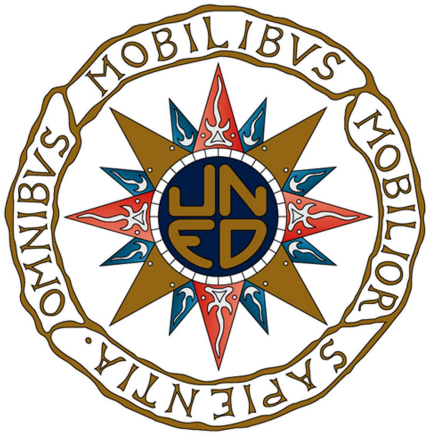

SISTEMAS EMPOTRADOS

Ricardo de Figueiredo Minelli

SISTEMAS EMPOTRADOS

Ricardo de Figueiredo Minelli

TRABAJO BASADO EN ARDUINO

Tutorizada por

David Moreno Salinas, José Sánchez Moreno

Índice general

1 Intrudución

1.1 Temperatura controlada por un sensor DS18B20, ventilador de 4 pines Noctua NF-A4x20 5V PWM y sistema de control PID.

En este proyecto consiste en el diseño de un sistema de regulación de temperatura, utilizando un sensor de temperatura Dallas DS18B20, un ventilador de cuatro pines (PWM Signal(+5V), RPM Speed Signal, +5V, Groud(GND)) y un Arduino Mega con control PID. Este proyecto propone estabilizar la temperatura en un valor predefinido. Si la temperatura varía, intentará estabilizarse mediante un control. En este proyecto vamos a diseñar un control proporcional simple.

Para hacer este control usaremos:

- Arduino Mega 2560
- Noctua NF-A4x20 5V PWM Quiet Premium Fan 4-Pin 5V Version (40x20mm Brown)
- DS18B20 temperature sensor module
- RGB LED SMD sensor module
- RGB 5mm LED Modul
- 16x2 LCD KEYPAD Display HD44780 Screen Module
- 10K ohm Resistor 1/4w (0.25 Watt) (3x)

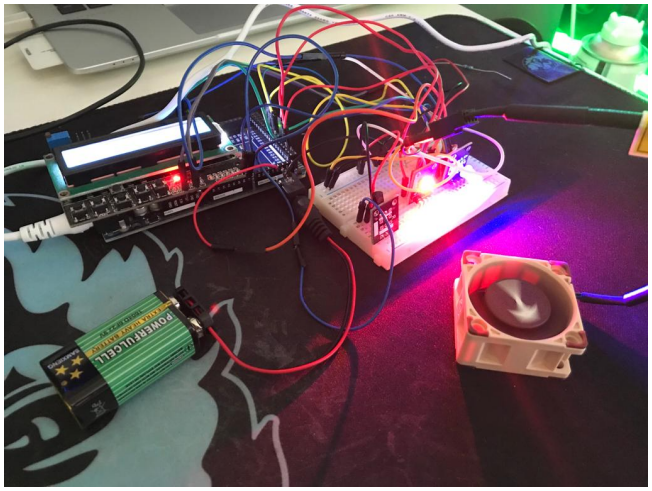

Figura 1.1: Controlador PID - Arduino

Video [Min21] con el controlador PID en Arduino en funcionamiento

2 | Implementación

Como nuestro controlador se compone de varios pasos, explicaremos cada paso por separado, y luego juntamos todo. Primero explicaremos cómo obtenemos la temperatura, luego cómo controlamos la velocidad del ventilador, luego medimos su velocidad y finalmente, aplicamos el control PID.

2.1 Midiendo la temperatura

Este paso consiste en tomar el valor analógico del sensor y convertirlo a grados Celsius. La biblioteca de Dallas Temperature es una biblioteca específica de hardware que maneja funciones de nivel inferior y por este motivo obviaremos la función de transferencia.

```
1  //Configurar una instancia de oneWire para comunicarse con
       cualquier dispositivo OneWire
2  OneWire oneWire(SensorTemp);
3  //Pasar la referencia oneWire a la biblioteca DallasTemperature
4  DallasTemperature sensors(&oneWire);
5
6  sensors.begin();   //Pon en marcha la biblioteca
7
8  // Se inicia la medicion de temperatura ...
9  sensors.requestTemperatures();
10 float leitura = sensors.getTempCByIndex(0);
11 Serial.print("Temp :  ");
12 Serial.println(leitura);
```

2.2 Controlando la temperatura (Consumo)

Usamos on/off con histéresis para el ahorro de energía, cuando la temperatura se encuentra por debajo de 30 grados centigrados el ventilador se apaga. Nos ayudamos del módulo RGB de 3 colores para identificar en que rango está la temperatura. Abajo de 30 grados (Led Azul), entre 30 y 55 (Led Verde) ó por encima de 55 grados (Led rojo).

```
1  //On/off con histeresis.
2  if (valor != previousValor){
3    // Valor por encima del umbral maximo
4    if ((( (sensors.getTempCByIndex(0)) > ( 55 ) ))){
5         digitalWrite(Led_Rot, HIGH);
6         digitalWrite(Led_Gruen, LOW);
7         digitalWrite(Led_Blau, LOW);
8       }else{
9         digitalWrite(Led_Rot, LOW);
10        digitalWrite(Led_Gruen, LOW);
11        digitalWrite(Led_Blau, LOW);
12      }
13   // Valor entre dos umbrales (region media)
14   if (( ( (sensors.getTempCByIndex(0)) <= ( 55 ) ) && ( (sensors.
        getTempCByIndex(0)) >= ( 30 ) ) )){
15        digitalWrite(Led_Rot, LOW);
16        digitalWrite(Led_Gruen, HIGH);
17        digitalWrite(Led_Blau, LOW);
18        }else{
19        digitalWrite(Led_Rot, LOW);
20        digitalWrite(Led_Gruen, LOW);
21        digitalWrite(Led_Blau, LOW);
22        }
23   // Valor por debajo del umbral minimo
24   if (( (sensors.getTempCByIndex(0)) < ( 30 ) )){
25        Output = 0;
26        digitalWrite(Led_Rot, LOW);
27        digitalWrite(Led_Gruen, LOW);
28        digitalWrite(Led_Blau, HIGH);
29        }else{
30        digitalWrite(Led_Rot, LOW);
31        digitalWrite(Led_Gruen, LOW);
32        digitalWrite(Led_Blau, LOW);
33        }
34 }
35 // Guarda el estado actual como ultimo estado, para la pr xima
       vez a traves del bucle
36 previousValor = valor;
```

2.3 Controlamos la velocidad del ventilador

Este paso es bastante sencillo. El ventilador en sí tiene un circuito interno que le permite regular la temperatura.

Para ello basta con conectar el cable azul a la salida de un puerto con pwm. Entonces, dependiendo del valor que ponga en pwm, se ejecutará más rápido o más lento. Para ayudar a ver los efectos, hemos agregado código que nos permite ingresar el valor que queremos para pwm a través de la terminal.

```
1   if (Serial.available() > 0){
2     String recebido = leStringSerial();
3     valor = recebido.toFloat();
4     Serial.println(valor);
5   }
```

2.4 Medición del número de revoluciones por minuto (RPM) del ventilador

El sensor que nos permite medir las rpm del enfriador funciona así: cada vez que una de las hélices pasa por el sensor magnético dentro del enfriador, envía una señal al puerto digital. Con esto podemos medir cuántas señales se enviaron al puerto en un período de tiempo determinado para establecer el valor de rpm.

```
1  void rpm(){
2  ContFan++;
3  }
4  void setup(){
5      pinMode(pwm, OUTPUT);
6      pinMode(NumRpm, INPUT);
7      Serial.begin(9600);
8      attachInterrupt(0, rpm, RISING);
9      SetTunings(4.2,0.01,0);
10     sensors.begin();  //Pon en marcha la biblioteca
11     lcd.begin(16, 2); //Configurar la pantalla LCD
12     lcd.setCursor(0, 0);
13     //Configurar las salidas de los LEDs de control y la entrada
           del control de bateria
14     pinMode(Led_Rot, OUTPUT);
15     pinMode(Led_Gruen, OUTPUT);
16     pinMode(Led_Blau, OUTPUT);
17     pinMode(A6,INPUT);
18     pinMode(Led_Bat_R, OUTPUT);
19     pinMode(Led_Bat_G, OUTPUT);
20     pinMode(Led_Bat_B, OUTPUT);
21 }
22 // Calculamos e imprimimos
23   Calc = ((ContFan * 60)/2);
```

```
24    Serial.print (Calc, DEC);
25    Serial.print (" rpm\r\n");
```

2.5 Control PID (SAMPLE TIME)

Tanto el cálculo integral como el derivado están directamente influenciados por el tiempo. Por lo tanto, los intervalos de tiempo irregulares pueden producir resultados irregulares. Por lo tanto, si se llama al PID a intervalos de tiempo constantes, los resultados serán mucho más consistentes.

Después de implementar una muestra de tiempo, se puede ver una diferencia muy significativa, especialmente en la derivada. Antes, la derivada parecía fuera de control. Ahora varía con más estabilidad.

```
1     void SetTunings(double Kp, double Ki, double Kd)
2  {
3     double SampleTimeInSec = ((double)SampleTime)/1000;
4     kp = Kp;
5     ki = Ki * SampleTimeInSec;
6     kd = Kd / SampleTimeInSec;
7  }
8  //------------------------------------------------------
9  void SetSampleTime(int NewSampleTime)
10 {
11    if (NewSampleTime > 0)
12    {
13       double ratio  = (double)NewSampleTime
14                       / (double)SampleTime;
15       ki *= ratio;
16       kd /= ratio;
17       SampleTime = (unsigned long)NewSampleTime;
18    }
19 }
20 // LLamamos a la funcion
21 SetTunings(4.2,0.01,0);
```

3 | Derivative Kick

Cuando hay un cambio en el punto de ajuste, ocurre un cambio repentino en el error. Esto se debe a que la derivada de este cambio genera un número muy grande.

Pero en nuestro caso este cambio no supuso mucha diferencia, ya que el efecto derivado no tiene mucho efecto, al igual que el cambio brusco de temperatura.

```
void Compute()
{
   //Cu nto tiempo desde la ultima vez que calculamos
   unsigned long now = millis();

   int timeChange = (now - lastTime);
      if(timeChange>=SampleTime)
      {

      //Calcular todas las variables de error de trabajo
      error = Setpoint - Input;
      ITerm += (ki * error);
      dInput = (Input - lastInput);

      //Calcular salida PID
      Output = kp * error + ITerm - kd * dInput;

      //Recuerda algunas variables para la proxima
      lastInput = Input;
      lastTime = now;
      }
}
```

3.1 ON-THE-FLY TUNING CHANGES

La mejora "On-the-fly tuning changes" (Cambios de ajuste sobre la marcha) permite al usuario modificar los valores de Kp, Ki y Kd en tiempo de ejecución.

Esto evita tener que detener el sistema para modificar los valores constantes generando un sobrecalentamiento sin una acción inmediata del controlador PID para aumentar las RPM.

La implementación de esta funcionalidad es simple como se muestra a continuación:

```
{
    double SampleTimeInSec = ((double)SampleTime)/1000;
    kp = Kp;
    ki = Ki * SampleTimeInSec;
    kd = Kd / SampleTimeInSec;
}
```

3.2 Control de la carga de la batería (Consumo)

Se controla la carga de la batería con histéresis y se muestra a través de un LED RGB, se pueden implementar diversas acciones a partir de este control. Hemos dejado solamente con los LEDs para que podamos visualizar los resultados del control.

```
  //Control de voltaje de la bateria con histeresis.
  void bat() {
    Bat_Lectura = analogRead(A6);
    volt = Bat_Lectura /1023 * 5.0; // Convertir el valor de ADC en
        voltaje
    Serial.print("Baterry:");
    Serial.println(volt);
    // Dependiendo del voltaje encender el Led?
    if (volt >= maximo){
      digitalWrite(Led_Bat_R, LOW);
      digitalWrite(Led_Bat_G, HIGH);
      digitalWrite(Led_Bat_B, LOW);
    }else if (volt < maximo && volt >= medio){
      digitalWrite(Led_Bat_R, LOW);
      digitalWrite(Led_Bat_G, LOW);
      digitalWrite(Led_Bat_B, HIGH);
    }else if (volt < medio){
      digitalWrite(Led_Bat_R, HIGH);
      digitalWrite(Led_Bat_G, LOW);
      digitalWrite(Led_Bat_B, LOW);
    }
  }
```

Bibliografía

[Min21] Ricardo Minelli. “Sistemas Enpotrados - Trabajo basado en Arduino”. En: (2021). URL: https://youtu.be/inKSCRp1wqY.