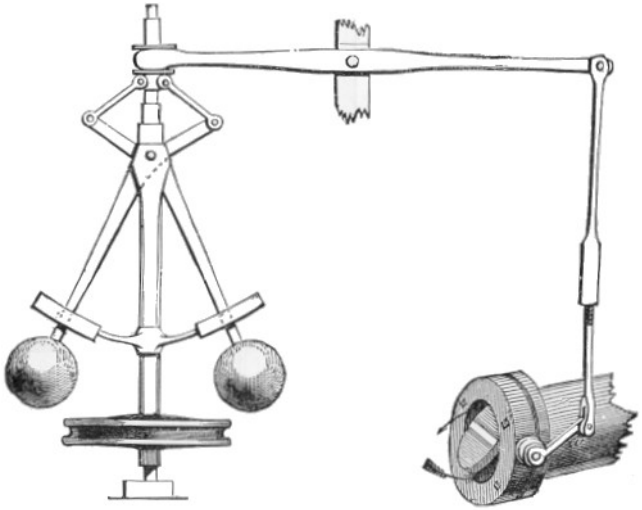
MECATRÓNICA – Apuntes Actualizados 2020

Licenciatura en Artes Electrónicas UNTREF

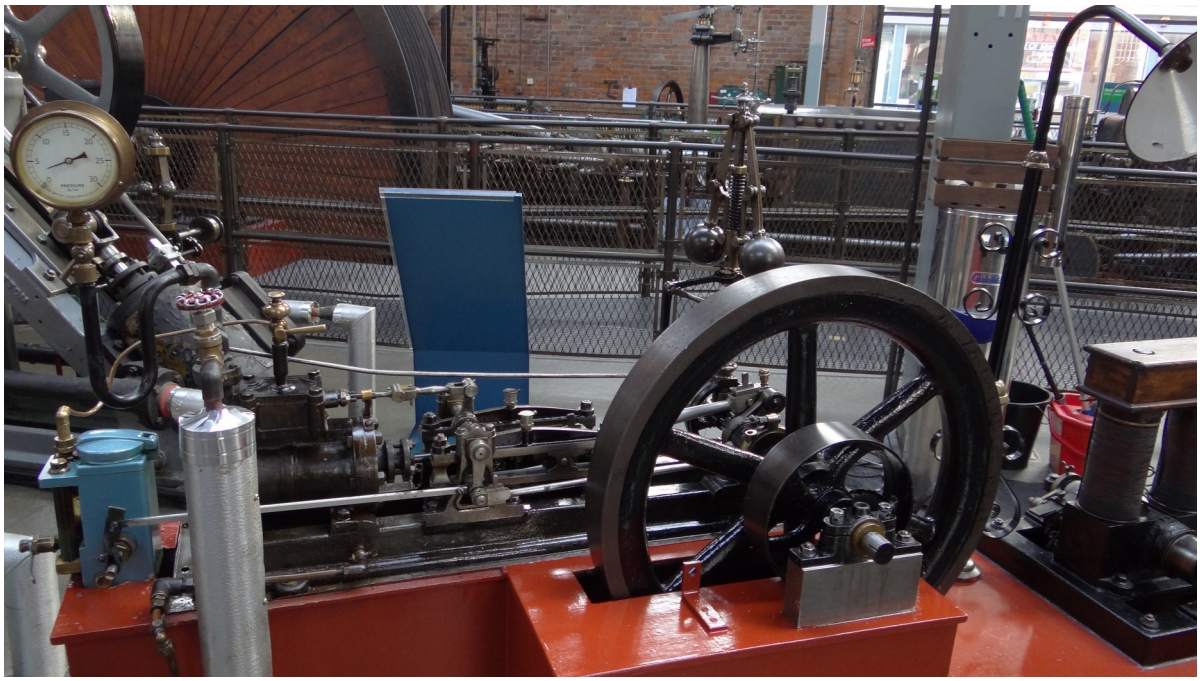
Introducción

Como hemos visto, la revolución industrial ocurrió al substituirse la producción manual o artesanal, de un sólo artículo a la vez, por la producción en masa o en serie. Esto fue posible gracias a que el "*governor*", creado por James Watt en 1788, permitió el control automático y estable de la velocidad de la maquinaria de vapor.

Ver video "Introducción al concepto de Control": https://youtu.be/KcNewOfDpkI



Posteriormente, el motor eléctrico, los sistemas hidráulicos, etc. reemplazaron a las máquinas de vapor y se desarrollaron los sistemas automáticos de control industrial.



Una máquina de Vapor con Governor en el Museo de Manchester

Sistemas de Control

Un sistema de control es un conjunto de dispositivos que interactúan para mantener el estado de cierta variable dentro de un determinado rango deseado de valores.

Por ejemplo el conjunto de una cañería de alimentación de agua que llena un tanque, la válvula que abre y cierra el ingreso del líquido, un desagote por el cual sale el líquido para su uso y el propio tanque conforman un sistema en el que usualmente es necesario un sistema de control para mantener el nivel del líquido por encima de determinado valor, sin que se derrame.

Son elementos indispensables en todo Sistema de Control, de cualquier tipo:

- Una **Variable** a controlar (en el ejemplo, el nivel de agua)
- Un **Actuador** capaz de modificar dicha variable (la válvula de llenado)
- Un **Set Point** o valor deseado de la variable (el tanque casi lleno, sin derramarse)

Usualmente intervendrán otros elementos externos que influyen sobre el sistema, como el consumo de agua por la salida, alguna posible pérdida por diversas razones, un llenado adicional por agua de lluvia, etc. que también actúan sobre la variable, pero en forma no controlada por el sistema y que se conocen como **Perturbaciones**.

Clasificación

Según el objeto del control podemos clasificar a los a los Sistemas en dos grupos:

1. Control de movimiento

Se conocen también como servomecanismos: Controlan posición, velocidad, aceleración, dirección y sentido. Son los de uso más habitual en nuestras aplicaciones mecatrónicas de arte.

2. Control de procesos

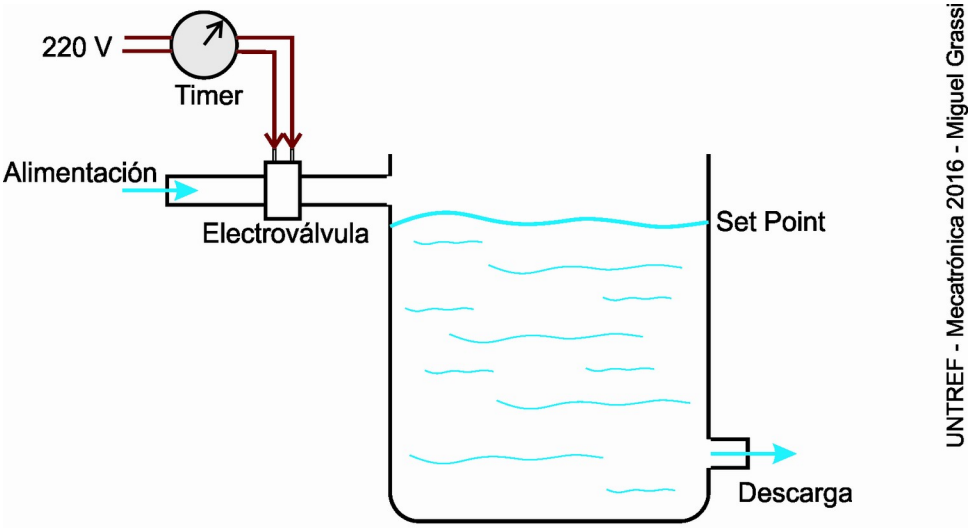
Controlan variables físicas o químicas tales como: temperatura, presión, caudal, nivel de líquido en un tanque, pH (acidez), humedad o composición química, entre otras. Son de uso habitual en la industria.

Sin embargo, la clasificación más importante se hace según el sistema reciba o no información del estado de la variable que está controlando (realimentación o *feedback*) en lo que se conoce como sistemas de control a lazo abierto o a lazo cerrado.

Lazo abierto

Son sistemas de control en los que el elemento que ejerce la acción sobre la variable controlada NO recibe información alguna sobre el estado de la misma y actúa según un modelo predeterminado que ya tiene incorporado.

Se aplican a problemas simples, donde el comportamiento de las variables es constante y conocido. Por ejemplo, un tanque que se sabe que se vacía durante la noche, puede llenarse todas las mañanas mediante una válvula de paso accionada eléctricamente por medio de un timer. Simplemente el timer acciona la válvula durante un determinado tiempo previamente calculado y se alcanzará el nivel deseado. Nótese que es necesario que la presión de ingreso del agua sea siempre similar, que el consumo nocturno sea constante y que no existan perturbaciones que puedan influir significativamente (por ejemplo llenado adicional por agua de lluvia).

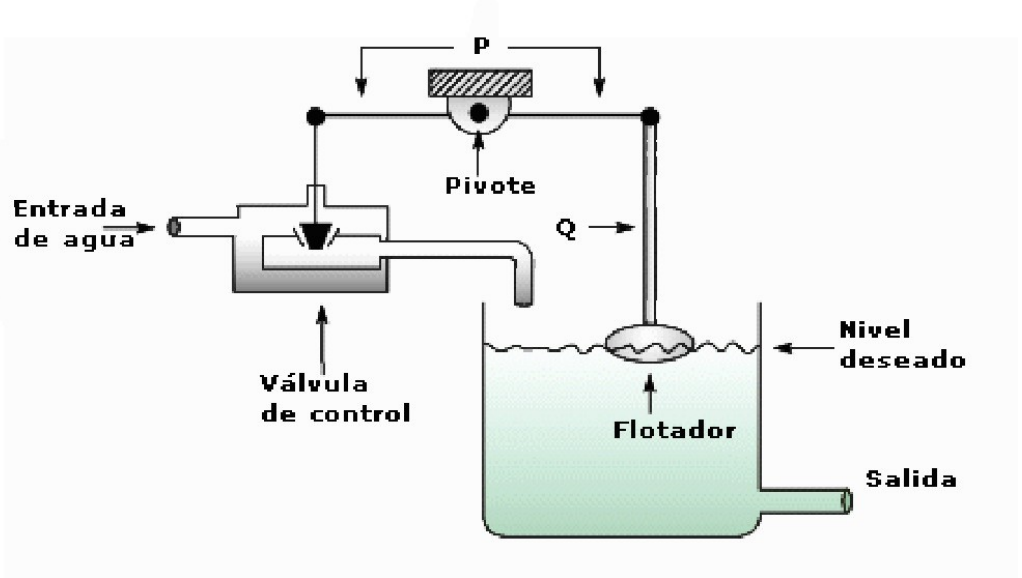


Sistema de control a lazo abierto - Ejemplo

Los sistemas de lazo abierto son simples, baratos, fáciles de implementar y mantener y son aplicables en misiones no críticas. En nuestro ejemplo, podría ocurrir que debido a una variación en el consumo u otra perturbación, el nivel no bajara lo habitual y suficiente durante la noche, por lo que a la mañana se produciría un desborde. Esto podría ser irrelevante si se trata de un tanque de riego en un jardín, pero muy problemático si el tanque está en el interior de un edificio. Si se quiere evitar esta circunstancia (o la posibilidad de quedarse sin contenido a mitad del tiempo de uso previsto) deberá apelarse a un sistema más complejo, que verifique periódicamente el nivel de llenado actual y lo compare con el nivel deseado, lo que constituye un sistema de lazo cerrado.

Lazo cerrado

Son automáticos y operan sin interrupción, ni participación externa.



En la figura se muestra un sistema a lazo cerrado en donde la válvula se abre o cierra automáticamente, de acuerdo con las variaciones de nivel, para mantener el mismo constante

A los elementos ya mencionados propios de todo sistema de control (Variable, Actuador, *Set Point*) se le agregan ahora:

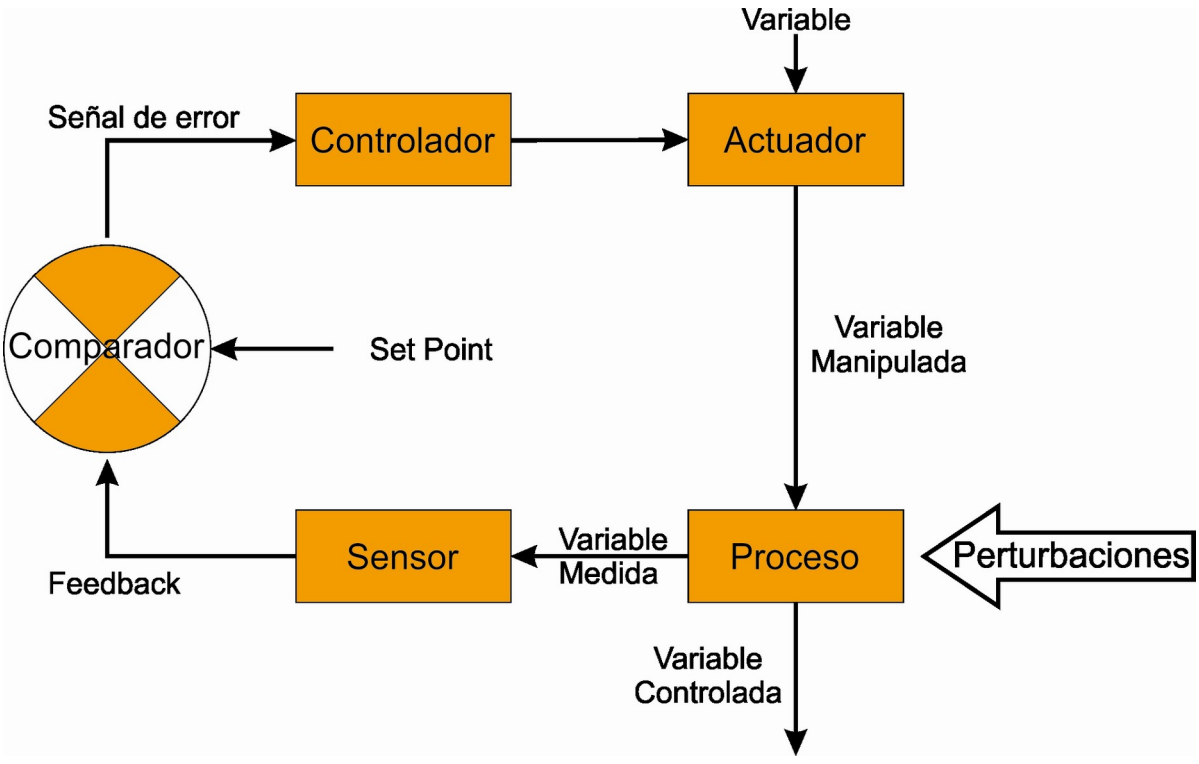
- Un **Sensor** o elemento de medición de la variable
- Un **Comparador** (mecánico, eléctrico, electrónico, digital, etc.) que compara el *Valor Medido* de la variable con el *Set Point* deseado**.**

Esta comparación dará lugar a la llamada **Señal de error**, que es simplemente la diferencia entre el Valor Actual de la variable y el Valor Deseado de la misma. En un sistema digital (Arduino o similar) se implemente mediante una simple resta entre ambos valores. En un sistema mecánico se hace mediante algún dispositivo de palancas y pívot, como en el ejemplo del gráfico, u otro artilugio equivalente.

Nótese especialmente que la palabra Error NO se refiere a que haya nada que esté funcionando mal o cosa parecida. Es simplemente la nomenclatura habitual para referirse a la diferencia entre el valor actual de la variable y el Valor deseado.

Podemos caracterizar más formalmente un sistema de Lazo Cerrado de la forma que se presenta a continuación.

Elementos constituyentes de un sistema de control a lazo cerrado



Cada bloque representa un elemento del sistema y ejecuta una función en la operación de control. Las líneas entre los bloques muestran las señales de entrada y salida de cada elemento, y las flechas, la secuencia de acciones en el orden en que ocurren.

Variable controlada: Se refiere a la variable cuyo valor debe mantenerse igual al de referencia, durante el proceso

Variable medida: Es el valor actual de la variable que se desea controlar. Para hacerlo, es necesario medirlo (sensarlo) en el proceso y compararlo con el valor de referencia.

Sensor: Es el Instrumento (o conjunto de ellos) que mide el valor actual de la variable en el procesor y produce señal/es de salida proporcionales al valor de esta variable.

Señal de retroalimentación: Es la salida del instrumento de medida que se reingresa al sistema, alimentando al comparador

Set point: Es el valor Valor de referencia o valor deseado de la variable controlada

Comparador: Compara el valor de referencia con el valor medido. Puede ser mecánico, electrónico, digital, etc.

Señal de error: Es la salida del Comparador. Representa la diferencia entre el valor deseado y el medido y puede ser positivo o negativo, según la variable medida sea mayor o menor que el valor deseado. Su nombre no debe inducir a pensar que indica cuando algo funciona mal, sino que es la denominación standard usada para este valor.

Actuador: Es el dispositivo que ejecuta las acciones que conducen a la variable controlada a adquirir el valor de referencia

Variable manipulada: Es la variable que se manipula para cambiar las condiciones de la variable controlada. En un horno, la válvula del gas se abre o cierra para cambiar el valor del flujo de gas que alimenta al quemador. Si el flujo aumenta, lo hace también la temperatura, que es la variable controlada.

Perturbación: Es cualquier factor responsable de cambiar el valor de la variable controlada y que está fuera del control del sistema

Controlador: Recibe la señal de error y produce los ajustes necesarios para minimizarla. Para nosotros, usualmente un microcontrolador o una PC corriendo un programa o algoritmo que determine las acciones a tomar.

Control discreto (On/Off)

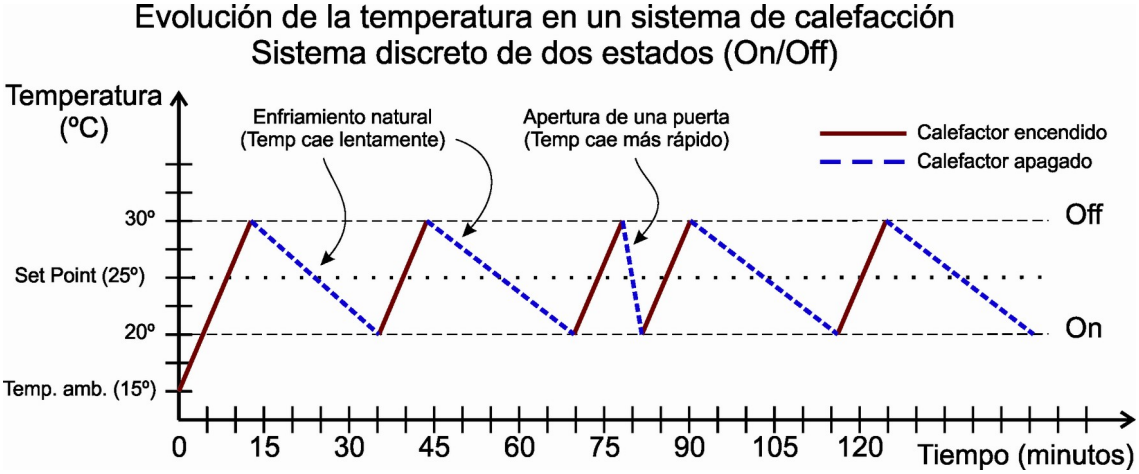
Es un sistema de control de lazo cerrado en el que el actuador tiene solo dos estados (accionado o no, tira o empuja, etc.) En líneas generales, cuando la variable medida supera el valor del *Set Point* el sistema acciona en un sentido y cuando la medición cae por debajo de ese valor, el sistema acciona en el otro sentido. Es el caso de una plancha eléctrica, de la calefacción o de un termotanque hogareño, por mencionar los ejemplos más comunes.

En todos esos ejemplos, cuando la temperatura alcanza un cierto valor el actuador es desconectado (resistencia eléctrica en la plancha y el calefactor, quemador en el termotanque). Cuando la temperatura cruza el *Set Point* en sentido contrario, la acción se invierte, es decir, se conecta el actuador.

Según esta descripción, el dispositivo debería conectarse y desconectarse continuamente mientras la variable mantiene valores cercanos al Set point. Si pensamos por ejemplo en el sistema de calefacción de un aire acondicionado frío/calor, el compresor debería estar permanentemente encendiendo y apagando, tratando de mantener la temperatura exactamente en el valor seteado. Sabemos sin embargo, por experiencia, que esto no es así. Usualmente el ciclo real es el siguiente:

- Al inicio el compresor se activa y la temperatura del ambiente sube,
- El compresor se desactiva al llegar a cierto valor (Supongamos 30º)

- El ambiente se comienza a enfriar pero el compresor no se vuelve a activar apenas a los 29.99º sino que lo hace bastante después
- El compresor se vuelve a activar cuando la temperatura del ambiente es de 20º



Este comportamiento hace que la temperatura, se mantenga no en un valor exacto (por ejemplo 25º) sino en un **Rango**, entre 20º y 30º. SI bien esto es menos preciso, tiene el importante beneficio adicional de la estabilidad.
El actuador no está permanentemente encendiendo y apagando (lo que generaría importante *stress* térmico, desgaste prematuro y gran consumo de energía) sino que actúa por períodos.

Ahora bien, ¿cómo se logra este efecto, si el sensor tiene sólo dos estados (On/Off)? ¿Por qué el sistema es estable y no comienza a oscilar encendiendo y apagando constantemente el actuador alrededor de los 25º C?

Esto se debe a una característica natural que presentan la mayoría de los sensores físicos, y que veremos a continuación.

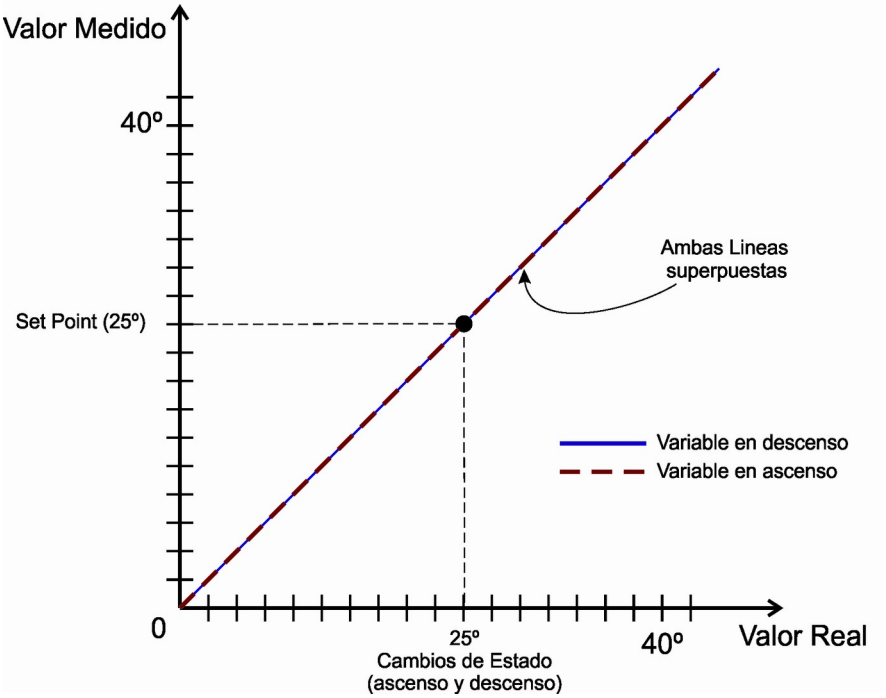
Histéresis

Es una característica presentada por los sensores físicos, a consecuencia de la cual el valor medido por el sensor es diferente si la variable se incrementa progresivamente en un sentido que si lo hace en sentido contrario

En un sensor de dos estados, esto provoca que el cambio de estado no se produzca exactamente en el mismo punto si la variable va en aumento, que si va en disminución.

Para entenderlo mejor, supongamos primero un sensor ideal de temperatura (sin la mencionada característica de histéresis). Construimos un gráfico que representa en el Eje X el valor real de la temperatura y en el Eje Y el valor que mide el sensor. Como se ve, tratándose de un sensor ideal, su curva de funcionamiento tendría la forma de una recta inclinada a 45º, tanto cuando la temperatura va ascendiendo como cuando la temperatura va descendiendo, ya que en ambos casos la temperatura indicada por el sensor es exactamente la misma e idéntica a la temperatura real.

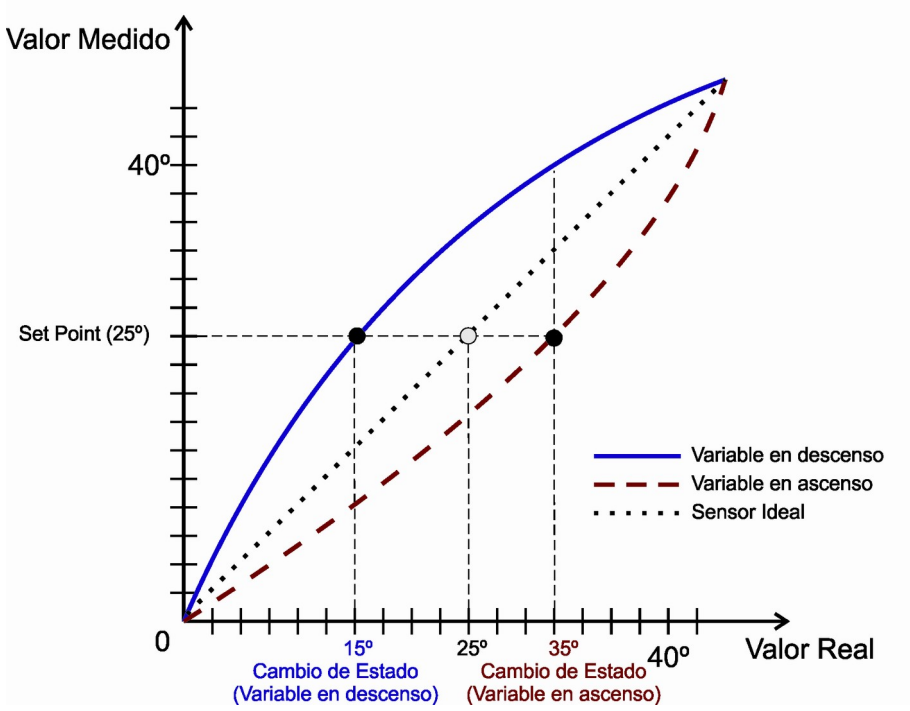
Curvas de funcionamiento de un sensor ideal (sin Histéresis)



Esto implicaría que para un S*et Point* dado, por ejemplo 25º C, el actuador desconectaría apenas la temperatura real supere ese valor y, ni bien la temperatura caiga unas pocas milésimas por debajo de ese valor, se volvería a conectar, entrando en una situación de inestabilidad poco deseable, conectándose y desconectándose continuamente.

En un sensor real, en cambio, la curva de funcionamiento del sensor cuando la variable va en ascenso es distinta a la curva que presenta cuando la variable va en descenso, como se indica en el gráfico siguiente.

Curvas de funcionamiento de un sensor real (con histéresis)



Si bien en este ejemplo se han exagerado los valores a fin de que el fenómeno resulte más evidente, resulta claro que la medición nunca es exacta y que, además, ese error en la medida no es constante sino que depende de si la variable viene creciendo a lo largo del tiempo o si viene disminuyendo.

Cuando la temperatura está en ascenso vale la curva inferior (línea roja de guiones) o sea que para un *Set Point* de 25º C la desconexión se produciría recién a los 35º reales.

Al empezar a enfriarse el sistema, en cambio, la curva que empieza a regir es la superior (línea azul continua) correspondiente a la variable en descenso. Como se ve en el gráfico, el sensor volverá a conectarse recién cuando la temperatura alcance los 15º C y el ciclo se reiniciará.

Gracias a esta característica del sensor, el sistema tendrá un rango de funcionamiento que mantendrá la variable entre 15º y 35º, sin oscilaciones rápidas alrededor del *Set Point*, cosa que, como ya se dijo, podría resultar problemática.

Es decir que una característica indeseable como la imprecisión en la medida y el comportamiento no constante (histéresis) se aprovecha en estos casos para lograr un efecto beneficioso

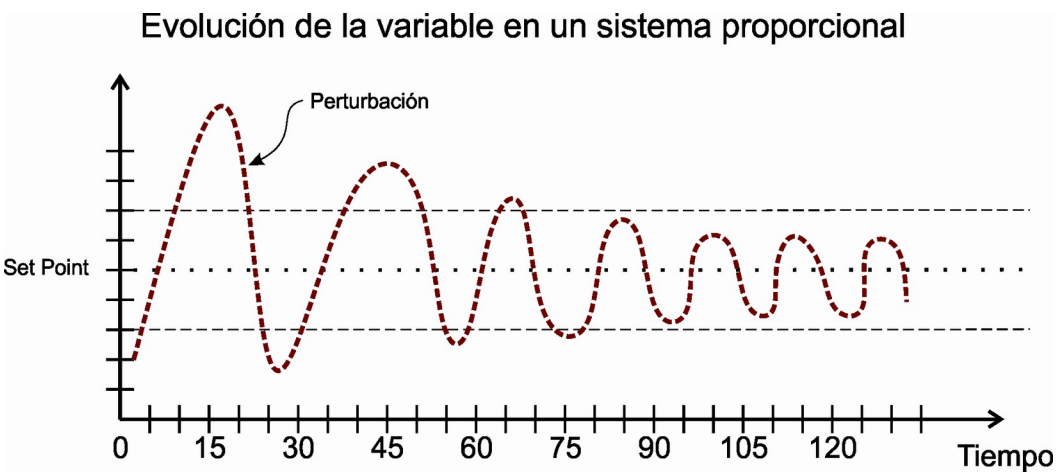
Aplicaciones

Los sistemas de control discretos y sus sensores con histéresis resultan adecuados para fenómenos lentos y de poca precisión, tolerantes a un rango en lugar de un valor exacto y en el cual no haya perturbaciones importantes. Son de bajo costo y de gran estabilidad. Son especialmente aplicables con actuadores que tienen sólo dos estados, encendido / apagado o similar.

Control proporcional

En los controles de tipo proporcional, el actuador debe ser capaz de ejercer una acción variable en intensidad. Esto permite que su acción se pueda regular proporcionalmente con respecto a la señal de error.

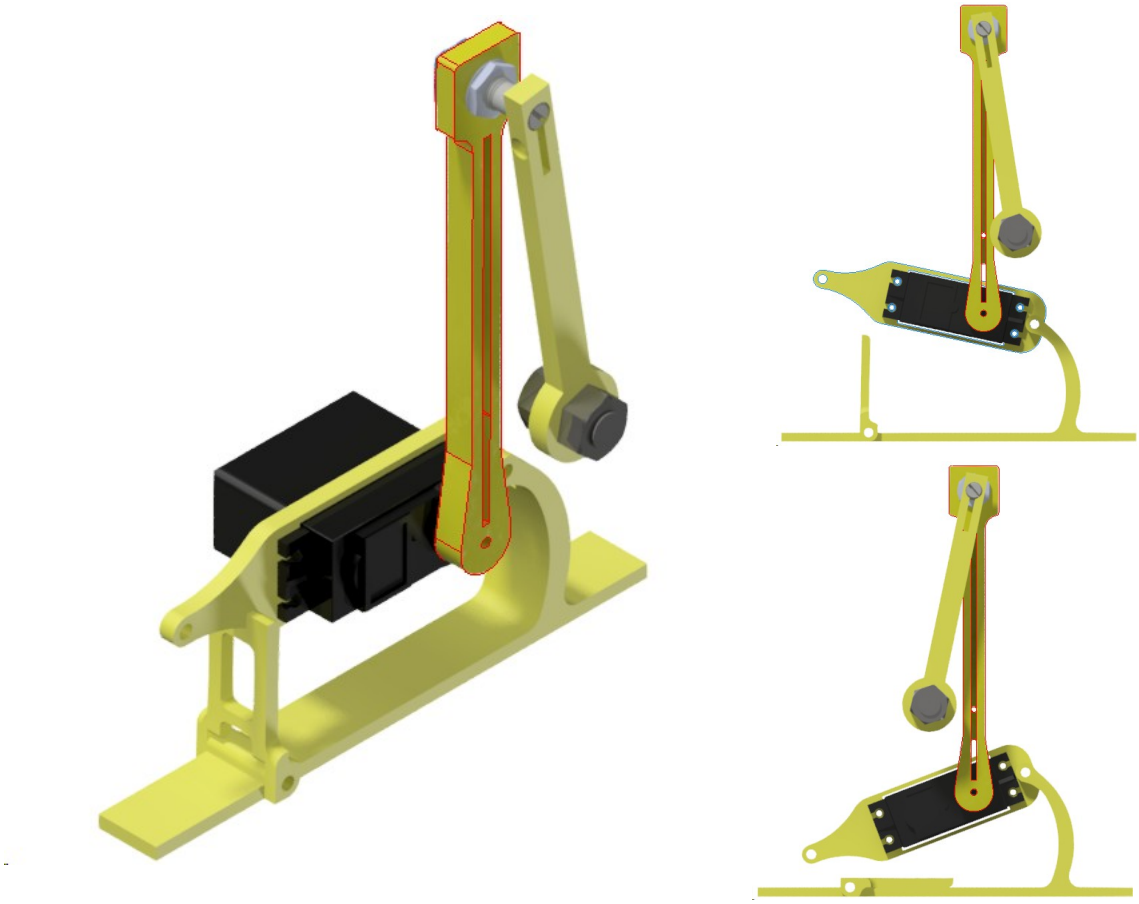
Es decir, si la señal de error (diferencia entre el valor medido y el *Set Point*) es grande, la acción será intensa, para compensar esa diferencia. En cambio, si la señal de error es pequeña, la intensidad de la actuación también lo será, tratando de “pasarse” lo menos posible para el otro lado debido a una acción correctora demasiado intensa. Esto resulta en un comportamiento progresivo que tiende a ir aproximando gradualmente la variable al S*et point*, posiblemente sin alcanzarlo exactamente, pero con muy buena reacción frente a las perturbaciones de magnitud importante.



Resulta particularmente adecuado para fenómenos en los los errores inducidos por las perturbaciones son importantes y en los que el actuador tiene mucha potencia correctora.

Ejemplo de lazo de control Proporcional (P)

Desarrollamos un ejemplo simple de control proporcional, usando como sensor un péndulo rígido, con un peso (bulón y tuerca) en el extremo, adosado al eje de un potenciómetro común. La idea de este ejemplo es controlar la posición de la pieza larga (bordeada con color rojo en las imágenes) de modo que se mantenga vertical cuando movemos con la mano, hacia arriba o abajo, la pieza en la que está montado el servo.



Ver el video explicativo: https://youtu.be/AterXl-AQd0

Programa de Arduino usado en el ejemplo del video:

```
/*
 * Control de un péndulo invertido simple, sensado con potenciómetro
 * utilizando un lazo de tipo Proporcional.

 * Miguel Grassi - www.miguelgrassi.com.ar
*/

#include <Servo.h>

const int pinPote = A1;  // Analog input pin, al que se conecta el potenciómetro
const int pinServo = 16; // Servo output pin

int valorPote = 0;        // valor leido del pote
int valorServo;   // valor a enviar al servo.

// Limites para la carrera del servo
int centroServo=850; // Valor central entre los límites
int rangoServo =600; // Rango total de carrera permitida al servo 400
int limInfServo ;//  Limite Inferior para el servo. Se calcula en el setup, en base a centro y rango
int limSupServo ;//  Limite Superior para el servo. Se calcula en el setup, en base a centro y rango

// Limites para la lectura del potenciómetro
int rangoPote = 435;
int setPoint=512;
```

```
// Variables para el control proporcional
int senalError; // Siendo el actuador un servo, no enviaremos directamente la señal de error al mismo
int correccion; // sino que la calcularemos como una corrección a sumar a la posición actual del servo

Servo miServo;

void setup() {
  miServo.attach(pinServo);  // Atacha el servo con el pin correspondiente
  valorServo=centroServo; // Inicializa la posición del servo en el valor central
  miServo.writeMicroseconds(valorServo);  // y posiciona el servo en ese valor
  //Calcula los Limites del servo para evitar que golpee los extremos
  limInfServo=centroServo -rangoServo; // Determina Limite inferior del servo
  limSupServo=centroServo +rangoServo; // Determina Limite superior del servo
}


void loop() {
  // Leer el valor del pote
  valorPote = analogRead(pinPote);
  // Compara con el setPoint
  senalError=valorPote-setPoint; // EN ESTA LINEA SE RESUELVE TODA LA PROPORCIONALIDAD

  //Restringe los valores a los limites establecidos y calcula la corrección necesaria
  constrain(senalError,-rangoPote/2, rangoPote/2);
  correccion = map(senalError, -rangoPote/2, rangoPote/2, -rangoServo/2, rangoServo/2);
  valorServo=valorServo - correccion;
  if (valorServo>limSupServo) valorServo=limSupServo;
  if (valorServo<limInfServo) valorServo=limInfServo;

  // Envia el valor corregido al servo
  miServo.writeMicroseconds(valorServo);

  // da tiempo  para que el servo llegue a la posición
  delay(300);
}
```

Lazo de Control PID

Como vimos entonces, el control a Lazo cerrado o Control Realimentado se refiere a una operación que en presencia de perturbaciones tiende a reducir la diferencia entre la salida de un sistema y la entrada de referencia -o *Set Point*- de manera continua y automática, tratando de mantener la diferencia por debajo de un margen previamente determinado, Recordamos que la diferencia entre el valor medido y el valor deseado la denominamos “Señal de error”.

Para ello es necesario implementar en el controlador un algoritmo, es decir una serie de pasos y cálculos, que determine las acciones a tomar. Por ejemplo, si estamos controlando la temperatura de un proceso químico y se está enfriando, se trata de determinar cuánto debe abrirse la válvula del gas para mantener la temperatura dentro del rango, sin recalentarlo.

Como se vio, el criterio de proporcionalidad entre la Señal de error y la acción correctora resulta muy eficaz para hacer converger rápidamente un sistema que ha sufrido una perturbación hacia el rango de valores deseado, evitando que caiga en una inestabilidad indeseable.

Sin embargo este modelo no se comporta bien frente a perturbaciones constantes en un determinado sentido o frente a otras, de sentido cambiante, pero que presenten en muy rápida secuencia. Particularmente, si hay una Señal de Error permanente, pero de pequeña magnitud, el control proporcional no la corregirá, pese a que ese error acumulado a lo largo del tiempo podría ser un problema significativo. Para eso hace falta un sistema de control que tenga en cuenta no sólo la magnitud del error, sino su comportamiento a lo largo del tiempo.

En los primeros años del siglo XX se comenzaron a desarrollar sistemas de control que contemplaban esos aspectos. Hacia 1922 un ingeniero ruso que trabajaba para la marina norteamericana buscando desarrollar un sistema de piloto automático para barcos, observó que los timoneles actuaban teniendo en cuenta no sólo al valor de error presente en el rumbo en cada instante, sino la tendencia general de las perturbaciones (por ejemplo un viento cruzado permanente) así como la velocidad de la variación de otras perturbaciones como el oleaje.

En consecuencia desarrolló un modelo matemático que tiene en cuenta, además de el criterio proporcional, la sumatoria (lo que en matemáticas se conoce como Integral) a lo largo del tiempo de las Señales de error que se van registrando y asimismo la velocidad con que varía dicha Señal de error (en matemáticas, la Derivada). Este sistema de control se conoce como Lazo PID, debido justamente a las iniciales de Proporcional, Integral y Derivativo.

A diferencia del Control Proporcional, que considera sólo el valor de la Señal de Error en el instante actual para determinar la acción de corrección que ejercerá el actuador, en un Control PID, es necesario además ir sumando en forma acumulada los valores de Señal de Error medidos y también determinar que tan rápido se presentan los cambios en la Señal de error.

Así, la acción de corrección que ejercerá el actuador estará determinada por tres factores:

- El factor Proporcional (P) que determina la reacción de acuerdo al valor la Señal de Error existente en el instante actual. O sea, la Señal de Error en función del tiempo: e(t)

- El factor Integral (I) considera la reacción basado en la acumulación (sumatoria) de los errores a lo largo del tiempo: $\sum e(t)$
  Esta sumatoria, si se toman valores de tiempo suficientemente cercanos en el tiempo (lo que se conoce como diferencial t, y se indica dt), se denomina Integral y se indica $\int e(t)\,dt$

- El factor Derivativo (D) o sea la velocidad con la que se presentan los cambios en la señal de error, lo que se conoce como Derivada de la Señal de error en función del tiempo y se indica como el cociente $d\,e(t) / dt$

Debe tenerse en cuenta entonces que:

- El control proporcional será insensible a un error de poca magnitud aunque sea permanente

- El control Integral actuará cuando el error –aún sea pequeño- se mantenga a lo largo del tiempo

- El control Derivativo sólo tendrá incidencia si hay cambios en el valor de la Señal de error. Si el mismo es constante, no tendrá influencia (La derivada de una constante es cero)

Por supuesto, cada factor tendrá su signo positivo o negativo, según resulten en una corrección en un sentido o en sentido contrario. La corrección total saldrá de la sumatoria algebraica de los tres (es decir, teniendo en cuenta el signo de cada uno).

Ahora bien, la sumatoria directa de los tres sería válida si los tres tuvieran el mismo grado de importancia, y esto nunca es así. En general el factor más importante para determinar la reacción del sistema de control es el factor Proporcional (o sea, el valor de la Señal de error en el instante actual, indicado como P). Los otros dos factores (I y D) influirán en mayor o menor medida, según el tipo de fenómeno de que se trate y de las perturbaciones que sufra.

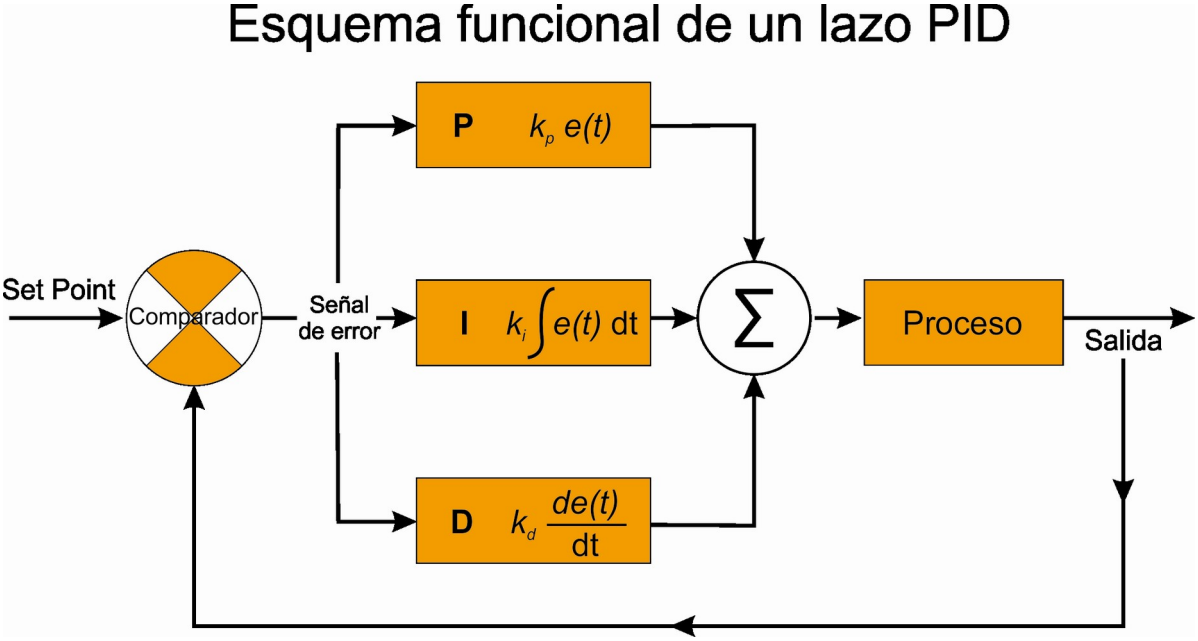
Para establecer cuánto debe influir cada uno de éstos factores en el cálculo del total de la reacción del actuador basta con multiplicar cada factor por un coeficiente k que determinará su peso en el valor total. O sea:

$$\text{Corrección} = k_p P + k_i I + k_d D$$

Qué valor deberá tener cada coeficiente k dependerá absolutamente del tipo de fenómeno que se quiera controlar en cada caso, y se determina empíricamente.

La sumatoria de los tres factores P, I, D -ponderada cada una por una constante Kp, Ki, Kd, respectivamente- determina la acción correctiva final que se tomará sobre la variable controlada. Ajustando las constantes K en el algoritmo se puede conseguir un control muy preciso sobre el grado de respuesta al error, así como la prevención de las sobrecargas y de las oscilaciones bruscas del sistema.

El siguiente esquema representa gráficamente lo expuesto:



Aplicaciones

Este tipo de lazo de control es adecuado para gran cantidad de fenómenos, especialmente aquellos en los que no se conoce demasiado sobre las leyes de variación del sistema y las perturbaciones. Si bien no garantiza la completa estabilidad del sistema, puede acotar el error a un rango suficientemente pequeño en la mayoría de los casos y regresarlo al mismo con muy buena velocidad de respuesta frente a diversos tipos de perturbaciones.

Implementación práctica

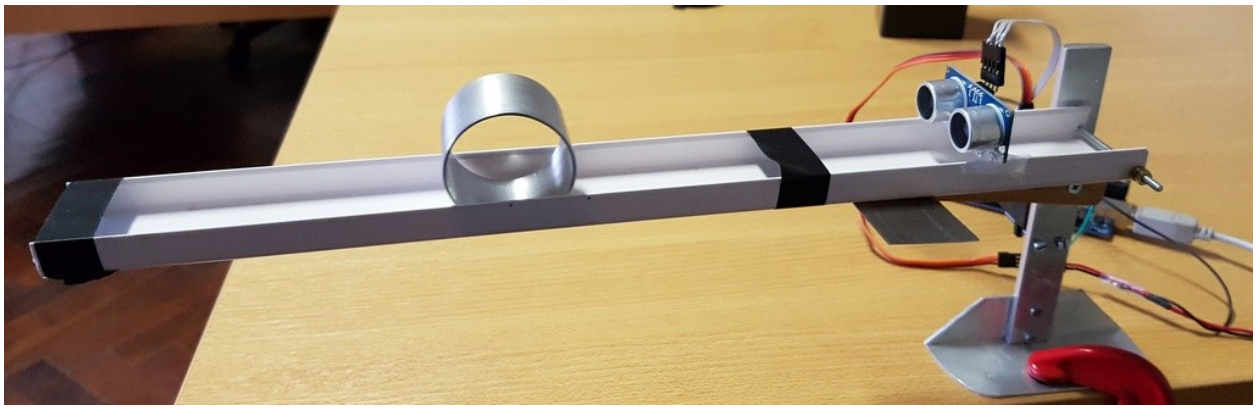
El desarrollo de un algoritmo PID ofrecía un considerable grado de dificultad hasta la aparición de los microcontroladores. Gracias a éstos se ha simplificado enormemente su programación. Existen además diversas librerías para la mayoría de los micros más

comunes que convierten su uso en una tarea muy simple, como es el caso de Arduino, en el siguiente ejemplo.

Se construyó una rampa que puede inclinarse a un lado u otro mediante un servo controlado por una placa Arduino. Sobre esa rampa se desplaza libremente un cilindro pulido que correrá hacia uno u otro lado, siguiendo la inclinación de la rampa.

En un extremo de la rampa, conectado al mismo Arduino, se ubica un sensor ultrasónico tipo SR04, capaz de medir la distancia del cilindro al extremo de la rampa, con una precisión de 1 cm. El sensor se opera mediante la librería NewPing, disponible en http://playground.arduino.cc/Code/NewPing .

El problema consiste en mantener permanentemente el cilindro lo más cerca posible de un determinado valor de distancia (*Set Point*), establecido en este caso en 13 cm, inclinando, mediante el servo la rampa hacia uno u otro lado. Se puede perturbar el sistema dando al cilindro toques con un lápiz u otro elemento similar o soplándolo para desplazarlo de su posición.



Como se ve, en el código, implementar el lazo, en este caso mediante la librería PID_v1 que puede descargarse de http://playground.arduino.cc/Code/PIDLibrary, se reduce simplemente a darle valores tentativos a las tres constantes k y poner el sistema a funcionar, observando su comportamiento.

Se van modificando sobre la marcha los valores de estas constantes según se quiera obtener un comportamiento más agresivo, de convergencia más rápida, mayor sensibilidad, etc. Este proceso se conoce como Sintonizar el lazo PID y se suele considerar valores relativos de 80% para el coeficiente Kp, 15% para el Ki y 5% para el Kd como un buen punto de partida para iniciar el análisis.

```
/*
 *        RampaPID_SR04_Servo_1_0_Centímetros
 *
 * ---------------  Programa de ejemplo de un lazo PID para Arduino  ------------------------
 *
 * Este programa lee, usando un sensor ultrasonico SR04, la posición de un cilindro que rueda
 * en una rampa y posiciona el servo que inclina la rampa para llevar el cilindro al SetPoint
 *
 * Utiliza la Libreía NewPing para el sensor ultrasónico y la librería PID_v1 para el lazo PID
 * http://playground.arduino.cc/Code/NewPing    http://playground.arduino.cc/Code/PIDLibrary
 *
 * Escrito por Miguel Grassi
 * www.miguelgrassi.com.ar
 *
 * V 1.0 2016 Julio 14 - Ultima modificación: 2016 Agosto 10
*/

#include<Servo.h>
#include<PID_v1.h>
#include<NewPing.h>
```

```
#define TRIGGER_PIN  12              // Trigger Pin del Sensor UltraSonico
#define ECHO_PIN 11                  // Echo Pin del Sensor UltraSonico
#define MAX_DISTANCE 35              // Distancia máxima (cm)
const int SERVO_PIN = 50;            // Servo Pin

// Estos son los factores que hay que ajustar para sintonizar el lazo PID:
float Kp = 3;                        // Factor Proporcional
float Ki = 1;                        // Factor Integral
float Kd = 2;                        // Factor Derivativo

int PosHorizontal=1620;              // Valor en ms para el servo que posiciona la rampa exactamente horizontal
int rango=90;                        // Angulo total (min a max) en ms, que debe alcanzar servo  (1º = 5.5 ms aprox o sea que 90 ms son unos 16º ))
int min, max;                        // Valores máximos de angulo superor e inferior del servo calculados simétricamante respecto a la horizontal

double Setpoint, Input, Output;      // Variables para operar con la librería PID

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);          // Inicializa el objeto NewPing para el sensor ultrasónico
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT );  // Inicializa el objeto PID
Servo servo;                                                 // Declara el Servo

void setup() {
 Serial.begin(115200);               // Inicializa el puerto serie
 servo.attach(SERVO_PIN);            // atacha el servo al pin correspondiente
 min = PosHorizontal-rango/2;        // Calcula el valor min del servo
 max = PosHorizontal+rango/2;        // Calcula el valor min del servo
 servo.writeMicroseconds(PosHorizontal);       //Posiciona el servo en el centro (rampa horizontal)
 Input = posicion();                 // Carga en la variable Input la distancia a la que está el clilindro, devuelta por la función posicion()
 Setpoint = 13;                      // Posición deseada en cm
 myPID.SetMode(AUTOMATIC);           // Set PID object myPID to AUTOMATIC (see playground for more details)
 myPID.SetOutputLimits(min, max);    //  Pone los limites del PID
}

void loop()
{
  Input = posicion();             // Carga en la variable Input la distancia a la que está el clilindro, devuelta por la función posicion()
  myPID.Compute();                // Calcula mediante el PID la posición que debe adoptar el servo (entre 1590 y 1680 ms)
  servo.writeMicroseconds(Output); // Le pasa ese valor al servo (writeMicroseconds acepta Entre 1000 y 2000 mS)
  printData(Setpoint, Input, Output);     // Llamo a la function que envía Setpoint, Input, y Output por el puerto Serie
}


//*************************************************************************************************************************
//Función posicion() - Lee la posición del cilindo y retorna ese valor en cm. Evita devolver el valor cero (lectura fuera    // de rango) así como
cualquier valor superor al extremo más lejano de la rampa, para que los valores devueltos sean
// consistentes
//*************************************************************************************************************************
float posicion() {
  delay(36);                          // Un pequeño delay para estabilidad
  int medida;                         // Declaro la variable para leer la medida
  unsigned int uS = sonar.ping_median(4); // leo 4 veces el tiempo que demora al pulso en ir y venir y promedia, para un resultado más estable
  medida = uS / US_ROUNDTRIP_CM;          // Paso el tiempo a cm
  if (medida==0) medida=30;               // Me aseguro que ante una lectura fuera de rango devuelva 30 cm
  if (medida>30) medida=30;               // Me aseguro que ante una lectura mayor a 30 cm no devuelva más que eso
  return medida;                      // Retorna la medida
}


//*************************************************************************************************************************
//Función PrintData() - Envía Setpoint, Input, y Output por el puerto Serial. Puede enlentecer algo la lectura, de modo que
//se puede dejar de invocar una vez que están establecidos los valores y terminada la fase de depuración
//*************************************************************************************************************************
void printData(float set, float in, float out) {
  Serial.print("Setpoint = ");
  Serial.print(set);
  Serial.print(" Input = ");
  Serial.print(in);
  Serial.print(" Output = ");
  Serial.print(out);
  Serial.print("\n");
}
```