Laboratorio: Control de posición de Motor DC

Victor Christian Paredes Cauna

Obtención de data

1.1 Descripción

En el presente laboratorio se implementará un controlador de posición y velocidad para un motor DC con encoder. Para ello, se usará un driver para motores DC basado en el circuito integrado L298. Utilizando ArduinoIO se realizará la lectura de posición y velocidad angular para identificar los funciones de transferencias correspondientes. Adicionalmente, se diseñarán lazos de control en Simulink y se usaran bloques de Arduino para implementarlos en nuestro sistema físico.

1.2 Materiales

Los materiales necesarios para ésta experiencia se muestran en la Fig. 1.1 y se detallan debajo.

1. Arduino UNO o superior
2. PC con Matlab
3. Matlab Support Package for Arduino Hardware
4. Driver Puente H - L298
5. Cables Jumper
6. Motor DC de 12V - 36V (máximo)
7. Bateria/FuenteV: Tipo de fuente de voltaje dependiente del voltaje del motor.

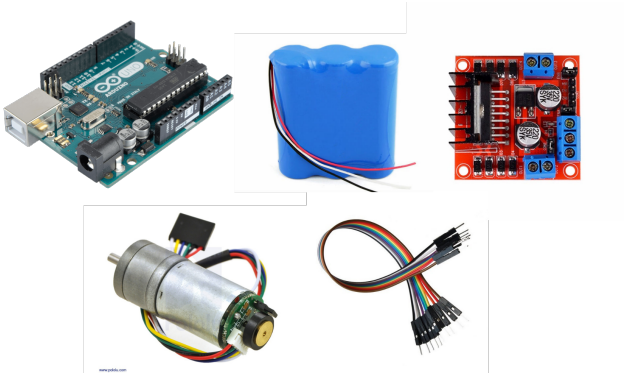

Figure 1.1: Materiales

1.3 Procedimiento

1.3.1 Conexión Driver L298 con el motor DC

El módulo L298 tiene la posibilidad de controlar hasta 2 motores DC, por ello tiene pines que afectan sólo a una salida y el resto de pines a la otra salida de voltaje para el motor DC. Los pines que se conectarán del driver L298 se muestran en la Fig. 1.2 y cada uno de ellos se describe debajo:

1. V_m : Voltaje de alimentación del motor, depende del voltaje nominal del motor. (12V por ejemplo)

2. V_{logico} : Salida de 5V del regulador interno, si el voltaje es mayor de 12V desconectar el jumper y alimentar 5V de manera externa.

3. ENA : Habilita el voltaje aplicado el motor conectado a M1 y M2 cuando la entrada es 5V, deshabilita el driver cuando la entrada es 0V.

4. $IN1, IN2$: Pines que determinan el sentido del voltaje aplicado al motor DC. Un sentido es indicado por IN1 = 5V, IN2 = 0V y el otro sentido por IN1 = 0V, IN2 = 5V.

5. $IN4, IN4, ENB$: Mismo funcionamiento que los pines anteriores pero para el segundo motor con salidas M3 y M4.

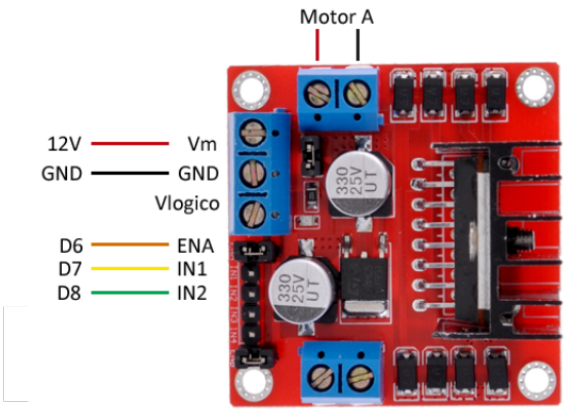

Figure 1.2: Posible conexión L298 con Arduino UNO

1.3.2 Conexión del Encoder con Arduino

El encoder de cuadratura nos permite medir la variación en posición angular de un eje. Es común encontrar algunos motores que presentan encoders en la parte trasera del mismo. Las señales que otorga éste sensor son:

1. Motor+: Usualmente un cable rojo para alimentación del motor.
2. Motor-: Usualmente un cable negro para tierra del motor.
3. GND: Tierra eléctrica.
4. Power: Entrada lógica para funcionamiento de motor. Revisar valores máximos y mínimos de voltaje.
5. Pin A: Señal *A* del encoder para contar pulsos.
6. Pin B: Señal *B* del encoder para determinar la dirección de movimiento.

Se necesita conectar el Pin A y el pin B a *pines con interrupción de arduino* para tener la mejor medición de este sensor. Así que conectaremos Pin A con D2 y Pin B con D3.

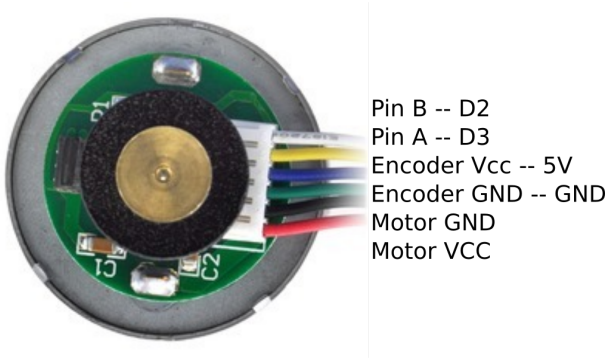

Figure 1.3: Cables del encoder

1.3.3 Implementación

Una posible implementación con un motor de 12V conectado a las salidas M3 y M4 se muestra a continuación:

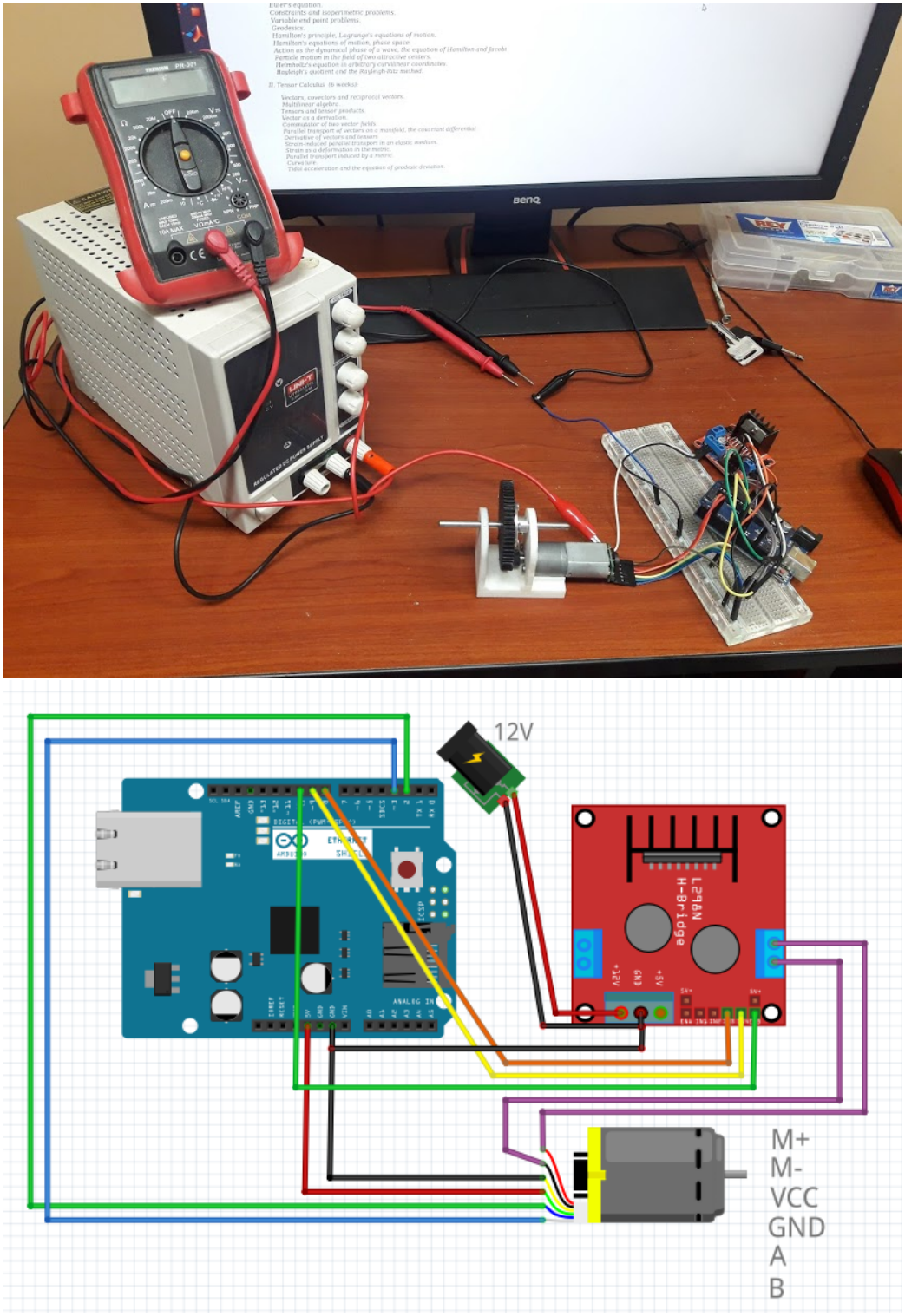

Figure 1.4: Implementación de las conexiones para realizar la identificación del motor DC

1.4 Usando ArduinoIO para la captura de datos

En este caso usaremos ArduinoIO (Legacy) para acceder a las funciones de lectura del encoder incremental. Además se controlaran los pines digitales de Arduino para activar el driver L298, como se muestra en la Fig. 1.5.

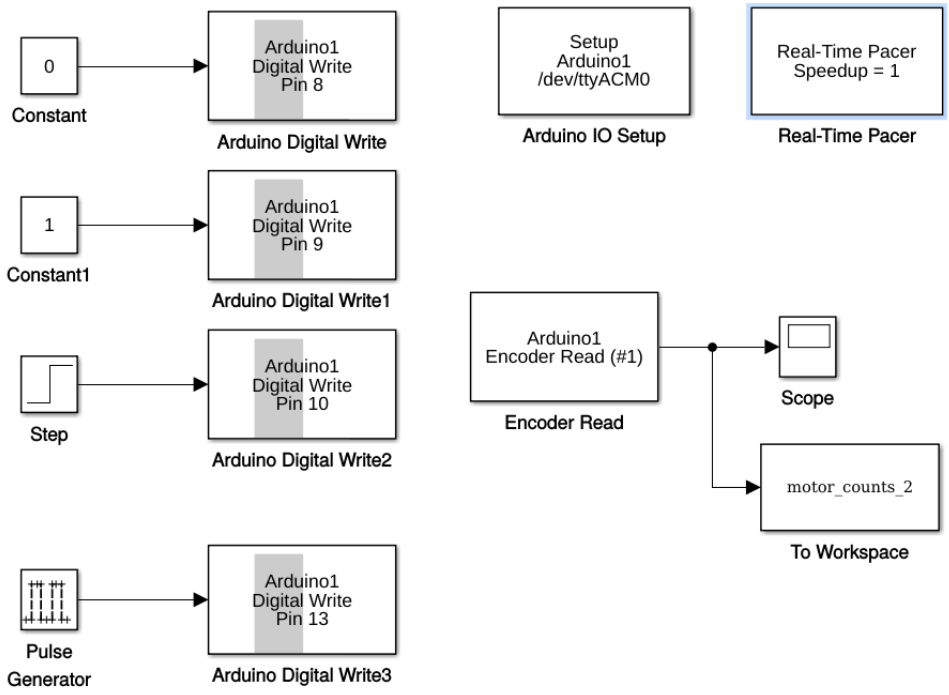

Figure 1.5: Diagrama de bloques inicial en Simulink

1.4.1 Rollover

Al revisar el gráfico del scope se observa que los contadores del encoder llegan a una valor máximo (o mínimo) y su valor sufre una discontinuidad de tipo overflow (o underflow), llamada *Rollover*, como se muestra en la Fig. 1.6

Para corregir el *Rollover* es necesario agregar un desfase a la señal, la siguiente lógica compara el incremento de los contadores y en base al resultado agrega un desfase positivo o negativo de 65535. La ultima parte de los bloques permiten manetener el valor del desfase para las demás señales. Nótese que un *Unit Delay* es un bloque que accede a la señal en un tiempo previo es decir $s(k-1)$ para una

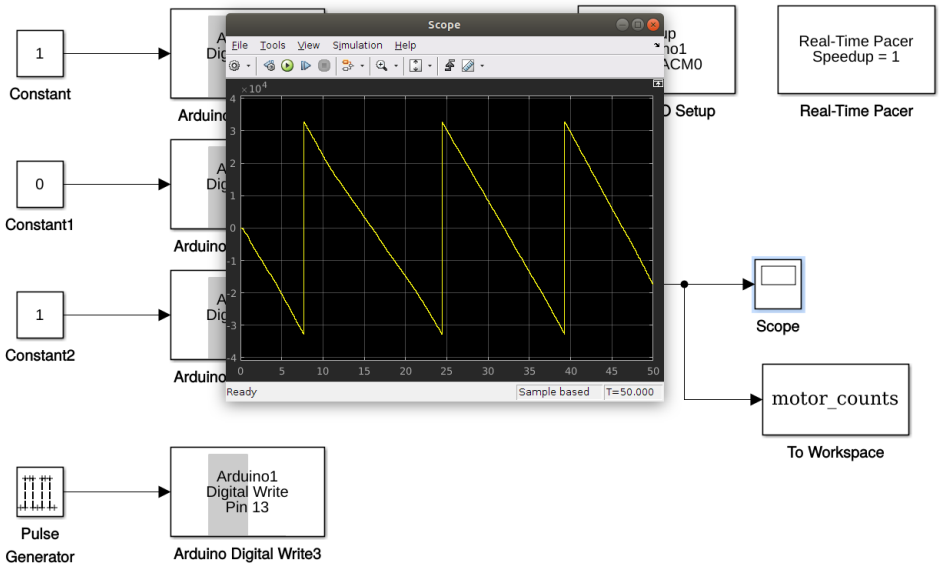

Figure 1.6: Rollover en la medición de los contadores del encoder

entrada $s(k)$.

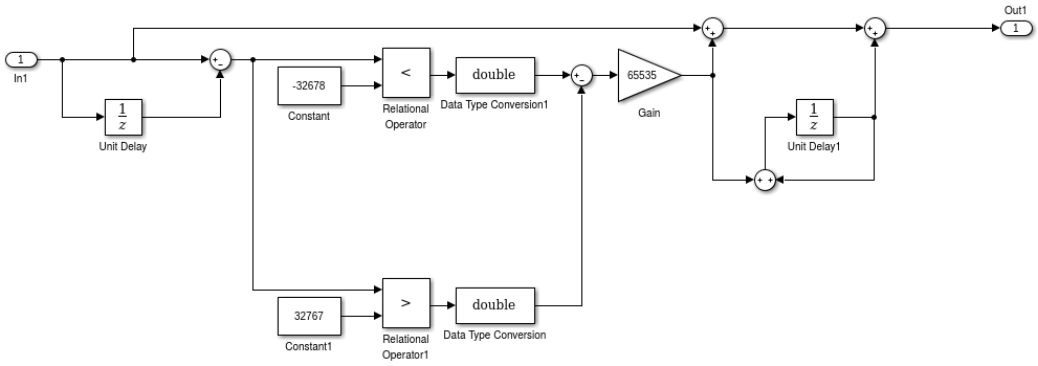

Figure 1.7: Correccion en el rollover del encoder

1.5 Procesamiento de la data

La data obtenida esta representada por el numero de contadores obtenidas del encoer, como se muestra la Fig. 1.8 para 3 segundos de simulación.

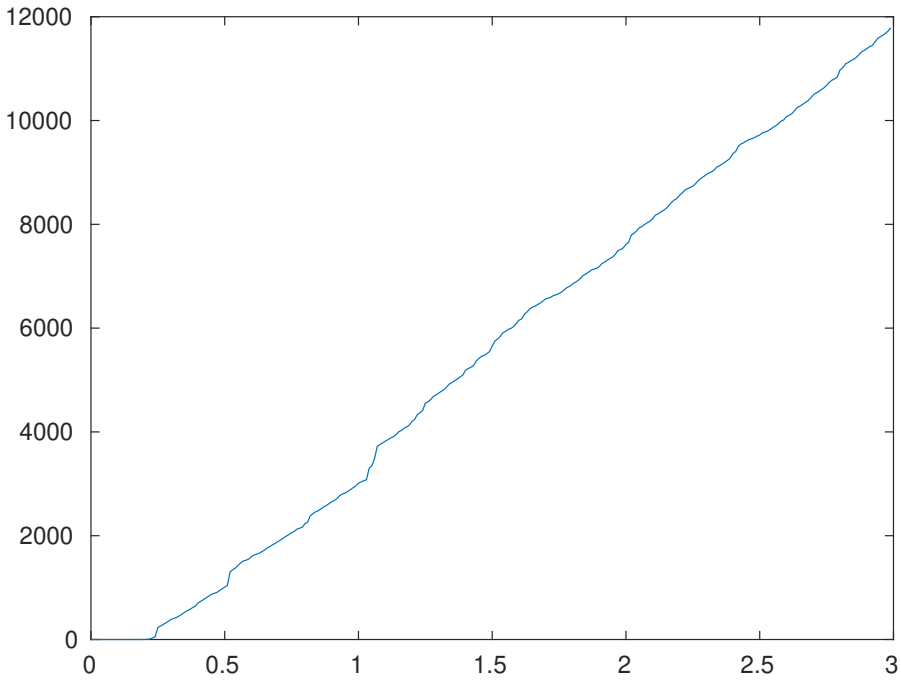

Figure 1.8: Data del encoder indicando posición basada en contadores

Para obtener la velocidad se necesita diferenciar la posición, en MATLAB podemos aplicar el comando *diff*, obteniendo la Fig. 1.9. En Simulink podemos llamar al bloque de derivación o implementar una derivada discreta basada en la diferencia de las señales entre el tiempo de muestreo.

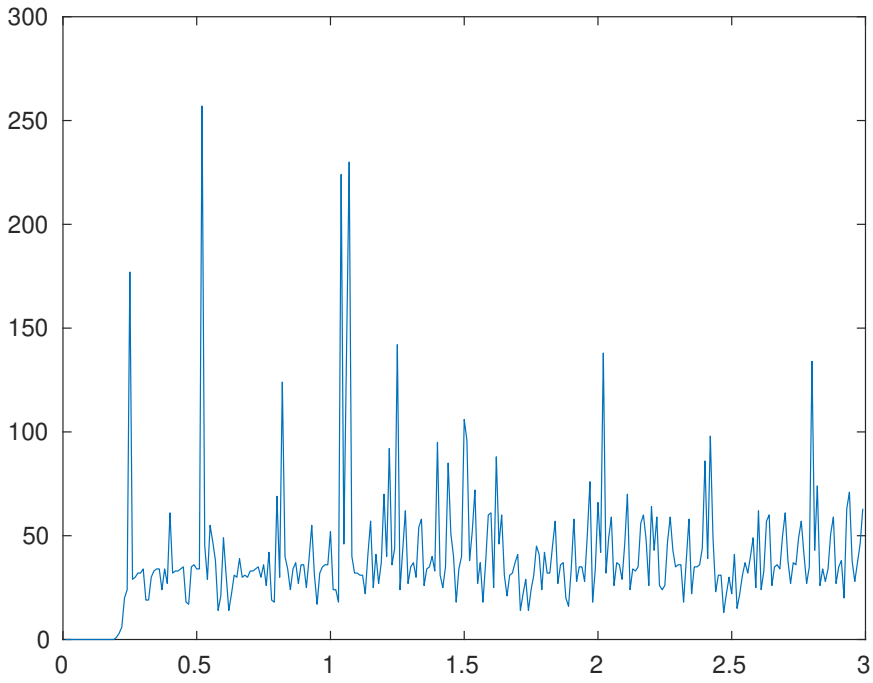

Figure 1.9: Velocidad de rotación

1.5.1 Filtrado de la data

En la práctica usualmente un ingeniero de control se encuentra con data ruidosa, es decir, que contiene elementos en la salida que no corresponden solamente a su modelo, sino también a señales externas, calidad del motor y los encoders y diferencias con el modelo real, como por ejemplo la vibración. Para poder seguir usando la data es necesario *filtrarla*. Si observamos el comportamiento de la data, observamos que se presenta como una señal aleatoria de alta frecuencia, usaremos un filtro RC para atenuar el ruido de la velocidad del motor, como se muestra en la Fig. 1.10

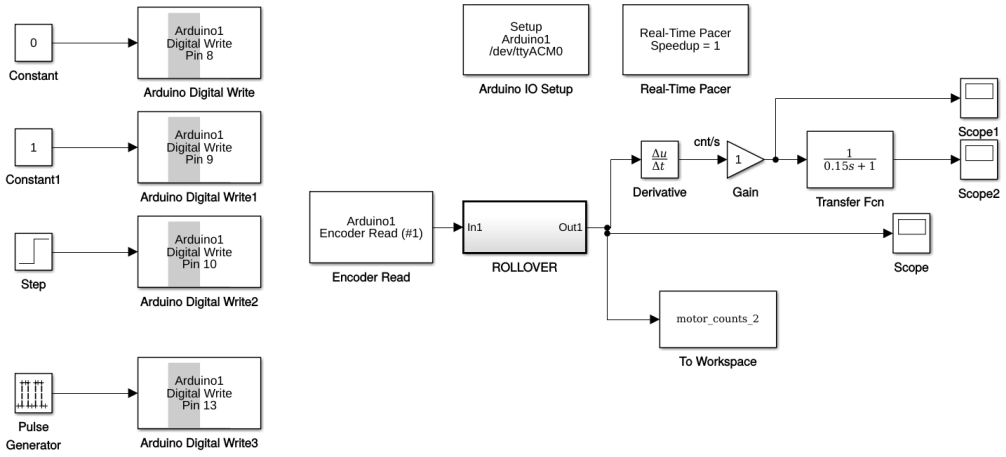

Figure 1.10: Filtrado de la señal de velocidad

1.5.2 Identificación del Modelo

Usando la data filtrada se procede a identificar el modelo, como se muestra en la Fig. 1.11.

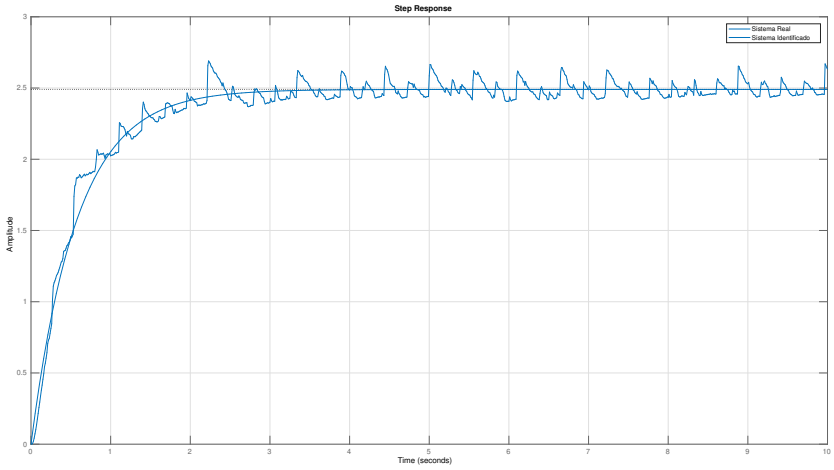

Figure 1.11: Identificación de la función transferencia

```
close all
load('counts')
load('dcounts')

plot(counts.Time, d_counts.Data)

%% IDENTIFIED SYSTEM
% Ganancia:
% media de los ultimos valores de la respuesta del sistema
Gain = sum(d_counts.Data(401:end))/(length(d_counts.Data) - 400);
Ts = 2.3; % Tiempo de asentamiento

s = tf('s');
G = Gain / (Ts/4*s + 1);
hold on
step(G, counts.Time(end))
legend('Sistema Real','Sistema Identificado')

%% Position Transfer Function
Gp = 1/s * G;
% 5ms
Kp = 3;
Kd = 0.02; % Digital 1
Ki = 1;% Digital: 0.005
C = pid(Kp, Ki, Kd);

D = Gp * C / (1 + Gp * C);
figure
step(D,10)
```

Chapter 2

Implementación en Arduino: Control de Posición

2.1 Requerimientos

Para implementar el controlador en arduino necesitamos dos librerias externas:

1. Libreria de Encoder para Arduino / Teensy
2. PWM para Arduino UNO/Mega

2.2 Notas

El motor DC a baja velocidad exhibe un comportamiento no lineal debido a la fricción, cuando se le aplica un voltaje bajo es posible que el motor no se mueva debido a la fricción estática.

La velocidad del motor necesita ser filtrada para poder hacer uso adecuado del derivador en el controlador PID, pues si se presentan picos la derivada resulta ser muy grande.

En conclusión, la dinámica puede incluir términos no lineales a baja velocidad y para usar correctamente el término derivativo se necesita desarrollar un filtro que permita el pase de frecuencias bajas pero atenue las frecuencias elevadas correspondientes al ruido.

Para realizar el controlador usando los filtros se debe usar Simulink o un microcontrolador con mejores capacidades que Arduino UNO.

A continuación se seguirá el procedimiento de *tunning*, o ajuste de gananacias para un controlador PID. Recordemos que la ganancia proporcional disminuye el error en estado estacionario y disminuye el tiempo de asentamiento pero puede generar oscilaciones o picos no deseados, el término integrador elimina el error en estado estacionario y el término derivativo puede suavizar la respuesta del controlador para disminuir los picos generados.

2.3 Código

```
#define ENCODER_OPTIMIZE_INTERRUPTS
#include <Encoder.h> // pins 2, 3, 18, 19, 20, 21
// Encoder tiene 20CPR -> 20 cuentas por revolucion
// Motor tiene reduccion de 488
// Cada cuenta (count) tiene una resolucion a la salida de:
// count * 360 / ( 20 * 488)
#include <PWM.h>

int32_t frequency = 20000; //20Khz
int pwmval = 180;
int8_t pinA = 7;
int8_t pinB = 8;
bool once = true;
bool stopAll = true;

Encoder EncoderM1(2,3);
long encoderSignal;
long positionM = -999;

// POSITION PD
float KpPos = 1.5;
float KdPos = 0.000005;
float KiPos = 5;

float Reference = 4000;

void setup() {
  // put your setup code here, to run once:
```

```
  Serial.begin(115200);
  Serial.println("Encoder");

  InitTimersSafe();
  bool success = SetPinFrequency(9, frequency);
    if(!success) {
      Serial.print("error1");
    }

  pinMode(pinA, OUTPUT);
  pinMode(pinB, OUTPUT);

  digitalWrite(pinA, HIGH);
  digitalWrite(pinB, LOW);

  // Configurando el encoder: Posici n Inicial es el nuevo cero
  EncoderM1.write(0);

}

void loop() {
  // put your main code here, to run repeatedly:
  encoderSignal = EncoderM1.read();

  //Serial.println(encoderSignal);
  positionM = encoderSignal;

  if(once) {
    pwmWrite(9, 0);
    once = false;
  }

  if(!stopAll) {
    PD_position(Reference);
    pwmWrite(9, pwmval);
  }
```

```
  delay(5); // 5ms -> 1/(5ms) = 200 Hz
}

void serialEvent() {
  while(Serial.available()){
    char tempo = Serial.read();
    if(tempo == '1') { // Mover el motor en sentido 1
      digitalWrite(pinB, LOW);
      digitalWrite(pinA, HIGH);
      stopAll = true;
      pwmWrite(9, 200);
    }else if(tempo == '2'){ // Mover en sentido 2
      digitalWrite(pinA, LOW);
      digitalWrite(pinB, HIGH);
      pwmWrite(9, 200);
      stopAll = true;
    }else if(tempo == '0'){ // Detenerse
      pwmWrite(9, 0);
      stopAll = true;
    }else if(tempo == 'p') { // Aplicar controlador
      stopAll = false;
    }

  }
}

void PD_position(int target) {
  static float oldPos = 0;
  static float errorSum = 0;
  float error_d;
  float error;
  // Error en velocidad
  error_d = (positionM - oldPos)/0.005;
  error = target - positionM;
  if(error < 400 && error > -400) {
      errorSum += error * 0.005;
      if(error < 2 && error > -2) { // Detener si error es muy bajo
        errorSum = 0; // 2 Counts = 2*360/(20*488)=0.07grados
```

```
    }
  }else{
    errorSum = 0;
  }


  float control_signal =
  KpPos*error + KdPos*error_d + KiPos*errorSum;
  Serial.print(error * 2.0 * 360.0 / ( 20.0 * 488.0 ));
  Serial.print("    ");
  Serial.println(control_signal);

  if(control_signal >= 0) {
      digitalWrite(pinA, LOW);
      digitalWrite(pinB, HIGH);
  }else {
      digitalWrite(pinB, LOW);
      digitalWrite(pinA, HIGH);
      control_signal *=-1;
  }

  control_signal = (control_signal <= 254 ? control_signal : 254);
  pwmval = floor(control_signal);

}
```

2.4 Resultados

El vídeo del experimiento se puede encontrar en este enlace.

El error final se muestra en el terminal serial de arduino (Fig. 2.1), donde se especifica el error y la señal de control residual. Se observa que el error final es cero (en general no siempre tiende a cero pero puede ser muy pequeño) debido al efecto del integrador.

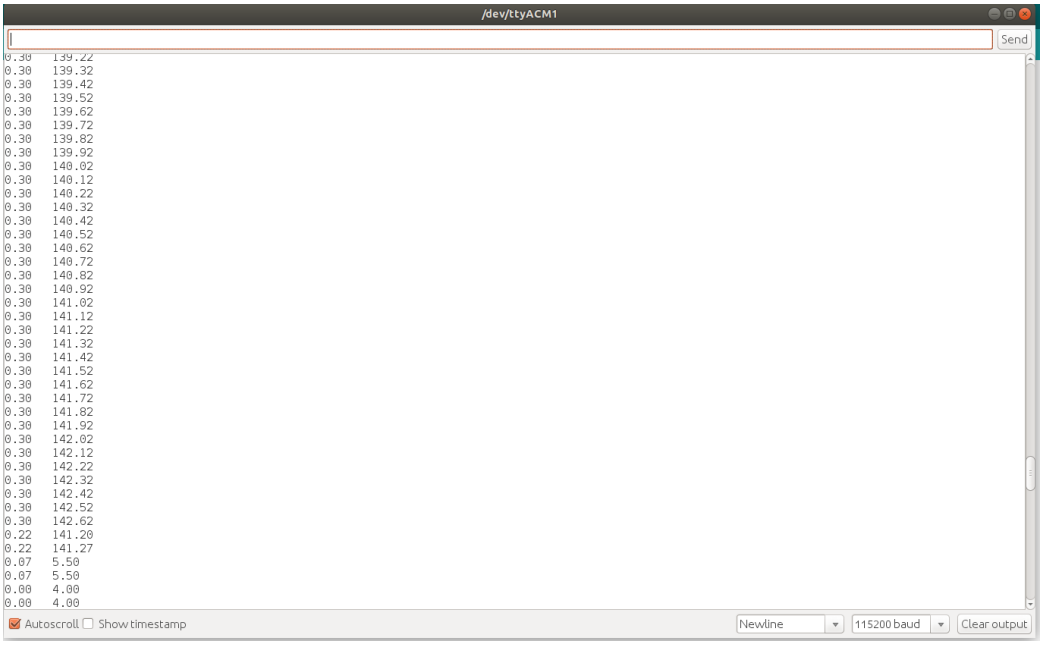

Figure 2.1: La primera columna es el error y la segunda la señal de control