Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingeniería y Diseño Industrial

Grado en Ingeniería Electrónica Industrial y Automática
Regulación Automática

Controlador con arduino de la velocidad de un ventilador de pc

Alejandro Casanova Martín (53872)
EE309
Enero 2020

Índice

Introducción: **2**

Descripción de la planta a controlar: **2**

Montaje: 2

Respuesta ante un escalón de la cadena abierta: 2

Modelo: 3

Descripción del sistema de control: **4**

Montaje y Funcionamiento: 4

Video demostrativo: 6

Implementación Software: 6

Equivalente del Sistema de Control: 9

Resultados de la Simulación: 10

Resultados Experimentales: 10

Conclusiones: **11**

Bibliografía: **11**

Introducción:

La idea del trabajo ha sido implementar con arduino un controlador que mantenga un ventilador de pc funcionando a una velocidad estable (1500 rpm). Para fines demostrativos se utilizará un segundo ventilador que, trabajando a favor o en contra del primero, actúe como perturbación.

Descripción de la planta a controlar:

Montaje:

La planta consiste en un ventilador de pc de 12V. Para poder controlar su velocidad con los 5V que nos proporcionan los pines de la placa arduino, se ha utilizado un MOSFET **IRLZ44Z** y una fuente de alimentación con salida de 12V. Se ha elegido específicamente el transistor mencionado por su alta velocidad de respuesta y por trabajar a tensiones V_{GS} entre 0 y 5 voltios. Conectando el ventilador como se muestra en la *Figura 1*, se puede regular su velocidad variando el ancho de pulso de la señal PWM introducida en la puerta del transistor

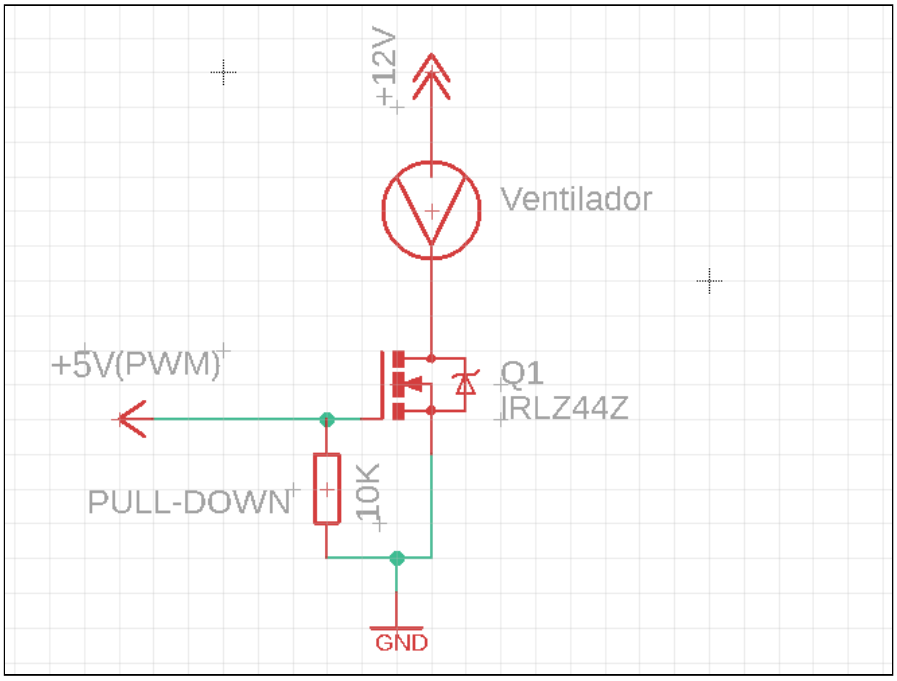

Figura 1. Montaje de la planta

Respuesta ante un escalón de la cadena abierta:

Se ha alimentado el sistema con un escalón de tensión máxima (5V sin PWM) para observar su respuesta temporal sin realimentación. Midiendo la velocidad del ventilador

cada 0.1 segundos desde el reposo hasta el régimen permanente, se ha desarrollado la siguiente gráfica:

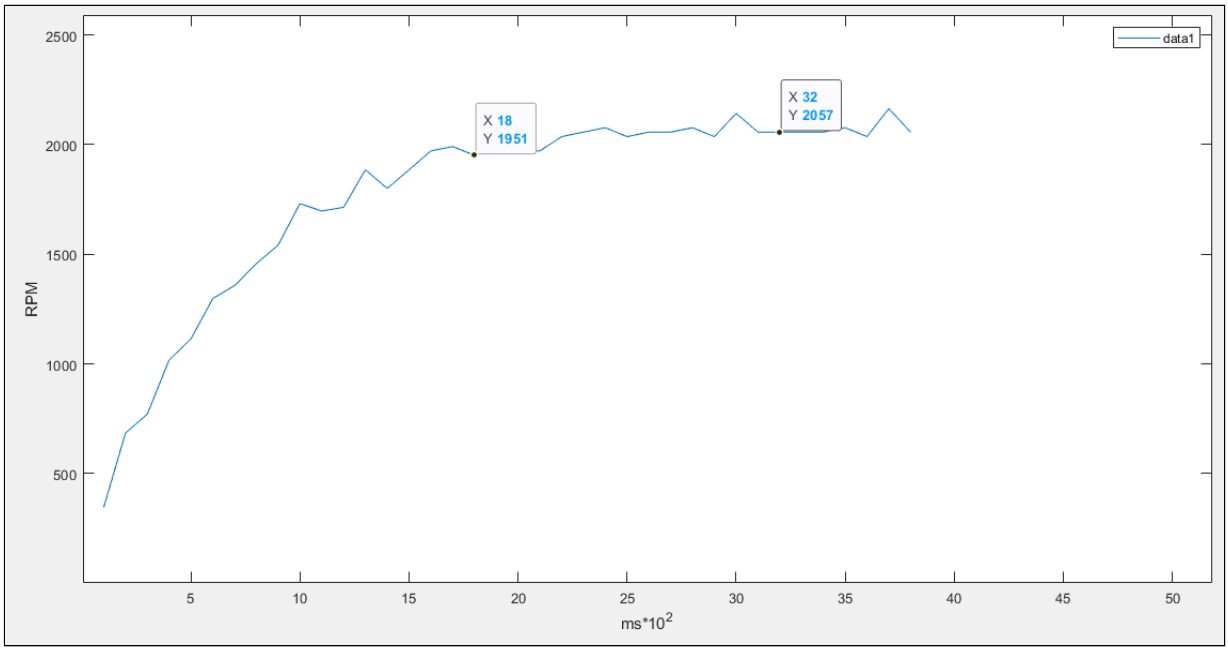

Figura 2. Respuesta al escalón de la cadena abierta

Se puede observar que el sistema se puede aproximar a un primer orden con ganancia estática **Ke = 2050 rpm** y tiempo de establecimiento **ts=1,8** segundos.

Modelo:

Para un tiempo de establecimiento de 1,8 segundos se obtiene una constante de tiempo de aproximadamente 0,6 segundos. Por lo tanto la planta puede modelarse mediante la siguiente función de transferencia:

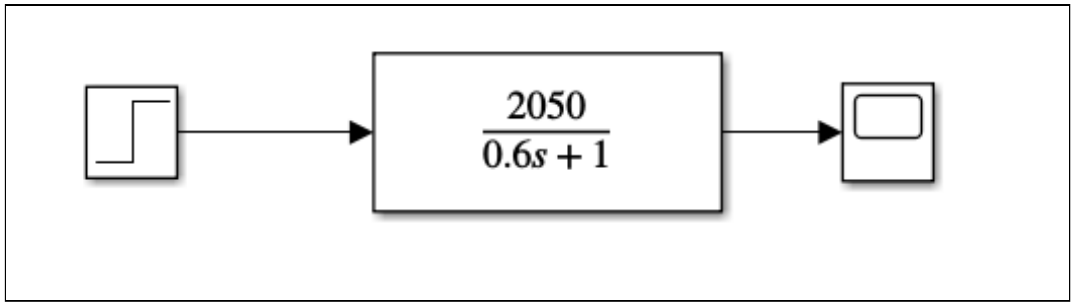

Figura 3. Modelo de la planta

Realizando la simulación en simulink se puede comprobar que la respuesta al escalón del modelo se corresponde con la respuesta observada experimentalmente.

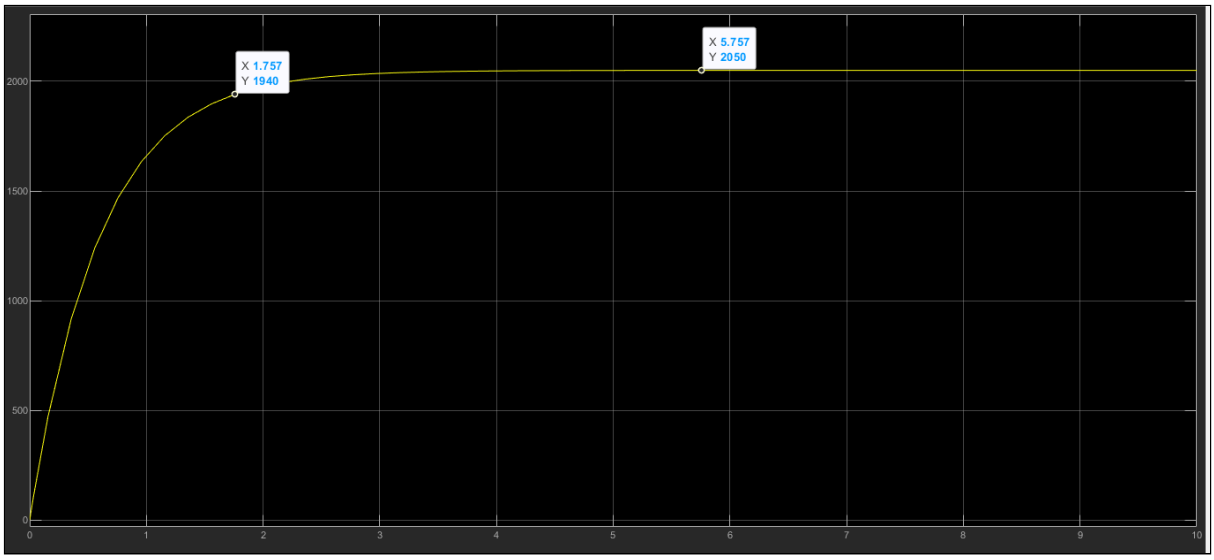

Figura 4. Simulación de la respuesta ante un escalón del modelo de la planta

Descripción del sistema de control:

Montaje y Funcionamiento:

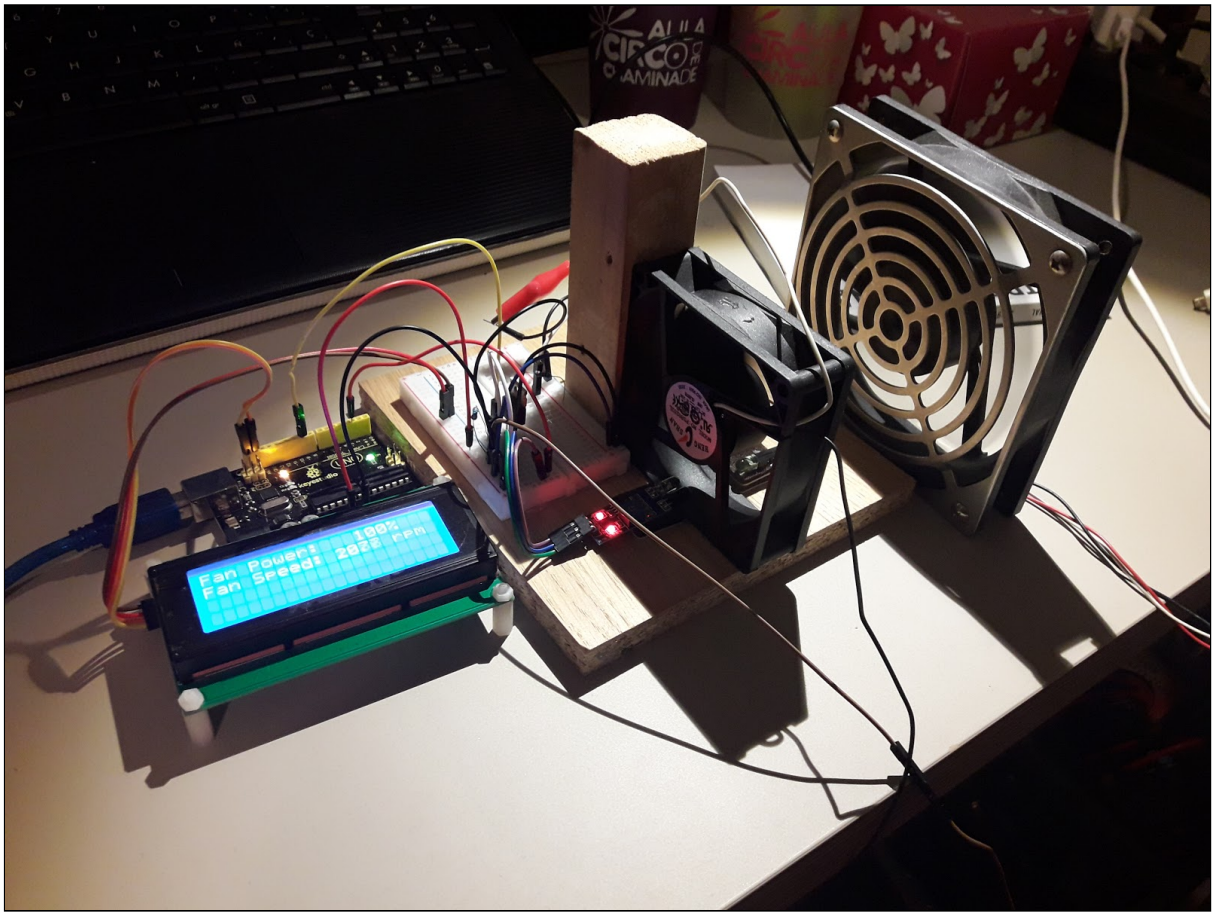

Figura 5. Maqueta del sistema de control

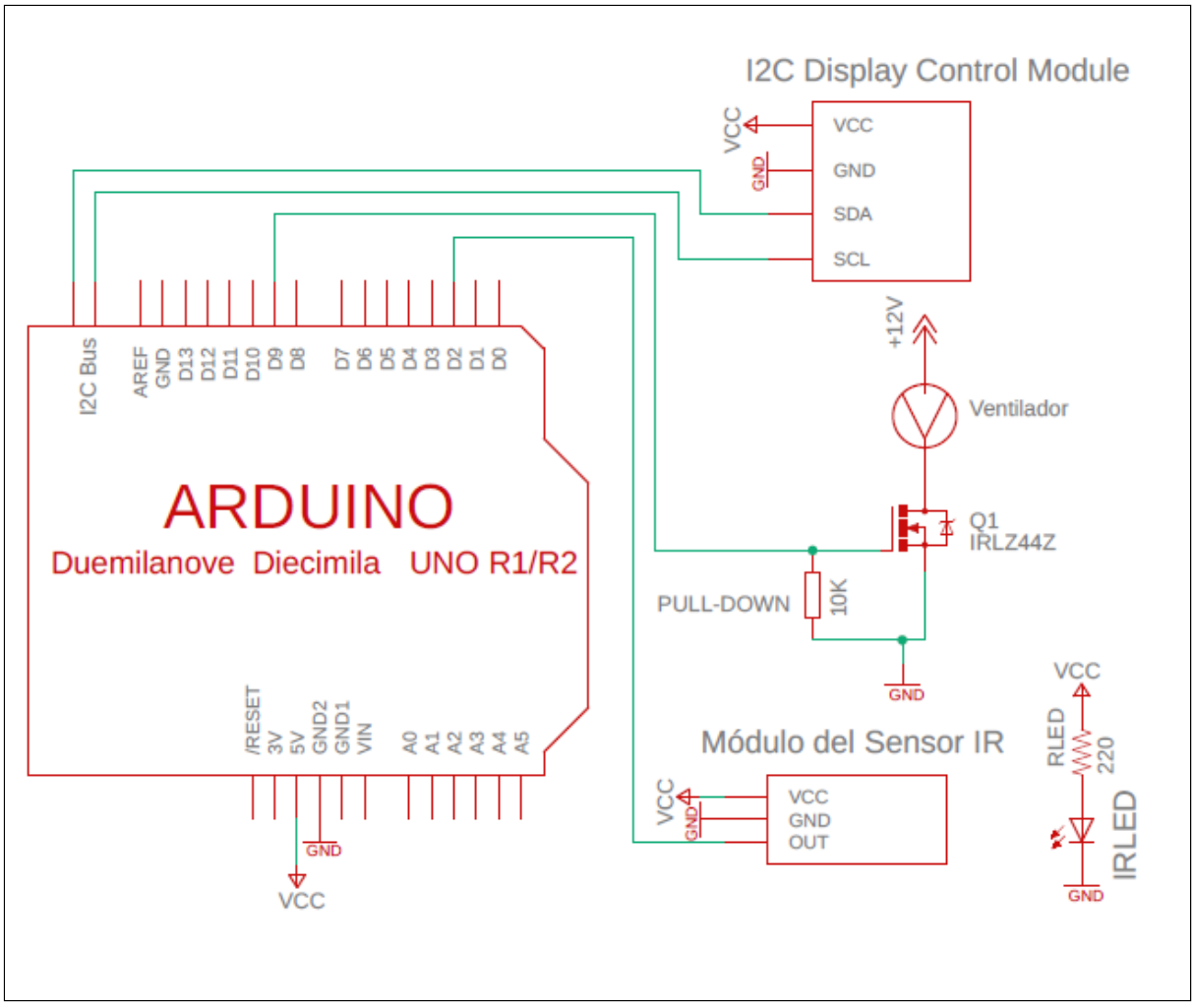

Figura 6. Esquema de Montaje del Sistema de Control

El sistema de control consiste en una placa Arduino UNO, un display lcd con módulo de control I2C, un módulo de sensor infrarrojo con salida digital, y un LED infrarrojo.
A un lado del ventilador se ha dispuesto el sensor infrarrojo y en el lado opuesto el led, de forma que por cada interrupción del haz infrarrojo, la señal de salida del sensor sufrirá un flanco de subida y otro de bajada. Se calculará la velocidad del motor contando el número de flancos de subida o bajada de la señal en un determinado intervalo de tiempo.

La velocidad del ventilador ha sido controlada variando el ancho de pulso (PWM) de la señal introducida en la puerta del MOSFET. *(véase Figura 1.)*

Simultáneamente, el sistema medirá la velocidad del ventilador, ajustará la señal PWM según corresponda para seguir la referencia, y mostrará en el display tanto la velocidad medida (r.p.m.) como el ancho de pulso de la señal de entrada (%). Se verá en la descripción del código que el control del sistema es por ciclos. La duración de cada ciclo vendrá determinada por el tiempo de muestreo utilizado para la medida de velocidad del ventilador.

Video demostrativo:

Se ha realizado un video explicativo en el que se muestra el funcionamiento de la maqueta. Puede ser accedido mediante el siguiente enlace: *https://youtu.be/wGxLK6ayyK8*

Implementación Software:

A continuación se incluyen varias capturas del programa utilizado para realizar el control del sistema. El código incluye breves comentarios explicativos. En **color gris** se han desarrollado dichos comentarios e incluido aclaraciones.

```
int gatepin=9;
int tacpin=2;
int gate=0;            //Valor de tensión inicial en la puerta(PWM)
int wref=1500;         //Vel. angular de referencia
int fact=50;           //Factor de reducción de la señal de error
int sampletime=250;    //Tiempo de muestreo del tacómetro

//Variables del tacómetro
volatile float rev = 0;
volatile int rpm;
int time;
int oldtime=0;

//Variables del display
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

const int  en = 2, rw = 1, rs = 0, d4 = 4, d5 = 5, d6 = 6, d7 = 7, bl = 3;
const int i2c_addr = 0x27;

LiquidCrystal_I2C lcd(i2c_addr, en, rw, rs, d4, d5, d6, d7, bl, POSITIVE);
```

Figura 7.1. Código: Declaración de variables

En la *Figura 7.1.* se observa la primera parte del código. Se ha comenzado declarando las variables necesarias: **"gatepin"** y **"tacpin"** son los pines digitales del arduino utilizados (el primero conecta con la puerta del mosfet y el segundo con la salida del sensor infrarrojo), **"gate"** es el valor del ancho de pulso de la señal PWM (debe estar entre 0 y 255), **"wref"** la frecuencia (en r.p.m.) a la que deberá estabilizarse el ventilador, **"fact"** es el factor por el que se dividirá la señal de error (*Figura 7.4.*) antes de sumarla a la entrada, y **"sampletime"** es la duración del periodo de muestreo durante el cual se contarán las revoluciones del ventilador. A continuación se declaran las variables utilizadas para la medida de velocidad, que serán explicadas más adelante (*Figura 7.4.*) .Finalmente se incluyen las librerías y se declaran las variables que permiten controlar el display lcd a través del bus I2C de la placa arduino.

```
void setup() {
  pinMode(tacpin, INPUT);
  pinMode(gatepin, OUTPUT);
  lcd.begin(20,4);
  attachInterrupt(0,count,RISING); //Rutina de Interrupción del tacómetro
}
```

Figura 7.2. Código: Función "setup"

En la *Figura 7.2.* se observa función **"setup"**, en la que se define la configuración de los pines digitales utilizados (entrada o salida), el tamaño del display (4 filas x 20 columnas), y la rutina de interrupción que se utilizará para medir la velocidad del ventilador. Por cada flanco de subida recibido en el pin digital 2, se llamará a la función **"count".** (Véase la *Figura 7.3.)*

```
void count() //Función de interrupción
{
rev++; //Aumenta el contador de rpm por cada parpadeo del led
}

void display(float pwr, int spd){ //Función de control del display lcd

  //Muestra la potencia del vent. en la fila 1
  lcd.setCursor(0,0);
  lcd.print("Fan Power:");
  lcd.setCursor(13,0);
  lcd.print(pwr, 0);
  lcd.print("%  ");

  //Muestra la vel. del vent. en la fila 2
  lcd.setCursor(0,1);
  lcd.print("Fan Speed:");
  lcd.setCursor(11,1);
  lcd.print(spd);
  lcd.print(" rpm    ");

}
```

Figura 7.3. Código: Funciones auxiliares "count" y "display"

La función **"count"** responderá a las interrupciones programadas anteriormente aumentando en 1 la variable **"rev"** cada vez que sea llamada. La función **"display"** representará en el display lcd las dos variables que se le introduzcan.

```
void loop() {

  attachInterrupt(0,count,RISING);     //Empieza a contar parpadeos
  delay(sampletime);                   //Espera
  detachInterrupt(0);                  //Para de contar
  time=millis()-oldtime;               //Tiempo pasado contando
  rpm=(rev/time)*60000/7;              //Calcula rpm (el ventilador tiene 7 aspas)

  if(rpm<wref-25) gate=gate+((wref-rpm)/fact+1);
  if(rpm>wref+25) gate=gate+((wref-rpm)/fact-1);
  if(gate>255) gate=255;         //Evita que la variable supere su valor máximo

  display(gate/2.55, rpm);       //Muestra los dos parámetros del vent. en el display
  analogWrite(gatepin, gate);    //Actualiza la tensión (PWM) en la puerta
  oldtime=millis();              //Actualiza el cronómetro
  rev=0;                         //Reinicia el contador

}
```

Figura 7.4. Código: Función "loop"

La función loop contiene el código principal. En las primeras tres líneas se activa la rutina de interrupción, se deja un tiempo de espera que ha sido definido al inicio del código (*Figura 7.1*), y se vuelve a desactivar la rutina de interrupción. En el tiempo de espera entre la activación y desactivación se ha llamado a la función **"count"** y se ha aumentado en 1 la variable "rev" tantas veces como parpadeos haya recibido el sensor infrarrojo. A continuación se calcula el tiempo pasado contando parpadeos y se almacena en la variable **"time"**. Dividiendo **"rev"** entre **"time"** se obtienen los parpadeos por milisegundo, multiplicando por 60000 los parpadeos por minuto, y dividiendo entre 7 las revoluciones por minuto (el ventilador tiene 7 aspas). El resultado del cálculo anterior se almacena en la variable **"rpm"**.

En régimen permanente se ha observado que la lectura de las r.p.m. del ventilador oscila en torno a ± 25 rpm. Esto se puede deber a las irregularidades del ventilador y el sensor o al ruido externo. Las líneas 6-8 de código se encargan de ajustar la señal de alimentación de la planta en función de la lectura de velocidad. Si dicha lectura se encuentra fuera de la zona de seguridad de ± 25 rpm, a la señal de alimentación **"gate"** se le sumará el error de velocidad (**"wref"-"rpm"**) dividido entre el factor **"fact"**. Dicho error será negativo cuando **"rpm"** sea mayor que **"wref"**.

Nota: Dado que la variable "gate" debe ser un entero, para velocidades muy próximas a la deseada la señal de error se vuelve despreciable y el sistema no llega a entrar en la zona de seguridad. Para compensar esta falta de precisión, se ha sumado una unidad a la señal de error para que esta nunca se anule y el sistema siempre pueda estabilizarse dentro de dicha zona.

Para evitar que la variable **"gate"** sature (esto ocurría especialmente en el arranque) se ha impuesto en la línea 9 una medida de seguridad.
Finalmente, se muestran en el display la velocidad del ventilador y el valor de la alimentación que recibe la planta (en % de ancho de pulso), se actualiza la variable

“oldtime” (que permitirá calcular el tiempo pasado contando parpadeos en el siguiente ciclo), y se reinicia el contador de parpadeos **“rev”**.

Equivalente del Sistema de Control:

El sistema definitivo consiste en la planta, un regulador PI y un retardo puro en el lazo de realimentación. El retardo representa el tiempo de muestreo que utiliza el sistema real para determinar la variable a la salida (las rpm del ventilador), que en este caso es **0.25 segundos.**

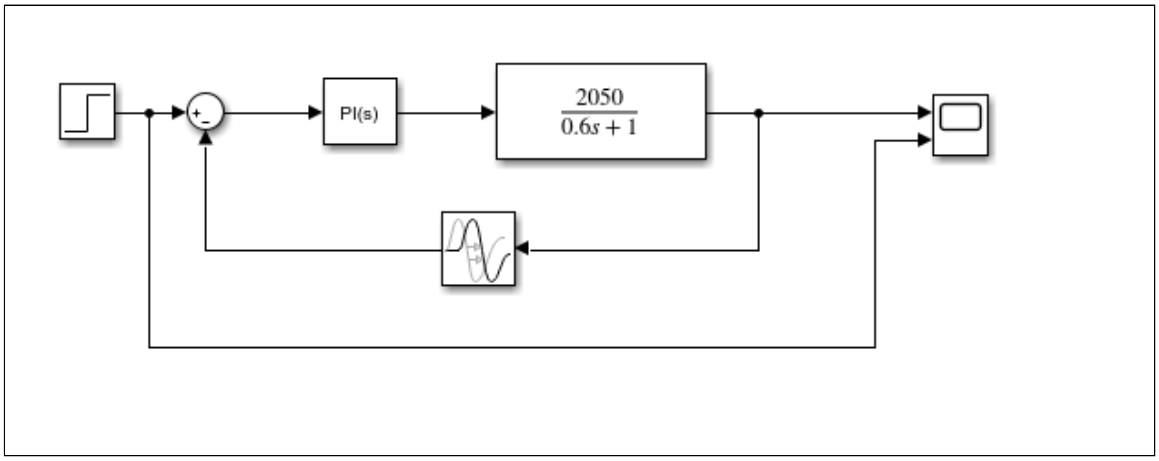

Figura 8. Diagrama de bloques del sistema de control implementado en Simulink

Para determinar los parámetros del regulador PI se ha utilizado el segundo ajuste de Ziegler-Nichols, consistente en dar valores a la ganancia en cadena cerrada hasta conseguir que el sistema oscile (punto de inestabilidad). De esta forma se ha obtenido un valor crítico de ganancia **Kcr = 0.0025** y un periodo de oscilación **Pcr = 0.6 segundos.**
Con los anteriores valores y las relaciones de la *Figura 9* se han obtenido los parámetros del regulador PI: **Kp = 0.001125** y **Ti = 0.5**.

TIPO DE CONTROLADOR	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{P_{cr}}{1.2}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Figura 9. Parámetros del controlador PID por Ziegler-Nichols II

Resultados de la Simulación:

Introduciendo al sistema de control (*Figura 8.)* un escalón de valor 1500 se obtiene la salida mostrada en la *Figura 10*. Los valores obtenidos son los siguientes: tiempo de establecimiento $\mathbf{t_s}$ **= 3.076 segundos**, tiempo de pico $\mathbf{t_p}$ **= 1.6 segundos**, sobreoscilación $\mathbf{M_p}$ **= 47.87%**, tiempo de subida $\mathbf{t_r}$ **= 1.296 segundos**, y error de posición $\mathbf{e_p}$ **= 0%**.

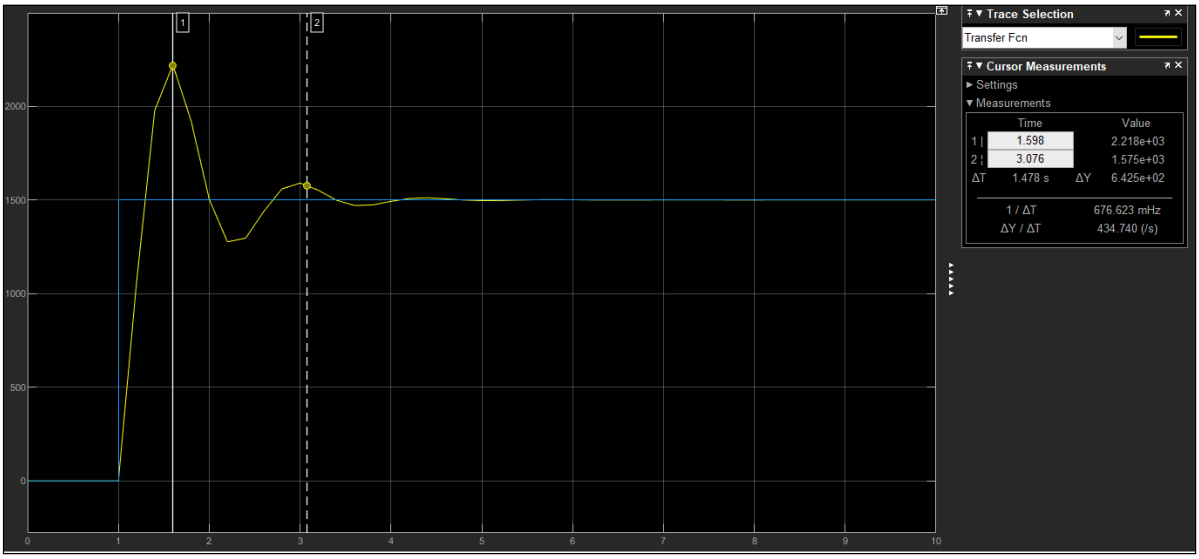

Figura 10. Respuesta del sistema de control ante un escalón

Resultados Experimentales:

Tomando los valores de velocidad durante el arranque y la estabilización del sistema se ha obtenido la respuesta mostrada en la *Figura 11*. Se han medido aproximadamente los siguientes valores: $\mathbf{t_s}$ **= 5 segundos**, $\mathbf{t_p}$ **= 2.75 segundos**, $\mathbf{t_r}$ **= 2 segundos**, $\mathbf{M_p}$ **= 25.67 %**, $\mathbf{e_p}$ **= 0%**.

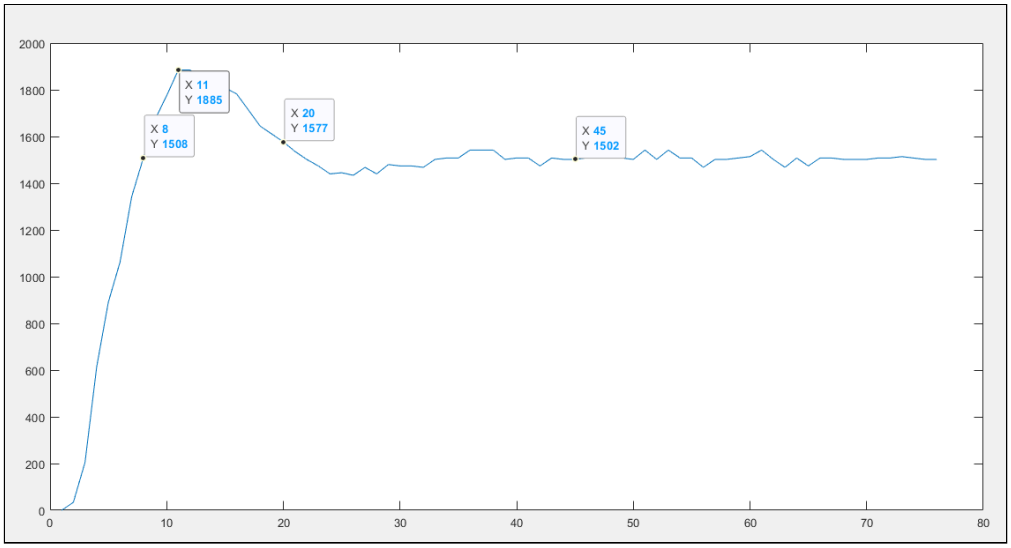

Figura 11. Respuesta experimental del sistema de control

Nota: por motivos de comodidad, en el eje x de la anterior figura, cada unidad representa 0.25 segundos.

Conclusiones:

Como solución se ha optado por ajustar el sistema de control para lograr un carácter sub-amortiguado. Con ello se ha conseguido un tiempo de establecimiento pequeño, pero también una gran sobreoscilación. Se puede observar en las *Figuras 10 y 11* que la mayor diferencia entre los resultados experimentales y los simulados es el valor de la sobreoscilación inicial, lo cual se debe en gran medida a la incapacidad del ventilador de superar las 2000 revoluciones por minuto.

Durante el desarrollo de la maqueta, resultó evidente la gran diferencia entre el trabajo con modelos teóricos (ideales) y modelos experimentales (reales), no solo por las irregularidades de los componentes físicos, sino también por los inesperados obstáculos que frecuentemente dificultan el desarrollo experimental. La lectura del tacómetro, por ejemplo, no resultó tan sencilla como se esperaba. La idea de utilizar la salida del sensor de efecto Hall presente en ciertos ventiladores resultó imposible, ya que se mezclaban los flancos de la señal PWM de alimentación con los de la señal de salida de dicho sensor (esto era inevitable debido a la construcción interna de los ventiladores de tres cables). Para el sensor infrarrojo se probaron diferentes configuraciones que acababan obteniendo lecturas muy irregulares. Finalmente se optó por añadir un segundo led (el propio sensor incluye un led para detectar objetos frontalmente) al otro lado del ventilador y medir las interrupciones del haz de luz entre ambos (las lecturas con esta configuración resultaron considerablemente más regulares).

Durante el desarrollo del programa, se pudo apreciar la facilidad con la que se puede modelar el comportamiento de un sistema realimentado mediante código. Con lógica condicional se pudo implementar cómodamente funciones como la limitación del valor máximo de una variable o el establecimiento de un rango de valores seguros en torno a los cuales pudiese oscilar el sistema durante el régimen permanente.

Bibliografía:

DroneBot Workshop (Mar 19, 2018) *Using LCD Displays with Arduino,* Disponible en: https://www.youtube.com/watch?v=wEbGhYjn4QI&list=PLNqouRGmE3BIws0W4cb7hbRgwRjDIgJml&index=9

Manuel Vargas Villanueva, Manuel Berenguel Soria y Teodoro Álamo Cantarero (1980-) TUTORIAL DE ANÁLISIS Y CONTROL DE SISTEMAS USANDO MATLAB . [Online]. Disponible en: http://www.ieef.upm.es/moodle/mod/resource/view.php?id=1212

Carlos Platero Dueñas y Miguel Hernando Gutiérrez (2014-) Apuntes y Presentaciones de Ingeniería de Control. [Online]. Disponible en: http://www.ieef.upm.es/moodle/course/view.php?id=52

Arduino Language Reference (2020), Disponible en: https://www.arduino.cc/reference/en/