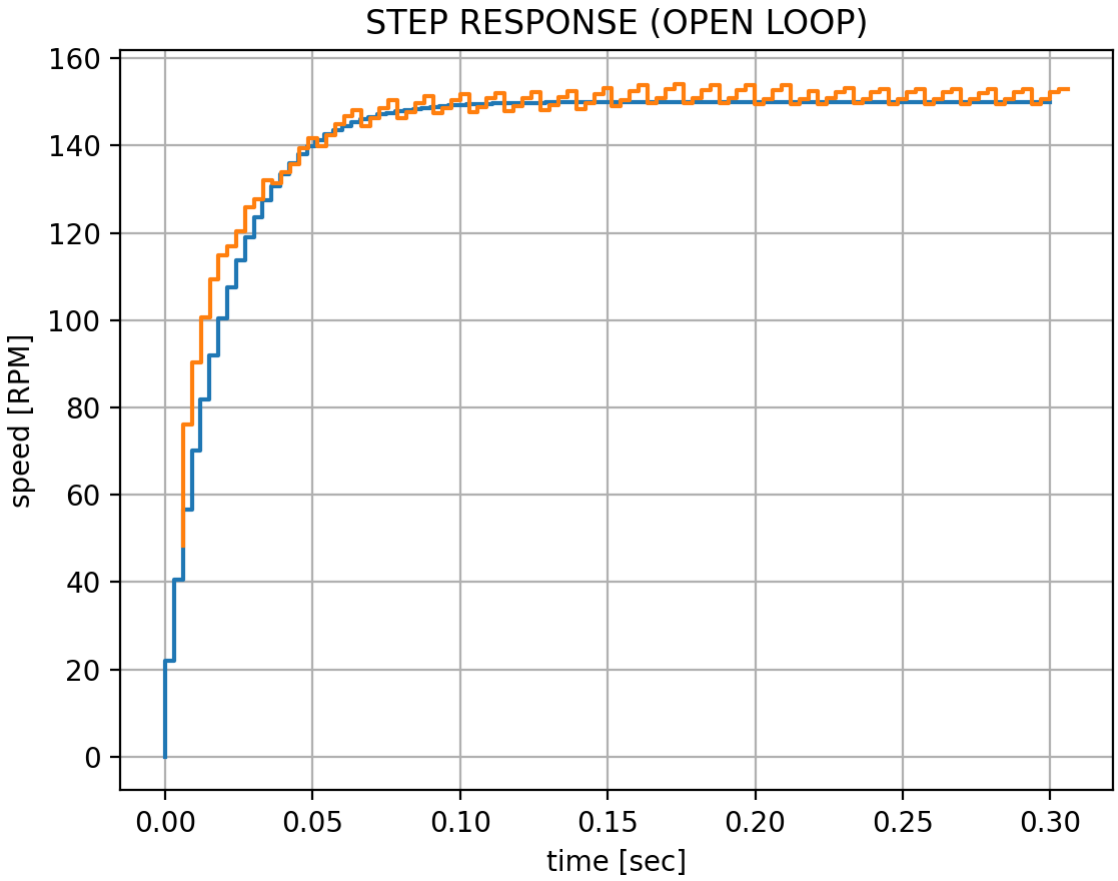

Figure 7: Comparación de las respuesta al escalón de los sistemas real (medido: naranja) y modelado (en tiempo discreto: azul)

Se puede ver que el modelo matemático del motor es muy similar al sistema real. Por lo tanto se concluye la identificación del sistema del motor y ahora ya es posible diseñar el controlador.

2 Controlador PI

Como bien se mencionó, se quiere diseñar un controlador pensado para una aplicación que requiere un arranque suave y mantener la velocidad. Además se deben cumplir los siguientes requisitos:

- Tiempo de establecimiento < 1.5 segundos.
- Error de estado estacionario $< 1\%$.
- Overshoot $< 7\%$.

Realizando el diseño del control en tiempo discreto (tiempo de sample 30 ms) en Matlab (Apéndice A), se diseño un controlador PI (loop shaping) capaz de rechazar perturbaciones de entrada de tipo escalón y mantener la velocidad.

$$C(s) = \frac{0.19 \cdot (0.1s + 1)}{s} \rightarrow Kp = 0.019 \quad Ki = 0.19$$

La respuesta al escalón del sistema a lazo cerrado se muestra a continuación (Figura 8).

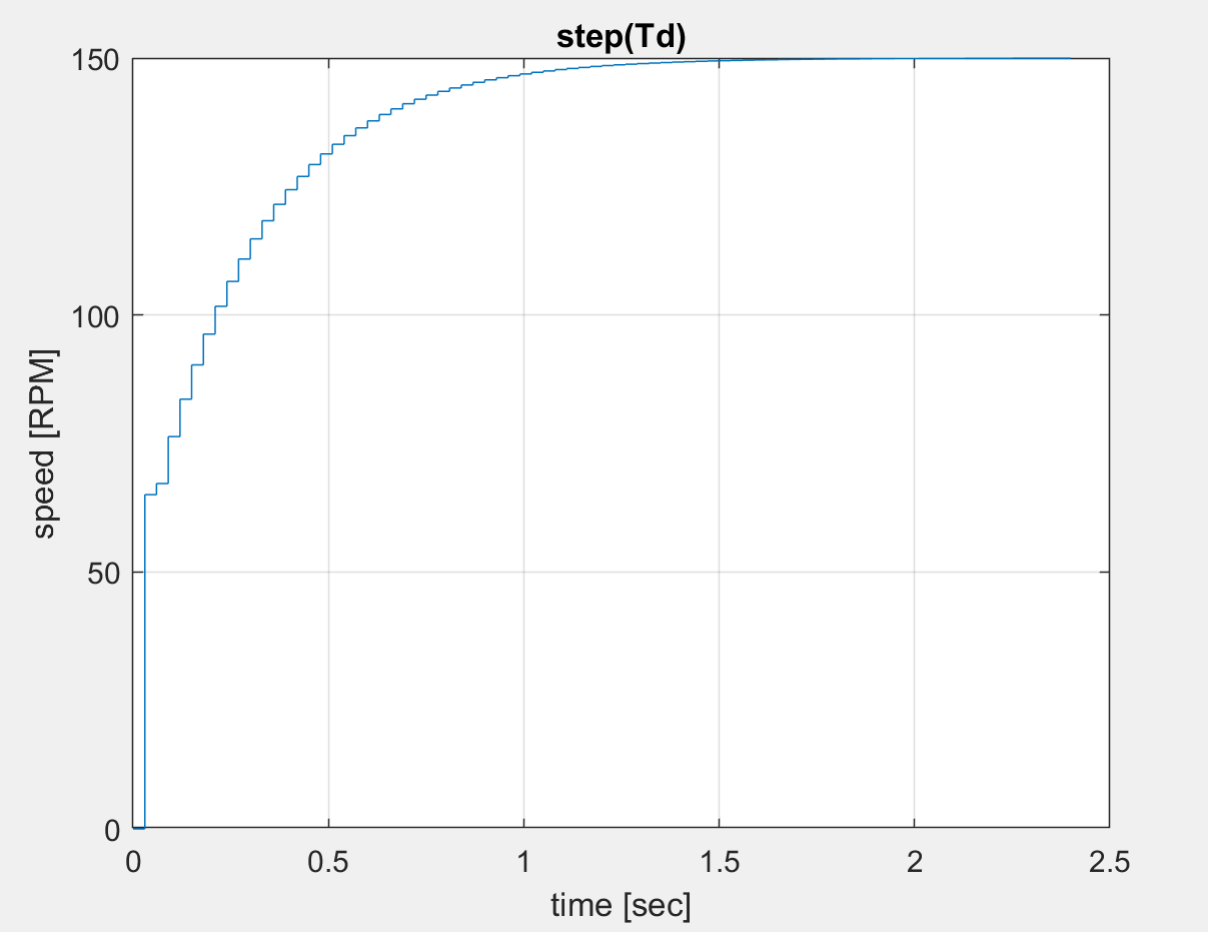

Figure 8: Respuesta al escalón del sistema discreto a lazo cerrado.

Para verificar que el controlador rechaza perturbaciones de entrada de tipo escalón, se optó por simular el sistema a lazo cerrado utilizando Simulink (Figuras 9 y 10).

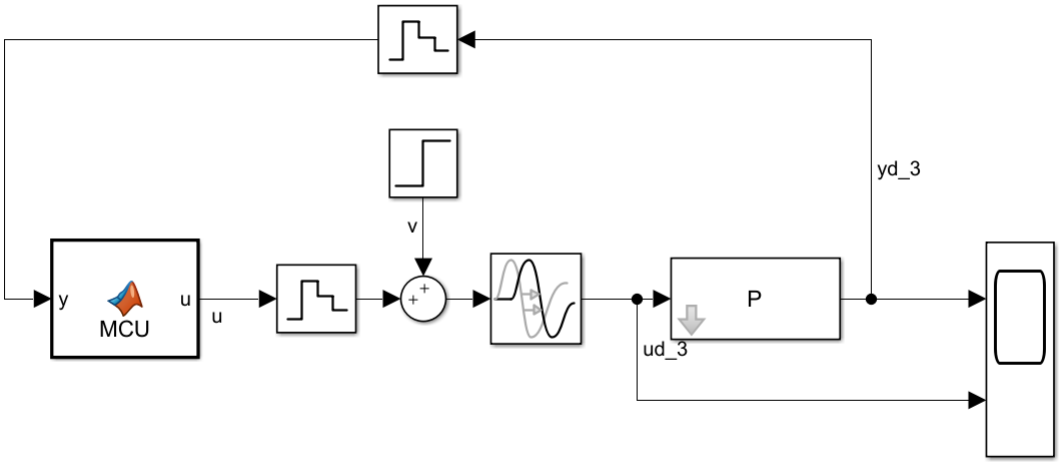

Figure 9: Circuito del sistema a lazo cerrado simulado en Simulink.

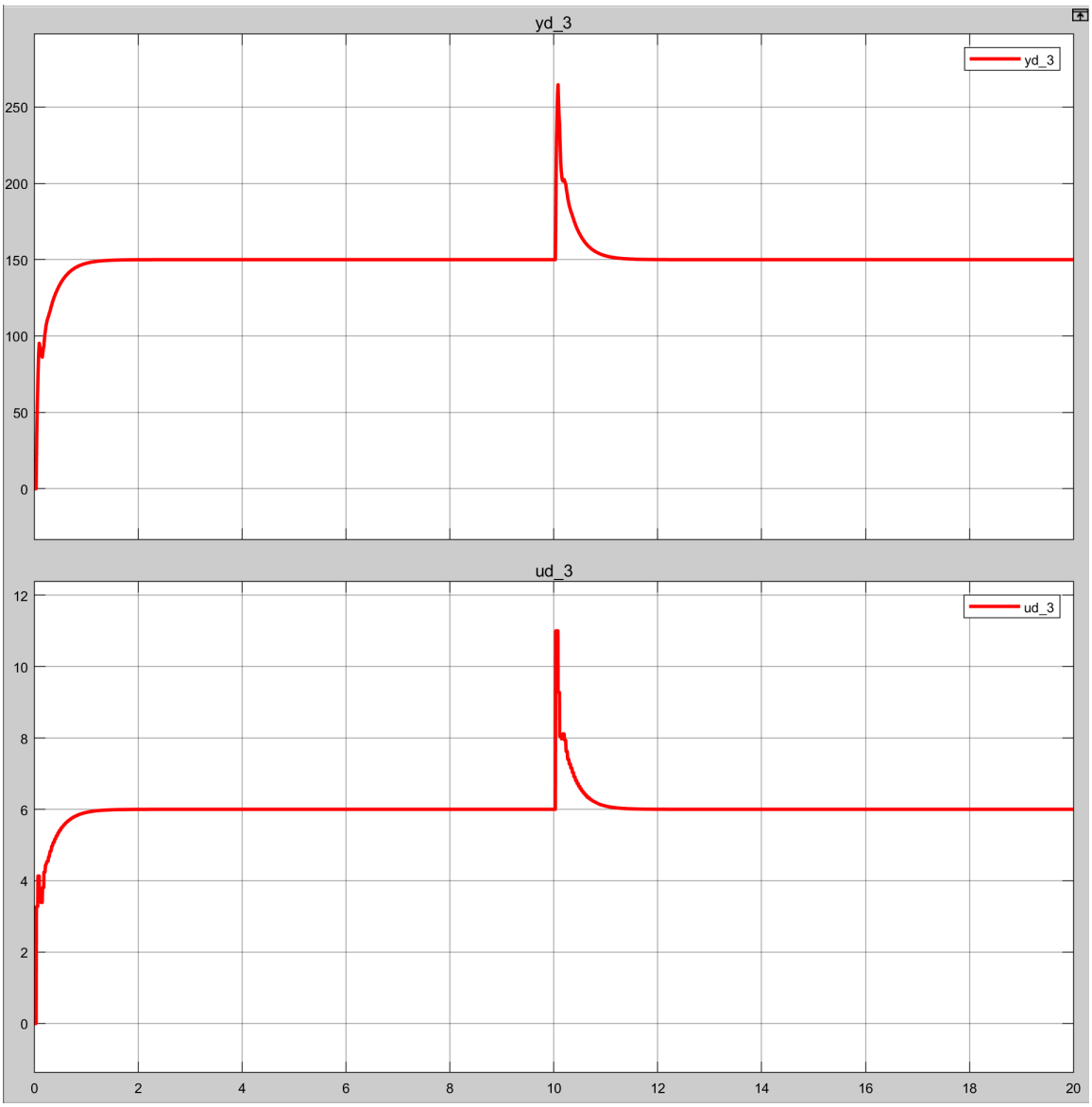

Figure 10: Respuesta frente a una perturbación de entrada de tipo escalón.

Como se puede ver, los resultados son satisfactorios y lo que resta es implementar el controlador en el MCU y comprobar el comportamiento real del motor.

3 Motor Controlado

Tras implementar el controlador en el MCU, se decidió medir la respuesta al escalón del sistema real a lazo cerrado. El controlador en tiempo discreto tiene la siguiente expresión en forma diferencial.

$$u(n+1) = u(n) + 0.02185 \cdot e(n+1) - 0.01615 \cdot e(n)$$

Puesto que el controlador se diseñó con un tiempo de sample de 30 ms, nuevamente se verifica con el test pin que el controlador funciona con dicho tiempo de sample (Figura 11).

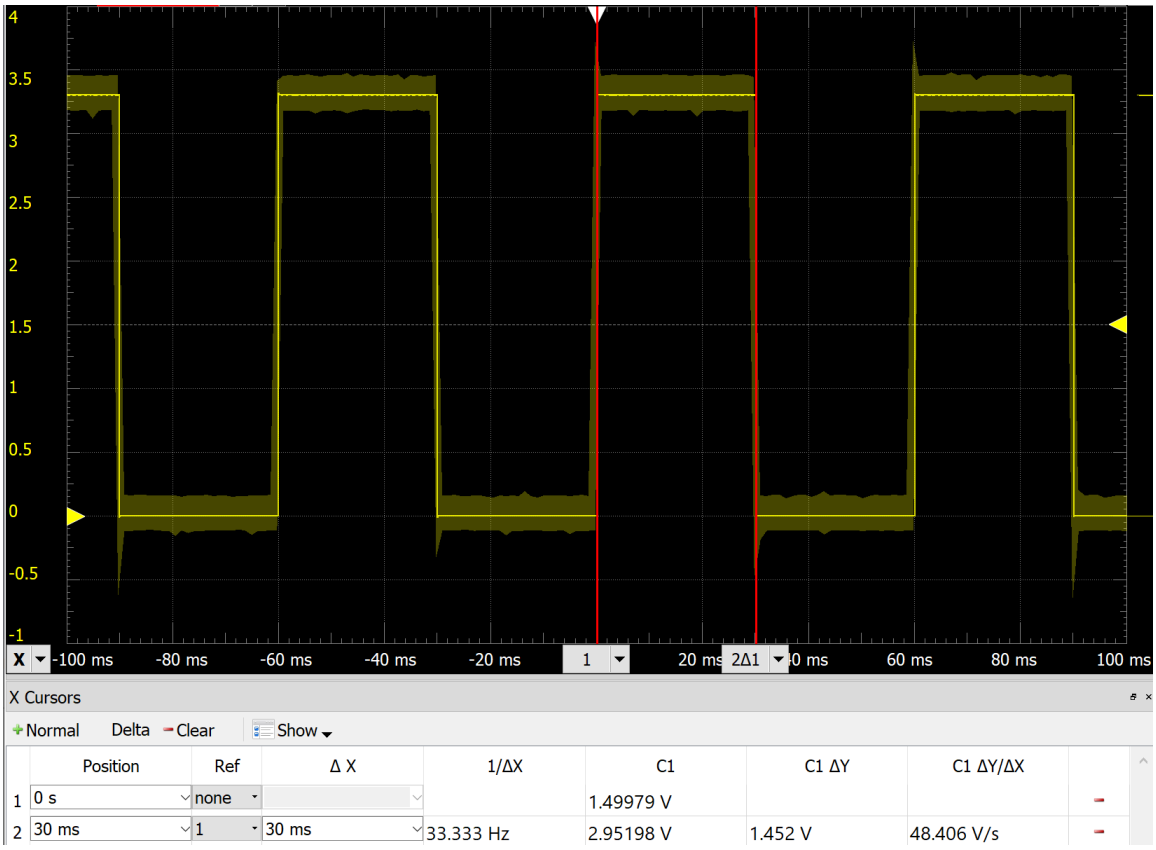

Figure 11: Señal del test pin. Para asegurarse que el tiempo de sample sea exacto.

A continuación se comparan las respuestas al escalón de los sistemas real (medido) y modelado a lazo cerrado (Figura 12).

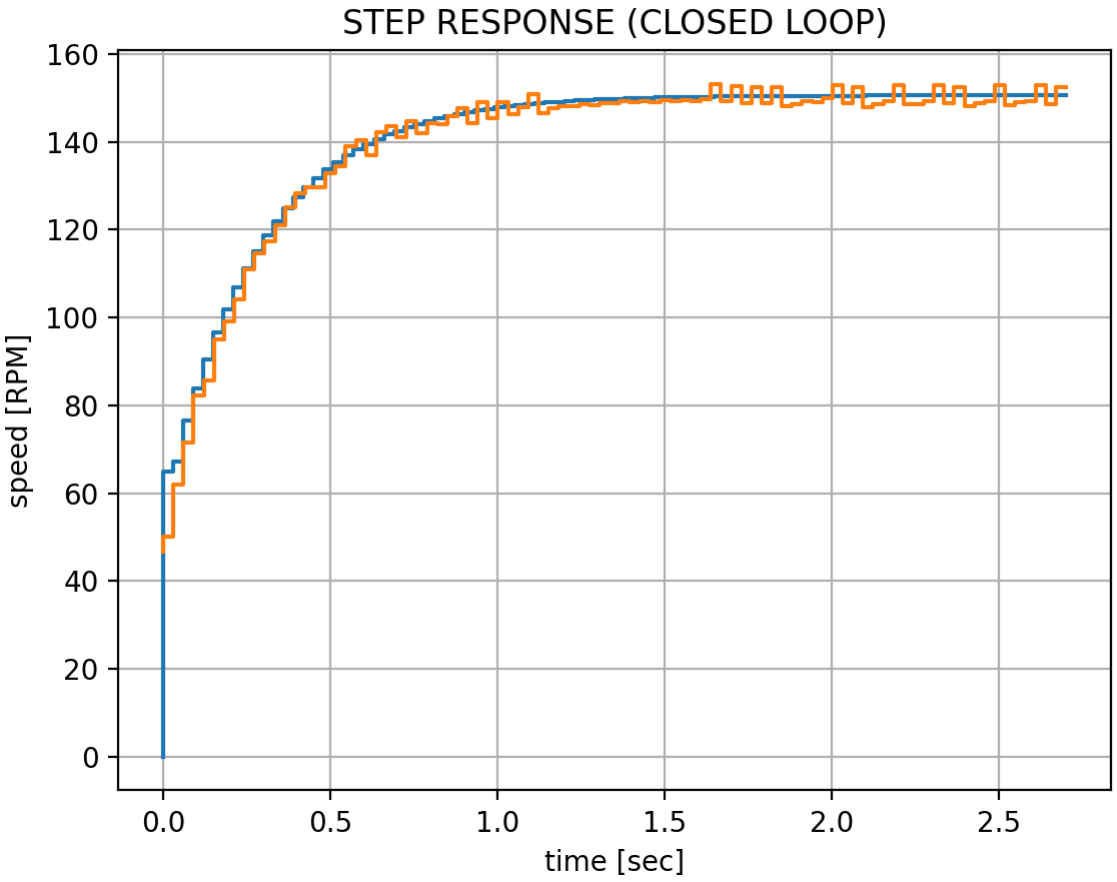

Figure 12: Comparación de las respuestas al escalón de los sistemas real (medido: naranja) y modelado (en tiempo discreto: azul) a lazo cerrado.

Con este resultado se culmina el trabajo pudiendo observar que el controlador implementado en el sistema real cumple perfectamente con los requisitos y es fiel al controlador diseñado a través del modelo indentificado.
Por último cabe mencionar que, mediante experiencia, se observó que el motor no tiene la suficiente torsión para girar los engranajes de la caja reductora cuando el mismo es alimentado a una tensión eficaz de 2V, lo que es equivalente a una señal PWM de duty cycle de 16.65%. Con esta tensión, el mortor gira a 50 RPM. Por lo tanto, se implementó un anti windup cuyo límite inferior es 2V y límite superior es 12V.

4 Controlador Alternativo

En caso que la aplicación no requiera un arranque suave pero si mantener la velocidad constante, una acción integral como controlador es una buena alternativa.

$$C(s) = \frac{0.35}{s}$$

A continuación se comparan las respuestas al escalón del sistema a lazo cerrado con el controlador alternativo y con el controlador anterior (Figura 13).

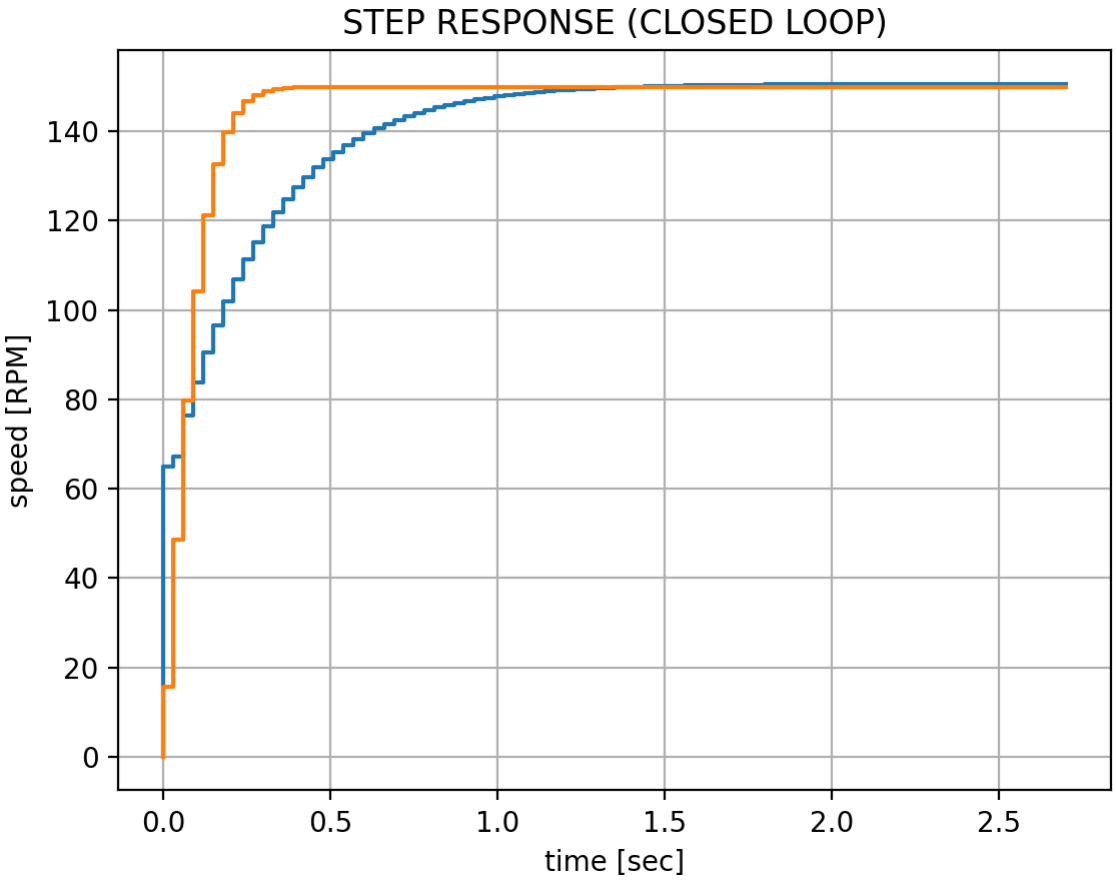

Figure 13: Comparación de las respuestas al escalón de los sistemas a lazo cerrado con el controlador alternativo (naranja) y con el controlador anterior (azul) en tiempo discreto (tiempo de sample 30 ms).

Se puede ver que el nuevo controlador es más rápido y por ende tendrá una reacción más rápida frente a perturbaciones de entrada de tipo escalón.

Apéndice A

Para este trabajo se requirió programar en Python (para obtener las muestras por comunicación serial), en Matlab (para diseñar el controlador) y en C (para el MCU). A continuación se muestran cada uno de los códigos utilizados.

Códigos en Python

El primer código utilizado en python ha sido para Recibir datos muestreados del motor a lazo abierto para poder obtener su respuesta al escalón y así realizar la identificación del sistema.

```
Ts = 0.003 #sample period.
N = 100 #amount of samples.
data = []
t = np.linspace(0,N*Ts,N)

s = serial.Serial(port='COM3', baudrate=115200)

f = open("DCmotor.txt", "w")
for i in range(N):
```

```
10      data.append(struct.unpack('<f', s.read(4))[0])
11      f.write(str(data[i]))
12      f.write(',')
13  f.close()
14
15  plt.step(t, data)
16  plt.grid()
17  plt.show()
```

El segundo código utilizado en python ha sido para realizar la identificación del sistema obteniendo el modelo matemático del motor. Se comparan la respuesta al escalón (a lazo abierto) medida y con el modelo matemático tanto en tiempo continuo como discreto.

```
1  data = [...]
2  Ts = 0.003 #sample period.
3  N = 100 #amount of samples.
4  tdata = np.linspace(2*Ts,(N+2)*Ts,N)
5  t = np.linspace(0*Ts,(N+0)*Ts,N)
6
7  P = signal.TransferFunction([25],[0.019,1]) #continuous time plant.
8  t, y = signal.step(P, T = t)
9  plt.plot(t, 6*y)
10
11 plt.title('STEP RESPONSE (OPEN LOOP)')
12 plt.ylabel('speed [RPM]')
13 plt.xlabel('time [sec]')
14
15 plt.step(tdata, data)
16 plt.grid()
17 plt.show()
18
19 Pd = signal.TransferFunction([3.6515],[1,-0.8539],dt=Ts) #discrete time plant.
20 t, y = signal.dstep(Pd, t = t)
21 plt.step(t, 6*np.squeeze(y))
22
23 plt.title('STEP RESPONSE (OPEN LOOP)')
24 plt.ylabel('speed [RPM]')
25 plt.xlabel('time [sec]')
26
27 plt.step(tdata, data)
28 plt.grid()
29 plt.show()
```

El tercer código utilizado en python ha sido para muestrear el motor a lazo cerrado para poder obtener su respuesta al escalón y compararlo con el que se obtuvo en el diseño del control.

```
1  data = [...]
2  Ts = 0.03 #sample period.
3  N = 90 #amount of samples.
4  tdata = np.linspace(0*Ts,(N+0)*Ts,N)
5
6  Td = signal.TransferFunction([0.43362,-0.32],[1,-0.7725,-0.11434],dt=Ts) #discrete time
       system.
7  t, y = signal.dstep(Td, t = tdata)
8  plt.step(t, 150*np.squeeze(y))
9
10 plt.title('STEP RESPONSE (CLOSED LOOP)')
11 plt.ylabel('speed [RPM]')
12 plt.xlabel('time [sec]')
13
14 plt.step(tdata, data)
15 plt.grid()
16 plt.show()
```

El cuarto y último código utilizado en python ha sido para graficar en tiempo real la velocidad del motor y así poder observar el funcionamiento del controlador diseñado para mantener cierta velocidad.

```
1  fig, axis = plt.subplots()
2  fig.suptitle('DC MOTOR', fontsize=12)
```

```
axis.set_ylabel('speed [RPM]')
axis.set_xlabel('time [sec]')
axis.set_ylim(45, 305)
axis.set_xlim(0, 3)

t0 = 0
N = 100
Ts = 0.03
data = np.zeros(N)
t = np.linspace(0*Ts,(N+0)*Ts,N)
plt.grid()
splt, = plt.step([], [])

s = serial.Serial(port='COM3', baudrate=115200)

def update(frame):
    global data,t0

    data = np.append(data, struct.unpack('<f', s.read(4))[0])
    data = data[1:]
    splt.set_data(t, data)

    return splt,

ani = FuncAnimation(fig, update, blit=True, interval=1)
plt.show()
```

Códigos en Matlab

El primer código utilizado en Matlab para poder diseñar un controlador para controlar la velocidad del motor. Dicho controlador ha sido un controlador PI.

```
clear; %clear the workspace.
clc;
close all;

s = tf('s'); %laplace variable.
P = zpk(25/(0.019*s+1));
bode(P);
title('bode(P)');
grid on;

Ts = 0.03; %period sample.
Pd = c2d(P,Ts,'zoh') %plant discretization.
figure;
[y,t] = step(Pd);
stairs(t,y);
title('step(Pd)')
grid on;

k = 0.19; %0.35; %0.19;
C = k*(s/10+1)/s; %PI.
Cd = c2d(C,Ts,'tustin') %control discretization.
Td = feedback(Cd*Pd,1)
figure;
[y,t] = step(Td);
stairs(t,150*y);
title('step(Td)');
xlabel('time [sec]');
ylabel('speed [RPM]');
```

```
29 grid on;
```

El segundo código utilizado en Matlab Simulink para poder simular la dinámica del sistema y verificar que el sontrolador rechace perturvaciones de entrada de tipo escalón.

```
1  function u = MCU(y)
2
3  r=150; %reference speed (RPM)
4  persistent e_; %static float e = 0; (in C).
5  persistent u_; %static float u = 0; (in C).
6  if (isempty(e_) && isempty(u_))
7      e_ = 0;
8      u_ = 0;
9  end
10
11 e_1=r-y;
12 u_1=u_+0.02185*e_1-0.01615*e_;
13
14 u_=u_1;
15 e_=e_1;
16
17 u=u_1;
```

Códigos en C

El primer código utilizado en C ha sido para muestrear el motor a lazo abierto para poder obtener su respuesta al escalón. Los datos muestreados son transmitidos a la PC con el protocolo UART.

```
1  float motorRPM = 0; //motor speed (RPM).
2  bool newRPM = false;
3  bool measure = false;
4
5  int main(void)
6  {
7    /* USER CODE BEGIN 1 */
8    uint8_t Ts = 3; //the sample rate (milliseconds) -> SYSTEM IDENTIFICATION
9    uint32_t t0, t1; //variables to measure time.
10   uint8_t *data2send;
11   /* USER CODE END 1 */
12
13   /* Initialize all configured peripherals */
14   MX_GPIO_Init();
15   MX_USART2_UART_Init();
16   MX_TIM2_Init();
17   MX_TIM3_Init();
18   /* USER CODE BEGIN 2 */
19   HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
20   HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);
21   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, false);
22   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, true);
23   /* USER CODE END 2 */
24
25   while (1)
26   {
27     HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_7); //test pin, to measure sample time.
28     t0 = HAL_GetTick(); //get current time in milliseconds.
29     measure = true;
30     while(!newRPM) {
31
32     }
33     measure = false;
34     data2send = (uint8_t*) &motorRPM;
35     HAL_UART_Transmit(&huart2, data2send, 4, HAL_MAX_DELAY); //1float = 4bytes.
```

```
    newRPM = false;
    t1 = HAL_GetTick(); //get current time in milliseconds.
    if((int32_t)(Ts-(t1-t0)) > 0)
      HAL_Delay(Ts-(t1-t0)-1);
  }
}
```

El segundo código utilizado en C ha sido para Controlar el motor con el controlador PI discreto. A través de la UART se transmiten los datos muestreados de la velocodad en tiempo real del motor.

```
float motorRPM = 0; //motor speed (RPM).
bool newRPM = false;
bool measure = false;

float PIcontroller(float e_1);

float PIcontroller(float e_1) {
  static float e_ = 0;
  static float u_ = 0;
  float u_1;

  u_1 = u_ + 0.02185*e_1 - 0.01615*e_;

  u_ = u_1;
  e_ = e_1;

  return u_1;
}

int main(void)
{
  /* USER CODE BEGIN 1 */
  float refRPM = 150; //speed reference (set point).
  float uV;
  uint8_t Ts = 30; //the sample rate (milliseconds) -> CONTROLLER
  uint32_t t0, t1; //variables to measure time.
  uint8_t *data2send;
  /* USER CODE END 1 */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, false);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, true);
  PWMsetpulse(333);
  /* USER CODE END 2 */

  while (1)
  {
    /* USER CODE BEGIN 3 */
    HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_7); //test pin, to measure sample time.
    t0 = HAL_GetTick(); //get current time in milliseconds.
    measure = true;
    while(!newRPM) {

    }
    measure = false;
    data2send = (uint8_t*) &motorRPM;
    HAL_UART_Transmit(&huart2, data2send, 4, HAL_MAX_DELAY); //1float = 4bytes.
    //uV = IAcontroller(refRPM - motorRPM)*166.667;
    uV = PIcontroller(refRPM - motorRPM);
    if(uV >= 2 && uV <= 12)
      PWMsetpulse(uV*166.667);
    else {
```

```
60          if(uV > 12)
61            PWMsetpulse(2000);
62          if(uV < 2)
63            PWMsetpulse(333);
64        }
65        newRPM = false;
66        t1 = HAL_GetTick(); //get current time in milliseconds.
67        if((int32_t)(Ts-(t1-t0)) > 0)
68          HAL_Delay(Ts-(t1-t0)-1);
69      }
70      /* USER CODE END 3 */
71  }
```