Grado en Ingeniería Informática

Memoria de Trabajo

Informática Industrial

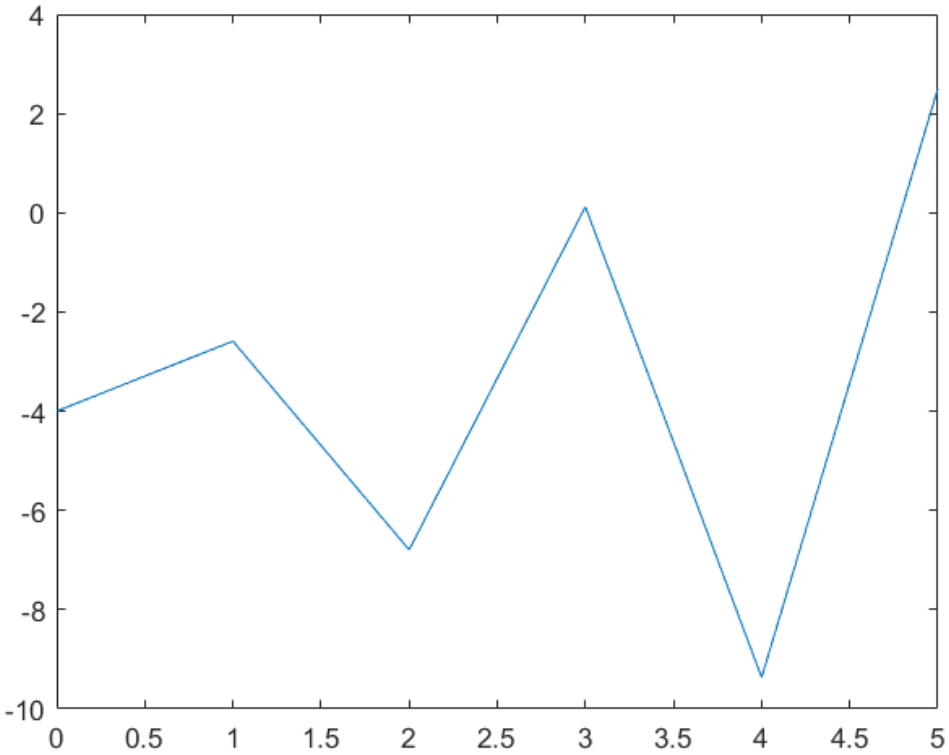
OSCAR JIMENEZ FERNANDEZ

1-11-2018

Introduciéndonos en Matlab

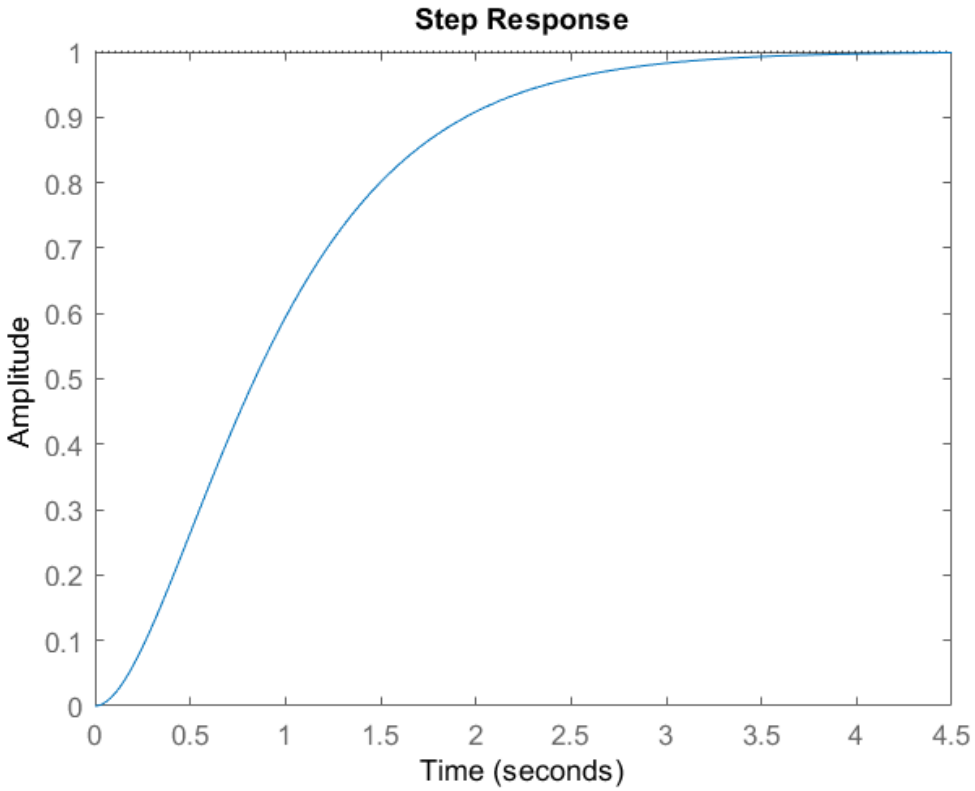
Representando funciones:

```
valores = [0:1:5];

y1 = 10*sin(3*valores)-4;

plot(valores,y1);
```



Respuesta con amortiguamiento crítico para la función de transferencia con un valor de Wn = 2.

```
Wn = 2;
Xi = 1;
s = tf('s');
G = Wn^2 / (s^2 + (2*Xi*Wn)*s + Wn^2);
step(G);
hold on;
```


Step Response
Amplitude
Time (seconds)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5

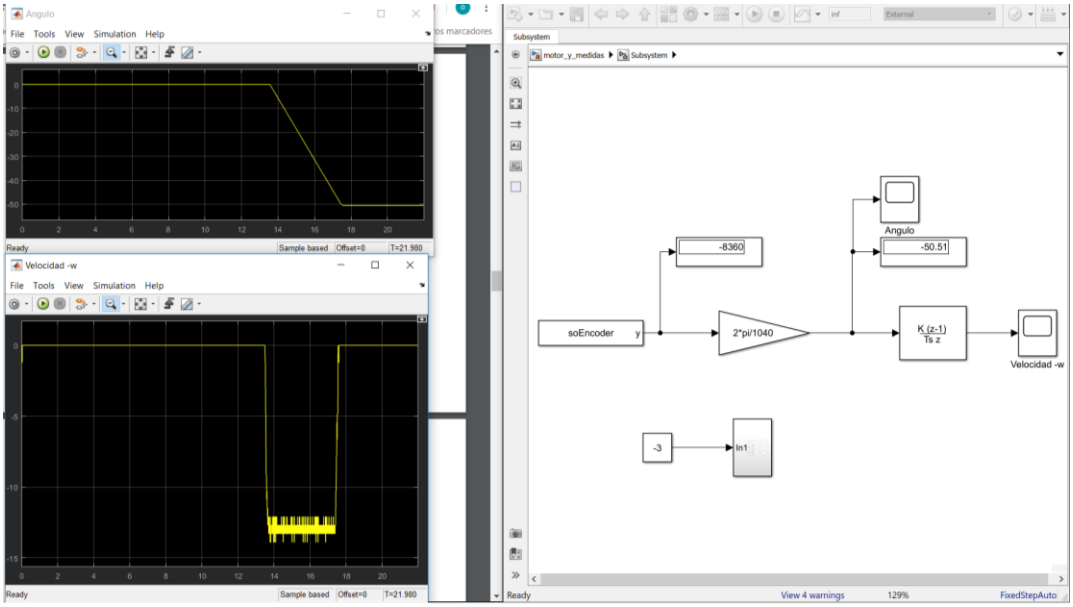
Prácticas de Laboratorio

Ejercicio 1.- Realizar un programa C++ para Arduino Mega que controle la velocidad y el sentido de giro del motor. Medir también simultáneamente el ángulo de giro y enviarlo por el puerto serie.

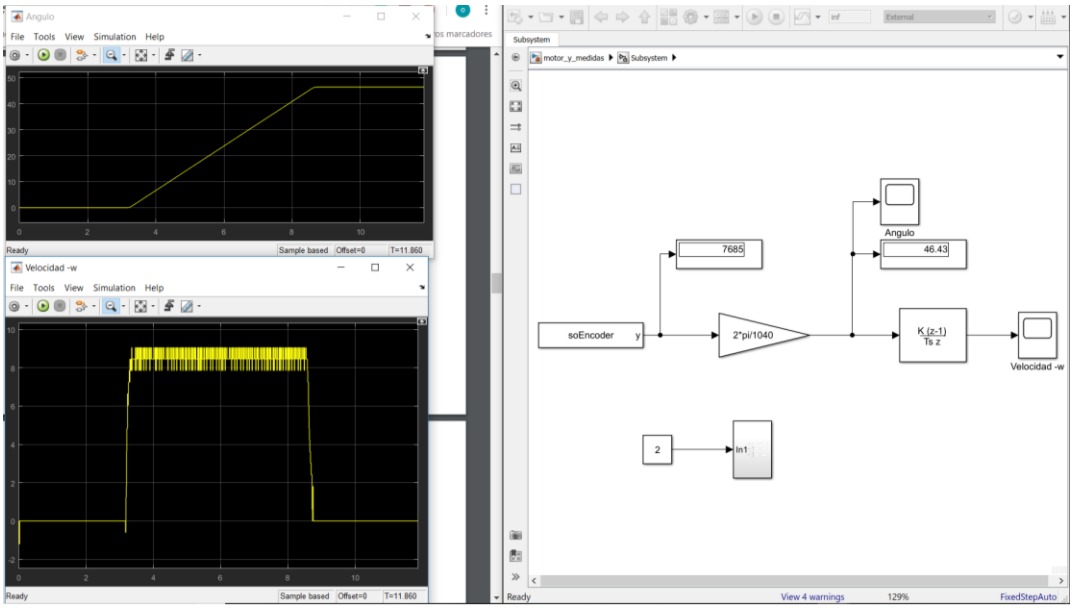
```
const int CHA =2;
const int CHB = 3;

const int PWM_Motor = 9;  //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
int Volt = 2;
int valorPWM8=0;
int pulsos = 0;
volatile double angulo = 0;

void Encoder_CHA(){
  if(digitalRead(CHA)==digitalRead(CHB)){
    pulsos--;
  }else{
    pulsos++;
  }
}

void Encoder_CHB(){
  if(digitalRead(CHA)!=digitalRead(CHB)){
    pulsos--;
  }else{
    pulsos++;
  }
}

void setup() {

  Serial.begin(9600);
  // put your setup code here, to run once:
  attachInterrupt(0, Encoder_CHA, CHANGE);
  attachInterrupt(1, Encoder_CHB, CHANGE);
  //Configuramos 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);

  angulo = float( pulsos * (2*PI/1040));
  if (Volt>0){
    valorPWM8 = byte(Volt*30);    // Volt positivo entre 0 y 8.5 voltios (255/8.5)
    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);
  }

}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.println(angulo);

}
```

Ejercicio 2.- Realizar un programa Simulink para medir y controlar en lazo abierto la posición del ángulo de giro en radianes y la velocidad de giro (es la derivada del ángulo de giro).

En este primer caso, el voltaje introducido, en función del cual varía la velocidad y el sentido de giro es de -3v, observando, así como el ángulo decrementa al activar el servo, donde podemos observar que tiene además cierto ruido con picos entorno al valor de velocidad angular al que tiende.

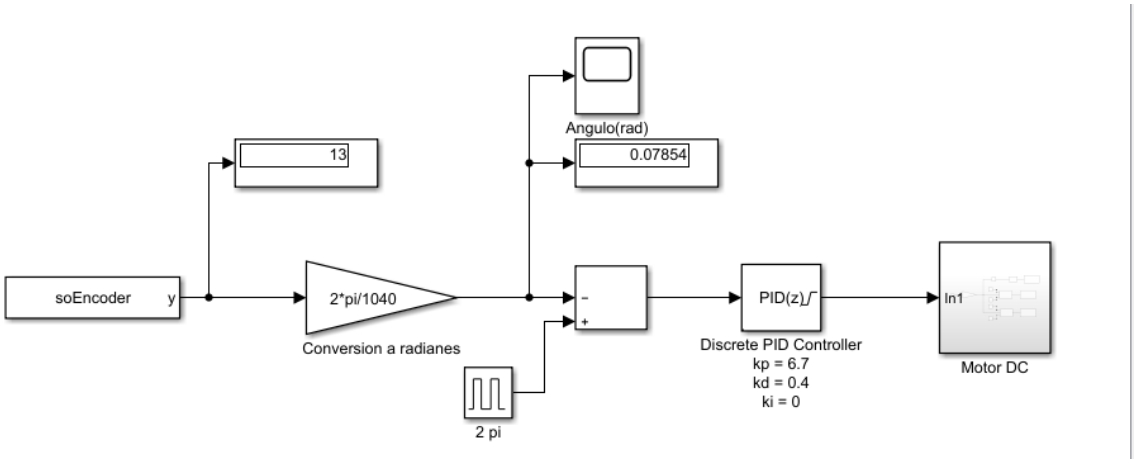


En este caso, el sentido de giro es positivo, observando al igual que antes como aumenta el giro con la activación del servo, pero en vez de decrementar el ángulo, se incrementa.

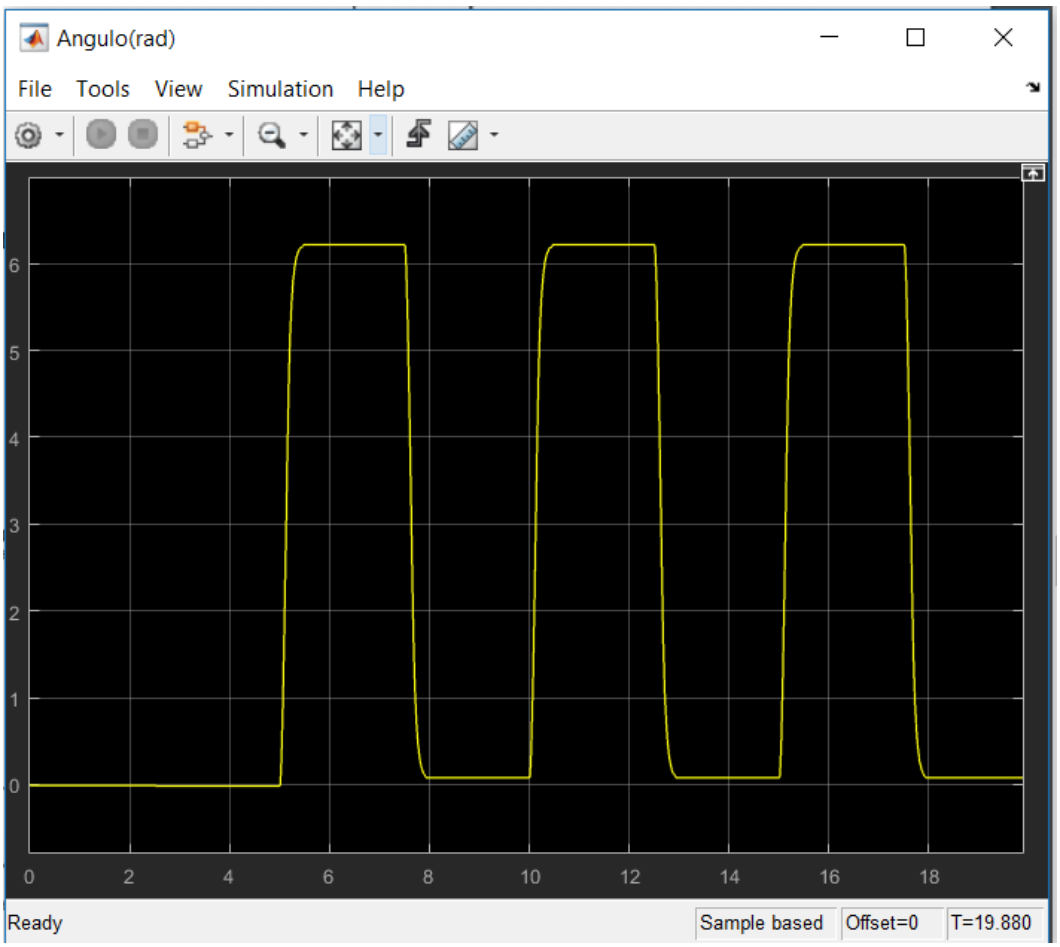


Ejercicio 3.- Realizar un control PID de la posición angular, con Simulink.

Realizamos el montaje del control PID de la posición angular de la siguiente forma:



El valor de consigna oscila se establece en 6.28 radianes, obteniendo así la siguiente respuesta:



Ejercicio 4.- Programar un control PID del ángulo de giro del motor en C++ para Arduino Mega.

```
#include <TimerOne.h>
#include <Arduino.h>

double Kp = 6;        //0.2
double Ki=  0.3;       //0.02
double Kd = 0.5;        //0.1

double Pn=0; //Salida del PID para muestra n
double Pn1=0;  // salida PID para muestra n-1

double En =0; // Error para muestra n
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;

const float Ts =0.005;
const float Tm =5000; // periodo de muestreo en seg  0.051
float consigna=6.28;  // angulo de consigna 2*PI radianes
volatile double angulo_n1 = 0;
volatile double angulo_n = 0; //angulo medido a partir de pulsos del encoder
volatile int pulsos = 0;//pulsos del encoder
double Volt = 2;  //voltaje inical en sentido horario
double valorPWM8 = 0; //

const int CHA =2;
const int CHB = 3;
const int PWM_Motor = 9;
const int AIN1 = 11;
const int AIN2 = 10;

void Encoder_CHA(){
  if(digitalRead(CHA)==digitalRead(CHB)){
    pulsos--;
  }else{
    pulsos++;
  }
}

void Encoder_CHB(){
  if(digitalRead(CHA)!=digitalRead(CHB)){
    pulsos--;
  }else{
    pulsos++;
  }
}

void control(){

  angulo_n = float( pulsos * (2*PI/1040));      // conversión de pulsos a radians
  En=consigna-angulo_n; // calculo error para la muestra actual
  Pn=Pn1+(q0*En+q1*En1+q2*En2);

  //Actualizamos para preparar el motor para la siguiente vuelta
  En2 = En1;
  En1 = En;
  Pn1 = Pn;

  if(Pn>8.5){
    Pn=8.5;
```

```
  ,
  if(Pn < -8.5){
    Pn = -8.5;
  }

  valorPWM8 = abs(Pn*30); //Cada voltio son 30 valores de pwm
  analogWrite(PWM_Motor, valorPWM8);
  if(Pn>0){
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, HIGH);
  }
  else {
    digitalWrite(AIN1, HIGH);  // se cambia el sentido de giro
    digitalWrite(AIN2, LOW);
  }

}

void defQvalues(){
  q0 = Kp+(Ki*Ts)/2+Kd/Ts;
  q1 = -Kp+(Ki*Ts)/2-(2*Kd)/Ts;
  q2 = Kd/Ts;
}

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

  pinMode(CHA,INPUT_PULLUP);
  pinMode(CHB,INPUT_PULLUP);
  pinMode(AIN1,OUTPUT);
  pinMode(AIN2,OUTPUT);
  pinMode(PWM_Motor,OUTPUT);

  digitalWrite(AIN1,HIGH);
  digitalWrite(AIN2,HIGH);

  //lectura de encoder por interrupciones
  attachInterrupt(0, Encoder_CHA, CHANGE);
  attachInterrupt(1, Encoder_CHB, CHANGE);

  defQvalues();

  Serial.begin(9600);

  Timer1.initialize(Tm);
  Timer1.attachInterrupt(control);
  Timer1.start();
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.println(angulo_n);
}
```

Los valores mostrados por el display para el ángulo son los incluidos en el archivo valoresAngulo.

0.00	1.34	3.67
0.01	1.49	3.79
0.04	1.64	3.93
0.07	1.79	4.04
0.19	2.11	4.27
0.27	2.28	4.37
0.36	2.44	4.47
0.45	2.60	4.57
0.68	2.92	4.74
0.80	3.08	4.82
0.92	3.24	4.89
1.06	3.38	4.96

5.10	6.12	6.23
5.16	6.13	6.23
5.21	6.13	6.23
5.27	6.14	6.23
5.37	6.15	6.23
5.42	6.16	6.23
5.46	6.16	6.23
5.50	6.17	6.23
5.58	6.18	6.23
5.61	6.19	6.23
5.64	6.19	6.24
5.68	6.19	6.24
5.74	6.19	6.24
5.76	6.19	6.24
5.79	6.20	6.24
5.81	6.20	6.24
5.85	6.22	6.24
5.87	6.22	6.24
5.89	6.22	6.24
5.91	6.23	6.25
5.94	6.23	6.25
5.96	6.23	6.25
5.98	6.23	6.25
5.99	6.23	6.26
6.02	6.23	6.26
6.03	6.23	6.27
6.04	6.23	6.27
6.05	6.23	6.27
6.07	6.23	6.27
6.08	6.23	6.27
6.10	6.23	6.27
6.10	6.23	

Ejercicio 5.-

1. Diseñar un controlador PD de posición angular del motor, con sobrepasamiento máximo Mp= 0.1 y tiempo de asentamiento ts= 0.7 seg.

2. Utilizar los valores Kp y Kd obtenidos, y la función de transferencia del motor para simular el control angular del motor mediante Simulink. Observar el valor máximo del voltaje de salida del controlador PD. A continuación, ajustar los valores Kp y Kd con el "autotunning" del módulo PID de Simulink.

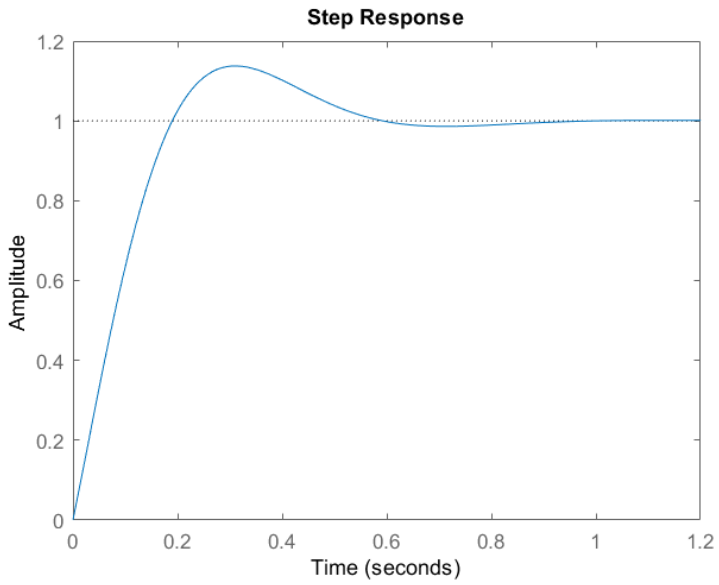
3. Volver a probar el control PD digital real con Simulink (Ejercicio 3) con dichos valores. Comparar la respuestas simulada y real.

Diseñamos el controlador PD de la siguiente forma:

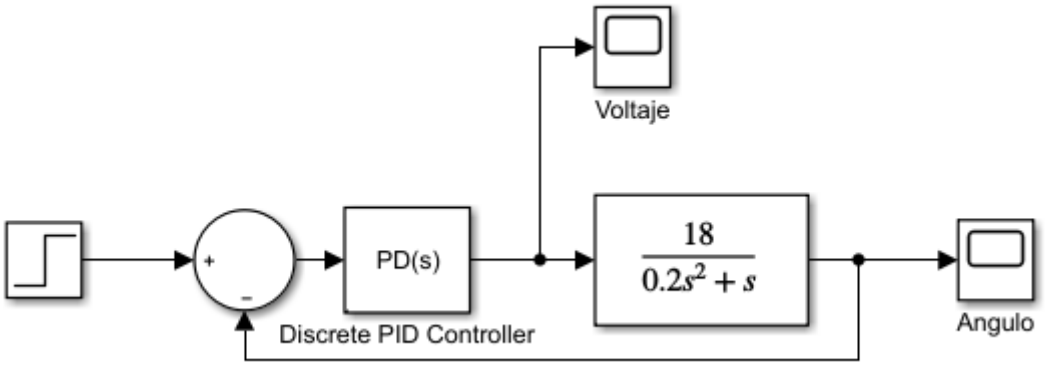
```
Ts = 0.01;
Mp = 0.1;
ts = 0.7;
K = 18;
T = 0.2;
xi = abs(log(Mp)/sqrt(pi^2+log(Mp)^2));
wn = 4/(xi*ts);
kd = (2*xi*wn*T-1)/K;
kp = (wn^2)*T/K;
fdtPD = tf([kd,kp],1);
sysc = tf([K],[T 1 0]);
fdtlazoc = feedback(fdtPD*sysc,1);
step(fdtlazoc);
```

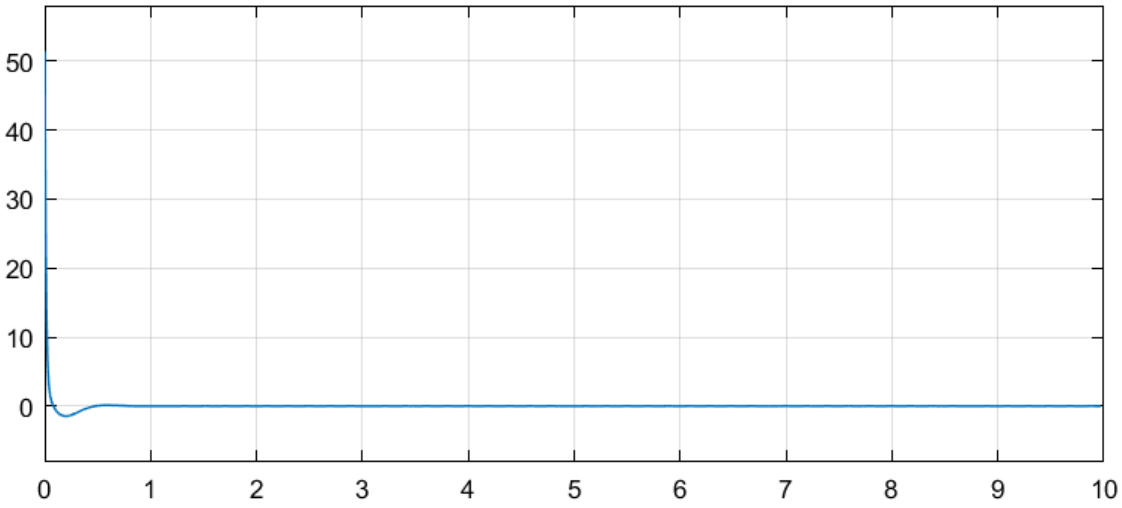
Obtenemos los siguientes valores y respuesta:

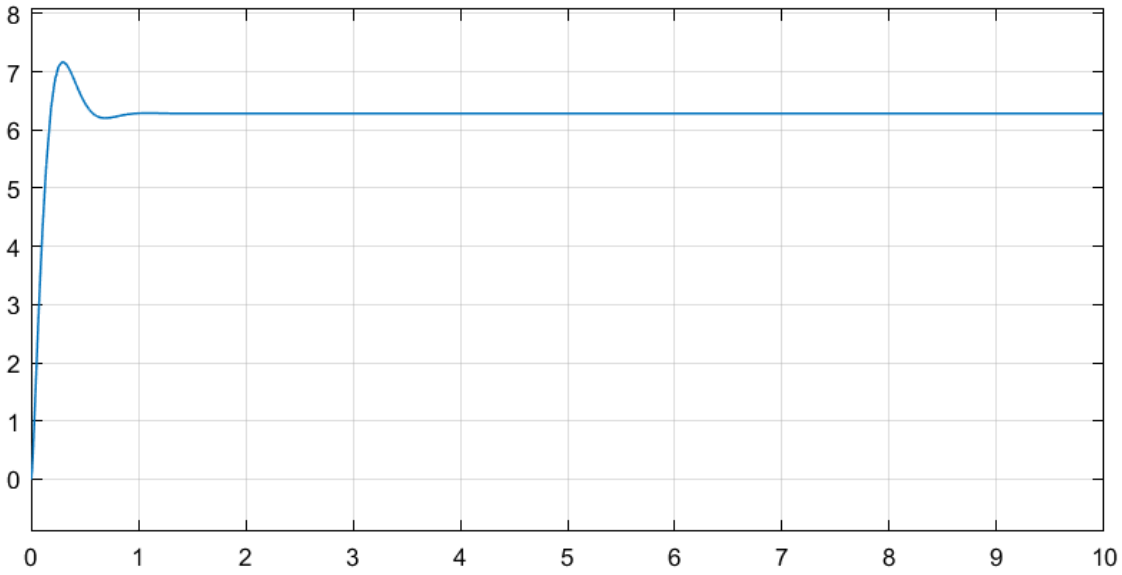
kd	0.0714
kp	1.0382



Simulamos la respuesta del controlador para los mismos valores de constantes obtenidos en el ejercicio anterior en Simulink:

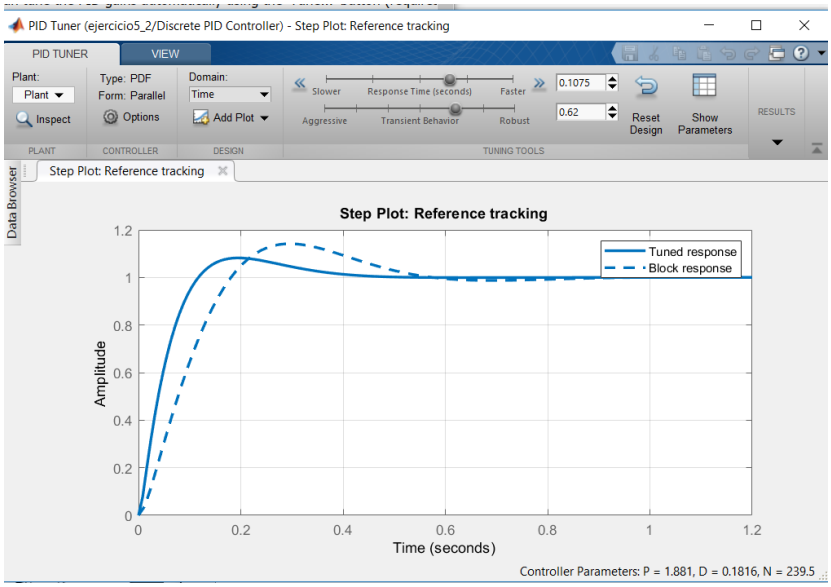


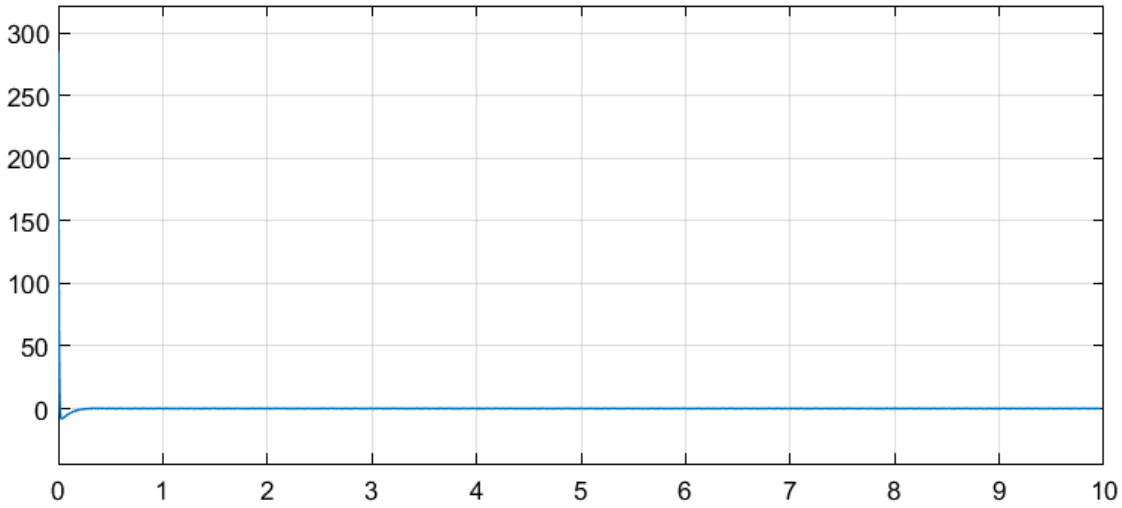


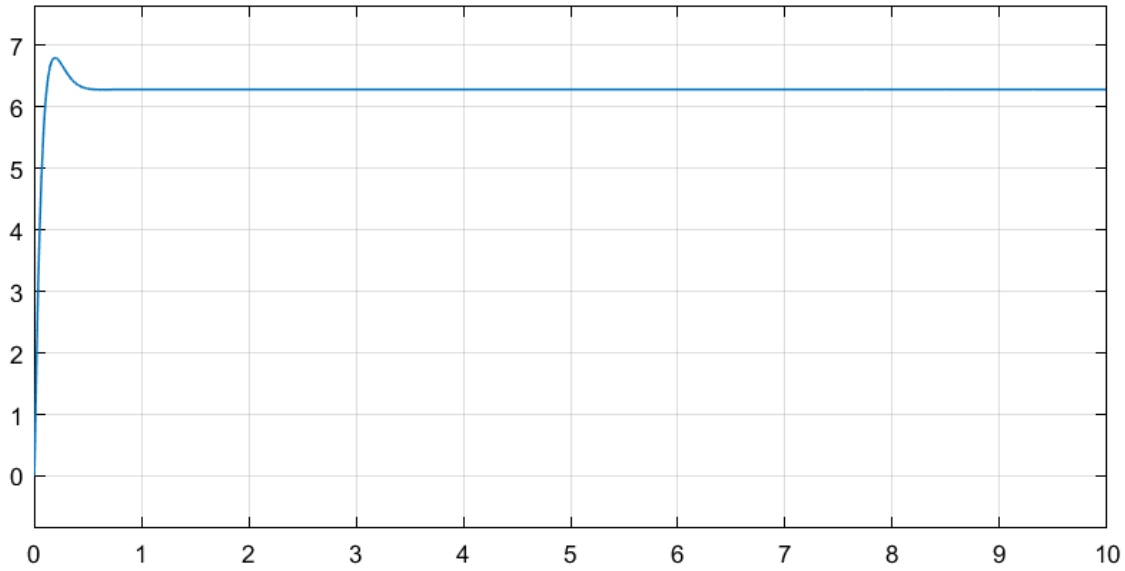


En este caso podemos ver como al no tener el valor de voltaje acotado, se sobredispara por encima de 50v.

Realizamos ahora el ajuste con autotunning del módulo PID de Simulink:

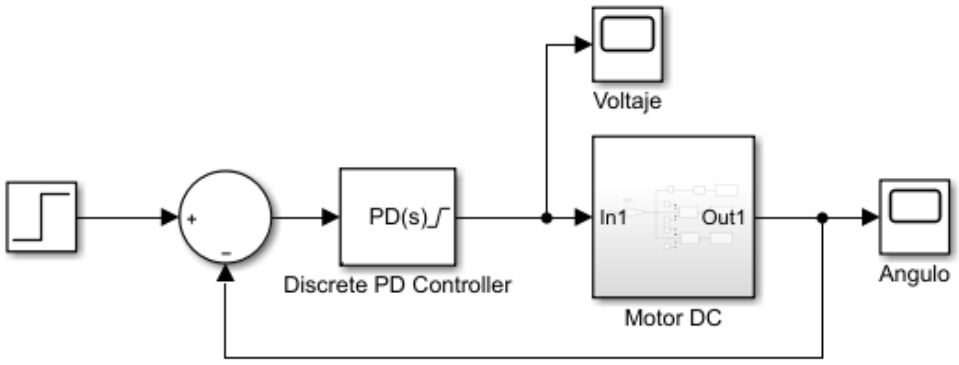


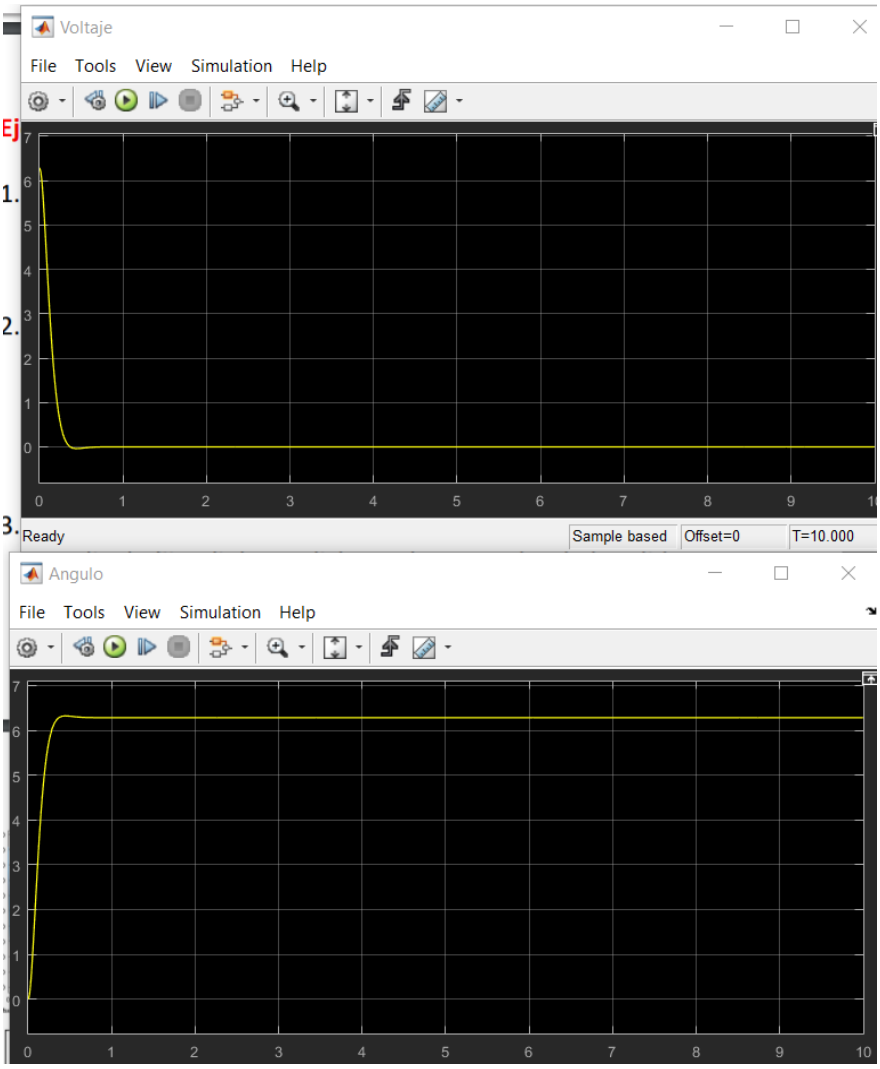




En este caso, la respuesta es más rápida y más controlada, pero para ello incrementa la respuesta de voltaje muy notablemente, pasando de 50 a más de 250 v.

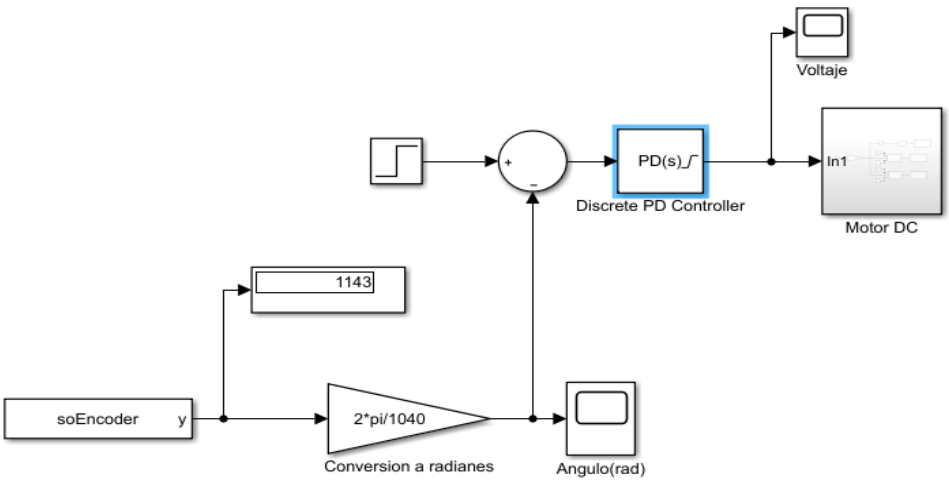
Para controlar este efecto, acotamos los valores del PD entre 8.5 y -8.5v obteniendo los siguientes resultados:

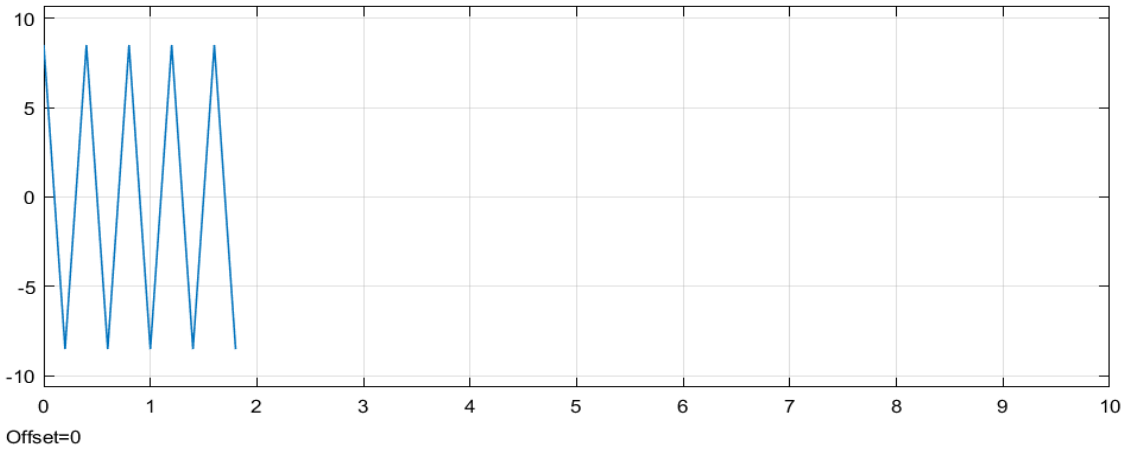


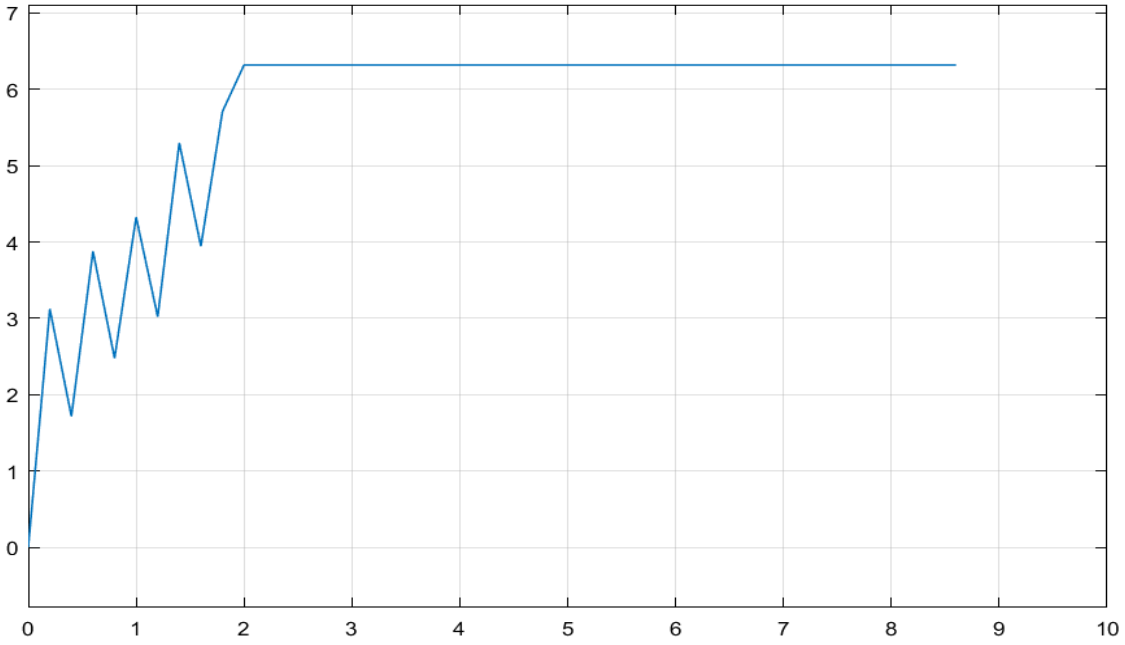


Como se puede observar, la respuesta es mucho mejor que en los anteriores casos, con un menor sobredisparo, menor tiempo en alcanzar el valor de consigna, y un valor de voltaje inferior a 7v.

Por último simularemos el PD digital real con Simulink y compararemos la respuesta:







Podemos observar como alcanza el valor de consigna deseado a los 2 segundos, comprobando simultaneamente el valor del voltaje, en orden inverso para controlar el valor de consigna.

Esta respuesta es pero que la obtenida con las anteriores simulaciones, ya que tardamos más en alcanzar el valor de consigna, y las oscilaciones son notables.

Ejercicio 6.-

Control de velocidad angular del motor:

6.1 Simulación con Simulink: Realizar una simulación de control PI de velocidad angular con Simulink. La velocidad angular de consigna será de 10 rad/seg.

a) Probar inicialmente con Kp= 0.05; Ki=0.2. b) Obtener los valores Kp y Kd con Matlab (TuningPI) c) Posteriormente, utilizar el ajuste automático ("tune") del módulo PID.

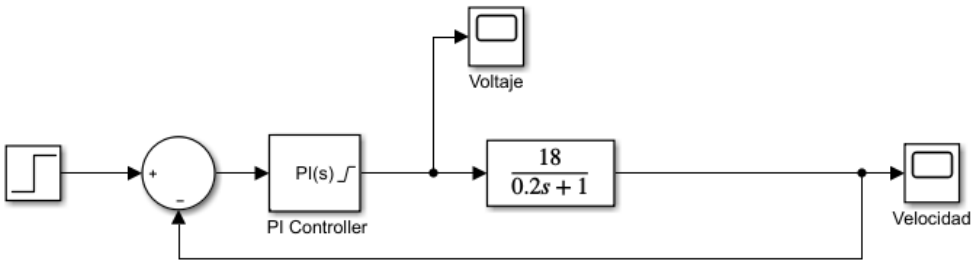
La f.d.t con respecto a la velocidad angular es:

$$\frac{\omega(s)}{V(s)} = \frac{K'}{Ts+1} = \frac{18}{0.2s+1}$$

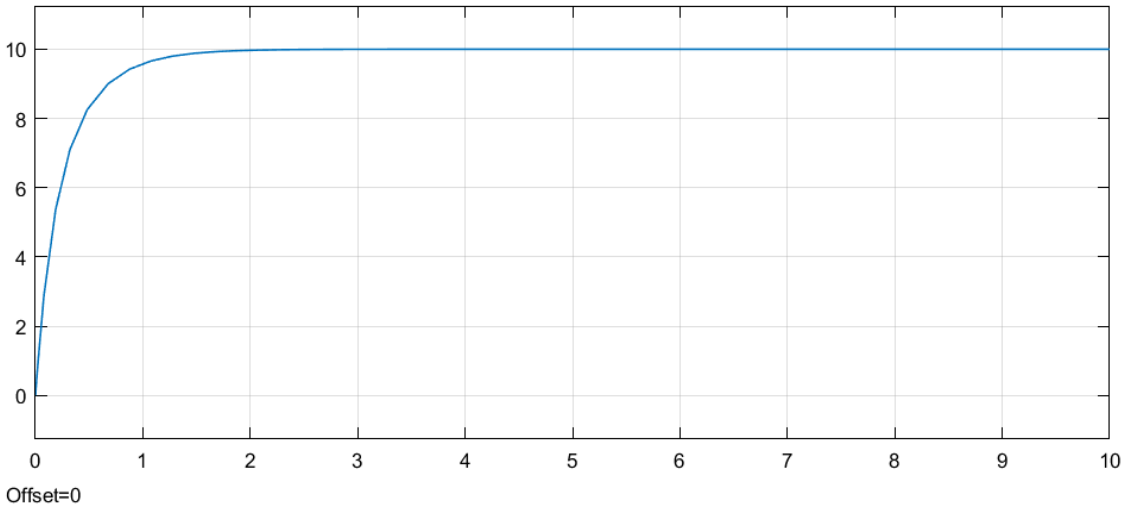
6.2 Control PID real de velocidad con Simulink: Realizar un control PI real de velocidad angular del motor, con Simulink. (Usar el filtro Pasa-baja).

6.3 Realizar el mismo control PID de velocidad en C++.

Ponemos el valor de consigna en el escalón a 10 como valor final y usamos la función de transferencia dada, empleando como valores del PI los proporcionados en el enunciado:



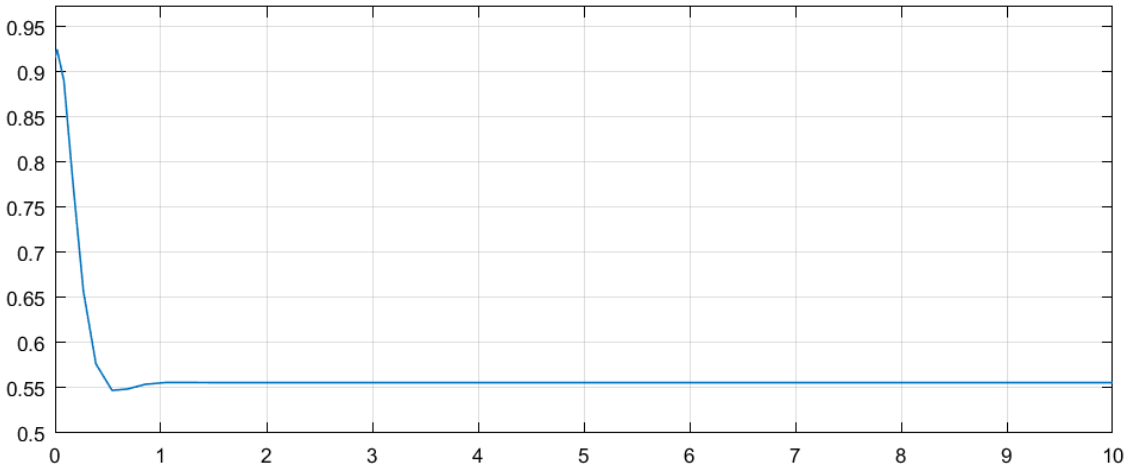
Obtenemos la siguiente respuesta para la velocidad:

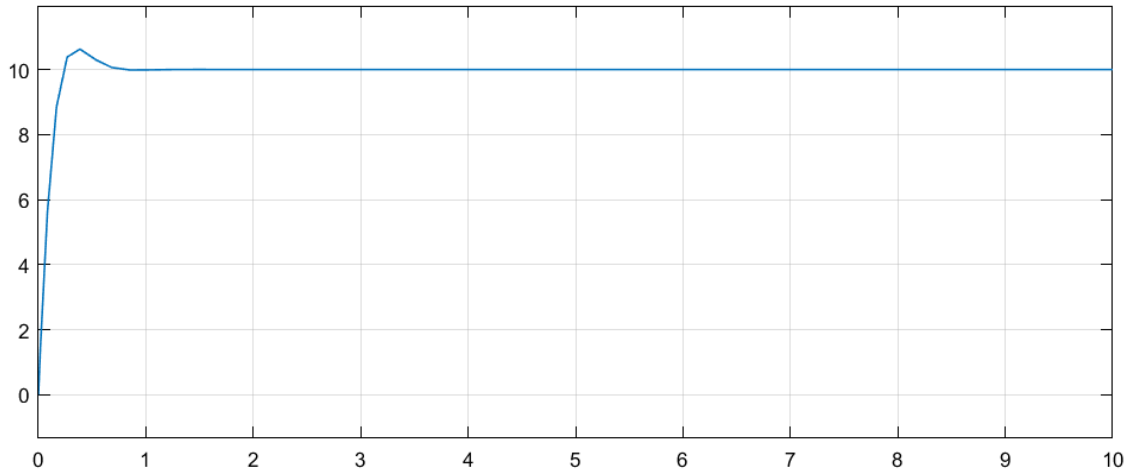


Obtenemos ahora los valores de kp y ki con Matlab empleando TuningPI:

```
s = tf('s');
sys = 18/(0.2*s + 1);
fdt_PID = pidtune(sys,'pid',10);
fdt_PID
```

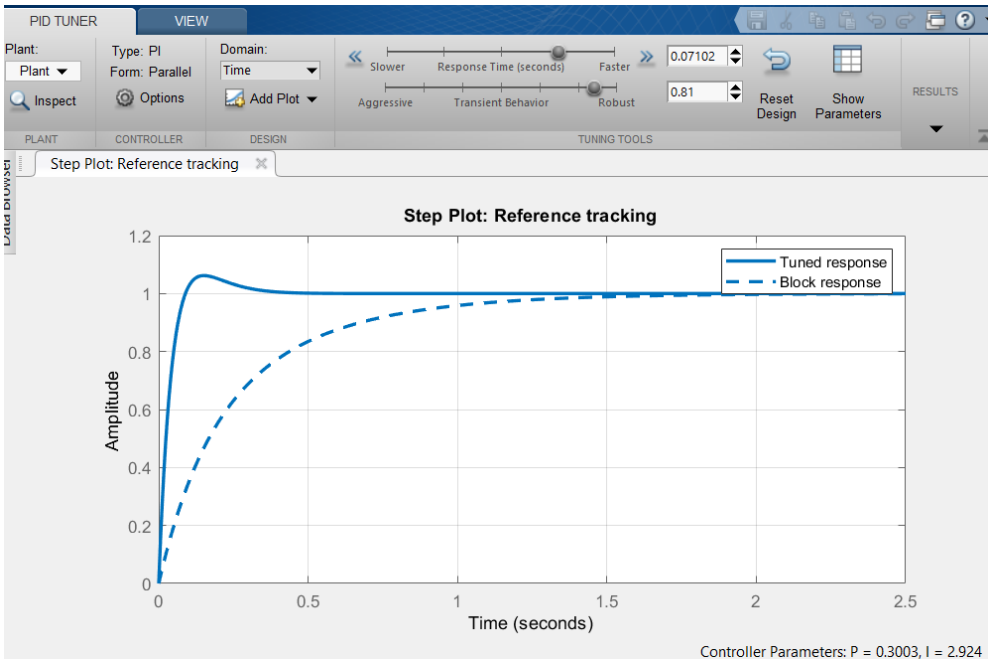
```
>> RE5

fdt_PID =

                1
  Kp + Ki * ---
                s

  with Kp = 0.0915, Ki = 0.84
```



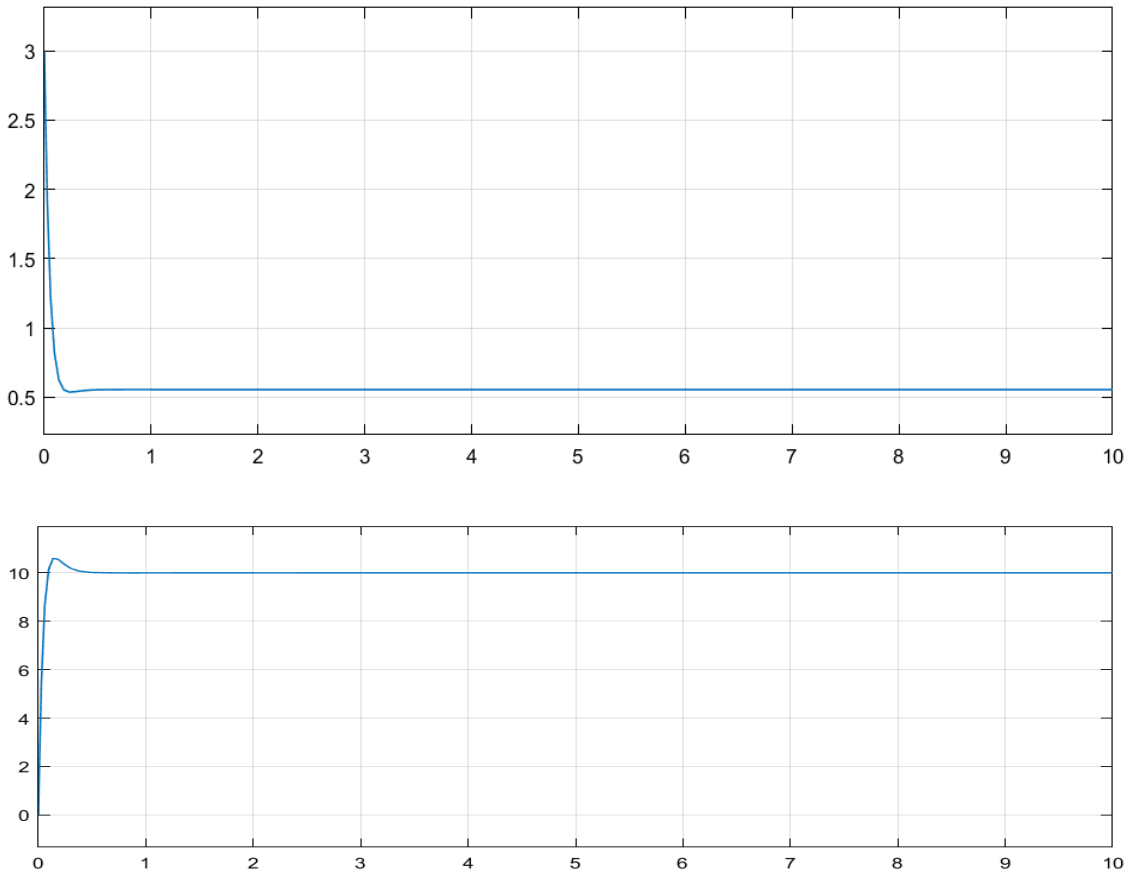


La respuesta obtenida en este caso es peor que la obtenida para el anterior ajuste.

Ajustamos ahora los valores con la función tune:

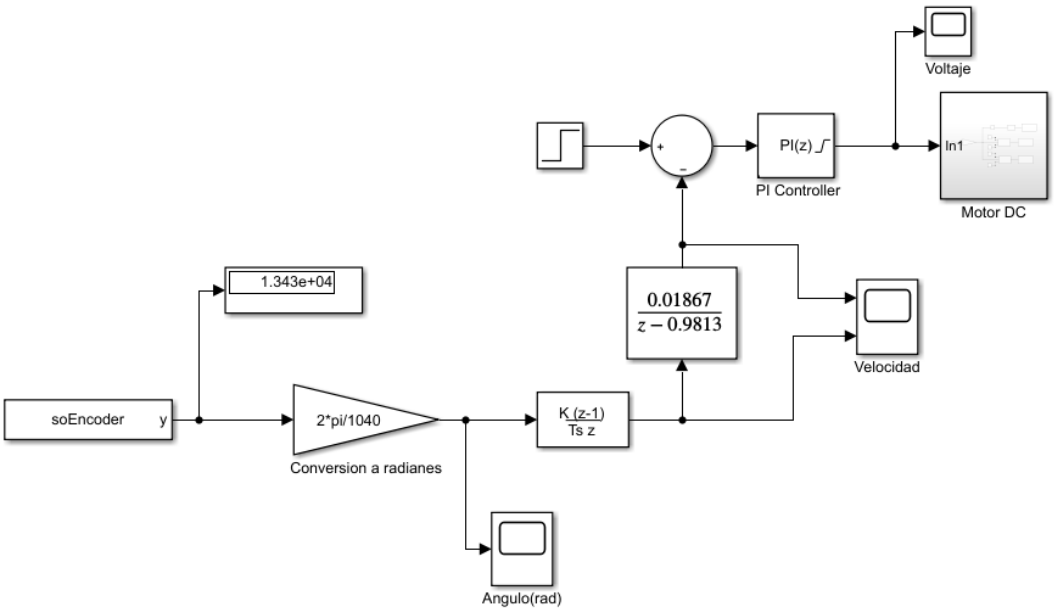


Obtenemos la siguiente salida:

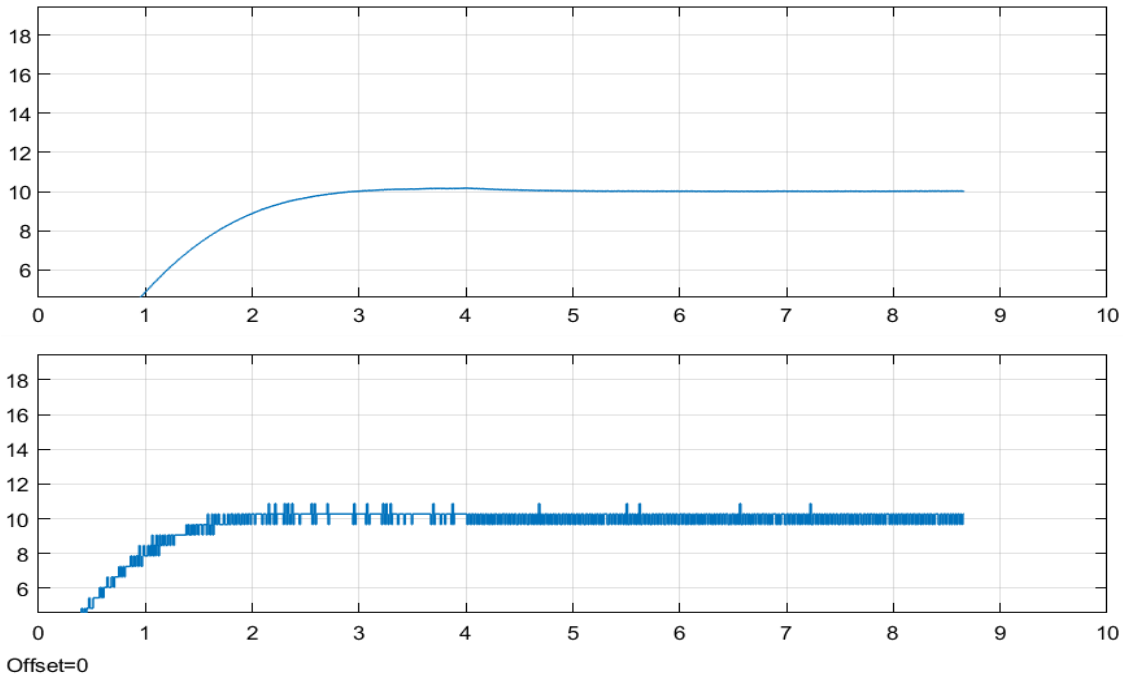


Obtenemos una respuesta más rápida, aunque menos controlada con un mayor sobrepasamiento.

Para el siguiente ejercicio probamos el control PI de velocidad con el motor, empleando un filtro de Pasa-baja:



Mostramos la salida de la velocidad angular antes y después de la aplicación del filtro para poder observar los efectos de este sobre la salida:



Podemos observar la notable diferencia, reduciéndose el ruido tras el efecto del filtro de Pasa-baja.

Probamos ahora en C++:

Para este caso, nos basamos en el esqueleto del código del ejercicio 4 y lo único que tendremos que modificar serán las siguientes líneas del módulo de control:

```
void control(){

  angulo_n = float( pulsos * (2*PI/1040));      /
  angulo_n = alpha*angulo_n+(1-alpha)*angulo_n1;
  velocidad_angular = (angulo_n - angulo_n1)/Ts;
  angulo_n1 = angulo_n;
  En=consigna-velocidad_angular; // calculo erro
  Pn=Pn1+(q0*En+q1*En1+q2*En2);
```

Ahora el valor de consigna es la velocidad angular deseada, que calculamos derivando el incremento angular respecto del tiempo.

La respuesta que hemos obtenido es la siguiente: