**Control de intensidad luminosa:**

**Identificación del sistema y control del sensor**

# Introducción

En esta práctica se pretende diseñar un sistema de control de luminosidad. El sistema será realimentado y tendrá un control PI, se controlará la intensidad a la que brilla una bombilla modificando el factor de servicio de la señal PWM con la que se alimenta al circuito de la bombilla desde el micro, y se medirá la intensidad luminosa con el circuito desarrollado en la práctica 2.a y 2.b

# Montaje del actuador

# Para poder controlar la bombilla de 12 V y 0.5 A que se va a usar como actuador a través del microcontrolador, es necesario utilizar un transistor. El circuito final es el representado en Ilustración 1.

# Diagrama Descripción generada automáticamente

Ilustración 1:Circuito para controlar la bombilla

# Donde la resistencia se ha diseñado para que el transistor sature cuando el micro pone la salida correspondiente a 3.3 V.

# Ensayo de la planta

# Para poder diseñar correctamente el control PI, es necesario obtener la planta. Por ello, se realizará un ensayo de un escalón en el mando.

# Tal como se ha especificado antes, el mando es el factor de servicio de un PWM. La señal del mando en el ensayo es la representada en Ilustración 2.

# Gráfico, Gráfico de cajas y bigotes Descripción generada automáticamente

Ilustración 2:Mando aplicado en el ensayo

# Se tomarán datos de la intensidad luminosa desde t = 0.8s hasta t=1.2s y se enviarán por la UART para poder ser registrados y procesados. La medida de la intensidad luminosa se hace tal como se especificó en la práctica p2a y p2b.

# El código para realizar el ensayo es el siguiente:

#include <xc.h>

#include <stdio.h>

#include <stdlib.h>

#include "uart.h"

#include "config.h"

#include "interpolar\_sensor.h"

#include "pwm.h"

#include "idle.h"

#include "adc.h"

**int** main(**void**)

{

**char** send\_data[9];

inicializarReloj();

TRISB &= 0x0FFF;

// Inicializaciones

inicializarUART(115200);

inicializarTareaIdle(10);

inicializarADCPolling(0x20);

inicializarPWM((1 << 10), 1000);

activarPWM((1 << 10));

**unsigned** **int** lectura;

**unsigned** **int** lux = 0;

**unsigned** **int** t = 0;

**unsigned** **int** pwm;

**while** (1)

{

// Tarea 1

pwm = ControlBulb(t);

// Control de la intensidad de la bombilla

// Tarea 2

lectura = leerADCPolling(5);

lux = interpolarSensor(lectura);

// Medida de la intensidad luminosa

// Tarea 3

// Envío de datos por UART

**if**(t>=800 && t<=1200){

sendData(lux, pwm);

}

tareaIdle();

t++;

// if(t>3500){return 0;}

}

**return** 0;

}

**void** sendData(**unsigned** **int** lux, **unsigned** **int** pwm)

{

**char** send\_data[20];

sprintf(send\_data, " %d, %d;\n", pwm, lux);

putsUART(send\_data);

}

**int** ControlBulb(**unsigned** **int** t)

{

**if** (t >= 1000 && t <= 3000)

{

setDcPWM((1 << 10), 10000);

**return** 100;

}

**else** **if** (t < 1000)

{

setDcPWM((1 << 10), 5000);

**return** 50;

}

**else** **if** (t > 3000)

{

setDcPWM((1 << 10), 0);

**return** 0;

}

}

# Identificación del sistema y diseño del controlador

# El resultado del ensayo se presenta en Ilustración 3.

# Gráfico Descripción generada automáticamente

Ilustración 3:Resultados del ensayo

Mediante la herramienta pidTuner, los datos se procesan, se obtiene la planta, y se diseña el control PI.

La planta se modela mediante la siguiente función de transferencia:

Donde k = 10.467 y Tp = 0.02.

El control PI sobre el error tiene la siguiente función de transferencia:

Donde los parámetros obtenidos son Kp =0.18 y Ki =7.12.

# Diseño del programa de control

El programa se implantará con tres tareas dentro de un bucle scan de 1ms de periodo de muestreo.

* Una tarea para realizar la medida de la intensidad luminosa.
* Otra tarea para enviar al PC los datos del ensayo, que también la tenemos.
* Control PI.

Esta última es la única tarea que se va a explicar, ya que las otras dos tareas son triviales.

## Tarea de control

La tarea de control tendrá como objetivo generar una señal de mando, que devolverá como salida. Para ello se usarán las señales de referencia y salida para ajustar el nivel de actuación de la salida, en centésimas de porcentaje.

Dado que se trata de un control PI el mando se calculará con una composición ponderada del error instantáneo y acumulado, donde el coeficiente con el que se pondera es el Kp y el Ki previamente identificados.

### Componente proporcional

La componente proporcional del error se computa en cada instante, en este caso lo hemos guardado en una integer, error. Este tipo es apropiado ya que una resta de unsigned int puede ser negativa, pero nunca puede ser un número decimal.

### Componente integral

La componente integral por su parte se trata del error acumulado de la salida, y por tanto ha de mantenerse entre llamadas a la función. Por ello, se ha decidido guardarlo como un static. Es relevante resaltar que se inicializa a cero, ya que en el instante inicial no hay error, y que se trata de un tipo float, ya que tenemos que guardar el área bajo la curva de error y para ello aproximamos por rectángulos cada periodo de muestreo (integral en adelanto).

Por último, para evitar que el control se haga inestable, dejamos de añadir componente integral si el mando se encuentra saturado, a esto se le llama Anti-windup.

Lo ultimo que hace la tarea es aplicar dicho mando en la bombilla con el módulo PWM y devuelve al main para que se pueda mandar por uart o procesar.

int ControlBulb(unsigned int salida, unsigned int referencia) // unsigned int \* p\_pwm, unsigned int \* p\_mando)

{

    int error;

    static float integral = 0; // se ha de mantener entre ejecuciones

    float mando;

    error = referencia-salida;

    integral += error \* (T\_S); // La integral es el error acumulado

    float errorP =KP\*error;

    float errorI =KI\*integral;

    mando = (KP\*error + KI\*integral);

    //saturaci�n

    if(mando >100){

        mando =100;

        integral -= error \* (T\_S);

    }

    if(mando < 0){

        mando = 0;

        integral -= error \* (T\_S);

    }

    //ajuste de mando

    setDcPWM((1 << 10), (int) (mando)\*100);

    return (int) mando;

}

# Resultados del control

Para comprobar la respuesta del control hemos realizado un ensayo donde se realiza un escalón a 500 lx desde un suelo de 300 lx. Se pueden ver los resultados del ensayo en la siguiente gráfica.

# 

Ilustración 4 Ensayo control PI

Se puede ver que la respuesta del control es muy buena, ya que conseguimos llegar a régimen permanente en menos de una décima de segundo y el overshoot es menor al 13%. Por otra parte, se puede ver que el mando nunca satura, aunque en el escalón de subida y bajada llega a estar cerca de ello. Esto nos muestra que no se debería aumentar la agresividad del control, ya que correríamos el riesgo de saturar y empeorar la respuesta, especialmente si se trata de escalones mayores. Se puede ver además que en los primeros instantes se produce un gran pico. Esto se debe a que al estar inicializado a cero, el error que se acumula en los primeros instantes le hace sufrir un overshoot.