
# UNIVERSIDAD NACIONAL DE AVELLANEDA

DPTO. DE TECNOLOGÍA Y ADMINISTRACIÓN

**INGENIERÍA EN INFORMÁTICA**

Sistemas de Control Automático

## Trabajo de Laboratorio 2

**Control Lineal de un Ascensor**

ESTUDIANTES

**Massimino, Matías**

**Colli, Alejandro**

**Eroles, Pedro**

DOCENTE

**Guillermo Federico Caporaletti**

2do Cuatrimestre 2023

# Índice

# Objetivo

Nuestro objetivo principal es controlar el sistema físico mediante la acción de un sistema de control de tipo Proporcional Derivativo. Se continuó utilizando el equipo utilizado en la práctica anterior.

La experiencia debe ser registrada para poder medir la respuesta del sistema y elegir los parámetros más adecuados para configurar nuestro controlador teniendo en cuenta: tiempo de respuesta, precisión y sobrevalor. Debemos evaluar su comportamiento real a partir de la experiencia.

Para eso, se va a proceder a modificar el código existente que se utilizó en la anterior presentación, agregando el control PWM, obteniendo el tiempo de derivación (TD) y la constante de proporcionalidad o ganancia (KP).

# Introducción

En este trabajo vamos estar utilizando un sistema de lazo cerrado o realimentado, en el cual exploraremos los principios teóricos detrás del control proporcional derivativo y su implementación practica para controlar nuestro sistema físico. Se llevarán a cabo experimentos con distintas pruebas y análisis para evaluar la respuesta transitoria, estabilidad, sobrevalor y eficiencia del sistema.

Posteriormente, en XCos se harán las simulaciones correspondientes, las cuales se ira variando tanto variando la ganancia como el derivativo, encontrando así la ganancia que más se adecue a nuestras necesidades.

Por último, con estos valores obtenidos, se procederá a realizar la experiencia de laboratorio.

# Modelo del Ascensor

Para la resolución del siguiente Trabajo Práctico, lo primero que se debe realizar es un diagrama en bloques que represente al sistema con el cual se trabaja; como así también la identificación de la planta, la variable controlada y la variable manipulada.

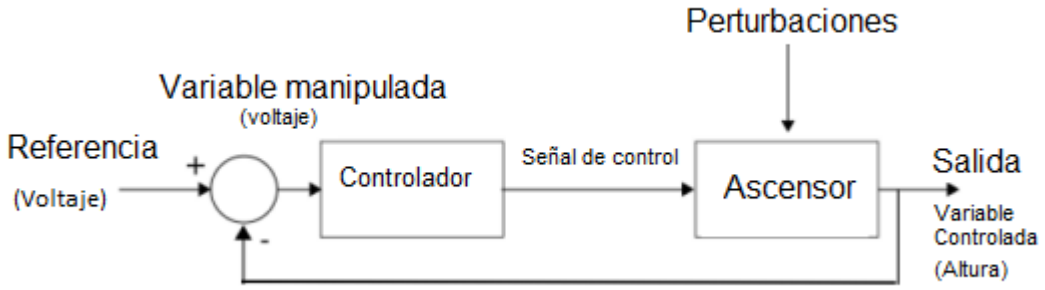


*Figura 1: Diagrama en bloques del sistema.*

El sistema que estamos controlando se puede representar en un diagrama en bloques, como se observa en la figura anterior. En nuestro sistema, la planta va a ser el ascensor, el cual va a contener un motor y un optoacoplador; la variable controlada va a ser la altura del ascensor; y finalmente la variable manipulada va a ser el voltaje que reciba el motor para subir el ascensor.

En nuestro sistema, la acción de control es del PD, en el cual se utiliza todo el rango de la señal manipulada. En nuestro caso de $0$ a $7V$. para controlar la velocidad de la planta.

Esto se hace multiplicando la señal de error por un coeficiente de proporcionalidad ($Kp$) y obteniendo con esto la corrección del sistema.

## Simulaciones

Antes de comenzar con el desarrollo, realizamos algunas simulaciones utilizando Scilab. Como ya teníamos el modelo desarrollado en la práctica anterior, utilizamos la transferencia obtenida para realizar las simulaciones.

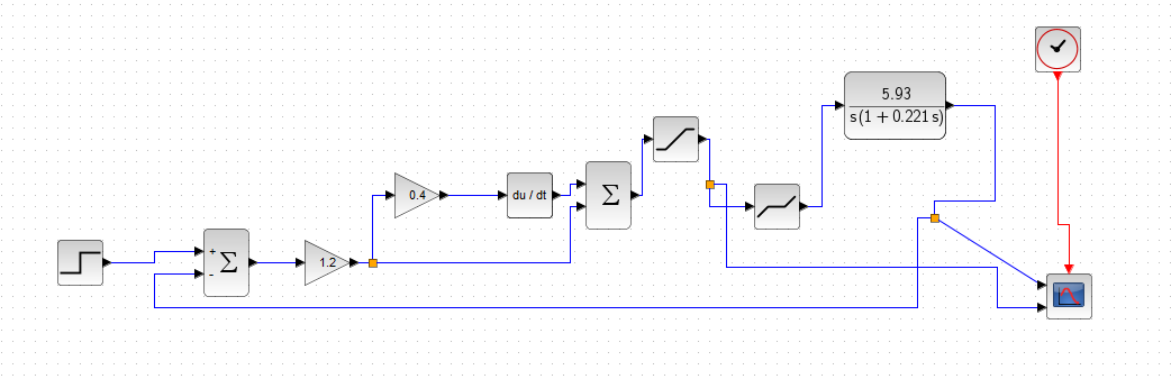


*Figura 2: Simulación control PD frente al escalón*.

En esta simulación, lo que tenemos es el control proporcional derivativo del ascensor frente a un escalón. Tuvimos en cuenta la zona muerta, el rozamiento y la saturación, por eso es que utilizamos una ganancia de 1,2.

Los resultados de la simulación fueron los siguientes.

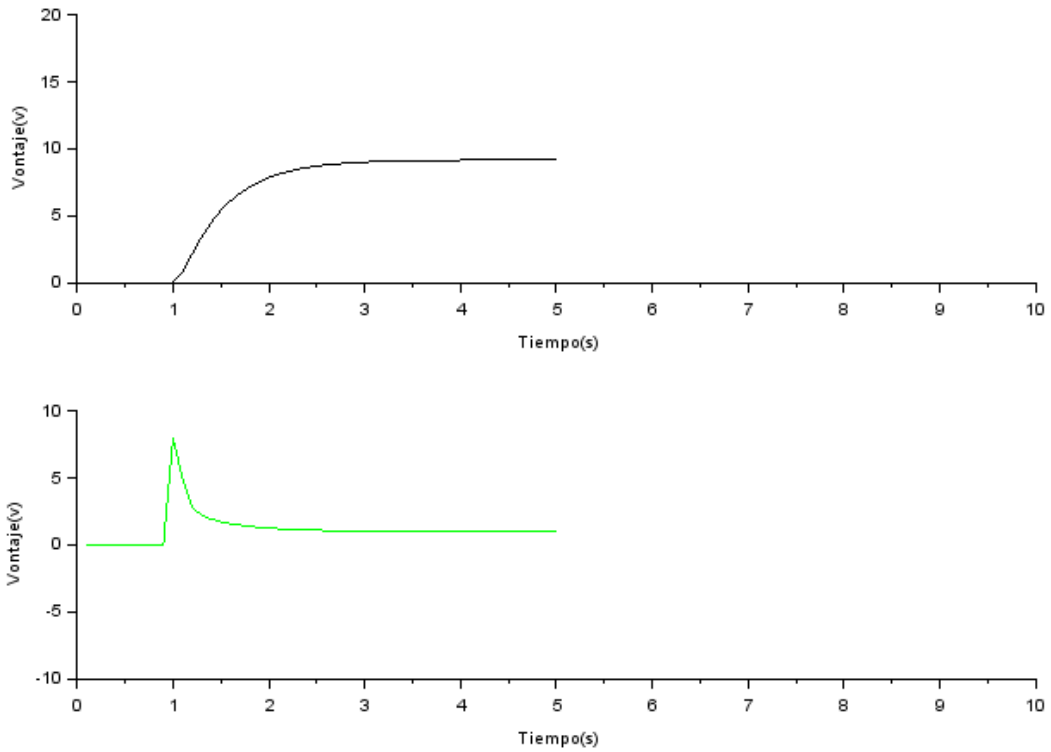


*Figura 3: Respuesta al escalón*

Obtuvimos un tiempo de establecimiento de $1{,}55\,s$ y un tiempo de crecimiento de $1{,}04\,s$. El retardo es de $0{,}4$

```
1  //·Ejemplo·de·control·integral
2
3  //·Función·de·transferencia·y·espacio·de·simulación
4  s=%s
5  t=0:0.01:10
6  G=·5.93/s/(1+0.221*s)
7  G=syslin('c',G)
8
9  KP·=·0.6
10
11
12
13 close()
14 close()
15 close()
16 figure(1)
17 evans(G*(1+s*0.4),10)
18 //evans(G*(1+s*0.2),10)
19 plot([0,-5],·[0,5],"--")
20
21
22 //·Pruebas·variando·el·TD
23 figure(2)
24 for·TD=[0,·0.1,0.2,0.4]
25 ····PD·=·KP·*·(1+·s*TD)
26 ····LC=(PD*G)/(1+PD*G)
27 ····Y=csim('step',t,LC);
28 ····plot(t,Y);
29 end
30
31 //Respuesta·frente·a·una·entrada·rampa
32
33 rampa=t*5;
34 for·i=·201:1:1001
35 ····rampa(i)=10;
36 end
37
38 figure(3)
39 plot(t,rampa,"r");
40 for·TD=[0,0.1,0.2,0.4]
41 ····PD·=·KP·*·(1+·s*TD)
42 ····LC=(PD*G)/(1+PD*G)
43 ····Y=csim(rampa,t,LC);
44 ····plot(t,Y);
45 end
```

*Figura 4: Script para simulaciones*

En esta segunda simulación lo que tenemos es un Script con el control proporcional derivativo del Ascensor.

Definimos la función de transferencia, y un $Kp$ de $0{,}6$. La figura 1 nos muestra el Evans.

La figura 2 nos muestra algunas pruebas variando el $TD$ frente a una entrada escalón.

La figura 3 nos muestra la respuesta frente a una entrada rampa.

Como resumen, este código se utiliza para realizar simulaciones y análisis de un controlador proporcional derivativo en un sistema representado por la función de transferencia. Analizamos la respuesta para varios valores de TD y evaluamos el comportamiento frente a entradas de escalón y rampa.

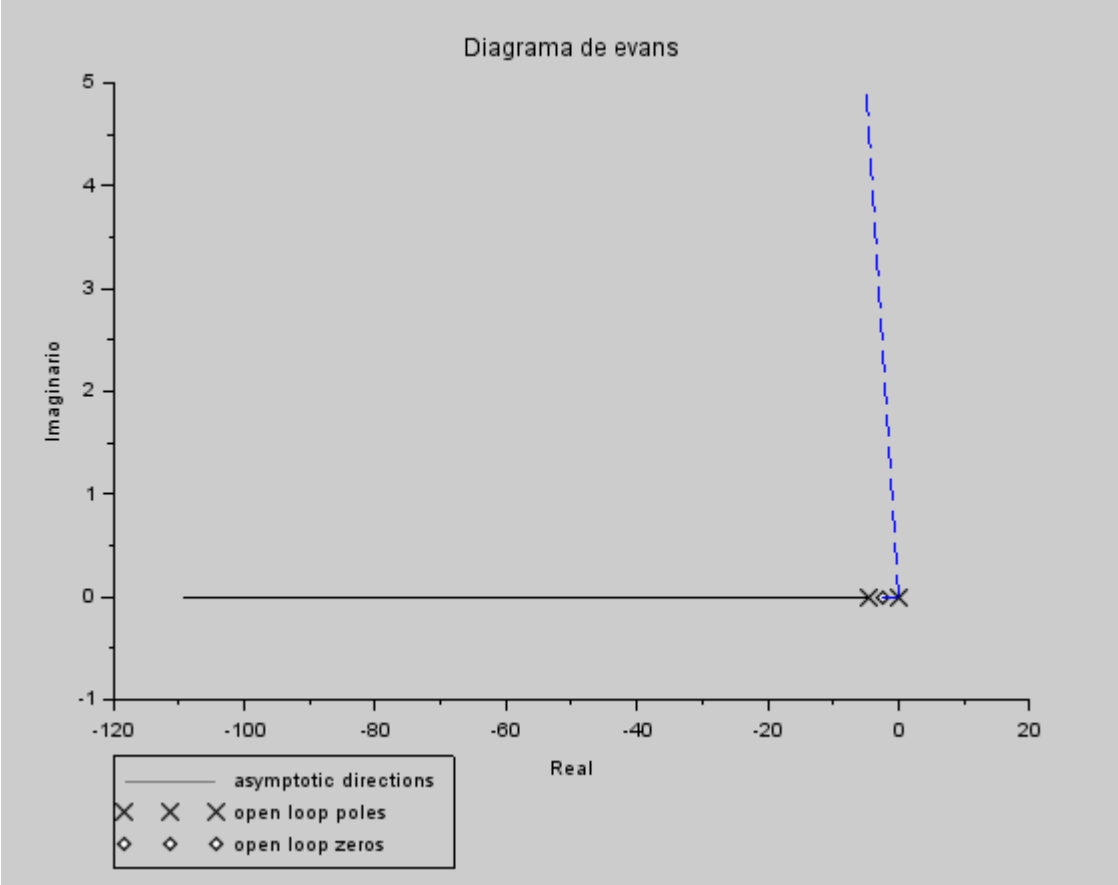


*Figura 5: Diagrama de Evans*

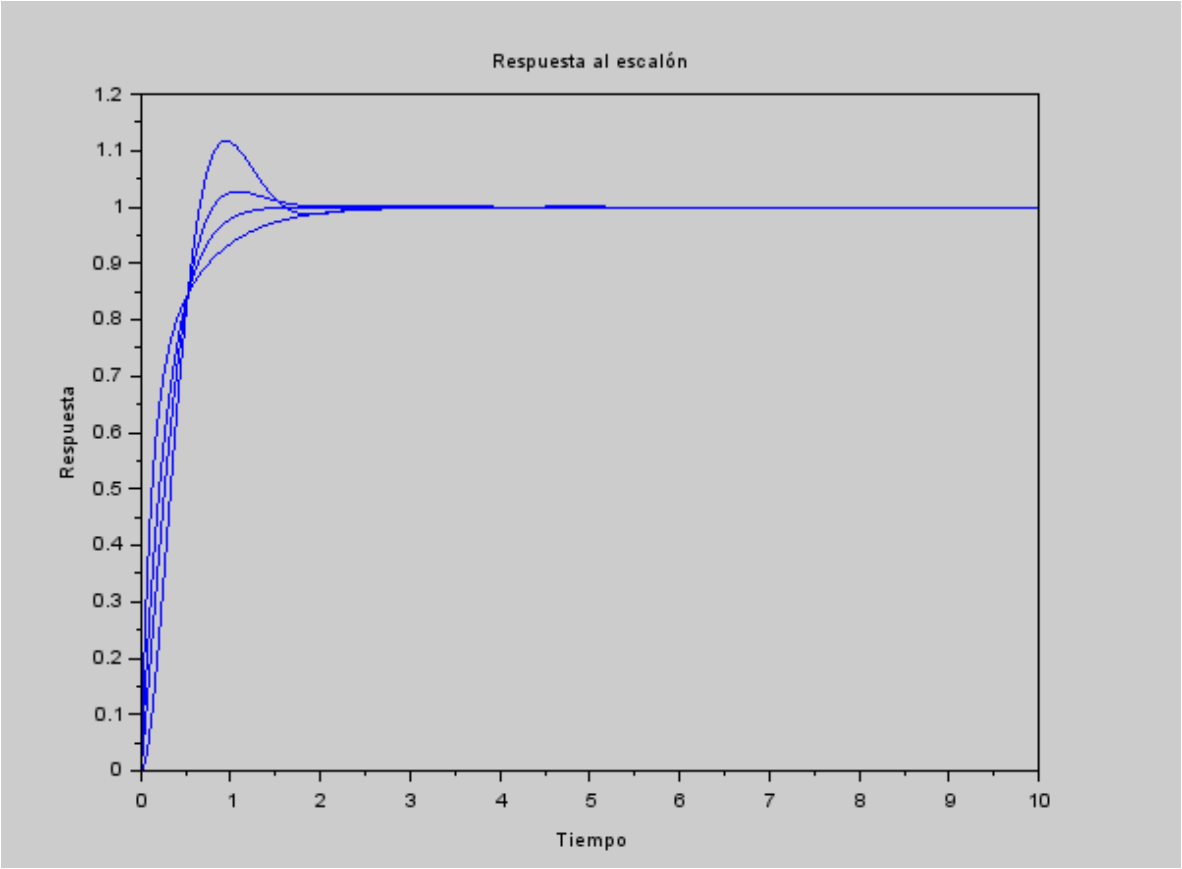


*Figura 6: Respuesta al escalón*

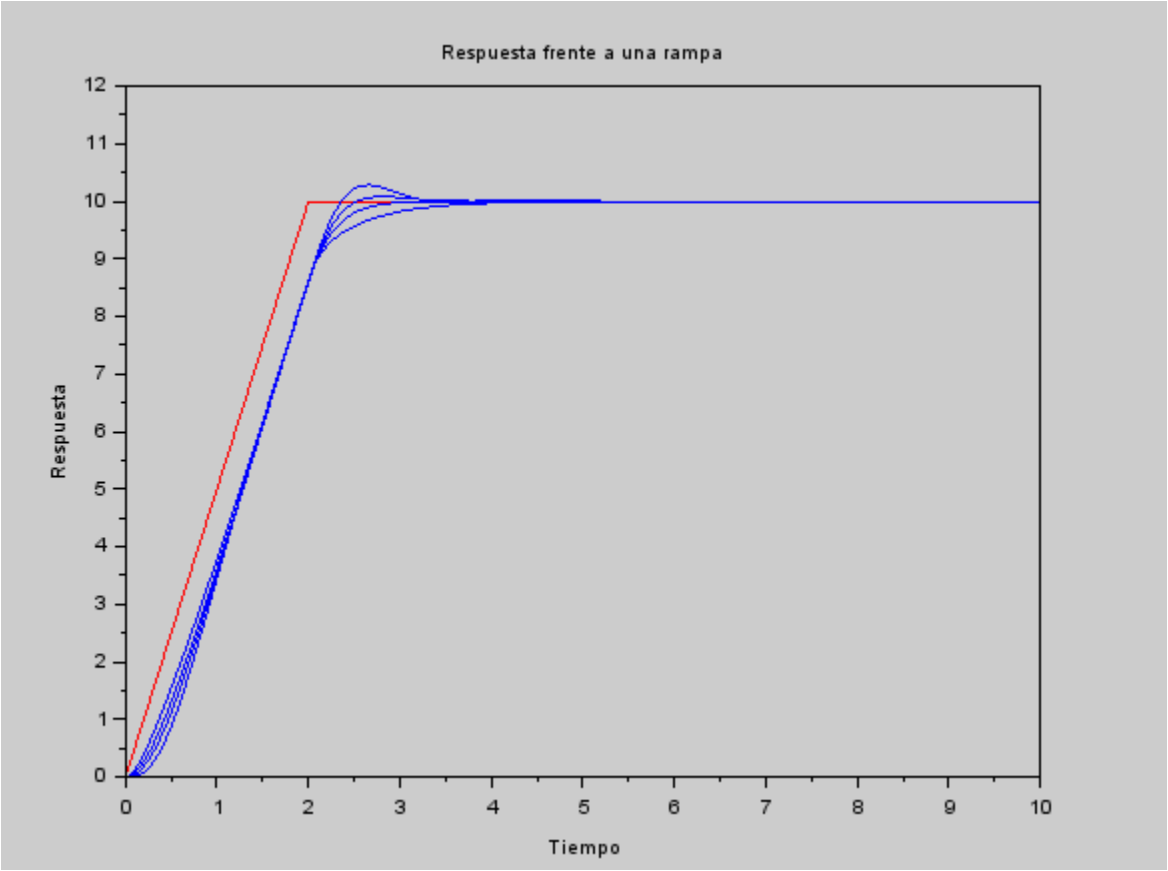


*Figura 7: Respuesta frente a una rampa*

# Equipamiento y componentes utilizados

Para esta práctica de laboratorio utilizamos los mismos componentes que en la práctica anterior, pero esta vez sumando el uso del PWM.

- Puente H
- Fuente de Tensión
- Pulsador
- Arduino Uno
- Optoacoplador

## PWM

PWM son las siglas de Pulse Width Modulation (Modulación por ancho de pulso). Para transmitir una señal, ya sea analógica o digital, se debe modular para que sea transmitida sin perder potencia o sufrir distorsión por interferencias.

PWM es una técnica que se usa para transmitir señales analógicas cuya señal portadora será digital. En esta técnica se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

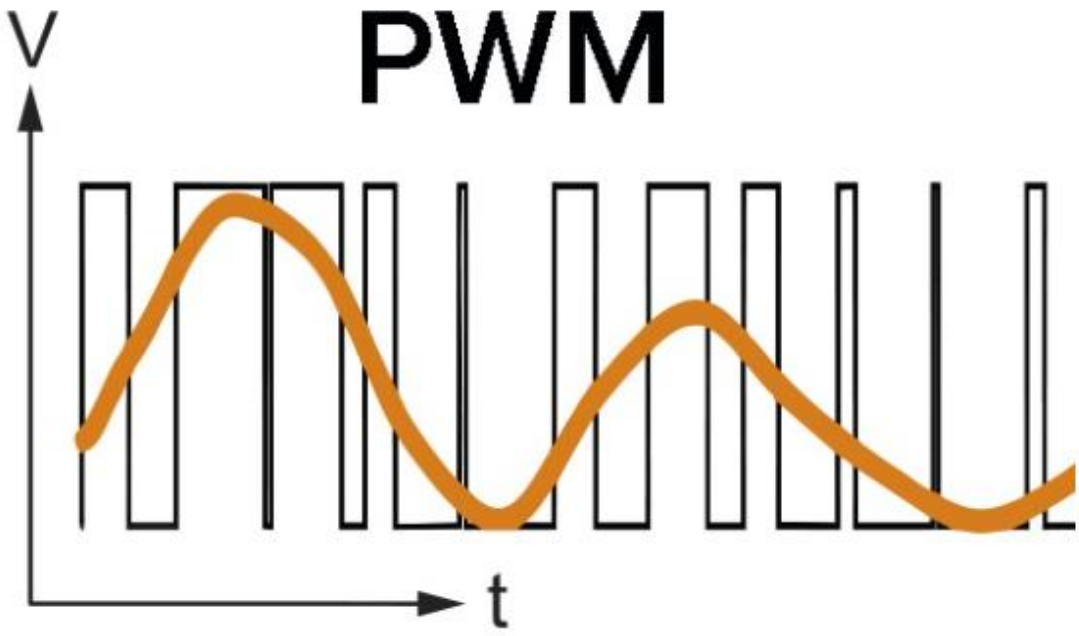


*Figura 8: PWM*

Esta modulación es muy usada para controlar la cantidad de energía que se envía a una carga, es una técnica utilizada para regular la velocidad de giro de los motores, regulación de intensidad luminosa, controles de elementos termoeléctricos o controlar fuentes conmutadas entre otros usos.

La mayoría de los automatismos, incluido Arduino, no son capaces de proporcionar una señal analógica. Sólo pueden proporcionar una salida digital de -Vcc o Vcc. (por ejemplo, $0V\ y\ 5V$). Entonces, para conseguir una señal analógica, la mayoría de los automatismos usan PWM. Se usa esta técnica porque como se ve en los ejemplos anteriores, no siempre quieres un valor digital de la señal (ON/OFF), si no que necesitaremos proporcionar un valor analógico de tensión que usarán para las aplicaciones deseadas.

En nuestro caso, utilizamos el PWM para controlar la potencia aplicada al motor, cuando el ascensor está subiendo ajustamos la potencia aplicada al motor para moverlo en función de la altura objetivo, y cuando el ascensor está bajando se utiliza de manera similar.

# Desarrollo

Como ya teníamos la implementación realizada de la práctica anterior, lo único que nos quedaba hacer era volver a realizar las conexiones, y agregar la conexión para el funcionamiento del PWM.

Esta vez, pudimos utilizar el pulsador para detener el sistema, cosa que en la anterior practica no habíamos podido realizar por falta de tiempo.

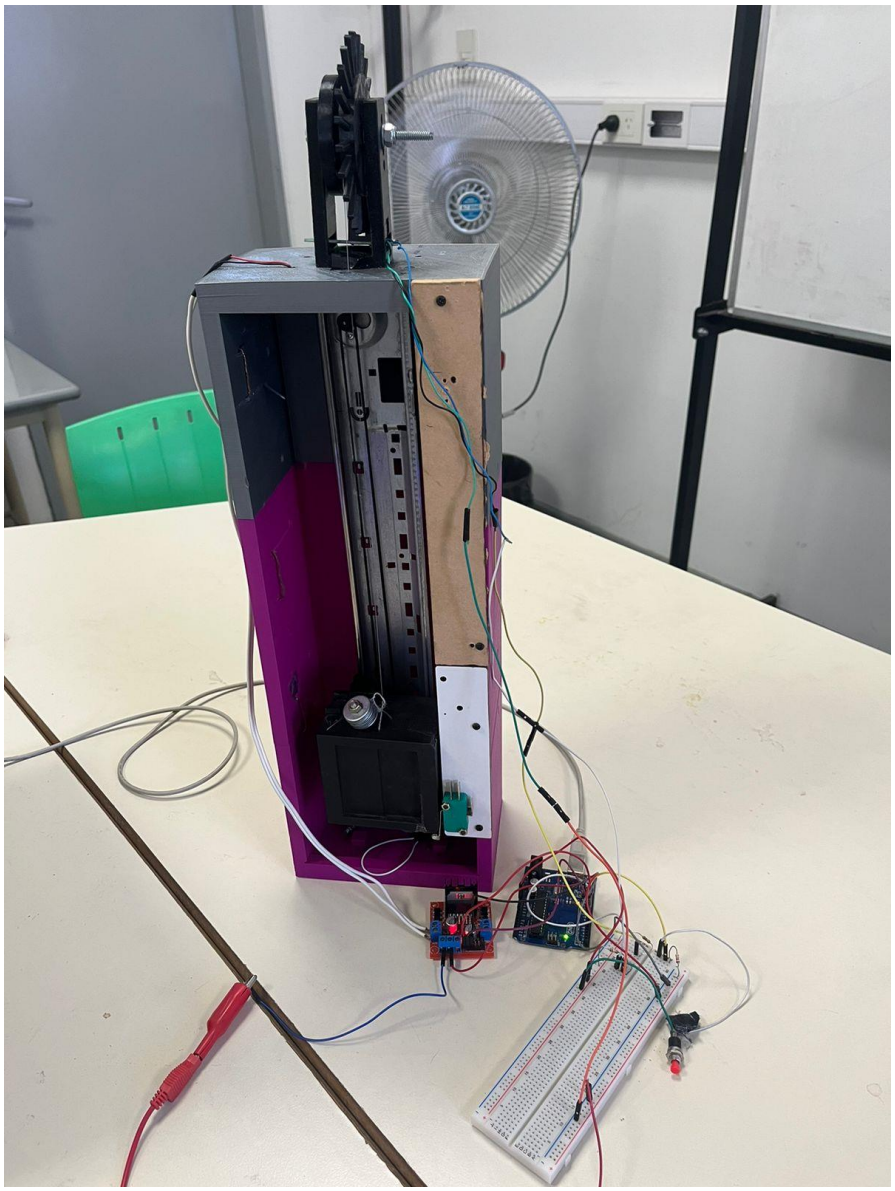

*Figura 9: Conexionado del Ascensor*

Luego de volver a realizar todo el conexionado, nos pusimos a modificar el código para que pueda funcionar nuestro sistema con un control proporcional derivativo.

Para el desarrollo de nuestro proyecto, contamos con un control proporcional derivativo, que a partir de las funciones “controlar” y “actuar” definidas en el código, decidimos si el motor debe subir, bajar, ir deteniéndose o detenerse en función de la posición en la que se encuentre y la posición establecida como objetivo.

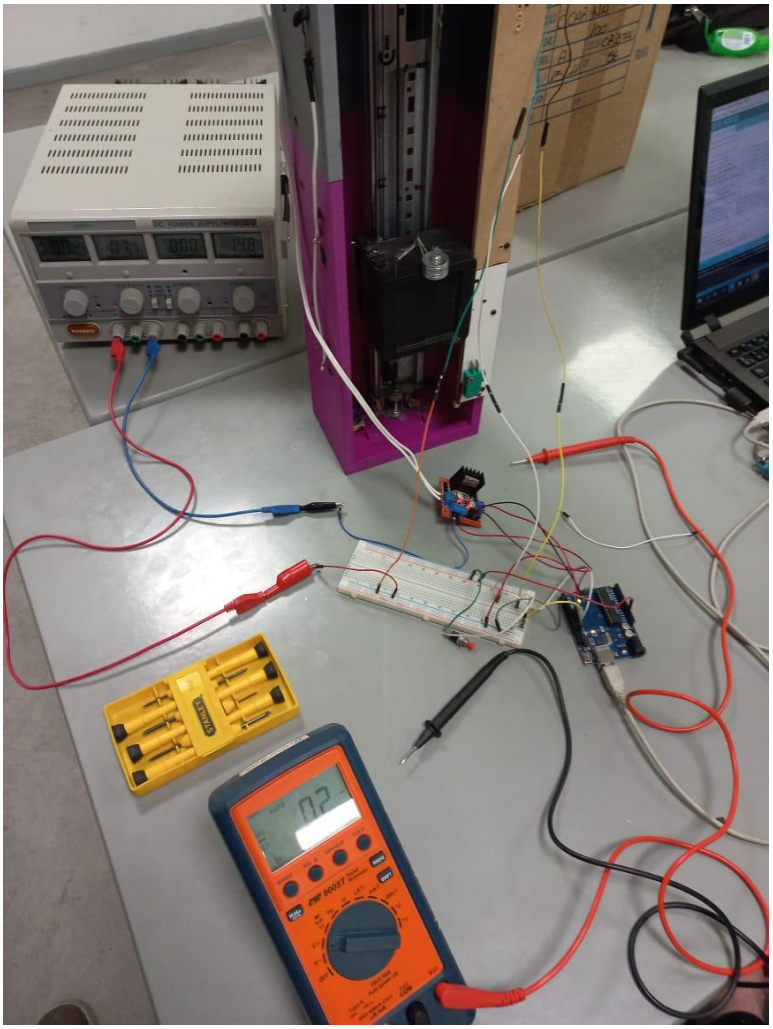

*Figura 10: Pruebas de las conexiones*

# Mediciones

Ya habiendo hecho el código y las conexiones de nuestro sistema, podemos pasar a la experiencia en sí. Lo que hicimos fue realizar varias mediciones, probando con distintos valores de $Kp$ y $Td$ para poder encontrar los valores que dieran la mayor precisión de nuestro sistema.

Nuestras mediciones las realizamos cada $50\,ms$. Específicamente, para la primera medición establecimos una altura objetivo de $10\,cm$, con un $Kp$ de 1,0 y un $Td$ de 0,2.

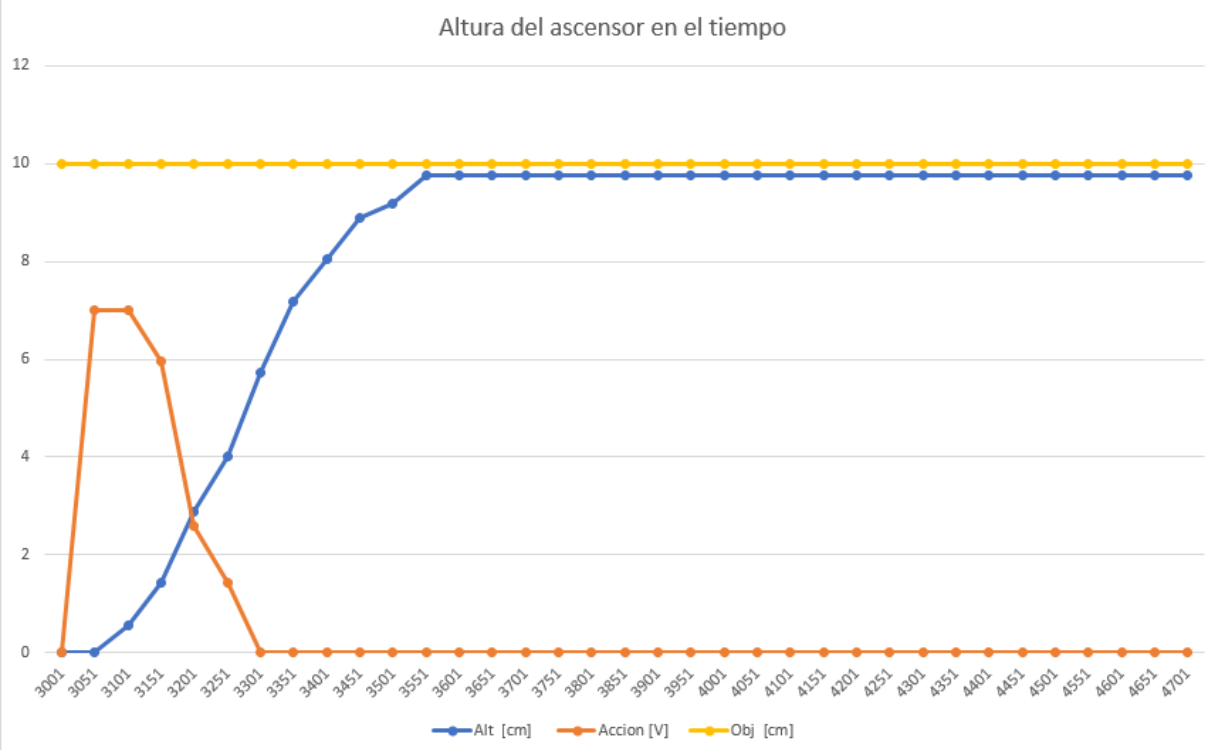


*Figura 11: Resultados de la primera medición.*

Como podemos observar, vemos que el ascensor llega hasta los $9{,}76\ cm$ y se detiene, lo cual indica una muy buena precisión, ya que el ascensor no sigue de largo y logra detenerse.

Para la segunda medición, lo que hicimos fue definir en nuestro código 3 pisos para el ascensor, con altura de $22cm$ para el piso $3$, $11\ cm$ para el piso $2$, y $0cm$ para el piso $1$. Para esto, utilizamos un $Kp$ de $2{,}6$ y un $Td$ de $0{,}2$.

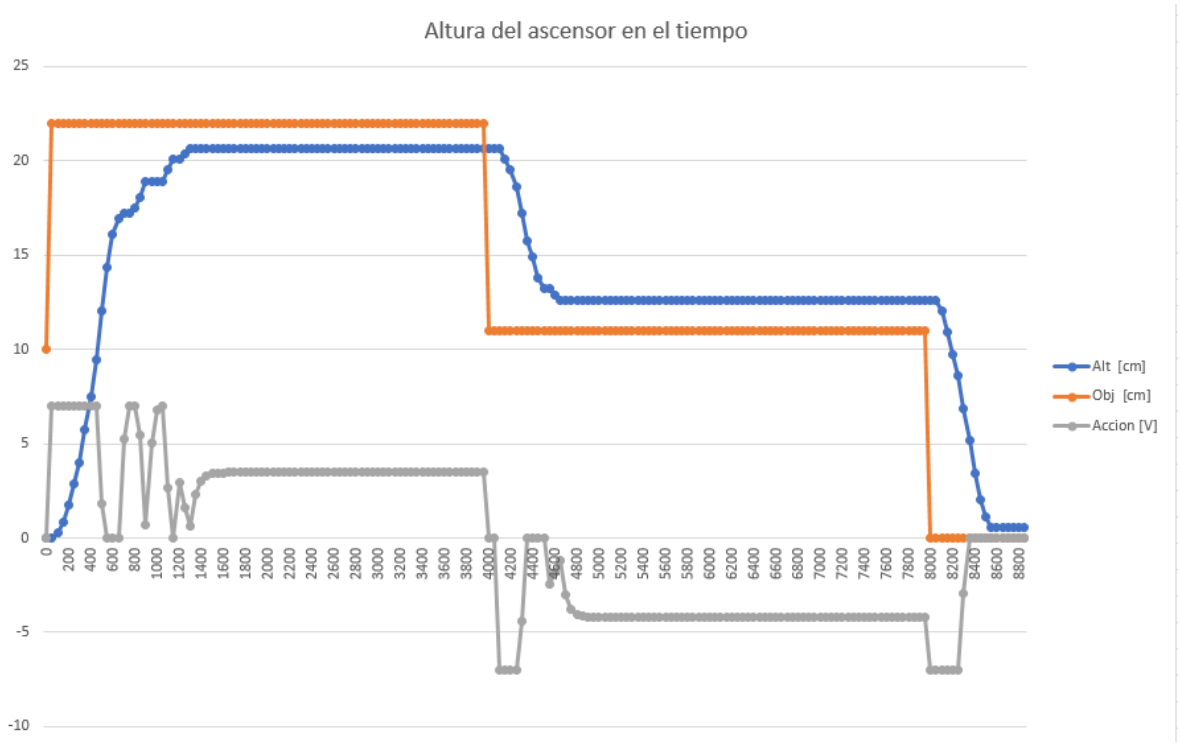


*Figura 12: Resultados de la segunda medición*

Como podemos observar, el ascensor se comporta de muy buena manera, ya que llega a sus 3 pisos objetivo sin pasarse de largo y deteniéndose cuando debe hacerlo.

Finalmente, para la tercer y última medición lo que hicimos fue utilizar un control proporcional integral. Utilizamos un $Kp$ de $3{,}2$, un $Td$ de $0{,}25$ y un $Ti$ de $2$.

Los resultados obtenidos fueron los siguientes

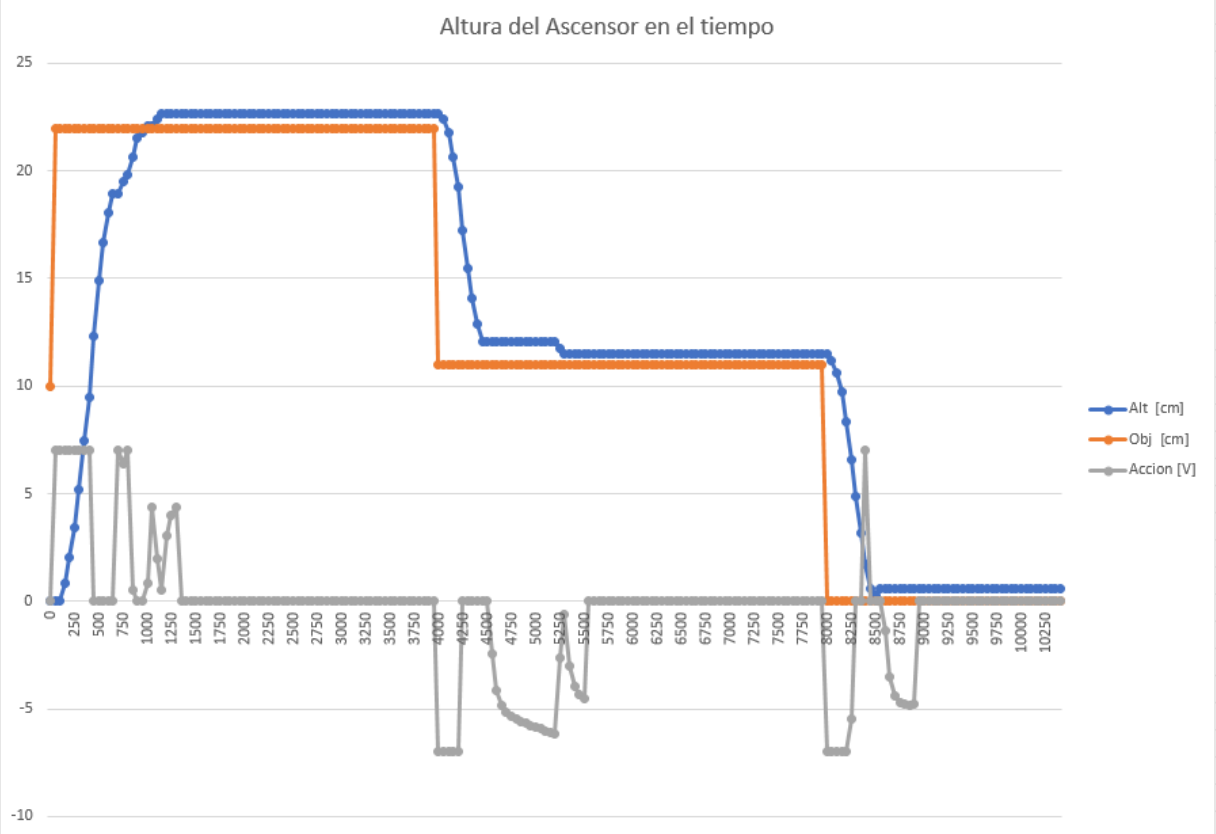


*Figura 13: Resultados de la tercera medición*

Una vez hechas las 3 mediciones, podemos afirmar que el tiempo de establecimiento es de $1300\,ms = 1{,}3s$ lo cual es muy parecido a lo que habíamos estimado en las simulaciones hechas.

# Conclusión

Como cierre, podemos decir que considerando nuestro objetivo principal al momento de hacer la practica era controlar el ascensor, logramos concluir el trabajo de laboratorio con éxito, haciendo que nuestro control proporcional derivativo funcione correctamente de acuerdo a los valores utilizados y la altura medida y el objetivo determinado. Pudimos comprobar la muy buena precisión que tiene nuestro sistema con la correcta elección de los tiempos de derivación y las constantes de proporcionalidad.

Esta vez, pudimos utilizar el pulsador para detener el sistema, y pudimos comprobar su correcto funcionamiento a la hora de presionarlo.

En comparación con la practica anterior, podemos decir que esta práctica se hizo más sencilla, ya que teníamos armado el ascensor, ya sabíamos cómo funcionaba el optoacoplador y los demás componentes, y en base al código esta vez al modularizar todo, fue mucho más claro y fácil de entender.

Afortunadamente, superamos estos desafíos y conseguimos completar con éxito nuestro trabajo de laboratorio, lo que dejó a todos nosotros muy satisfechos.

# Mejoras a desarrollar

Una mejora por realizar seria principalmente la posibilidad de agregar un tiempo de integración, un control integral además del control proporcional derivativo que se utiliza actualmente, este control integral podría darle mayor precisión a los desplazamientos u objetivos que tiene el ascensor a cada movimiento, reduciendo oscilaciones y dando mayor estabilidad, especialmente al momento de perturbaciones o cambios en las condiciones de la práctica. Durante la práctica se estuvieron haciendo experimentos con este control integral, pero el resultado no fue el esperado, debido a la disminución del voltaje que tiene en ese momento, no puede desplazarse correctamente para mejorar la precisión y no consigue terminar la acción, se tendría que verificar si es posible controlar mejor el voltaje para poder utilizarlo.

Otra mejora que se podría implementar, pero esta vez a un nivel físico del sistema, podría ser la incorporación de finales de carrera, tanto para el piso inferior como para el superior, así mismo se podría seguir agregar distintos pisos al ascensor. Por otro lado, que tanto el peso del ascensor como el del contrapaso sean el mismo, para que la medición obtenga una mayor precisión.

# Anexo

A continuación, se coloca el código fuente elaborado para el control proporcional derivativo que se usó para el ascensor, con sus respectivos módulos.

```
#include "Posicion.h"
#include "ControlPID.h"

// Definicion de pines
#define PIN_PH1           8    // Control para el pin 1 del motor A // Baja al alimentar el IN1
#define PIN_PH2           9    // Control para el pin 2 del motor A // Sube el motor al alimentar IN2
#define PIN_PH_PWM        10   // Elegir un pin que admita PWM //
#define PIN_EMERGENCIA    11   // Defino un pin para leer el boton de emergencia
#define PIN_LED           13   // Led de emergencia
#define PIN_OPTOACOPLADOR 12   // Elegir //Defino un pin para el optoacoplador

// Definicion de tiempos
#define DELTA_TIEMPO      50   // Intervalo entre muestras para control
#define INTERVALO_CONTROL 50   // en milisegundos
#define MUESTRAS_PARA_DETENER 6

// Constantes de parámetros de control PID
const float kp = 1.0;  // Ganancia proporcional
const float ti = 0;    // Ganancia integral (puedes ajustar según sea necesario)
const float td = 0.2;  // Ganancia derivativa

// Variables globales
controlPID controlAscensor (kp, ti, td);   // Instancia de la clase ControlPID
float altura_objetivo = 10;
float altura_medida = 0;
float accion_deseada_pwm;
const float altura_piso[3] = {0,11,22};
estado_motor accion_deseada = DETENIDO;

// Se establece una variable ALTURA en 0 (la cual será el destino del ascensor)
// Definimos una variable MOTOR que nos indique 0 si esta quieto, 1 si sube y 2 si baja
// Se establece una variable ALT_OBJ en 0
// El ascensor estará en la planta baja.
// float ALTURA = 0.0;  // unificar unidades de altura y tipo de variable
//estado_motor accion_motor = DETENIDO;   // Estado inicial del motor

unsigned long ultimo_tiempo = 0;
unsigned long tiempo_actual = 0;
unsigned long tiempo_inicial = 0;

// Declaración de funciones
void mostrar();
void detenerMaquina();
void controlar();
void actuar();
```

```
49  void setup() {
50      inicializarPosicion(PIN_OPTOACOPLADOR);
51      pinMode(PIN_PH1, OUTPUT);//Se inician los pines
52      pinMode(PIN_PH2, OUTPUT);
53      pinMode(PIN_EMERGENCIA, INPUT);
54      pinMode(PIN_PH_PWM, OUTPUT);
55      pinMode(PIN_LED, OUTPUT);
56
57      controlAscensor.LimitarSalida(true, -TENSION_MAXIMA, TENSION_MAXIMA);
58
59      // Inicialicen la conexión serial
60      Serial.begin(115200);  // poner velocidad mas rapida
61
62      // Encabezado de datos:
63      Serial.println("Encabezado de datos");
64      Serial.println("Tiempo\tAlt \tObj \tProp \tDer \tAccion");
65      Serial.println("[ms] \t[cm] \t[cm] \t[V] \t[V] \t[V]");
66
67      // Antes de comenzar
68      delay (3000); // Luego de 3 segundos arrancamos el programa.
69      ultimo_tiempo = millis();
70      tiempo_actual = ultimo_tiempo;
71      tiempo_inicial = ultimo_tiempo;
72      mostrar();
73
74  }
```

```
78  void loop() {
79    tiempo_actual = millis();
80
81    // Verificamos el botón de emergencia (PIN_EMERGENCIA)
82    if (digitalRead(PIN_EMERGENCIA) == HIGH) {
83      detenerMaquina(); // Realizamos la parada de emergencia
84    }
85
86    // Obtenemos la altura medida utilizando la función de Posicion
87    altura_medida = leerPosicion(accion_deseada);
88
89    if (millis() - ultimo_tiempo >= DELTA_TIEMPO) {
90      ultimo_tiempo += DELTA_TIEMPO;
91      //Analizamos altura objetivo
92      if(ultimo_tiempo - tiempo_inicial < 4000){
93        altura_objetivo = altura_piso[2];
94
95      }else if (ultimo_tiempo - tiempo_inicial < 8000){
96          altura_objetivo = altura_piso[1];
97
98      }else if(ultimo_tiempo - tiempo_inicial < 12000){
99          altura_objetivo = altura_piso[0];
100     }
101     // Llamamos a la función de control
102     accion_deseada_pwm = controlAscensor.Controlar(altura_objetivo - altura_medida);
103
104     // Llamada a la función de actuación
105     actuar();
106
107     // Mostramos información
108     mostrar();
109   }
110 }
```

```
113   // Parada de emergencia
114   void detenerMaquina() {
115     // Lógica para detener el ascensor de emergencia
116     digitalWrite(PIN_PH1, LOW);
117     digitalWrite(PIN_PH2, LOW);
118     analogWrite(PIN_PH_PWM, 0);
119     digitalWrite(PIN_LED, HIGH);
120     Serial.println("Parada de emergencia");
121
122     do {} while (1);
123   }
```

```
127   void actuar()
128   {
129     int pwmValue = map(abs(accion_deseada_pwm), TENSION_MIN_ACT, TENSION_MAXIMA, 0, 255);
130     if(pwmValue == TENSION_MIN_ACT){
131       pwmValue = 0;
132     }
133     static int cuenta = 0;
134     static int cuenta2 = 0;
135     switch (accion_deseada){
136       case DETENIDO:
137       cuenta2 = 0;
138         // Ascensor Detenido
139         if (altura_objetivo - altura_medida > TOLERANCIA ){
140           accion_deseada = SUBIENDO;
141         }
142         if (altura_medida - altura_objetivo > TOLERANCIA){
143           accion_deseada = BAJANDO;
144         }
145         break;
146
147       case SUBIENDO:
148         // Ascensor Subiendo
149         if(accion_deseada_pwm <= 0){
150           accion_deseada = DETENIENDO_SUB;
151         }
152         if(abs(altura_objetivo - altura_medida) <= TOLERANCIA){
153           cuenta2++;
154           if(cuenta2 > MUESTRAS_PARA_DETENER){
155             accion_deseada = DETENIDO;
156           }
157         }
158         break;
159
160       case BAJANDO:
161         // Ascensor Bajando
162         if(accion_deseada_pwm >= 0){
163           accion_deseada = DETENIENDO_BAJ;
164         }
165         if(abs(altura_objetivo - altura_medida) <= TOLERANCIA){
166           cuenta2++;
167           if(cuenta2 > MUESTRAS_PARA_DETENER){
168             accion_deseada = DETENIDO;
169           }
170         }
171         break;
```

```
173     case DETENIENDO_SUB:
174     cuenta2 = 0;
175       // Ascensor Deteniendose
176       cuenta++;
177       if ((abs(leerVelocidad())) < VELOCIDAD_MINIMA){
178         accion_deseada = DETENIDO;
179         cuenta=0;
180       }
181       if (cuenta > MUESTRAS_PARA_DETENER) {
182         accion_deseada = DETENIDO;
183         cuenta=0;
184       }
185       if(accion_deseada_pwm > 0){
186         accion_deseada = SUBIENDO;
187       }
188       break;
189 
190       case DETENIENDO_BAJ:
191       cuenta2 = 0;
192       // Ascensor Deteniendose
193       cuenta++;
194       if ((abs(leerVelocidad())) < VELOCIDAD_MINIMA){
195         accion_deseada = DETENIDO;
196         cuenta=0;
197       }
198       if (cuenta > MUESTRAS_PARA_DETENER) {
199         accion_deseada = DETENIDO;
200         cuenta=0;
201       }
202       if(accion_deseada_pwm < 0){
203         accion_deseada = BAJANDO;
204       }
205       break;
206   }
207 
```

```
208   switch (accion_deseada){
209
210     case DETENIDO:
211       // Ascensor Detenido
212       analogWrite(PIN_PH_PWM, 0);
213       accion_deseada_pwm = 0;
214       break;
215
216     case SUBIENDO:
217       // Ascensor Subiendo
218       digitalWrite(PIN_PH1, LOW);
219       digitalWrite(PIN_PH2, HIGH);
220       analogWrite(PIN_PH_PWM, pwmValue);
221       break;
222
223     case BAJANDO:
224       // Ascensor Bajando
225       digitalWrite(PIN_PH2, LOW);
226       digitalWrite(PIN_PH1, HIGH);
227       analogWrite(PIN_PH_PWM, pwmValue);
228       break;
229
230     case DETENIENDO_SUB:
231       // Ascensor Deteniendose
232       digitalWrite(PIN_PH2, LOW);
233       digitalWrite(PIN_PH1, LOW);
234       analogWrite(PIN_PH_PWM, 0);
235       accion_deseada_pwm = 0;
236       break;
237
238       case DETENIENDO_BAJ:
239       // Ascensor Deteniendose
240       digitalWrite(PIN_PH2, LOW);
241       digitalWrite(PIN_PH1, LOW);
242       analogWrite(PIN_PH_PWM, 0);
243       accion_deseada_pwm = 0;
244       break;
245   }
246   }
```

```
248  void mostrar() {
249      // Imprimimos los valores de las variables y del controlador PID
250      //Serial.print("Tiempo: ");
251      Serial.print(ultimo_tiempo - tiempo_inicial);
252      Serial.print("\t");
253
254      //Serial.print("ALTURA actual: ");
255      Serial.print(altura_medida);  // Cambiado de ALTURA a altura_medida
256      Serial.print("\t");
257
258      //Serial.print("ALTURA objetivo: ");
259      Serial.print(altura_objetivo);
260      Serial.print("\t");
261
262      //Serial.print("Acción Proporcional: ");
263      Serial.print(controlAscensor.ObtenerProporcional());
264      Serial.print("\t");
265
266      //Serial.print("Acción Derivativa: ");
267      Serial.print(controlAscensor.ObtenerDerivativo());
268      Serial.print("\t");
269
270      //Serial.print("Salida Controlador PID: ");
271      Serial.print(accion_deseada_pwm);
272      Serial.print("\t");
273
274      switch (accion_deseada) {
275      case SUBIENDO:
276          Serial.println("SUB");
277          break;
278      case DETENIDO:
279          Serial.println("DET");
280          break;
281     case DETENIENDO_SUB:
282          Serial.println("DNS");
283          break;
284      case BAJANDO:
285          Serial.println("BAJ");
286          break;
287      case DETENIENDO_BAJ:
288          Serial.println("DNB");
289          break;
290      }
291  }
```

```h
#ifndef ASCENSOR_H
#define ASCENSOR_H

#define INTERVALO_MINIMO  5       // Intervalo mínimo entre cambios de estados, en milisegundos.
#define DISTANCIA         0.287   // Distancia de un cambio de estado del optoacoplador, en cm.
  // Cambios de estados y desplazamiento:
  // La polea tiene 46 cambios de estados en una vuelta.
  // Diametro de polea = 42 mm => Recorrido en una vuelta pi*diametro = 131 mm
  // Cada cambio de estado = 131 mm / 46 = 2,87 mm = 0,287 cm
#define TOLERANCIA        1.0     // Define la tolerancia en centímetros
#define VELOCIDAD_MINIMA  1.0     // Define la velocidad mínima en cm/s
#define VEL_MAXIMA        35.6    // cm/s
#define VELOCIDAD_CONVERSION 5.93 // cm/V/s
#define TENSION_MAXIMA    7
#define TENSION_MIN_ACT   3

typedef enum { SUBIENDO, BAJANDO, DETENIENDO_SUB, DETENIENDO_BAJ, DETENIDO } estado_motor;

estado_motor leerUltimoMovimiento();
void  inicializarPosicion (int pin_opto);
float leerPosicion (estado_motor accion_deseada);
float leerVelocidad ();

#endif
```

```cpp
#include "Posicion.h"
#include "Arduino.h"

estado_motor ultima_accion = DETENIDO;
unsigned long tiempo_ultimo_cambio_0 = 0;
unsigned long tiempo_ultimo_cambio_1 = 0;
unsigned long tiempo_ultimo_cambio_2 = 0;
bool estadoOptoacoplador;
float altura = 0;
int pin_optoacoplador;

//*************************************************************************
void inicializarPosicion(int pin_opto)
{
  pin_optoacoplador = pin_opto;
  // Configuramos el pin del optoacoplador como entrada
  pinMode(pin_optoacoplador, INPUT);
  estadoOptoacoplador = digitalRead(pin_optoacoplador);
  altura = 0;
}

//*************************************************************************
estado_motor leerEstadoMotor()
{
  return ultima_accion;
}

```

```
29  float leerPosicion(estado_motor accion_deseada)
30  {
31
32    // Obtenemos el tiempo actual y leo estado de optoacoplador
33    unsigned long tiempo_actual = millis();
34    bool lectura_actual = digitalRead(pin_optoacoplador);
35
36    // Verificamos si hubo cambio de estado
37    if (estadoOptoacoplador != lectura_actual) {
38
39      // ANTIRREBOTE: verificamos si ha pasado un tiempo mínimo
40      if (tiempo_actual - tiempo_ultimo_cambio_0 >= INTERVALO_MINIMO) {
41        // Hubo un cambio de estado verificado!!!
42        estadoOptoacoplador = lectura_actual;
43
44        // Guardamos los tiempos de los dos últimos cambios de estado
45        tiempo_ultimo_cambio_2 = tiempo_ultimo_cambio_1;
46        tiempo_ultimo_cambio_1 = tiempo_ultimo_cambio_0;
47        tiempo_ultimo_cambio_0 = tiempo_actual;
48
49        // Actualizar la última acción si el motor estaba subiendo o bajando
50        if (accion_deseada == SUBIENDO) {
51          ultima_accion = SUBIENDO;
52        }
53        if (accion_deseada == BAJANDO) {
54          ultima_accion = BAJANDO;
55        }
56
57        // Calculamos la posición actual
58        if (ultima_accion==SUBIENDO) {altura+=DISTANCIA;};
59        if (ultima_accion==BAJANDO)  {altura-=DISTANCIA;};
60      }
61    }
62
63    //Devolvemos la altura
64    return altura;
65  }
66
68  float leerVelocidad()
69  {
70    float medicion = 0;
71    unsigned int delta_actual   = 0; // es el delta actual, que mide si reduce la velocidad o no
72    unsigned int delta_anterior = 0; // es el delta anterior, medido entre los deltas de tiempo actual y anterior al anterior.
73
74    if (0 == tiempo_ultimo_cambio_1){
75      // Por el momento no hubo ninguna interrupciòn almacenada
76      // Se supone una velocidad de 0 RMP.
77      medicion = 0;
78    } else {
79      delta_actual = millis() - tiempo_ultimo_cambio_1;
80      delta_anterior = tiempo_ultimo_cambio_0 - tiempo_ultimo_cambio_2;
81
82      if ( 0 != tiempo_ultimo_cambio_2 ) {
83        // Verifico que ya habia sido actualizado tiempo_ultimo_cmbio_2
84        // y elijo la menor velocidad (== a el mayor tiempo):
85        delta_actual = max(delta_actual, delta_anterior);
86      }
87
88      medicion = 1000 * 2 * DISTANCIA / (delta_actual);
89    }
90
91    return medicion;
92  }
93
94  estado_motor leerUltimoMovimiento()
95  {
96    return ultima_accion;
97  }
```

```
/*****************************************************************************************************
* Control PID
*****************************************************************************************************
* Sistemas de Control Automatico (SCA)
* Universidad Nacional de Avellaneda (UNDAV)
*
* Archivo:    pid_sca.h
* Version:    2.1.
* Fecha:      Noviembre 2023
* Novedades:  Reorganizacion de funciones en .h y .cpp
* Version Anterior: 2.0, diciembre 2021.
*****************************************************************************************************/

#define COHEFICIENTE_FILTRO 0.6

class controlPID            // Objeto para control Proporcional-Integral-Derivativo (PID)
{
private:

    float Salida;                 // La señal de control que va al acuador
    // o potencia de salida (sin asignar unidades)
    float Proporcional;           // Componente proporcional de la salida
    // (sin asignar unidades)
    float Integral;               // Componente integral
    float Derivativo;             // Componente derivativa
    float Compensacion;           // Contiene la compensacion resultante ante saturacion
    float CompensacionAnterior;   // Como usamos aproximacion trapezoidal de la integral,
    // necesitamos conservar el valor anterior.
    float Kp;                     // Constante proporcional (sin asignar unidades)
    float Ti;                     // Tiempo de integracion (en segundos)
    float Td;                     // Tiempo para la componente derivativa (en segundos)
    unsigned long TiempoAnterior; // Tiempo de la medicion anterior utilizando micros
    float ErrorAnterior;          // Señal de error anterior
    bool LimitaSalida;            // Indica si establecimos limites superior e inferior
    bool CompensaIntegral;        // Indica si establecimos la compensacion de integral
    float SalidaMax;              // Limite superior de la salida (y de la integral)
    float SalidaMin;              // Limite inferior de la salida
    const float MILLON = 1e6;     // Constante para convertir micros() a segundos.

```

```
public:

    // Constructor con lo minimo:
    controlPID(float KP,          // KP: Constante de proporcionalidad (puede ser negativo)
               float TI,          // TI: Tiempo de integracion (si es 0, no integra)
               float TD);        // TD: Tiempo de derivacion (si es 0 no deriva)

    // Para cambiar configuracion inicial:
    void ConfigurarPID(float KP, float TI, float TD);  //Mismos parametros que constructor.

    // Configura los limites de salida e indica si estan activados:
    bool LimitarSalida(bool RESPUESTA, float SMIN, float SMAX);

    // Activa o desactiva los limites de salida e indica si el limite de salida esta activado:
    bool LimitarSalida(bool RESPUESTA);  // No permite activar limites si antes no fueron
    // establecidos.

// Indica si el limite de salida esta activado:
    bool LimitarSalida();

    // Activa o desactiva la compansacion de integracion e indica si esta activado:
    bool CompensarIntegral(bool RESPUESTA);

    // Me indica si la compansacion esta activada:
    bool CompensarIntegral();

    // Calcula señal de control (salida) en funcion del error:
    float Controlar(float ERROR);

    // Apaga el PID manteniendo configuracion:
    void Apagar();                // No se modifican los valores de KP, TI y TD.
    // Tampoco los limites pre establecidos.

    float ObtenerIntegral();
    float ObtenerProporcional();
    float ObtenerDerivativo();
    float ObtenerSalida();
    float ObtenerCompensacion();

};
```

```
/*****************************************************************************************************
* Control PID
*****************************************************************************************************
* Sistemas de Control Automatico (SCA)
* Universidad Nacional de Avellaneda (UNDAV)
*
* Archivo:    pid_sca.cpp
* Version:    2.1.
* Fecha:      Noviembre 2023
* Novedades:  Reorganizacion de funciones en .h y .cpp
* Version Anterior: 2.0, diciembre 2021.
*****************************************************************************************************/

#include "ControlPID.h"
#include "Arduino.h"

/*****************************************************************************************************/

controlPID::controlPID(float KP, float TI, float TD)
// Constructor: incluye configuracion inicial del PID y valores predeterminados.
{  // Kp puede ser negativo (esto ultimo podria servir para controlar una planta cuya salida
   //                        tienda a bajar cuando aumente la señal de control. Ej.: heladera.)
   // Si Ti=0, el PID no lo tomar en cuenta
   // Si Td=0, el PID no lo tomar en cuenta
   // Inicializa integracion e impone false en limites y compensacion.
    ConfigurarPID(KP, TI, TD);
    LimitaSalida = false;
    CompensaIntegral = false;
}
```

```
32  void controlPID::ConfigurarPID(float KP, float TI, float TD)
33  // Configura las constantes basicas del control PID
34  // Sirve para cambiar configuracion inicial, sin modificar limites y banderas.
35  {  // Kp puede ser negativo (esto ultimo podria servir para controlar una planta cuya salida
36     //                        tienda a bajar cuando aumente la señal de control. Ej.: heladera.)
37     // Si Ti=0, el PID no lo tomar en cuenta
38     // Si Td=0, el PID no lo tomar en cuenta
39      Kp = KP;
40      Ti = TI;
41      Td = TD;
42      // Resetea valores de integracion.
43      TiempoAnterior = 0;
44      ErrorAnterior = 0;
45      CompensacionAnterior = 0;
46      Integral = 0;
47  }
48  //-------------------------------------------------------------------------------------------------
49
50  bool controlPID::LimitarSalida()
51  // Devuelve el valor de la variable privada LimitaSalida
52  // que indica si nuestro PID esta configurado para limitar su salida.
53  {
54      return LimitaSalida;
55  }
56  //-------------------------------------------------------------------------------------------------
57
58  bool controlPID::LimitarSalida(bool RESPUESTA)
59  // Configura si limitar la salida...
60  // No permite activar limites si antes no fueron establecidos.
61  {
62      LimitaSalida = RESPUESTA;
63      if (SalidaMax == 0 && SalidaMin == 0) {
64          // ...no voy a limitar porque no tengo limites definidos.
65          LimitaSalida = false;
66      }
67      return LimitaSalida;
68  }
```

```
71  bool controlPID::LimitarSalida(bool RESPUESTA, float SMIN, float SMAX)
72  // Configura si limitar la salida entre SMAX y SMIN.
73  // Se puede establecer los limites pero no activarlos aun.
74  // No activa con SMIN=SMAX.
75  // No activa si SMIN>SMAX.
76  {
77      SalidaMax = SMAX;
78      SalidaMin = SMIN;
79      LimitaSalida = RESPUESTA;
80      // La forma de desactivar este limite es:
81      // 1) Volviendo a llamar esta funcion con RESPUESTA=false
82      // 2) Llamando a LimitarSalida(false)
83      // Tambien se pueden poner limites muy grandes.
84      if (SalidaMax == SalidaMin) {
85          // ...no voy a limitar porque no tengo limites definidos.
86          LimitaSalida = false;
87      }
88      if (SalidaMin > SalidaMax) {
89          // ...no voy a limitar porque estan mal configurados.
90          LimitaSalida = false;
91      }
92      return LimitaSalida;
93  }
94  //---------------------------------------------------------------------------------------------------
95
96  bool controlPID::CompensarIntegral()
97  // Devuelve el valor de CompensaIntegral.
98  {
99      return CompensaIntegral;
100 }
101 //---------------------------------------------------------------------------------------------------
102
103 bool controlPID::CompensarIntegral(bool RESPUESTA)
104 // Establece si debo compensar la integracion cuando la salida esta saturada.
105 // Deben haberse preestablecido los limites de salida.
106 {
107     CompensaIntegral = RESPUESTA;
108     if (!LimitaSalida) CompensaIntegral = false;
109     return CompensaIntegral;
110 }
```

```
113  float controlPID::Controlar(float ERROR)
114  // Calcula Salida en funcion de la señal error y los parametros del PID
115  {
116      unsigned long TiempoActual = micros(); // Tomo tiempo actual para comparar con anterior
117
118      // PROPORCIONAL ---------------------------------------------------------------------------
119      Proporcional = Kp * ERROR;
120
121      // DERIVATIVO -----------------------------------------------------------------------------
122      if (TiempoAnterior > 0 && Td != 0) {
123          // Dos condiciones para componente derivativa:
124          // 1) Que no sea el primer calculo y 2) Td seteado
125          float Cuenta = Kp * Td * (ERROR - ErrorAnterior) * MILLON / (TiempoActual - TiempoAnterior);
126          Derivativo = COHEFICIENTE_FILTRO * Cuenta + (1-COHEFICIENTE_FILTRO) * Derivativo;
127      }
128      else {
129          Derivativo = 0;
130      }
131
132      // Debo compensar? ------------------------------------------------------------------------
133      Salida = Proporcional + Integral + Derivativo;
134      Compensacion = 0;
135      if (LimitaSalida && CompensaIntegral) {
136          // Si no hay limite de saturacion a la salida, no hay nada que compensar...
137          if (Salida > SalidaMax) {
138              // Debo saturar la salida porque supera el maximo...
139              Compensacion = Salida - SalidaMax;
140          }
141          if (Salida < SalidaMin) {
142              // Debo saturar la salida porque esta por debajo del minimo...
143              Compensacion = Salida - SalidaMin;
144          }
145      }
146
```

```
    // INTEGRAL ------------------------------------------------------------------------------
    if (TiempoAnterior > 0 && Ti != 0) {
        // Cumplidas las condiciones para integrar: (Si Compensacion==0, no va a compensar nada...)
        Integral += (Kp * (ERROR + ErrorAnterior) - (Compensacion + CompensacionAnterior))
            * (TiempoActual - TiempoAnterior)
            / (2 * Ti * MILLON);
        if (LimitaSalida) {
            // Debo saturar la integral:
            // (se supone que esto solo podria pasar si cambio los parametros de integracion)
            Integral = min(Integral, SalidaMax);
            Integral = max(Integral, SalidaMin);
        }
    }

    // Termina componente integral ----------------------------------------------------------

    // Calculo final completo:
    Salida = Proporcional + Integral + Derivativo;

    if (LimitaSalida) {
        // Debo saturar la salida:
        // (se supone que esto solo podria pasar si cambio los parametros de integracion)
        Salida = min(Salida, SalidaMax);
        Salida = max(Salida, SalidaMin);
    }
    TiempoAnterior = TiempoActual;
    ErrorAnterior = ERROR;
    CompensacionAnterior = Compensacion;
    return Salida;
    // Termina funcion PID ------------------------------------------------------------------
}
//-------------------------------------------------------------------------------------------

float controlPID::ObtenerIntegral()
{
    return Integral;
}
//-------------------------------------------------------------------------------------------

float controlPID::ObtenerProporcional()
{
    return Proporcional;
}
//-------------------------------------------------------------------------------------------

float controlPID::ObtenerDerivativo()
{
    return Derivativo;
}
//-------------------------------------------------------------------------------------------

float controlPID::ObtenerSalida()
{
    return Salida;
}
//-------------------------------------------------------------------------------------------

float controlPID::ObtenerCompensacion()
{
    return Compensacion;
}
//-------------------------------------------------------------------------------------------

void controlPID::Apagar()
{
    TiempoAnterior = 0;
    ErrorAnterior = 0;
    CompensacionAnterior = 0;
    Integral = 0;
    Proporcional = 0;
    Derivativo = 0;
}
```

# Bibliografía

PWM

https://solectroshop.com/es/blog/que-es-pwm-y-como-usarlo--n38

Kuo - “Sistemas de control automático - 7ed”

Codigo para control PID

https://github.com/sca-undav/Control_PID