CONTROL PID DE TEMPERATURA IMPLEMENTADO EN RASPBERRY PI3.

Maestría en Sistemas Inteligentes Multimedia

Lenguaje Ensamblador

Dr. Eduardo Castañeda

Alumnos: Ismael Duron, Francisco Pedraza

Indice

Introducción

Nuestro proyecto pretende mostrar una de las muchas capacidades del control por medio de sistemas embebidos. De esta manera, el desarrollo mostrado en este reporte fue utilizando una tarjeta Raspberry Pi 3 con sistema “Raspbian” integrado, capaz de ejecutar instrucciones del Micro-controlador Broadcom BCM2837 a bajo nivel.

De esta forma, mediante lenguaje C se pudo tener acceso a un ADC externo (MCP3202) por medio de la interfaz SPI. Esta conexión con ADC nos permite analizar los cambios de voltaje en un sensor de temperatura en grados Celsius (LM35). Dependiendo de la temperatura medida y el ajuste que se le dé dentro el programa, el micro-controlador enviara una señal PWM que ira variando de velocidad a un mini-ventilador de DC.

El acoplamiento del motor de DC se hace por medio de un puente H (LM293D) que permite dar potencia al motor de DC y hacerlo trabajar a diferentes velocidades según el valor del duty cycle del PWM, así como aislar nuestra Raspberry con el dominio de control de DC.

Cabe aclarar que para la programación en C, se utilizaron ejemplos vistos en clase dentro de la librería elaborada por el grupo “Aerospyce” la cual facilita el acceso a puertos GPIO, PWM y SPI de micro-controlador.

Marco Teórico

Los convertidores CD-CD son circuitos electrónicos de potencia que convierten un voltaje de corriente directa en otro nivel de voltaje de corriente directa o corriente continua. Actualmente existen dos métodos para realizar la conversión CD-CD.

1. **Convertidores Lineales:**
   Basados en el empleo de un elemento regulador que trabaja en su zona activa disipando energía.
2. **Convertidores Conmutados:**
   Los cuales se basan en el empleo de dispositivos semiconductores que trabajan en la conmutación (corte/conducción), regulando de esta forma el flujo de potencia a la salida del convertidor. A este tipo de convertidores también se les conoce como fuentes de alimentación conmutadas ya que poseen grandes beneficios en comparación con los convertidores lineales. Mientras que uno regulador de tensión utiliza transistores polarizados en su región activa de amplificación, las fuentes conmutadas utilizan los mismos conmutándolos activamente a altas frecuencias (20-100 kilociclos típicamente) entre corte (abiertos) y saturación (cerrados).

En comparación con las fuentes lineales, las conmutadas regularmente tienen menor peso y tamaño, tienen una eficiencia mayor y por ende el calentamiento también es menor. Contrariamente, son más complejas de elaborar y generan ruido eléctrico de alta frecuencia que deber ser cuidadosamente minimizado para no causar interferencias a equipos próximos a estas fuentes.

Ingeniería de Control

Se ocupa de los aspectos prácticos y teóricos en el control de sistemas y procesos, incluyendo aspectos tales como el análisis y diseño de sistemas regulados, diseño y sintonización de reguladores, utilización de sensores y actuadores, procesamiento digital de señal, etc. Se ocupa principalmente en el control de los sistemas dinámicos mediante el principio de la realimentación, consiguiendo de tal manera que las salidas de los sistemas se acerquen lo más posible a un comportamiento ya predefinido.

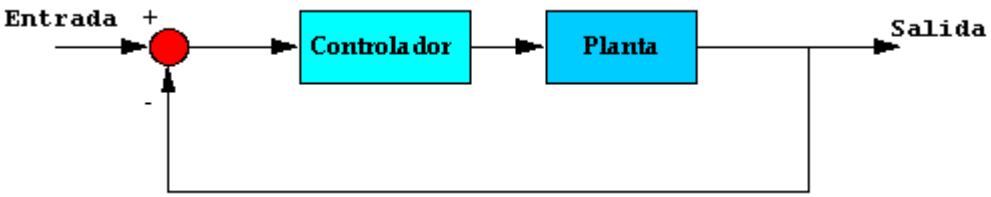

Figura 1

Planta: El sistema que se va a controlar, (en nuestro caso el motor). Una planta puede ser una parte de un equipo, tal vez un conjunto de los elementos de una máquina que funcionan juntos, y el objetivo es efectuar una operación en particular.

Procesos: Es una operación o un desarrollo natural progresivamente continuo, marcado por una serie de cambios graduales que suceden unos a otros de una forma relativamente fija y que conducen a un resultado o propósito determinados.

Controlador: Es quien provee la “excitación” de la planta, se diseña para controlar el comportamiento de todo el sistema.

Desarrollo

El **controlador proporcional** (Kp) va a tener el efecto de reducir el tiempo de elevación y reducirá, sin eliminar el error de estado estacionario. Un control integral (Ki) tendrá el efecto de eliminar el error de estado estacionario, pero puede empeorar la respuesta transitoria.

Un **controlador integral** (Ki) decremento del tiempo de elevación, incrementa tanto el sobre pico cuanto el tiempo de establecimiento, y elimina el error de estado estacionario. El controlador integral también reduce el tiempo de elevación e incrementa el sobre pico así como lo hace el controlador proporcional, el controlador integral elimina el error de estado estacionario.

En el esquema siguiente se puede apreciar el cambio respecto a la temperatura.

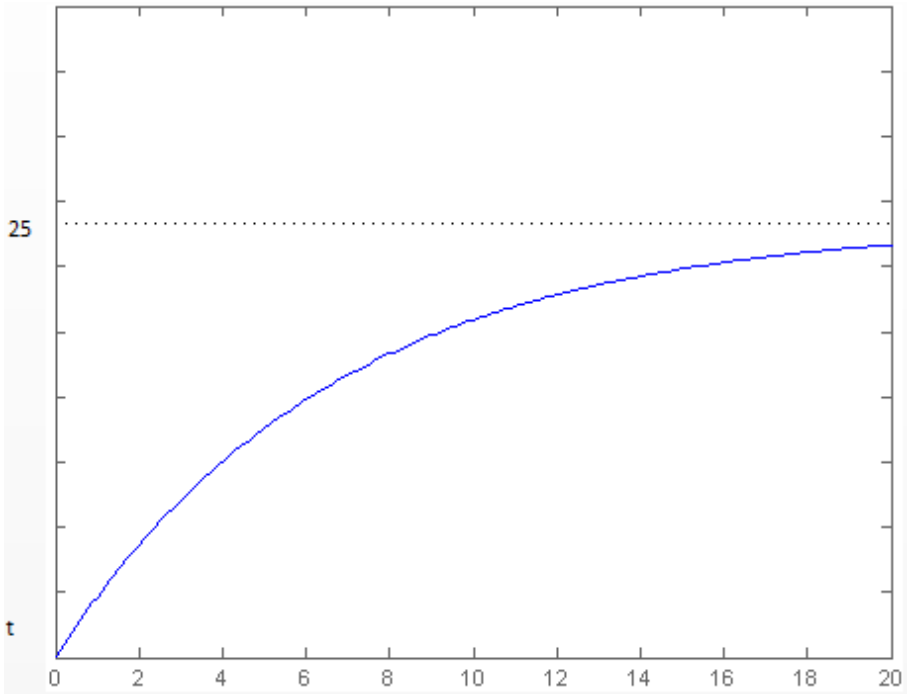

Figura 2

Control Discreto, con el fin de procesar la señal a controlar por medio de un sistema digital, es necesario convertir la señal en pulsos discretos con un convertidor análogo digital.

Teoremas

PID = Kpe + Kde + Ki ∑ e.

PID = Kpe + Kde + Ki ∑ e + Constante.

Error = Posición deseada – posición medida.

e' error = error actual – error anterior.

∑ error = error actual + error anterior.

Cabe mencionar que la medición no puede irse a valores negativos, al saturador hay que manejarlo de la siguiente manera.

1. El GPIO disponible para PWM es el 12.
2. La temperatura a regular es 25 grados.
3. Las variables, Kp, Kd y Ki nos ayudaran a tener el control apropiado sobre la planta. Kp=0.5, Ki=0.01, Kd=0.5. Estos valores se probaron aleatoriamente (prueba y error) hasta que se tuvo el control planeado sobre la planta, las variables del PID se ajustan a las mismas.

Esquemático de Raspberry, Sensor de temperatura (LM35), ADC (MCP3202), Puente (L293D) y planta (ventilador)

Con el fin de realizar una comunicación exitosa entre los dispositivos, con las interfaces SPI, MISO, MOSI se seleccionó cuidadosamente cada pin entre cada dispositivo.

1. El pin 21 de la Raspberry, correspondiente a SPIO MISO, estará conectado al pin 6 del MCP3202 correspondiente a DOUT.
2. El pin 23 de la Raspberry, correspondiente a SPIO SCLK, estará conectado al pin 7 del MCP3202 correspondiente a SCK.
3. El pin 24 de la Raspberry, correspondiente a SPIO CS0, estará conectado al pin 1 del MCP3202 correspondiente a CS.
4. El pin 19 de la Raspberry, correspondiente a SPIO MOSI, estará conectado al pin 5 del MCP3202 correspondiente a DIN.
5. El pin 12 de la Raspberry, correspondiente a GPIO 18, estará conectado al pin 2 del L293D, 1A.
6. El pin 6 de la Raspberry, estará conectado a tierra, comparte la misma con el sensor de temperatura LM35, y con el MCP3202.
7. El pin 2 de la Raspberry, estará conectado a la fuente de 5 volteos y compartirá la misma conexión con el MCP3202 el cual conecta al voltaje el pin 8 de VCC. Así mismo el sensor de temperatura LM35 con el pin 2.
8. En el sensor de temperatura LM35 el pin 3, correspondiente al voltaje de salida. Este a su vez ira conectado a VIN_CH0 del MCP3202.
9. Los pines 1 y 16 correspondientes a 1,2EN y VCC1 respectivamente, estarán conectados a voltaje, son los pines que harán el puente.

10. Los pines 3 y 6 del L293D correspondientes a 1Y y 2Y respectivamente, estarán conectados a la planta (ventilador).
11. Los pines 4, 5, 7, 12, 23, del L293D, correspondientes a GND1, GND2, 2A, GND3 y GND4 irán conectados a tierra.
12. El pin 8 del L293D, correspondiente a VCC2, ira conectada a la fuente de 8v.
    De esta manera el control del motor del ventilador fue exitosa y se logró un equilibrio entre la temperatura y el giro del motor.

En la figura 3 se puede observar un diseño esquemático de estas conexiones.

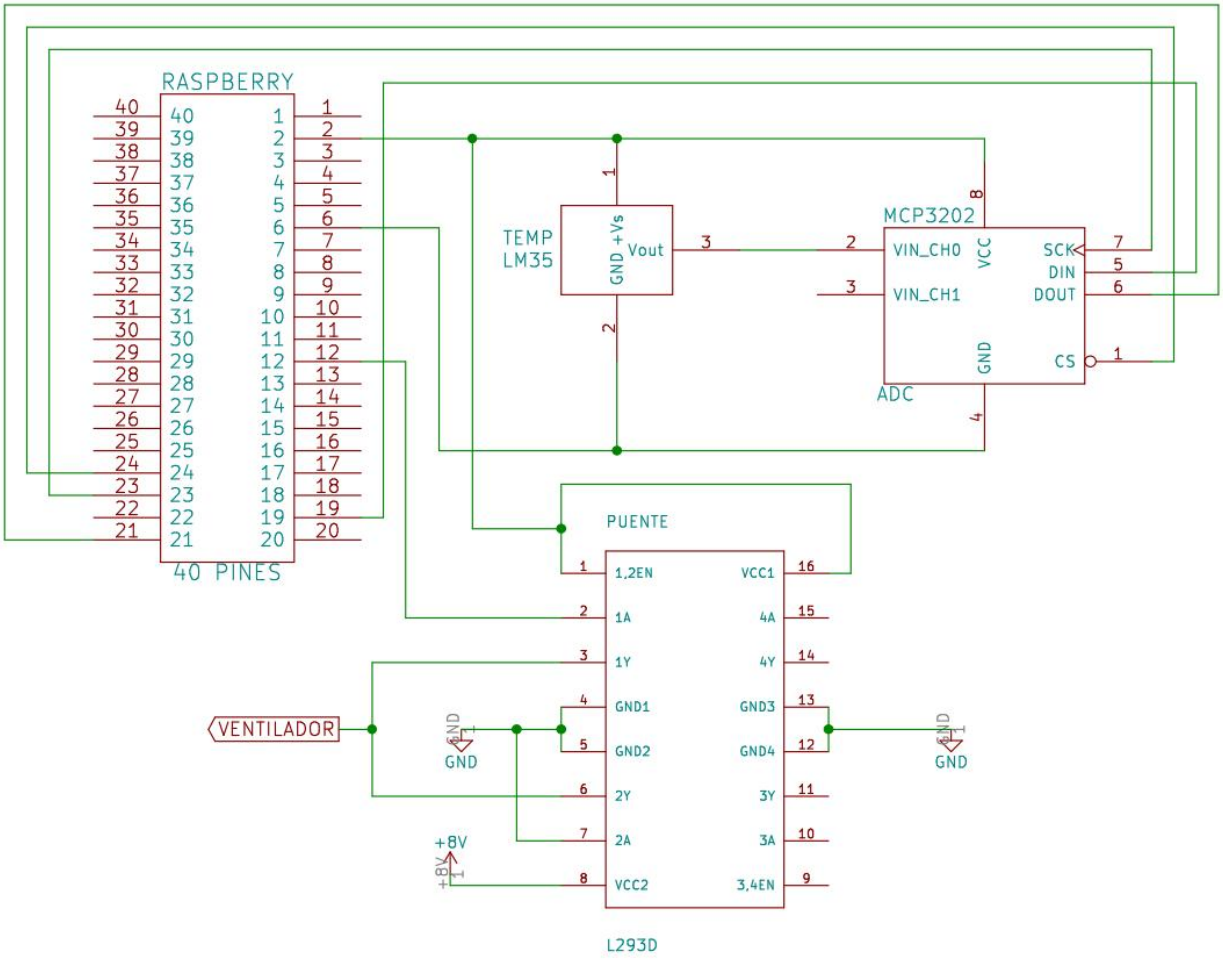

Figura 3

Se hizo uso del lenguaje C para la codificación, además de implementación de la librería bcm2835.

Código Fuente del programa

```
#include <bcm2835.h>
#include <stdio.h>

// Asignacion de GPIO disponible para pin de PWM
#define PIN RPI_GPIO_P1_12
// Canal 0 para PWM
#define PWM_CHANNEL 0
// Controla el rango maximo del PWM
#define RANGE 600

// Constante de temperatura donde queremos regular
#define TEMP 25

// Cabecera de funcion para lectura de ADC
float a2dRead (int canal);

// Cabecera de funcion para control PID
float PID_Controller (float set_point, float measured_value);

// Variables del PID, se ajustan Kp, Ki y Kd a valores determinados mediante prueba y
error.
float actual_error, error_previous, P, I, D, Kp=0.5, Ki=0.01, Kd=0.5;


int main(int argc, char **argv)
{
    if (!bcm2835_init())
    {
        printf("Falla en inicializacion de bcm2835_init. Estas corriendo como
root??\n");
        return 1;
    }

    if (!bcm2835_spi_begin())
    {
        printf("Falla en inicializacion de bcm2835_spi_begin. Estas corriendo como
root??\n");
        return 1;
    }

    //********************************************** Seccion de inicializacion del
PWM *****************************************************
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_ALT5);         // Poner el GPIO a modo
ALT5, para permitir que el Canal 0 de PWM salga de ese pin
    bcm2835_pwm_set_clock(BCM2835_PWM_CLOCK_DIVIDER_16);    // Divisor de reloj puesto
a 16.
    bcm2835_pwm_set_mode(PWM_CHANNEL, 1, 1);                // Con un divisor de 16 y
un rango de 1024, la frecuencia de pulsacion de un PWM,
    bcm2835_pwm_set_range(PWM_CHANNEL, RANGE);              // sera de 1.2MHz/600 =
2KHz, buena para hacer driving de un motor de DC.

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

    //********************************************** Seccion de inicializacion del
SPI ******************************************************
```

```
    bcm2835_spi_begin();
    bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);        // Default
    bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);                     // Default
    bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_65536);   // Default
    bcm2835_spi_chipSelect(BCM2835_SPI_CS0);                        // Default
    bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, LOW);        // Default

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

    float volt_0;   //Variable para lecturas de voltaje del sensor LM35.
    int canal_0;    //Variable para lectura del canal 0 del ADC MCP3202.

    while (1) //loop infinito de medicion
    {
        float pid_control;
        int pwm_value;
        float temp_read;

        canal_0 = a2dRead(0);       //Valor de la lectura de funcion ADC en canal 0.
        volt_0  = canal_0*5.0/4096;   //Voltaje de entrada en CH0, el cual tiene un
Vref=5V y 12 bits de granularidad.
        temp_read   = volt_0*100;   //Conversion de voltaje a temperatura (el LM35
tiene una relacion de 10mV por 1 grado Celsius)

        pid_control = PID_Controller(TEMP, temp_read);
        pwm_value = (pid_control*-1) + 400; //conversion a valor para PWM
        printf("El valor del PWM es %i y la temperatura es %.2f \r:  ",  pwm_value,
temp_read);

        if ((pwm_value>400) &&( pwm_value<4000))        //Rango ajustado para hacer
funcionar el ventilador
            bcm2835_pwm_set_data(PWM_CHANNEL,pwm_value);
        else if (pwm_value<400)                         //Si el valor del de
conversion para el PWM es menor de 400, entonces se apaga el ventilador
            bcm2835_pwm_set_data(PWM_CHANNEL,0);
    }

    bcm2835_spi_end();
    bcm2835_close();
    return 0;
}


float a2dRead (int canal)
    {
        int a2dVal;
        char dato[3];

        dato[0]=1;  // Primer byte transmitido
        dato[1]=( ((canal +2) << 6)); // Segundo Byte transmitido
        dato [2]=0; // Tercer Byte transmitido

        bcm2835_spi_transfern(dato, sizeof(dato));

        // Leyendo los datos
        a2dVal = 0;
        a2dVal = (dato[1]&0b00001111)<<8; // Byte mi¿½s significativo
        a2dVal |=  (dato[2] & 0xff);  // Byte menos significativo

        return a2dVal;
    }
```

```
//Aqui esta la deficion del control PID, Kp, Ki y Kd son las ganacias proporcional,
integral y diferencial.

float PID_Controller (float set_point, float measured_value)
    {
        error_previous = actual_error;  //error_previous tiene el error previo.
        actual_error = set_point - measured_value;
        // PID
        P  = actual_error;   //Error actual
        I += error_previous;  //Sumatoria de los errores previos
        D  = actual_error - error_previous;  //Restas de los errores previos con el
error actual.
        return Kp*P + Ki*I + Kd*D;   //Ajuste de Kp, Ki, Kd empiricamente a prueba y
error
    }
```

Referencias

EETech Media, LLC. (6 de Octubre de 2015). *allaboutcircuits.com*. Obtenido de Raspberry Pi Project: Control a DC Fan: http://www.allaboutcircuits.com/projects/raspberry-pi-project-control-a-dc-fan/

McCauley, M. (s.f.). *www.airspayce.com*. Obtenido de C library for Broadcom BCM 2835 as used in Raspberry Pi: http://www.airspayce.com/mikem/bcm2835/

Microchip Technology Inc. (2001). *www.mouser.com*. Obtenido de www.mouser.com-MCP3202: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=0ahUKEwjLhMmosqvMAhVB3GMKHTawCMgQFggoMAI&url=http%3A%2F%2Fww1.microchip.com%2Fdownloads%2Fen%2FDeviceDoc%2F21034D.pdf&usg=AFQjCNGZs9Qmvg1DB4Pe7cf9ZlFK6RD6Wg

RASPBERRY PI FOUNDATION. (s.f.). *Raspberry pi 3 model B*. Obtenido de Raspberry pi 3 model B: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Glosario

MOSI = Master Output Slave Input.

MISO = Master Input Slave Output.

GPIO = General Purpose Input/Output.

SPI = Serial Peripheral Interface.

PWM = Pulse Width Modulation.

MCP3202 = Convertidor Analógico a Digital de registro de aproximaciones sucesivas (SAR), de 12 bits señal diferencial, SOIC, SPI.