Prácticas Inf. Industrial: Control de Motor DC Simulink y C++

Óscar David López Arcos
75571640B
odlarcos@correo.ugr.es

Ejercicio 1

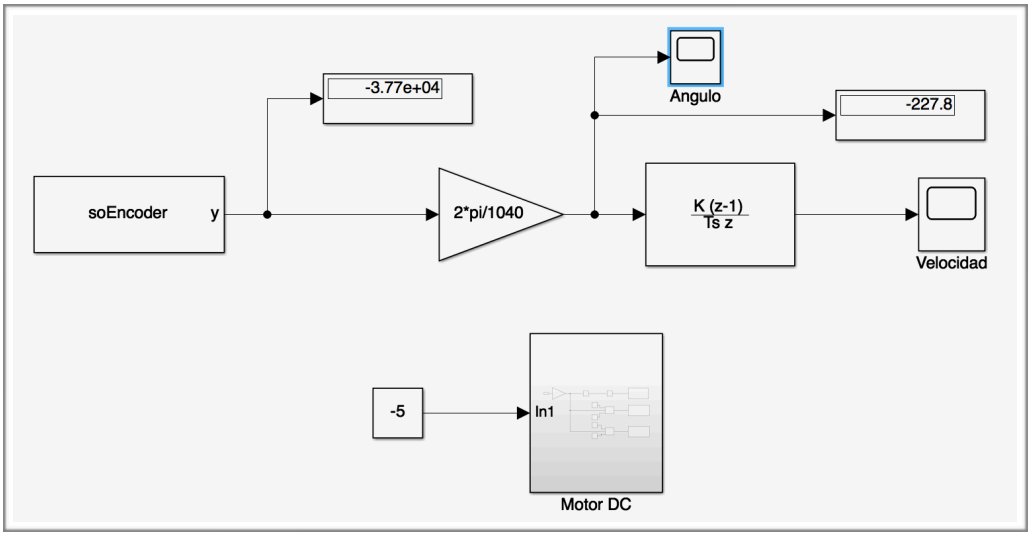
Definimos un programa en C que se contecte adecuadamente con el motor, estableciendo los pines necesarios para producir el giro en ambos sentidos. Calculamos la velocidad de giro en función del voltaje y modificamos el ángulo según aumente o disminuya la lectura de pulsos en el Encoder, imprimiéndolo posteriormente por el puerto serie. El programa obtenido sería:

```
#include <TimerOne.h>

const int CHA = 2; // Canal A del encoder del motor conectado a pin 2 (entrada)
const int CHB = 3; // Canal B del encoder del motor en pin 3

const int PWM_Motor = 9; // Pin Salida PWM para el motor DC
const int AIN1 = 11; // Enable AIN1 del puente H del motor
const int AIN2 = 10; // Enable AIN2 del puente H del motor

const int Volt = -3; // Valor del voltaje (-8.5 / 8.5)

double angulo=0; // angulo medido a partir de pulsos del encoder
double pulsos=0; // pulsos del encoder

void setup() {

  //La lectura de los encoder se realizara por interrupciones, una por cada canal
  attachInterrupt(0, Encoder_CHA, CHANGE); // INTERRUPCION MEGA para PIN 2
  attachInterrupt(1, Encoder_CHB, CHANGE); // INTERRUPCION MEGA para PIN 3

  //Configuracion de los pines de E/S
  pinMode(AIN1,OUTPUT);
  pinMode(AIN2,OUTPUT);
  pinMode(PWM_Motor,OUTPUT);

 //Comenzamos con el motor parado
  digitalWrite(AIN1,HIGH);
  digitalWrite(AIN2,HIGH);

  int valorPWM8;
  if (Volt>0){
    valorPWM8 = byte(Volt*30); // PWM está comprendido entre 0 y 255
    digitalWrite(AIN1, HIGH);
    digitalWrite(AIN2, LOW);
    analogWrite(PWM_Motor, valorPWM8);

  } else {
    valorPWM8 = byte(-Volt*30); // Volt negativo:
    digitalWrite(AIN1, LOW); // se cambia el sentido de giro
    digitalWrite(AIN2, HIGH);
    analogWrite(PWM_Motor, valorPWM8);
  }
```

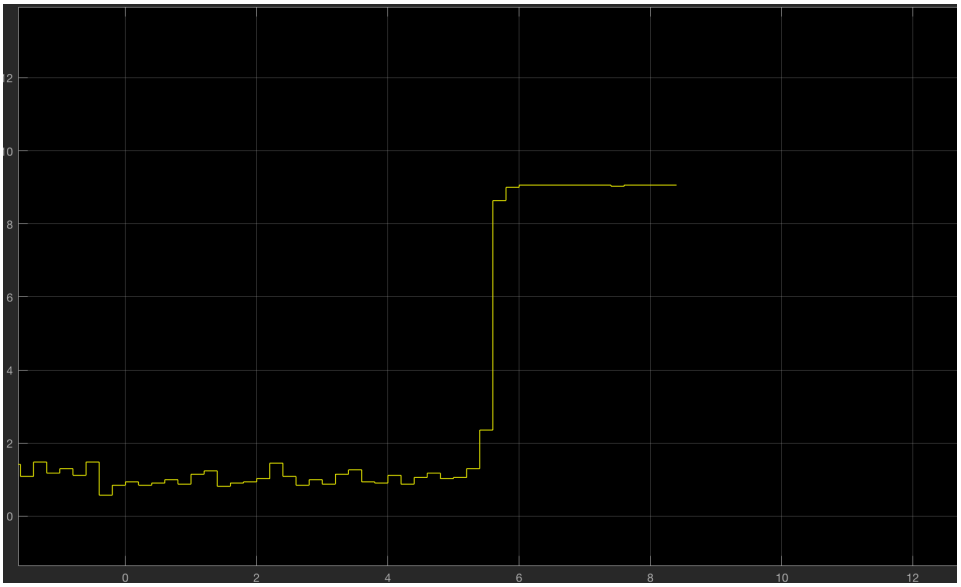
```
//Lectura de pulsos del canal A del encoder del motor en cada transición
void Encoder_CHA(){
  if(digitalRead(CHA)==digitalRead(CHB)){
    pulsos = pulsos-1;
  }
  else{
    pulsos = pulsos+1;
  }

  Serial.print( pulsos*2*3.1416/1040 );
}

//Lectura de pulsos del canal B del encoder del motor en cada transición
void Encoder_CHB(){
  if (digitalRead(CHA)!=digitalRead(CHB)){
     pulsos = pulsos-1;
  }
  else{
    pulsos = pulsos+1;
  }

  Serial.print( pulsos*2*3.1416/1040 );
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Ejercicio 2

Para este ejercicio, después de definir la estructura del motor en Simulink (estableciendo pines para el sentido de giro y velocidad en función del voltaje), diseñamos el siguiente programa que obtenga el ángulo de giro en función de los pulsos del encoder y calcule la derivada del ángulo para obtener la velocidad angular:

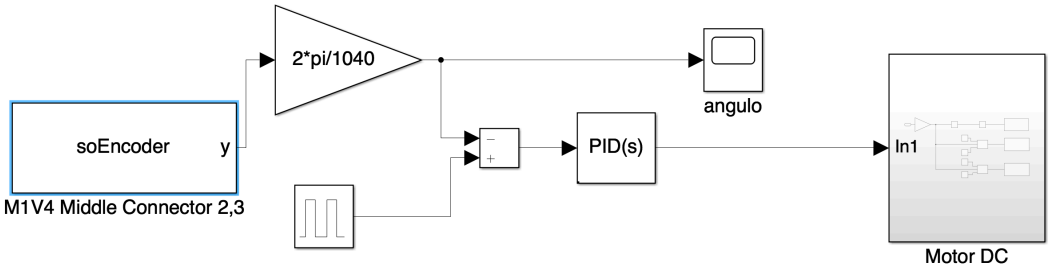


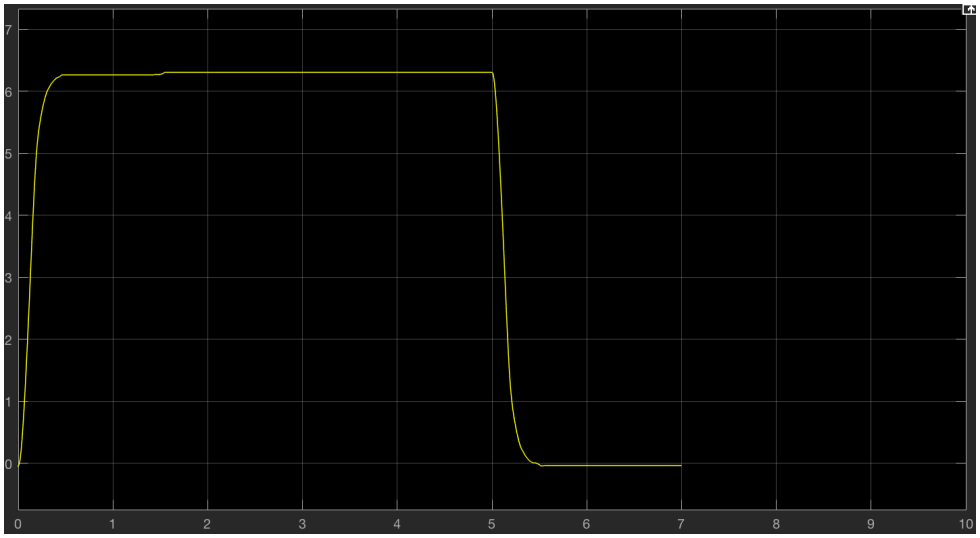
Si observamos la variación de velocidad, se dispara cuando encendemos el motor y se estabiliza al alcanzar la velocidad definida por el voltaje. El ángulo y los pulsos disminuyen constantemente debido a que el voltaje introducido es negativo.



Ejercicio 3

Generamos un cambio de consigna cada 5 segundos, de 0 a 2π radianes y viceversa. El movimiento del motor para igualarse al valor de la consigna será producido por la función de transferencia representada por el controlador PD de valores Kp=6.7, Kd=0.4. Obtenemos como resultado la siguiente gráfica.





Ejercicio 4

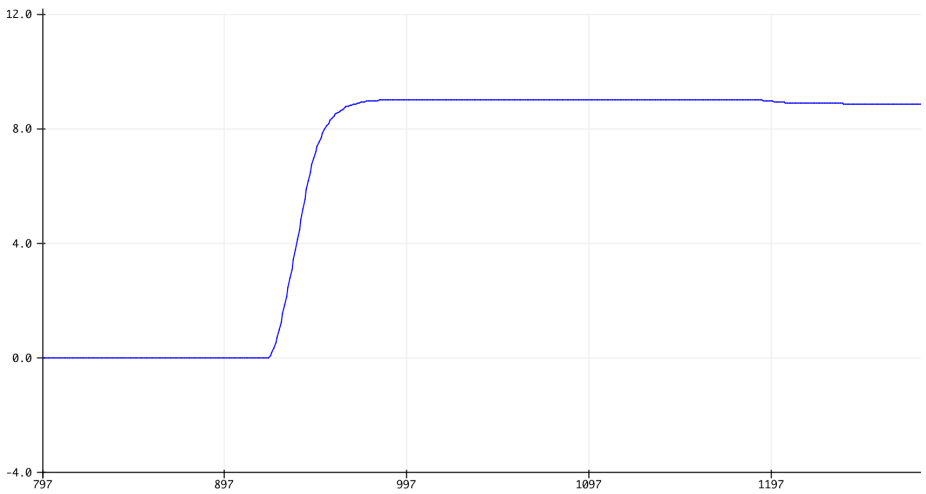
Definimos las variables necesarias para construir un controlador PID mediante la aproximación trapezoidal. Establecemos una rutina de control que se ejecutará cada 0.01s encargada de calcular el ángulo actual y la diferencia respecto a la consigna (Error) que usará en la fórmula del PID o para estimar la salida del voltaje del motor.

```
#include <TimerOne.h>

const int CHA = 2; // Canal A del encoder del motor conectado a pin 2 (entrada)
const int CHB = 3; // Canal B del encoder del motor en pin 3

const int PWM_Motor = 9; // Pin Salida PWM para el motor DC
const int AIN1 = 11; // Enable AIN1 del puente H del motor
const int AIN2 = 10; // Enable AIN2 del puente H del motor

double Kp = 6;
double Ki= 0.3;
double Kd = 0.5;

double Pn = 0; //Salida del PID para muestra n
double En = 0; // Error para muestra n

double Pn1 = 0; // salida PID para muestra n_1

double En1 = 0; //Error para muestra n_1
double En2 = 0; // Error para muestra n-2

double q0=0;
double q1=0;
double q2=0;

double Tm = 0.01; // periodo de muestreo en seg
double consigna = 6.28; // angulo de consigna
double angulo = 0; // angulo medido a partir de pulsos del encoder
double pulsos = 0; //pulsos del encoder

double SalidaPD = 0;
double EntradaPWM = 0;

int valorPWM8;
int inicio = 0;

void setup() {
  Serial.begin(9600);

  // Valores iniciales
  q0 = Kp+(Ki*Tm)/2+Kd/Tm;
  q1 = -Kp+(Ki*Tm)/2-(2*Kd)/Tm;
  q2 = Kd/Tm;
```

```
  // INPUT
  pinMode(CHA, INPUT_PULLUP);
  pinMode(CHB, INPUT_PULLUP);

  // put your setup code here, to run once:
  //La lectura de los encoder se realizara por interrupciones, una por cada canal
  attachInterrupt(0, Encoder_CHA, CHANGE); // INTERRUPCION MEGA para PIN 2
  attachInterrupt(1, Encoder_CHB, CHANGE); // INTERRUPCION MEGA para PIN 3

  //Configuracion de los pines de E/S
  pinMode(AIN1,OUTPUT);
  pinMode(AIN2,OUTPUT);
  pinMode(PWM_Motor,OUTPUT);

 //Comenzamos con el motor parado
  digitalWrite(AIN1,HIGH);
  digitalWrite(AIN2,HIGH);

  //Configuramos el temporizador para saltar la rutina de servicio cada 10ms
  Timer1.initialize(10000); // Ts=0.01 (10000 microsegundos)
  // cada 0.01 seg se salta a la rutina de servicio: "control"
  Timer1.attachInterrupt(control);
  Timer1.start(); // inicio de las interrupciones

}

//Lectura de pulsos del canal A del encoder del motor en cada transición
void Encoder_CHA(){
  if(digitalRead(CHA)==digitalRead(CHB)){
    pulsos = pulsos-1;
  }
  else{
    pulsos = pulsos+1;
  }
}

//Lectura de pulsos del canal B del encoder del motor en cada transición
void Encoder_CHB(){
  if (digitalRead(CHA) != digitalRead(CHB)){
     pulsos = pulsos-1;
  }
  else{
    pulsos = pulsos+1;
  }
}
```

```
void control(){

  // Controlador PID
  angulo = float(pulsos*(2*3.1416/1040 )); // conversión de pulsos a radianes

  En = consigna - angulo; // calculo error para la muestra actual
  Pn = Pn1 + (q0*En + q1*En1 + q2*En2); // Salida PIDx
  En2 = En1;
  En1 = En;
  Pn1 = Pn;

  // Acotar la salida Pn entre -8.5 y +8.5 voltios
  if(Pn > 8.5) Pn = 8.5;
  else if (Pn < -8.5) Pn = -8.5;

  Serial.println(angulo);

  if (Pn>0){
    valorPWM8 = byte(Pn*30); // PWM está comprendido entre 0 y 255
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, HIGH);
    analogWrite(PWM_Motor, valorPWM8);

  } else {
    valorPWM8 = byte(-1*Pn*30); // Volt negativo:
    digitalWrite(AIN1, HIGH); // se cambia el sentido de giro
    digitalWrite(AIN2, LOW);
    analogWrite(PWM_Motor, valorPWM8);
  }
}

void loop() {
}
```

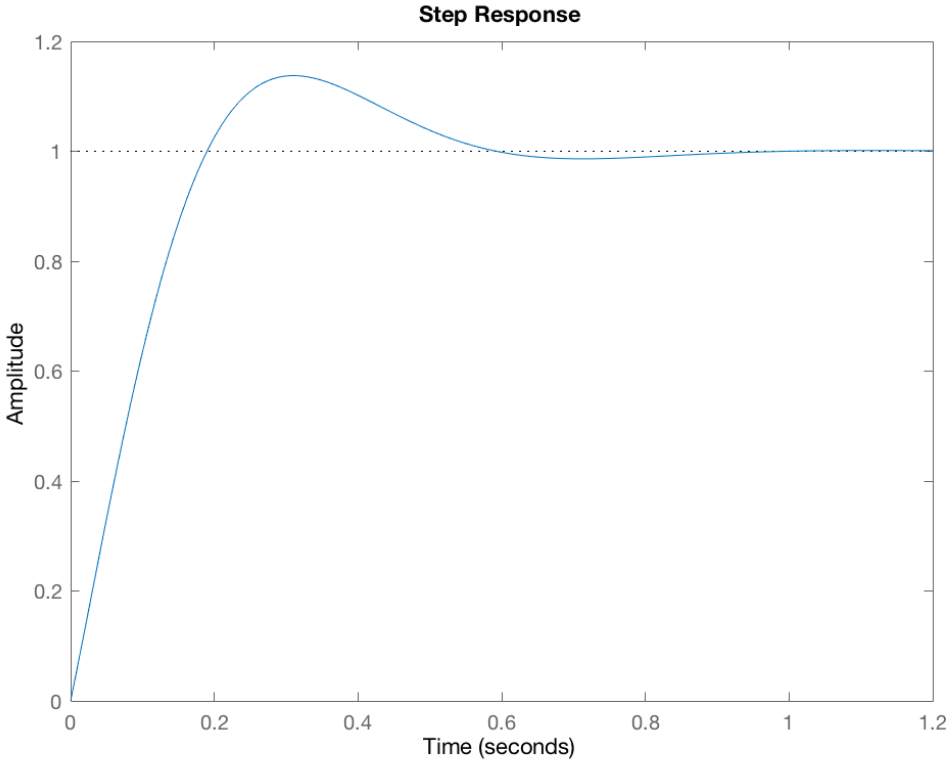
El resultado obtenido al imprimir el ángulo será girar una vuelta completa, ya que la consigna está establecida a 6,28 (2π radianes).



Ejercicio 5

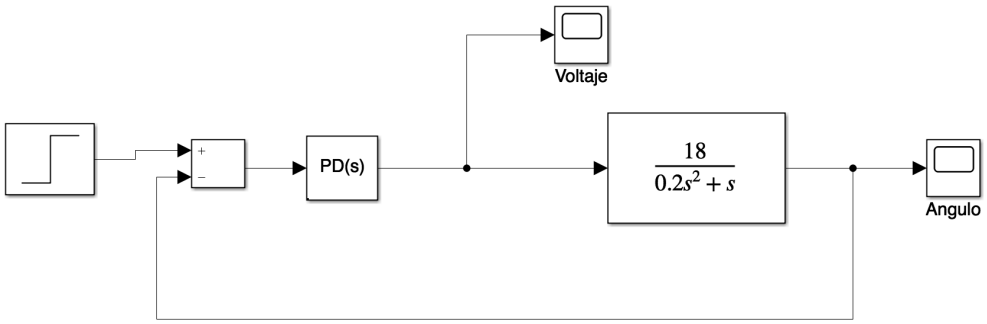
a) Aplicamos las ecuaciones de la función de transferencia del servo con un sobrepasamiento deseado del 0.0001% y un tiempo de establecimiento de 0.5s. Obtenemos como resultado los valores Kd = 0.0714 y Kp = 1.0382

```
% Especificaciones de diseño:
Ts=0.01; % periodo de muestreo
Mp=0.1;
ts=0.7; % Sobrepasamiento y tiempo de asentamiento
% Parametros de la f.d.t del motor
K=18; T= 0.2;
% Calculo de coef de amortiguamiento y frec. propia a partir de Mp , ts
xi=abs(log(Mp)/sqrt(pi^2+log(Mp)^2));
wn=4/(xi*ts);
p1=-wn*xi+j*wn*sqrt(1-xi^2);% polos sistema deseado
p2=-wn*xi-j*wn*sqrt(1-xi^2);
%Calculode KpyKd
Kd=(2*xi*wn*T-1)/K;
Kp=(wn^2)*T/K;
% Simulacion
fdtPD=tf([Kd Kp],1); % f.d.t del controlador PD: (Kp+Kds)
sysc=tf(K,[T 1 0]); % fdt continua motor servo
fdtlazoc=feedback(fdtPD*sysc,1);
step(fdtlazoc) % respuesta continua sistema controla
```



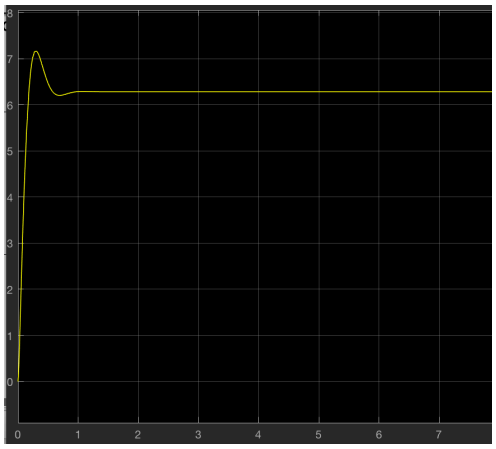
Puede observarse una respuesta bastante acertada , aunque el grado de sobrepasamiento es algo alto para el tiempo de asentamiento.

b) Representamos un PD básico con SimuLink con los valores obtenidos en el apartado anterior.

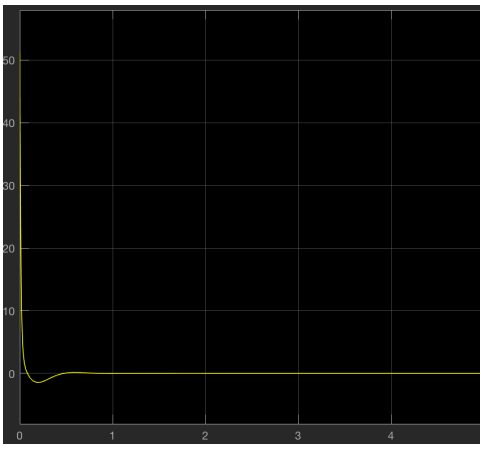


Si no limitamos el voltaje, el PID intentará alcanzar los 2π radianes (Valor de consigna) lo más rápido posible, llegando a alcanzar los 50V y produciendo bastante sobrepasamiento:

Comportamiento

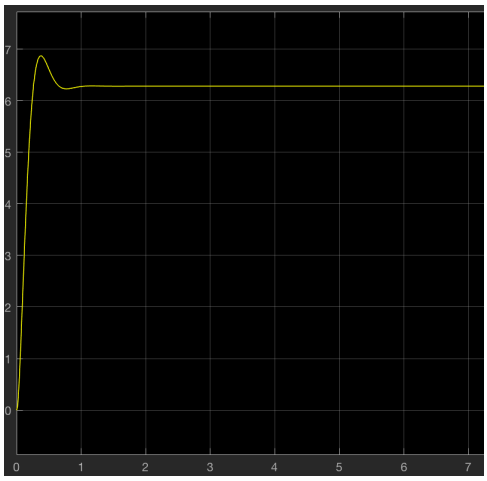


Voltaje

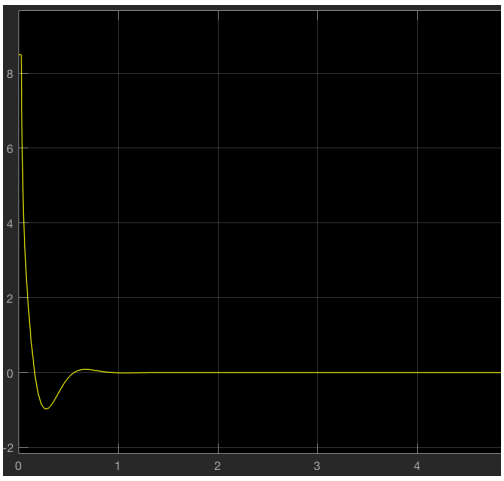


Si capamos el voltaje a +/- 8.5 Voltios, obtenemos una respuesta algo más lenta pero con menos sobrepasamiento:

Comportamiento

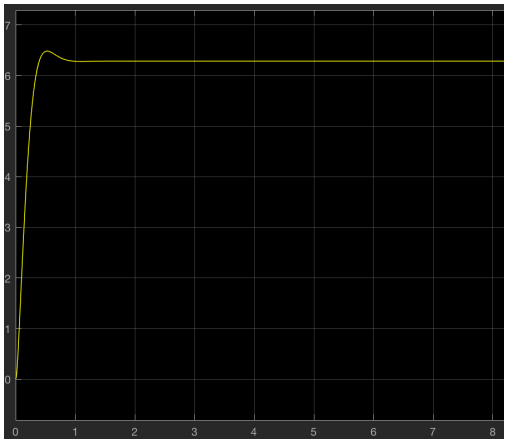


Voltaje

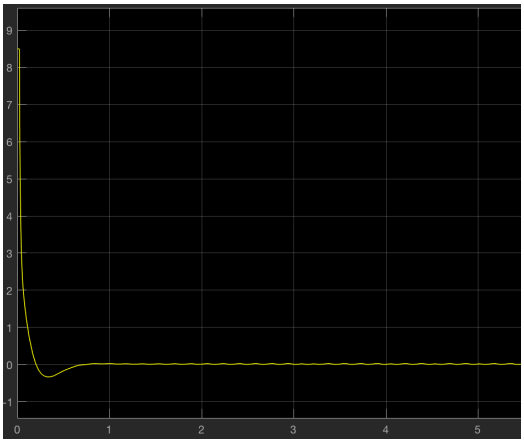


Por último, calculamos los valores óptimos para la función de transferencia, consiguiendo mejores resultados (Kp = 0.717 y Kd = 0.074):

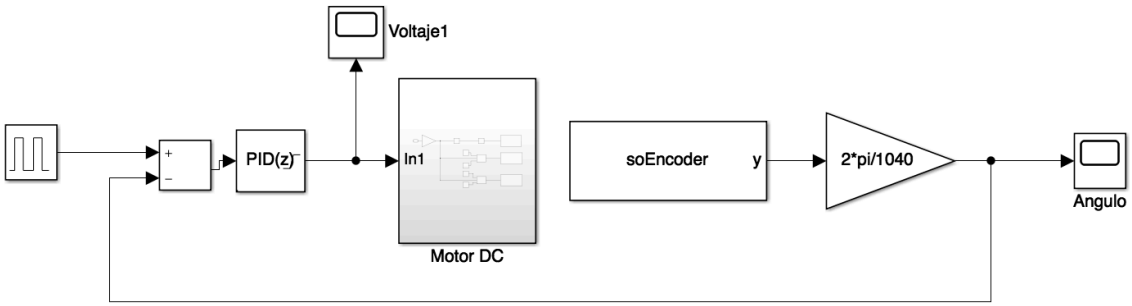
Comportamiento

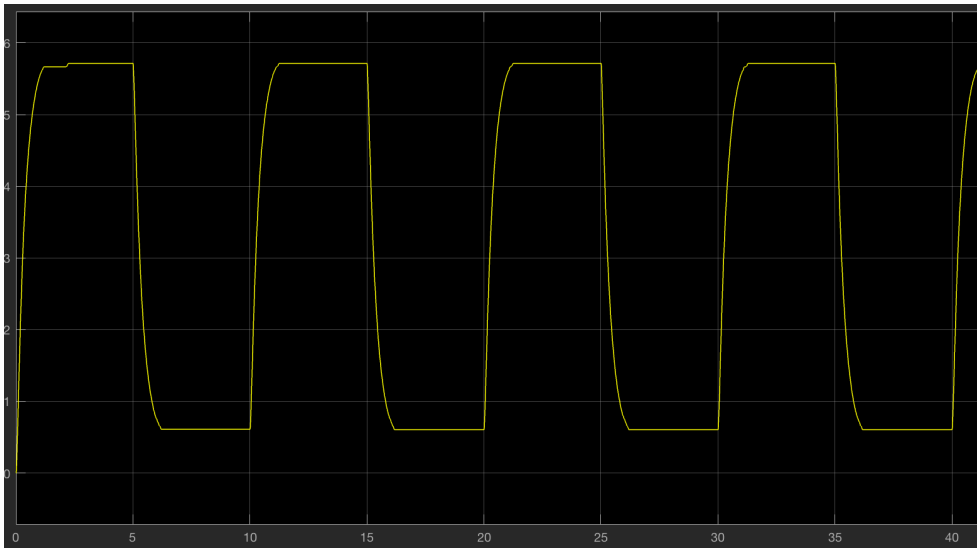


Voltaje



c) Probamos este último PD con el motor, y realizamos un cambio de consigna para que vuelva a girar, de 0 a 2π radianes.

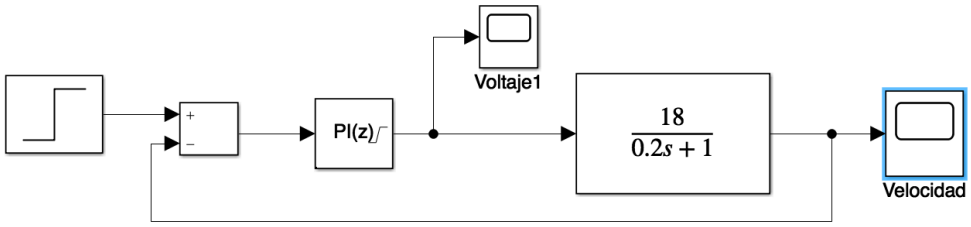




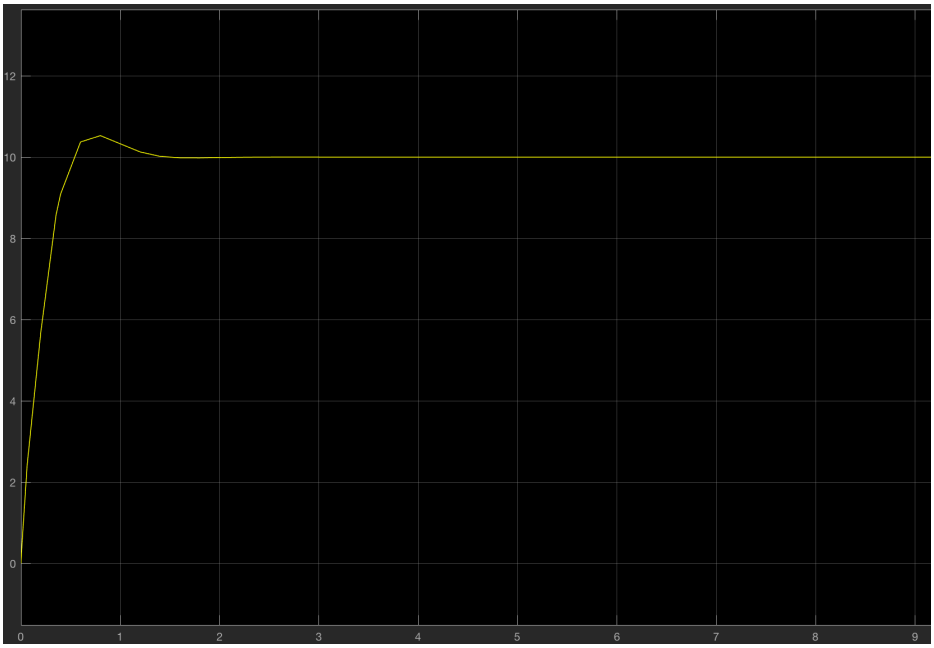
Al ser un controlador PD, nunca se establece al valor de la consigna, si no que mantiene un pequeño offset.

Ejercicio 6

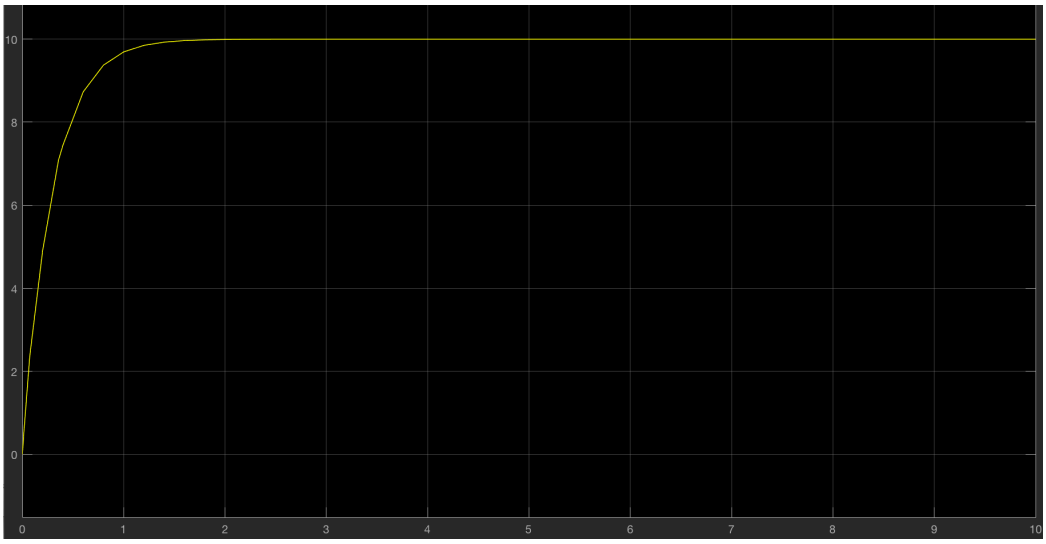
a) Representamos en Simulink un PI que dependa de la velocidad, con una consigna de 10 rad/seg. Utilizamos la fdt respecto a la velocidad angular.



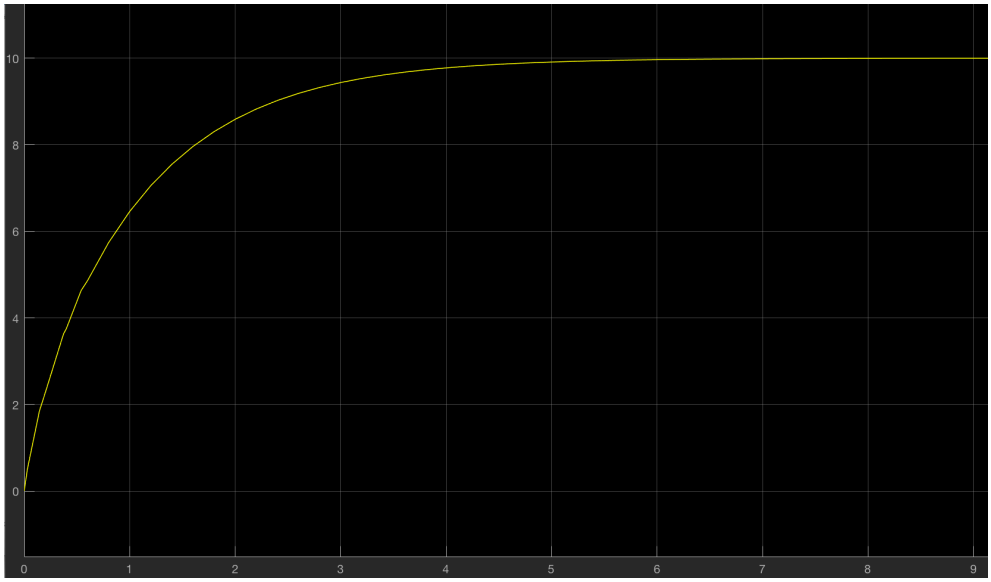
Para los valores Kp = 0.05 y Ki = 0.2, el resultado obtenido es:



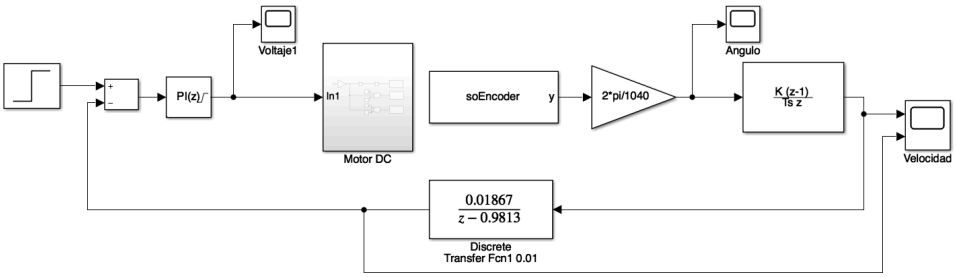
Para los obtenidos con TuningPi (Ki = 0.1380 y Kp = 0.0432), obtenemos:

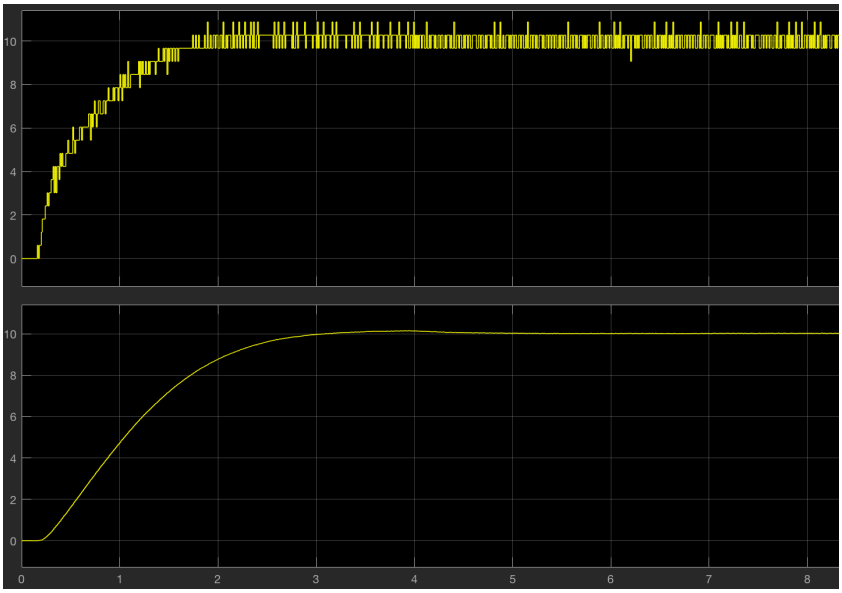


Y, por último, con los valores generados por el ajuste automático:



b) Al aplicar el mismo ejercicio sobre el motor real, debemos derivar el ángulo para obtener la velocidad angular, además de aplicar un filtro para suavizar la señal.





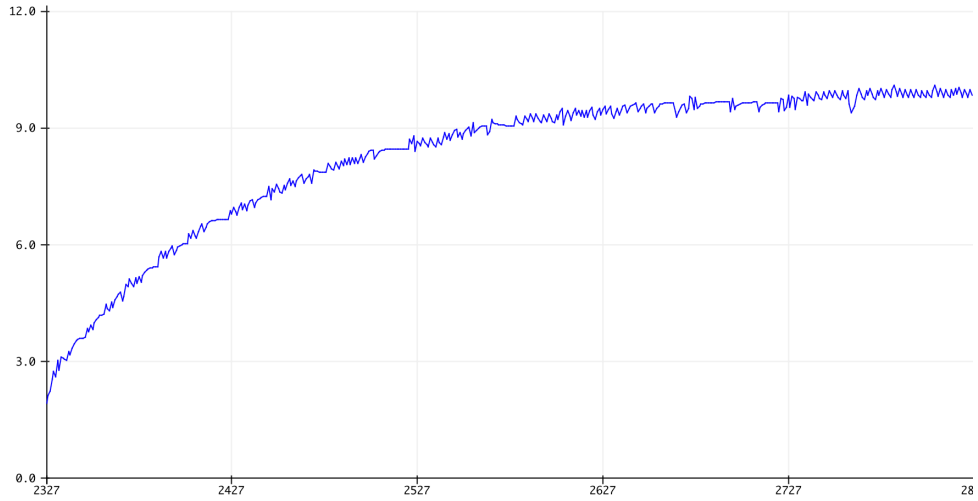
c) Por último, modificamos el código del ejercicio 4, estableciendo la consigna a 10 y realizando los cálculos necesarios para que nuestro controladorl PI dependa de la velocidad.

```
#include <TimerOne.h>

const int CHA = 2; // Canal A del encoder del motor conectado a pin 2 (entrada)
const int CHB = 3; // Canal B del encoder del motor en pin 3

const int PWM_Motor = 9; // Pin Salida PWM para el motor DC
const int AIN1 = 11; // Enable AIN1 del puente H del motor
const int AIN2 = 10; // Enable AIN2 del puente H del motor

//const int Volt = 0; // Valor del voltaje (-8.5 / 8.5)
double Kp = 0.05;
double Ki= 0.2;
double Kd = 0.0;

double Pn = 0; //Salida del PID para muestra n
double En = 0; // Error para muestra n

double Pn1 = 0; // salida PID para muestra n_1

double En1 = 0; //Error para muestra n_1
double En2 = 0;

double q0=0; // Variables aproximación trapezoidal
double q1=0;
double q2=0;

double Tm = 0.01; // periodo de muestreo en seg
double consigna = 10; // velocidad de consigna
double angulo = 0; // angulo medido a partir de pulsos del encoder
double velocidad = 0;
double angulo_ant = 0;
double alfa = 0.4;
double pulsos = 0; //pulsos del encoder
double SalidaPD = 0;
double EntradaPWM = 0;
int valorPWM8;

int inicio = 0;

void setup() {

  Serial.begin(9600);
  // Valores iniciales
  q0 = Kp+(Ki*Tm)/2+Kd/Tm;
  q1 = -Kp+(Ki*Tm)/2-(2*Kd)/Tm;
  q2 = Kd/Tm;
```

```
// INPUT
  pinMode(CHA, INPUT_PULLUP);
  pinMode(CHB, INPUT_PULLUP);

  // put your setup code here, to run once:
  //La lectura de los encoder se realizara por interrupciones, una por cada canal
  attachInterrupt(0, Encoder_CHA, CHANGE); // INTERRUPCION MEGA para PIN 2
  attachInterrupt(1, Encoder_CHB, CHANGE); // INTERRUPCION MEGA para PIN 3

  //Configuracion de los pines de E/S
  pinMode(AIN1,OUTPUT);
  pinMode(AIN2,OUTPUT);
  pinMode(PWM_Motor,OUTPUT);

 //Comenzamos con el motor parado
  digitalWrite(AIN1,HIGH);
  digitalWrite(AIN2,HIGH);

  //Configuramos el temporizador para saltar la rutina de servicio cada 10ms
  Timer1.initialize(10000); // Ts=0.01 (10000 microsegundos)
  // cada 0.01 seg se salta a la rutina de servicio: "control"
  Timer1.attachInterrupt(control);
  Timer1.start(); // inicio de las interrupciones

}

//Lectura de pulsos del canal A del encoder del motor en cada transición
void Encoder_CHA(){
  if(digitalRead(CHA)==digitalRead(CHB)){
    pulsos = pulsos-1;
  }
  else{
    pulsos = pulsos+1;
  }
}

//Lectura de pulsos del canal B del encoder del motor en cada transición
void Encoder_CHB(){
  if (digitalRead(CHA) != digitalRead(CHB)){
     pulsos = pulsos-1;
  }
  else{
    pulsos = pulsos+1;
  }
}
```

```
void control(){

  // Controlador PID
  angulo = float(pulsos*(2*3.1416/1040)); // conversión de pulsos a radianes

  angulo = alfa*angulo + (1-alfa)*angulo_ant; // Aplico el filtro de alisamiento

  velocidad = (angulo - angulo_ant) / Tm; // Calculo la derivada
  angulo_ant = angulo;

  En = consigna - velocidad; // calculo error para la muestra actual
  Pn = Pn1 + (q0*En + q1*En1 + q2*En2); // Salida PIDx
  En2 = En1;
  En1 = En;
  Pn1 = Pn;

  // Acotar la salida Pn entre -8.5 y +8.5 voltios
  if(Pn > 8.5) Pn = 8.5;
  else if (Pn < -8.5) Pn = -8.5;

  Serial.println(velocidad);

  if (Pn>0){
    valorPWM8 = byte(Pn*30); // PWM está comprendido entre 0 y 255
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, HIGH);
    analogWrite(PWM_Motor, valorPWM8);

  } else {
    valorPWM8 = byte(-1*Pn*30); // Volt negativo:
    digitalWrite(AIN1, HIGH); // se cambia el sentido de giro
    digitalWrite(AIN2, LOW);
    analogWrite(PWM_Motor, valorPWM8);
  }
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

En el código, puede verse como se calcula en ángulo de igual forma, pero se le aplica la derivada como vimos en clase después de un filtro de alisamiento. El resultado obtenido se aprecia como el motor asciende hasta 10, donde se estabiliza a pesar de las pequeñas oscilaciones.


12.0
9.0
6.0
3.0
0.0
2327
2427
2527
2627
2727
28


15.0
12.0
9.0
6.0
3.0
2502
2602
2702
2802
2902