Justin Yatsuhashi and Daniel Noronha

Professor Fuja

Embedded Systems

20 April 2022

EE 10115 Temperature Regulator Project Report

Executive Summary

The project being demonstrated is a temperature regulator which checks the current ambient temperature and adjusts the speed of a DC fan accordingly. It has two modes: Manual and Automatic. In the Manual mode, the user can control the fan speed directly. In the Automatic mode, the system will calculate the error between the current and user-set temperature, then will use this value to determine the fan's speed. To switch between these two modes, the user pushes a button. The system will also include an LCD to display the set temperature or fan speed and the temperature of the surroundings.

Background

The device is extremely useful for regulating the temperature of a room. In the automatic mode, it will spin faster in order to cool the room to the set temperature, then it will slow down to maintain that temperature. Meanwhile, in the manual mode, the user can simply set how fast they want the fan to spin in order to have the desired cooling effect. This device could be useful as an air conditioner in different buildings and rooms in order to regulate and maintain a comfortable temperature. For instance, it would be a very useful system for dorms without air conditioning at Notre Dame. Since it runs on battery power, it is very portable and can be used as a source of cooling in rooms without mains electricity or outdoors in hot weather.

The idea was an original design that was created for this project, but its functionality is similar to that of a standard air conditioning system (temperature sensing and actuation). Like an air conditioner, the temperature regulator allows the user to set the temperature and adjusts the cooling effect based on that setting. This means that the fan will cool the room to the desired temperature just like an air conditioning unit would do. Devices like this exist in the marketplace, but most of them require a lot of power to run and hence can only be used with mains (AC) electricity. An example of a similar embedded system would be a computer cooling fan used as a CPU cooler or case fan. This project device is meant for personal use by individuals as its low power consumption lets it run on battery power and makes it portable.

Functional Description and Product Operation

The temperature regulator has a manual and automatic mode which the user can toggle between by pushing a button. In both modes the LCD will show the current ambient temperature in degrees Celsius and Fahrenheit. The currently active mode is denoted by the letter "M" for manual and "A" for automatic which can be found at all times in the bottom right corner of the LCD display. The LCD contrast can be adjusted using a second (dedicated) potentiometer knob.

In the manual mode, the user turns a potentiometer knob to set the desired fan speed as a percentage (10%-100%), which is shown on the second line of the LCD.

In the automatic mode, the user turns the same potentiometer knob to set the desired temperature. The currently set temperature is shown on the second line of the LCD in degrees Celsius and Fahrenheit. The user can set a temperature between 21°C (70°F) and 27°C (81°F). This range was chosen because the fan can cool the surroundings to achieve the user's set temperature within a reasonable amount of time (for demonstration purposes).

Hardware

The temperature regulator employs a variety of different sensors and actuators.

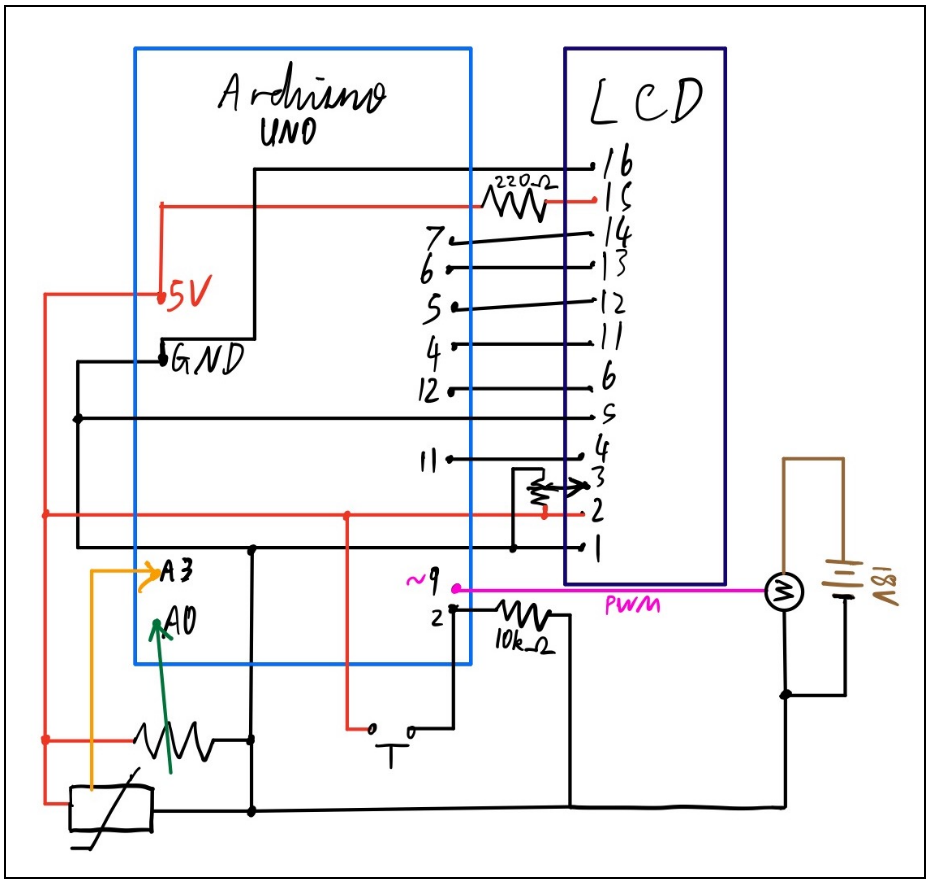
Sensors:

- Thermistor (temperature sensor)
- Potentiometer
- Push Button

Actuators:

- 120mm 4-pin PWM RGB Fan (12V DC motor)
- 16x2 LCD

Hardware Setup/Circuit Diagram



The thermistor connected to pin A3 is used to determine the current ambient temperature so the fan speed can be adjusted accordingly. The potentiometer on pin A0 is used to either set the desired temperature (in the automatic mode) or to set the fan speed (in the manual mode) by turning the knob. The push button conveniently switches the mode between automatic and manual by triggering an interrupt service routine on pin 2. The 120mm 4-pin PWM RGB Fan (12V DC motor) cools the surroundings and was chosen because its speed can be controlled directly with a PWM signal from digital pin 9, while the LCD displays information in a user-friendly manner. The fan's power pins are connected to an external circuit with 18V (from two 9V batteries in series) and to the Arduino circuit through its PWM pin.

Software

The code reads the three sensors, and uses the analog input (or digital in the case of the push button) to determine what signal to send to the two actuators.

Before the setup function, the LiquidCrystal.h header file is imported and some constants relating to fan speed and pin selection are set.

setup ()

The setup function sets the fan PWM pin mode to output and attaches a rising interrupt function (modeSwitch) to the push button on pin 2. Additionally, it sets the size of the LCD to 16x2 and displays a start screen with the project name and the names of its creators for 2.5 seconds before executing the rest of the code. Finally, it sets the initial mode as “manual.”

loop()

The loop function first clears the LCD then uses the analogRead() in order to obtain sensor values from the potentiometer and the thermistor. The program then uses the map function to convert the potentiometer value to a fan speed between 26 (10%) and 255 (100%). The fan speed is calculated as a percentage of 255. The user-defined function therm2cel converts the thermistor value to a temperature in degrees Celsius. The current temperature is printed to the LCD in degrees Celsius and Fahrenheit and the LCD cursor is set to the next line.

Since the mode is initially set to manual, the program will execute the else condition block, which instructs the fan to spin at the user-set speed using the analogWrite() function. The fan speed percentage is printed to the LCD display along with the letter “M” to denote that the system is in manual mode.

If the push button is pressed, the mode program will execute the if (automatic) condition block. It will first display the set temperature in degrees Celsius and Fahrenheit while also printing “A” for automatic. If the ambient temperature is more than 1°C above the set temperature, the fan speed is set to 100% (255). If it is within 1°C above the set temperature, the fan is set to 50% (127). If the current temperature is less than the set temperature, the fan will remain at 10% (26) to maintain the current temperature. Finally, the program will delay for a half-second before looping.

therm2cel()

This function takes the value read from the thermistor (pin A3), and converts it to millivolts (0V to 5000V) using the map function. Then, it determines the Celsius temperature using an equation based on the temperature-resistance relationship of the TMP36 thermistor: temp = (mV - 500.0) / 10. The function returns the temperature in degrees Celsius.

modeSwitch()

This function has no inputs or outputs. It is an interrupt service routine that toggles the value of the volatile boolean 'automatic' between false and true as long as it has been at least 200 milliseconds between two consecutive button pushes (to eliminate button bounce).

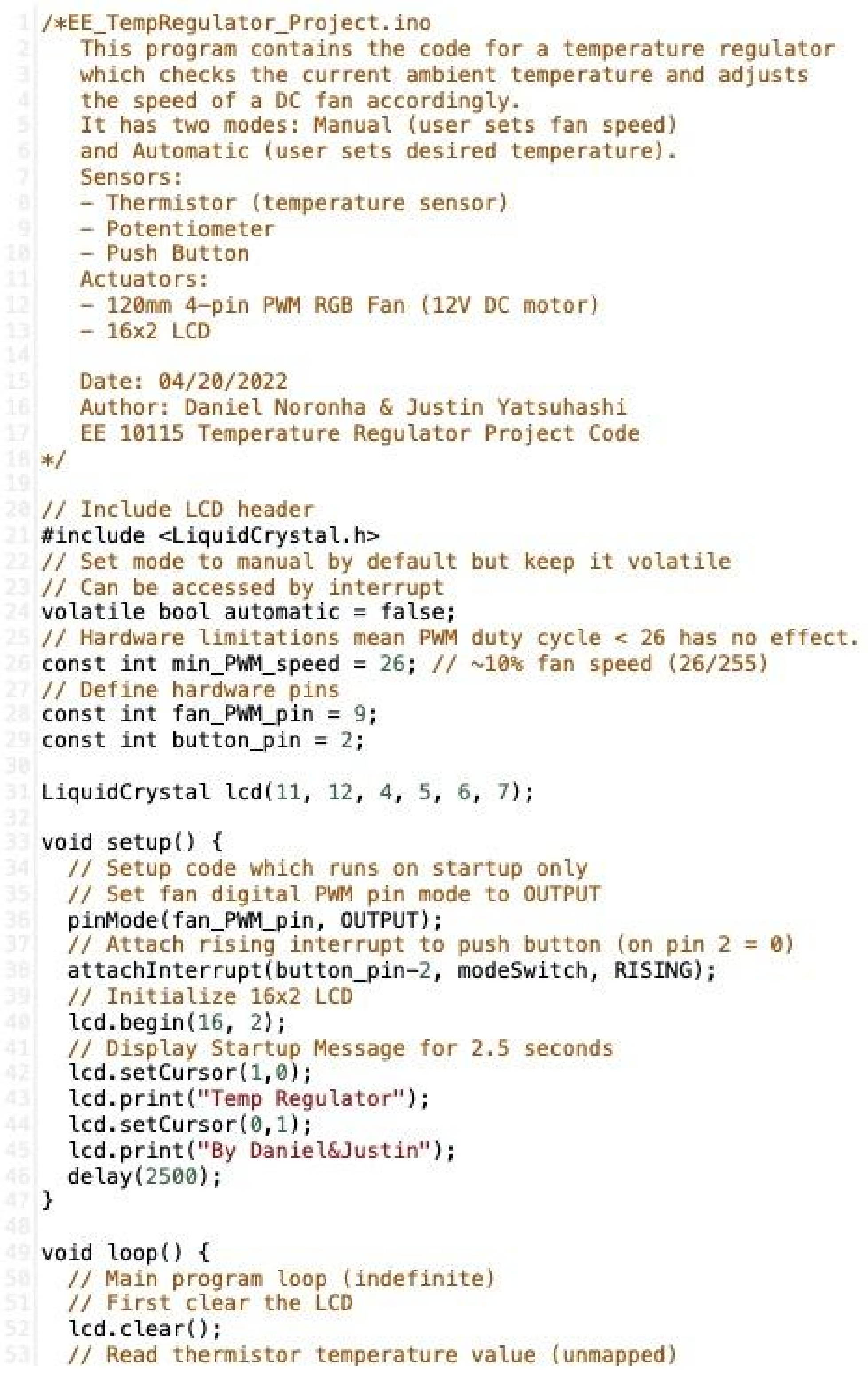
Program Flow

After initializing global variables, the program performs the setup() function and then proceeds to the loop() function. In the loop function(), the user-defined function therm2cel() is called to determine the current ambient temperature, before the program loops again. Pushing the button triggers the modeSwitch() interrupt service routine which toggles the system mode as needed.

Project Post-Mortem

The final product was almost the same as described in the initial proposal. The only change made was to display temperatures in Celsius and Fahrenheit instead of only in Celsius. Overall, the most interesting part of the project is its real-use application. The temperature regulator's function is basically the same as that of an air conditioning unit (except that it only uses a fan), and it was interesting to see how such a unit actually works. The system effectively uses a feedback mechanism to maintain the temperature that the user sets in the automatic mode and can lower the temperature quite quickly for a single fan. The most difficult part of the project was actually fitting all of the components and wires onto a single breadboard, and eventually the button had to be put onto a separate board because there was not enough room on the first one. If the project were to be done again, the system would be modified so that in automatic mode, the fan turns off completely when the temperature drops below the set temperature instead of spinning at idle (10% speed). The fan itself currently does not spin at less than 10% even with a PWM duty cycle of 0 (a design limitation). The provided MOSFET could not be used to stop the fan either because with a gate voltage of 5V, the RDS(on) is too high resulting in a very high voltage drop across the transistor (~15V/18V) when in the on state (which leaves too little voltage for the fan to run). This issue could be fixed by using a transistor with a lower RDS(on) value. Also, in automatic mode, a proportional control algorithm could be used instead of stepping between 0%, 50%, and 100% fan speed.

Appendix



```
/*EE_TempRegulator_Project.ino
   This program contains the code for a temperature regulator
   which checks the current ambient temperature and adjusts
   the speed of a DC fan accordingly.
   It has two modes: Manual (user sets fan speed)
   and Automatic (user sets desired temperature).
   Sensors:
   - Thermistor (temperature sensor)
   - Potentiometer
   - Push Button
   Actuators:
   - 120mm 4-pin PWM RGB Fan (12V DC motor)
   - 16x2 LCD

   Date: 04/20/2022
   Author: Daniel Noronha & Justin Yatsuhashi
   EE 10115 Temperature Regulator Project Code
*/

// Include LCD header
#include <LiquidCrystal.h>
// Set mode to manual by default but keep it volatile
// Can be accessed by interrupt
volatile bool automatic = false;
// Hardware limitations mean PWM duty cycle < 26 has no effect.
const int min_PWM_speed = 26; // ~10% fan speed (26/255)
// Define hardware pins
const int fan_PWM_pin = 9;
const int button_pin = 2;

LiquidCrystal lcd(11, 12, 4, 5, 6, 7);

void setup() {
  // Setup code which runs on startup only
  // Set fan digital PWM pin mode to OUTPUT
  pinMode(fan_PWM_pin, OUTPUT);
  // Attach rising interrupt to push button (on pin 2 = 0)
  attachInterrupt(button_pin-2, modeSwitch, RISING);
  // Initialize 16x2 LCD
  lcd.begin(16, 2);
  // Display Startup Message for 2.5 seconds
  lcd.setCursor(1,0);
  lcd.print("Temp Regulator");
  lcd.setCursor(0,1);
  lcd.print("By Daniel&Justin");
  delay(2500);
}

void loop() {
  // Main program loop (indefinite)
  // First clear the LCD
  lcd.clear();
  // Read thermistor temperature value (unmapped)
  int sensorVal = analogRead(A3);
  // Read potentiometer value
  int potVal = analogRead(A0);

  // Use therm2cel function defined below to map thermistor value to
temperature
  float temp = therm2cel(sensorVal);// temperature in °C
  // Display current temperature on first line
  lcd.print(temp);
  lcd.print("*C");
  lcd.print("  ");
  lcd.print(temp*1.8+32); // °C to °F
  lcd.print("*F");
  lcd.setCursor(0,1);
  // If the mode is automatic:
  if (automatic) {
    // Map knob setting to temperature range (21°C-27°C)
    int settemp = map(potVal, 0, 1023, 21, 27);
    // Display set temperature on second line
    lcd.print("Set: ");
    lcd.print(settemp);
    lcd.print("*C=");
    lcd.print(int(settemp*1.8+32)); // °C to °F
    lcd.print("*F A"); // Show selected mode as Automatic (A)
    // If the current temperature is less than 1°C above the
    // set temperature, then set fan speed to 50% (127/255)
    if (int(temp) == settemp) analogWrite(fan_PWM_pin, 127);
    // Otherwise, if the current temperature is more than 1°C above the
    // set temperature, then set fan speed to 100% (255/255)
    else if (temp > settemp) analogWrite(fan_PWM_pin, 255);
    // If the current temperature is less than the set temperature,
    // then set the fan to idle (10%)
    else analogWrite(fan_PWM_pin, 0);
  }

  else {
    // Manual Mode
    // Map knob setting to PWM fan speed (26-255)
    int fanSpeed = map(potVal, 0, 1023, min_PWM_speed, 255);
    // Express PWM fan speed as a percentage (10%-100%)
    int percent = (fanSpeed/255.0) * 100;
    // Send PWM signal to fan
    analogWrite(fan_PWM_pin,fanSpeed);
    // Display user-set fan speed percentage on second line
    lcd.print(percent);
    lcd.print("%");
    lcd.setCursor(15,1);
    lcd.print("M"); // Show selected mode as Manual (M)
  }
  delay(500);
  // Wait half a second to let user read display before clearing

}
```

```
float therm2cel(int thermval) {
  // User-defined function to map thermistor value to temperature (°C)
  // Map thermistor value to millivolts (0-5mV)
  float mVolts = map(thermval, 0, 1023, 0, 5000);
  // Use linear equation to convert voltage to Celsius temperature
  float temp = (mVolts - 500.0) / 10.0;
  return temp;
}

void modeSwitch() {
  // Interrupt Service Routine to toggle system mode selection (A/M)
  // Checks how long it has been since last processed interrupt
  // If it has been less than 0.2s, do not process interrupt
  // Eliminates button bounce
  static unsigned long last_processed_interrupt_time = 0;
  unsigned long current_interrupt_time = millis();
  if ((current_interrupt_time - last_processed_interrupt_time) > 200) {
      // Bit-flip boolean value of global volatile variable automatic (0/1)
      automatic = !automatic;// false=>true or true=>false
  }
  last_processed_interrupt_time = current_interrupt_time;
}

// End of Program //
```