Diseño de un Controlador PID con Interfaz Gráfica de Control para un Sistema de Equilibrio Bi-hélice

Alarcón Gacitúa, Gustavo Adolfo
Universidad Católica de Santa María, Arequipa, Arequipa, Perú, gustavoagacitua.89@gmail.com

Lajo Carpio, Daniel Adrian Fernando
Universidad Católica de Santa María, Arequipa, Arequipa, Perú, daniel.lajo89@gmail.com

Faculty Mentor:
Quispe Ccachuco, Marcelo
Universidad Católica de Santa María, Arequipa, Arequipa, Perú, mquispec@ucsm.edu.pe

ABSTRACT

This paper show how a one degree of freedom equilibrium system was controlled, this system consist in a beam with propellers at its ends. It is described the mechanical and electrical features of the controlled process. PID equation is detailed as well as the problems it presents once is implemented in real life. Solutions are shown for these problems while beginner PID algorithm is modified, this algorithm which is going to be implemented in Arduino. After PID algorithm is developed, second Ziegler-Nichols tuning rule is applied to the controller, because we have the physical plant but we do not the mathematical model because its complexity and non-linearity. Once right parameters are obtained, we proceed to develop the graphical interface using software Processing, in which we will draw some angle and the physical system adopt the right behavior to follow the set point. Finally, having the controller design implemented, the interface and the physical plant, we obtain a control module that can be used as an educative tool in the courses of classical control, having used non-expensive resources thanks to open source hardware and software tools.

Keywords: Control, IDE, PID, Arduino, Procesing.

RESUMEN

El presente trabajo muestra cómo se logró controlar un sistema de equilibrio de un grado de libertad, de una barra con propulsores en sus extremos. Se describen las características mecánicas y eléctricas del proceso controlado, se detalla la ecuación clásica del control PID, los problemas que presenta a la hora de ser implementado de manera real, y se plantean las soluciones para estos problemas a la vez que se va modificando el algoritmo de PID principiante, el cual será implementado en Arduino. Una vez desarrollado el algoritmo de PID, se describe el proceso de sintonización mediante el segundo método de Ziegler y Nichols, dado que se tiene la planta física, más no el modelo matemático debido a su complejidad y no linealidad. Obtenidos los parámetros de control adecuados, se procede a desarrollar la interfaz gráfica usando Processing, con la cual ante un determinado ángulo dibujado en la interfaz, el modelo físico sigue el comportamiento adecuado. En conjunto, diseñado el controlador, la interfaz, y teniendo la planta física se obtiene un módulo de control que puede ser utilizado como herramienta educativa para cursos de control clásico, habiendo utilizado pocos recursos gracias a las herramientas de software y hardware libre utilizadas.

Palabras claves: Control, Interfaz, PID, Arduino, Processing.

1. INTRODUCCIÓN

El algoritmo PID es una herramienta de control muy eficaz cuando se desea mantener una variable en un valor determinado. Este tipo de control resulta ser muy eficiente con los ajustes adecuados, pero son estos mismos los que hacen de este algoritmo un reto a la hora de aplicarlo en ciertos sistemas o plantas, es por ello que surgen distintos métodos de sintonización para plantas cuyos modelamientos matemáticos son muy complejos o poco prácticos. Por otro lado al ser enseñada esta área de control en las casas de educación de ingeniería, no siempre se cuenta con dispositivos físicos que ayuden a consolidar los conceptos aprendidos.

Es por ello que se decidió diseñar un controlador PID para un sistema de equilibrio bi-hélice, en el cual se pueda observar, sin necesidad de instrumentos, el significado físico de muchos conceptos tales como estabilidad, sobre-impulso, tiempo de establecimiento, error de estado estacionario, estabilidad marginal, entre otros. De manera que este sistema pueda ser utilizado como elemento auxiliar a la hora de enseñar las citadas técnicas y teoría de control. Por estas razones se ve la necesidad de tener un sistema eficiente pero sin necesidad de gastar muchos recursos, por ello se decidió usar hardware y software de fuente abierta, que además de ser económicos, se pueden modificar para requerimientos específicos.

2. DESCRIPCIÓN DEL PROCESO A CONTROLAR

La planta a controlar consta de una barra pivoteada en su centro de inercia (1 GDL), la cual, apoyada en una columna a través de un eje, es capaz de oscilar libremente.

Se desea controlar el ángulo de inclinación de esta barra, permitiendo así visualizar como se comporta el sistema ante un determinado ángulo de referencia, de manera que se pueda análisar de forma dinámica, cuando es que el sistema superó el estado deseado, en cuanto tiempo lo alcanzó y como se comportaría ante una eventual perturbación.

A los extremos de la barra se colocaron dos motores de corriente continua con hélices en sus ejes, generando asi un determinado empuje según la velocidad angular a la que trabajan los motores. En la parte superior central de la barra se colocó un acelerómetro para sensar el angulo de inclinación en cada instante de tiempo programado.

- Descripción Mecánica: Se tiene una base rectangular de madera de 600 mm de largo por 80mm de ancho y 10mm de espesor, que sirve de soporte mecánico para los dispositivos electrónicos de control y potencia, en el centro de la superficie superior se alza una columna de 200mm de alto, en su extremo superior se pivotea una barra de 400mm de largo y la cual como ya se mencionó, alojará a los motores DC y el respectivo acelerómetro. En la figura 1 se muestra el modelo diseñado en Autodesk Inventor 2013, y en la figura 2 el prototipo real construido.

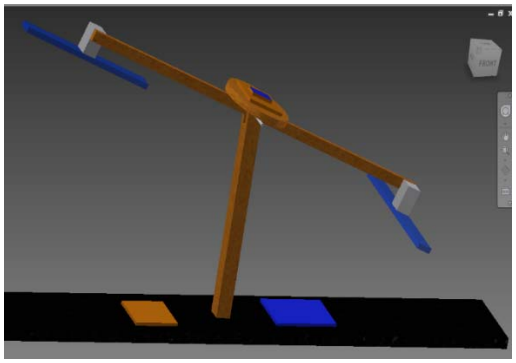

Figura 1: Modelo en Autodesk Inventor 2013

Figura 2: Prototipo Real

- Descripción Eléctrica: Cada motor de corriente continua, que hará las veces de impulsor, opera a 2.2 Vdc, cada uno estará gobernado por un transistor darlington NEC C4811 [7], que permiten operar la suficiente cantidad de corriente a un voltaje reducido. Su colector estará conectado al polo negativo del motor, su

base por medio de una resistencia de 1KΩ recibirán las señales de PWM (8 bits) provenientes Arduino, y el emisor de los mismos se conectará a tierra.

- Descripción Electrónica: Como hemos mencionado, se necesitan herramientas que sean económicas y a la vez eficientes, por ello encontramos en la plataforma Arduino una amplia gamma de controladores con diversas características. Elegimos el Arduino Leonardo [4], debido a que manteniendo un costo bajo, permite una gran velocidad de adquisición de datos (15000 muestras por segundo) 50% más rápido que el Arduino Uno a una resolución de 10 bits, que en conjunto con el acelerómetro, nos permiten obtener precisión de 0.36° grados sexagesimales. Además posee en su microcontrolador Atmega32u4 la capacidad de comunicación serial por puerto USB, sin necesidad de un conversor RS232 intermediario.

  Esto nos da la facilidad y ventaja de crear una interfaz de comunicación sencilla entre software y hardware a la vez que esta comunicación mantiene un buen nivel de solidez.

  El mismo Arduino Leonardo, gracias a su regulador de voltaje, suminstrará poder al acelerómetro, el cual devolverá su señal a Arduino quien la recogerá y la discretizará por medio de sus entradas analógicas en función de su resolución y taza de muestreo ya descritas.

3. ECUACIÓN DEL PID

De la documentación revisada sobre sistemas de control relacionados con el control PID [1][3][6], podemos destacar la siguiente ecuación:

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(t)dt + K_p T_d \frac{de(t)}{dt}$$

De la ecuación y diagrama podemos hacer las siguientes afirmaciones y llegar a un código principiante:

- **e(t)**: Error de la señal.
- **u(t)**: Salida del controlador y entrada de control al proceso.
- **Kp**: Ganancia Proporcional.
- **Ti**: Constante de Tiempo Integral
- **Td**: Constante de Tiempo Derivativo

3.1 CÓDIGO PRINCIPIANTE

```
/* Variables utilizadas en el controlador PID. */
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastErr;
double kp, ki, kd;
void Compute()
{
       /* Cuanto tiempo pasó desde el último cálculo. */
       unsigned long now = millis();
       double timeChange = (double)(now - lastTime);
       /* Calculamos todas las variables de error. */
       double error = Setpoint - Input;
       errSum += (error * timeChange);
       double dErr = (error - lastErr) / timeChange;
       /* Calculamos la función de salida del PID. */
       Output = kp * error + ki * errSum + kd * dErr;
       /* Guardamos el valor de algunas variables para    el próximo ciclo de cálculo. */
       lastErr = error;
       lastTime = now;
}
/* Establecemos los valores de las constantes para la sintonización. */
void SetTunings(double Kp, double Ki, double Kd)
{
```

```
kp = Kp;
ki = Ki;
kd = Kd;}
```

4. DISEÑO DEL CONTROLADOR

El diseño de nuestro controlador parte del PID de nivel principiante, el cual depuraremos hasta obtener un controlador de mejores prestaciones.

4.1 PROBLEMA: PERIODO DE FUNCIONAMIENTO

Los PID de nivel principiante, están diseñados para ejecutarse a periodos irregulares, esto puede traer 2 problemas:

- Se tiene un comportamiento inconsistente del PID, debido a que en ocasiones se lo ejecuta regularmente y a veces no.
- Hay que realizar operaciones matemáticas extras para calcular los términos correspondientes a la parte derivada e integral del PID, ya que ambos son dependientes del tiempo.

4.1.1 LA SOLUCIÓN

Asegurar el funcionamiento periódico del PID basado en un tiempo de ejecución predeterminado. El PID decide si debe hacer cálculos o retornar de la función. Una vez que nos aseguremos que el PID se ejecuta a intervalos regulares, los cálculos correspondientes a la parte derivada e integral se simplifican.

```
int SampleTime = 100; // Configuramos el tiempo de muestreo en 0,1 segundo.
void Compute()
{
unsigned long now = millis();
int timeChange = (now - lastTime);
// Determina si hay que ejecutar el PID o retornar de la función.
if(timeChange>=SampleTime)
{ …
…}
void SetTunings(double Kp, double Ki, double Kd)
{
double SampleTimeInSec = ((double)SampleTime)/1000;
kp = Kp;
ki = Ki * SampleTimeInSec;
kd = Kd / SampleTimeInSec;
}
void SetSampleTime(int NewSampleTime)
{
if (NewSampleTime > 0)
{
/* Si el usuario decide cambiar el tiempo de muestreo durante el funcionamiento, Ki y Kd tendrán que
ajustarse para reflejar este cambio. */
double ratio = (double)NewSampleTime / (double)SampleTime;
ki *= ratio;
kd /= ratio;
SampleTime = (unsigned long)NewSampleTime; }}
```

4.2 EL PROBLEMA: DERIVATIVE KICK

La modificación desarrollada a continuación, cambiará levemente el término derivativo con el objetivo de eliminar el fenómeno "**Derivative Kick**".

Este fenómeno, se produce por variaciones rápidas en la señal de referencia r(t), que se magnifican por la acción

derivativa y se transforman en componentes transitorios de gran amplitud en la señal de control.

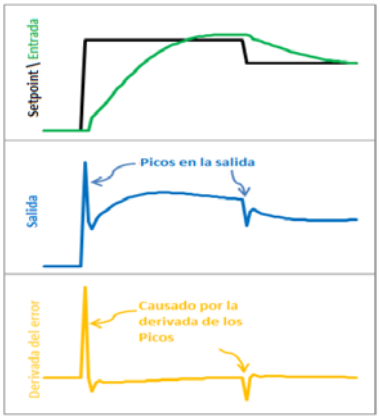

Figura 3: Derivative Kick

4.2.1 LA SOLUCIÓN

$$\frac{derror}{dt} = \frac{dSetpoint}{dt} - \frac{dInput}{dt}$$

Entonces, cuando el setpoint es constante.

$$\frac{dError}{dt} = -\frac{dInput}{dt}$$

```
double dInput = (Input - lastInput);
// Calculamos la función de salida del PID.
Output = kp * error + ki * errSum - kd * dInput;
…
// Guardamos el valor de algunas variables para el próximo ciclo de cálculo.
lastInput = Input;
…
```

4.3 EL PROBLEMA: CAMBIOS EN LA SINTONIZACIÓN

La posibilidad de cambiar la sintonización del PID, mientras el sistema está corriendo, es la característica más destacable del algoritmo del sistema de control.

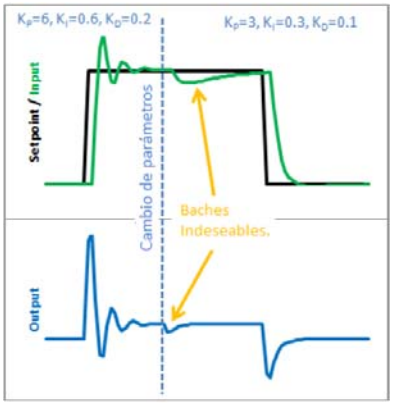

Figura 4: Ruido introducido por re-sintonización

Los PID principiantes, de hecho, actúan de forma errática si queremos configurar los valores de sintonización

mientras el sistema se encuentra en marcha.

La siguiente tabla muestra el estado del PID antes y después de que los parámetros han cambiado.

Tabla 1: Estado del PID antes y después de los cambios de parámetros

	Salida =	Kp *	Error +	Ki *	errSum -	Kd*	dInput
Justo antes	0.98	6	-0.01	0.6	1.73	-0.2	0.02
Justo después	0.49	3	-0.01	0.3	1.72	-0.1	-0.01

La salida se reduce a la mitad debido a que el término integral se reduce rápidamente, esto sucede debido a la interpretación de la integral. Esta interpretación funciona bien hasta que Ki cambia. Entonces, la suma de todos los errores se multiplica con el valor de Ki, siendo esto algo que definitivamente no queremos. Nosotros solo queremos que afecte a los valores que estén por adelante del momento de la sintonización.

4.3.1 LA SOLUCIÓN

Definimos una variable más llamada ITerm, en donde almacenaremos el resultado $(Ki * error)$. Este arreglo nos permitirá mantener la trama correspondiente de errores multiplicados por su Ki correspondiente.

```
double ITerm
…
ITerm += (ki * error);
…
Output = kp * error + ITerm - kd * dInput;
```

4.4 EL PROBLEMA: RESET WINDUP

El efecto WindUp aparece al arrancar el sistema o en cualquier otra situación, donde aparece un error muy grande durante un tiempo prolongado. Esto hará que el término integral aumente para reducir el error. Pero si nuestro actuador es limitado, con esto nos referimos a que la tensión que podemos aplicar se encuentra entre los 0 y 5 voltios (0 a 255, modulación por ancho de pulso de 8 bits), se saturará, pero el termino integral seguirá creciendo. Cuando el error se reduce, la parte integral también comenzara a reducirse, pero desde un valor muy alto, llevando mucho tiempo hasta que logre la estabilidad, generando fluctuaciones exageradamente grandes.

Figura 5: Efecto WindUp

4.4.1 LA SOLUCIÓN – PASO 1: RESTRICCIÓN DEL TÉRMINO INTEGRAL

Se configura el PID con los límites de salida mediante la creación de la función SetOutputLimits. Esta función controla el crecimiento del término integral cuando este alcance el límite configurado.

4.4.2 LA SOLUCIÓN – PASO 2: RESTRICCIÓN DE LA SALIDA DEL CONTROLADOR

Aunque el término integral ha sido acotado de forma segura, el término *Proporcional* y *Derivativo* añade pequeños valores, dando un resultado superior al límite de salida. Esto es inaceptable. Dado que si el usuario llama a la función SetOutputLimits, este asumirá que la salida se encontrara dentro los límites configurados. Así que el paso 2 consistirá en una suposición valida. Además de la restricción del término *Integral*, hay que acotar el valor de salida para que se mantenga dentro de los límites.

La modificación en resumen, sería la siguiente:

```
void SetOutputLimits(double Min, double Max)
{
if(Min > Max) return;
outMin = Min;
outMax = Max;
if(Output > outMax) Output = outMax;
else if(Output < outMin) Output = outMin;
if(ITerm> outMax) ITerm= outMax;
else if(ITerm< outMin) ITerm= outMin;
}
```

El resultado de la programación del controlador PID se encuentra de forma completa en Sketch Path [10]. En donde se solucionó un problema más: “Encendido y Apagado del PID”, que a su vez resultaba extenso para ser detallado en esta sección.

5. SINTONIZACIÓN DE LOS PARÁMETROS DEL PID

Para sintonizar las ganancias de nuestro controlador PID, se utilizaron los datos de adquisición del sensor, una placa Arduino Uno en paralelo con la placa Leonardo (PID) y el programa Oscilloscope de Sofian Audry [8], el código fuente del programa de adquisición será proporcionado vía Sketch Path [11].

5.1 SEGUNDO MÉTODO DE ZIEGLER-NICHOLS

Para el segundo método [6], que se realiza con el controlador trabajando en lazo cerrado, se deben poner a cero las ganancias integral y derivativa, de modo que el controlador actúe solo como un proporcional, aumentamos hasta obtener una oscilación sostenida (estabilidad marginal).

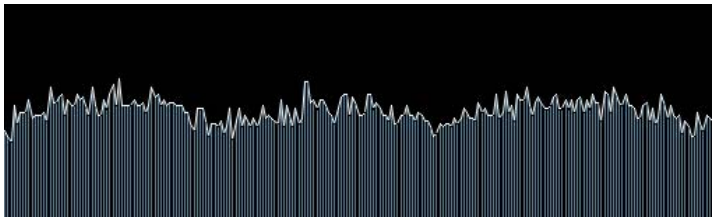

Figura 6: Respuesta oscilatoria del sistema con ganancia crítica

Tomando el tiempo de periodo de cada oscilación (Pcr) y la ganancia crítica (Kcr) del sistema con la cual se llegó a este estado, podemos conocer los parámetros del controlador diseñado.

Tabla 2: Método de Sintonización Basado en la Ganancia Crítica y el Periodo Crítico

Tipo de Controlador	Kp	Ti	Td
P	0.5 Kcr	∞	0
PI	0.45 Kcr	0.0833Pcr	0
PID	0.6 Kcr	0.5Pcr	0.125 Pcr

De nuestra adquisición de datos obtuvimos: Kcr = 0.9 y Pcr = 1.5

Kp = 0.54, Ki = 0.72, Kd = 0.1013

Encontrados los parámetros de sintonización, podemos implementarlos en nuestro código del controlador, y ver el comportamiento de la planta.

6. INTERFAZ DE CONTROL

La interfaz de control para el sistema de equilibro, fue desarrollada en el entorno de programación Processing[5], el cual nos permite crear fácilmente diferentes geometrías y figuras junto con sencillos métodos de comunicación entre arduino y el programa realizado, siendo una de sus mejores caracteristicas el de ser open-source.

La interfaz de control se divide en 3 secciónes:

1. Sección de Mando
2. Sección de Configuración de Parametros en tiempo real.
3. Sección de Visualización de Retroaliemntación.

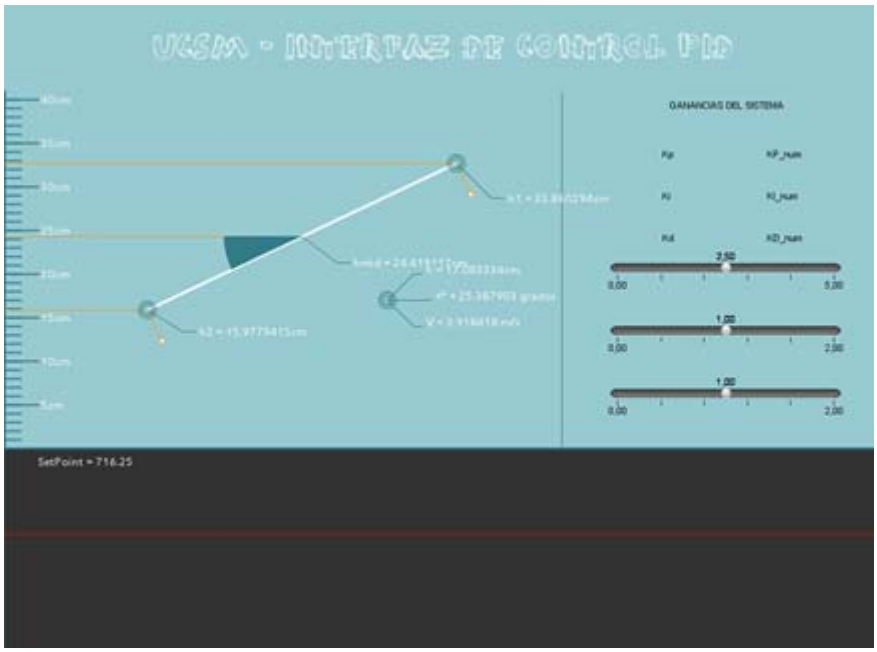

Figura 5: Interfaz de Control

El objetivo de esta interfaz de control es brindar una forma sencilla y dinámica de configuración del setpoint junto con los valores de sintonización. El modo de trabajo es mediante un click en la sección 1 y luego arrastrando ese punto hasta conseguir el ángulo de referencia deseado. Este dato se genera en tiempo real, mientras se manipula la distancia e inclinación de sistema dibujado. Una vez soltado el click, el setpoint es envíado por vía Serial hacía Arduino.

Los valores de sintonización puede ser modificados mediante los sliders de la sección 2.

En la programación se utilizo la librería de comunicación Serial provista por la página oficial de Processing y la librería G4P (guicomponents) para el desarrollo del entorno gráfico de usuario.

El proceso de envío de información hacia arduino es mediante flags de activación, los cuales estan dados por letras (s: setpoint, p: ganancia proporsional, i: ganancia integrativa, d: ganancia derivativo) las cuales viajan en formato BYTE teniendo que ser tratas con el siguiente filtro para su correcto manejo.

```
if(Serial.available() > 0){
    lectura = Serial.read();
    switch(lectura){
    case 's':
      stpnt = Serial.read();
      if(stpnt > 36){
        stpnt = stpnt - 256;
      }
      Setpoint = constrain(stpnt, -36, 36);
      Setpoint = map(Setpoint, -36, 36, 730, 620);
      break;
       …
```

Esta sección de código la podemos interpretar como los límites de trabajo del controlador, los cuales traducidos a processing, equivalen a -20° y 20° de inclinación.

Los detalles de diseño y consideración de factores para dibujo y comportamiento de la interfaz escrita en processing se encuentran en Sketch Path [12][13].

7. RESULTADOS:

Se logró controlar el proceso descrito mediante la implementación del algoritmo descrito en la plataforma Arduino. Conseguimos tener un sistema estable, con tiempos de establecimiento de 1.5 segundos, una recuperación ante perturbación de 2 segundos, una estabilidad ante distintas condiciones de procesos. Observamos que el primer método de Ziegler Nichols no es aplicable en esta planta, y que vía el segundo método de Ziegler-Nichols, sí se llegó a los parámetros desados. El comportamiento de la planta con el controlador implementado, se pueden observar en video por el siguiente enlace:

- http://www.youtube.com/watch?v=WEN918NJ87Y&feature=share

8. CONCLUSIONES:

Se pudo diseñar un controlador PID de mejores prestaciones al principiante con el cual se controló de forma eficiente el sistema de equilibrio planteado. Se consiguió la elaboración de una planta de bajo presupuesto, que pueda ser utilizada en el reforzamiento de la teoría de control y la que pudo establecer perfecta comunicación con Processing, un lenguaje de programación orientado al diseño y el arte, pero que para este caso, decidimos explotarlo como medio de desarrollo de interfaces innovadoras y sencillas.

Hemos visto un gran potencial en Processing y Arduino, como herramientas de investigación, pues pueden brindar muchas soluciones de bajo presupuesto a problemas de pre-grado como post-grado. Dependiendo solo de la habilidad y la creatividad para obtener el máximo potencial de estas plataformas.

9. AGRADECIMIENTOS:

Agradecemos a Dios todopoderoso y a Santa María patrona de nuestra universidad, a nuestros docentes por su desinteresado apoyo. Y nuestro reconocimiento y agradecimiento al profesor Seki Kensaburo, por ayudaros en el desarrollo e inspiración de este proyecto y brindarnos su tiempo y paciencia en la enseñanza de las técnicas de control.

10. REFERENCES

[1] Bolton William (2008). Mechatronics, 4th edition, Pearson Education Limited.
[2] Brett Beauregard. Librería Arduino PID, Obra liberada bajo licencia Creative Commons by-nc-sa.
[3] Dorf Richard C, Bishop Robert (2005). Sistemas de control Moderno, 10ª edición, Pearson educación S. A.
[4] Home page. www.arduino.cc
[5] Home page. www.processing.org
[6] Katsuhiko Ogata (2003). Ingeniería de Controla Moderna, 4ta edición, Pearson Educación, S. A. Madrid.
[7] NEC Corporation, Document N°. D15602E2V0DS00 (2nd Edition) – Datasheet
[8] Project Accrochanges: http://accrochages.drone.ws
[9] Resultado de Sintonización y puesta en marcha: http://www.youtube.com/watch?v=WEN918NJ87Y
[10] Sketchpad1: http://studio.sketchpad.cc/static/uploaded_resources/p.2097/PID_alpha_Arduino.pde
[11] Sketchpad2: http://studio.sketchpad.cc/static/uploaded_resources/p.2097/Oscilloscope_processing.pde
[12] Sketchpad3: http://studio.sketchpad.cc/static/uploaded_resources/p.2097/PID_Interfaz_Ganancias.pde
[13] Sketchpad3.1: http://studio.sketchpad.cc/static/uploaded_resources/p.2097/gui.pde

Authorization and Disclaimer