Temperature Operated Summer Fan

Rishabh Roy
ME 102B: Mechatronics Design, Fall 2020
December 8, 2020

Description:

I usually get hot easily during the summer months and it is not always feasible to turn the AC on. Sometimes, even when the AC is turned on, I still feel uncomfortable because I cannot feel the draft of the AC. Now you might be asking – why not use a regular fan and turn it on when I am hot as this way I can feel the draft? The reason is inconvenience – I would need to manually get up and turn on the fan when I am hot and turn off the fan when I am cold.

To rectify this, I decided to build a simple yet efficient temperature-controlled fan. The fan turns on when the ambient temperature of the surroundings is greater than some set max temperature (can be set by the user) and the fan turns off when the ambient temperature of the surroundings is less than a specified min temperature (can be set by the user). The device also has a manual stop switch that can be used to turn off the fan regardless of whether the temperature threshold for turning off the fan has been met or not.

This device also helps when I am sleeping as it controls when the fan is on or off based on the temperature. Previously I would have to get up to turn a fan on or off depending on if I felt hot or cold, but by using this I can get a good night's rest as the fan will automatically turn itself on or off during the night based on the temperature of the surroundings. This saves electricity as the fan is not always on and allows me to sleep comfortably.

***Figure 1**: Overall design of my machine housing*

Electromechanical Details:

This project required numerous fabricated elements. The 3 most important elements that I fabricated to ensure that the device worked as intended were the (1) housing, the (2) fan, and the (3) button pusher. I built all these components using items that were either included in the MicroKit we received for this class or I had at home.

(1) **Housing:**

I made the housing from an old shoe box I had lying around the house. One benefit of using the shoebox was that it already came with a top covering. I punched 3 holes in the base of shoe box. I inserted the wire connecting the microcontroller to my computer through the hole in the back of the box (H1) and the wires connected to the external power supply through the hole in the side of the box (H2). I put the shaft of my motor through a hole a created on the front of the box (H3) and attached the fan to it from the other side. I then used tape to seal off all the holes after the wires were fed through. I made sure that the hole for the motor shaft was a bit bigger than the diameter of the shaft to limit friction between the shaft and the box when the motor was spinning. I also created two holes on the top of the box – one to insert

my button pusher and one to insert a straw so that I could increase the internal temperature of the box. As a next step I would create a new housing unit that would me bade using laser-cut interlocking plywood.

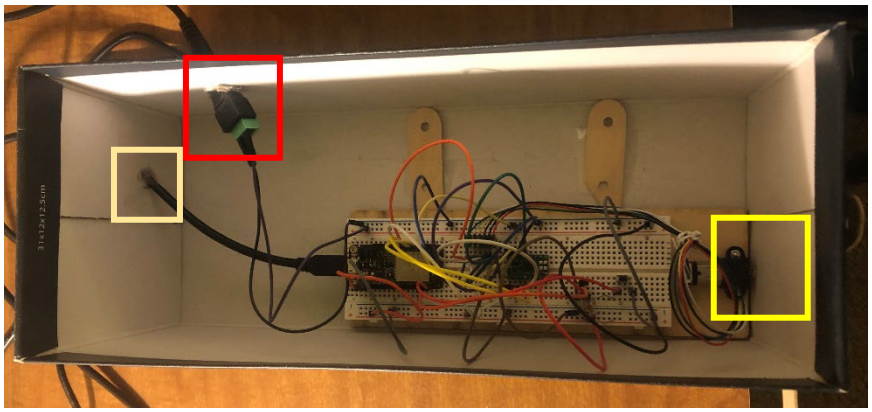

***Figure 2**: Overall design of my machine including housing. The cream square is H1, the red square is H2, and the yellow square is H3.*

(2) **Fan:**

I built the fan from the circular marker and the popsicle sticks that came with the MicroKit. I first cut the popsicle stick to a smaller size and then attached them to the circular marker using the brass pins that came with the MicroKit. To make sure that the popsicle sticks stayed in place I then taped them to the circular marker and to each other. This homebuilt fan worked to demonstrate the workings of the machine and provided a slight breeze that was sufficient in colling the ambient temperature near the sensor. As a next step, I would invest in an actual fan to provide for greater cooling capabilities.

***Figure 3**: Homebuilt Fan*

(3) **Button Pusher:**

I built the button pusher from a chopstick. I cut and sanded down one side of the chopstick to make sure it was of appropriate length. As a next step I would 3D print a custom object that would allow me to press the manual shut off button.

***Figure 4**: Button Pusher*

Circuit:

This project incorporated numerous elements into the final circuit. The 4 most important components in the circuit include the (1) ESP32 Feather Board, the (2) Micro Metal Gearmotor, the (3) DS18B20 Temperature Sensor, and the (4) Single Pole Single Throw Tactile Pushbutton. More details regarding these parts are listed below:

(1) **Assembled Adafruit HUZZAH32 – ESP32 Feather Board – with Stacking Headers:**

This microcontroller was included in the MicroKit we received for this class. I used this to interface with the other components of this circuit. I also flashed my code into it to run my device. The microcontroller is powered by connecting it to my computer and it runs on 3.3V. I attached the microcontroller to the base plate so that it would be easier to position and the assembly would be more compact.

(2) **Micro Metal Gearmotor HP 6V with Extended Motor Shaft:**

This motor was included in the MicroKit we received for this class. I used this to drive my fan according to my predefined state. I powered this motor with the 5V external power supply. I bolted the motor to the faceplate using a plastic bracket and attached the fan to the motor using a shaft hub and a circular marker. I controlled the speed and directionality of the motor using a H-bridge. All the components used alongside the Gearmotor were included in the MicroKit.

(3) **DS18B20 Temperature Sensor:**

This is a one wire digital temperature sensor, meaning that you only need to connect it to a single GPIO pin to communicate data with it. I bought this on Adafruit as it is solderless (already comes ready to use) and simple to interface with. I used two Arduino libraries to interface with this sensor – a onewire library by Paul Stroffregen and a Dallas library written by Miles Burton. I also used a 4.7kΩ pullup resistor with the sensor to ensure clear logic signals for the microcontroller to read (was included with the sensor purchase). The DS18B20 has a designated -67°F to +257°F temperature range with a ±1.8°F accuracy for temperatures between +14°F to +185°F. This is perfect for my application as ambient room temperature ranges from +60°F to +90°F. Both bounds are within the optimal temperature bounds of this sensor. I connected this peripheral to one of the GPIO (17) pins on my microcontroller.

(4) **Single Pole Single Throw Tactile Pushbutton:**

This switch was included in the MicroKit we received for this class. I used this switch to act as the manual stop that the user can use to turn the fan off even if the shutting off temperature has not been reached. I used a 22kΩ pullup resistor with this switch. This value was calculated in a previous assignment for this class. Like its use with the DS18B20 sensor, the pullup resistor ensures clear logic signals for the microcontroller to read. I connected this peripheral to one of the GPIO (21) pins on my microcontroller.

I debugged my circuit using print statements. They have been left in the code that I have attached to the appendix but have been commented out. I also used the print lines to gather data for my plots to demonstrate the effectiveness and working of my state machine.

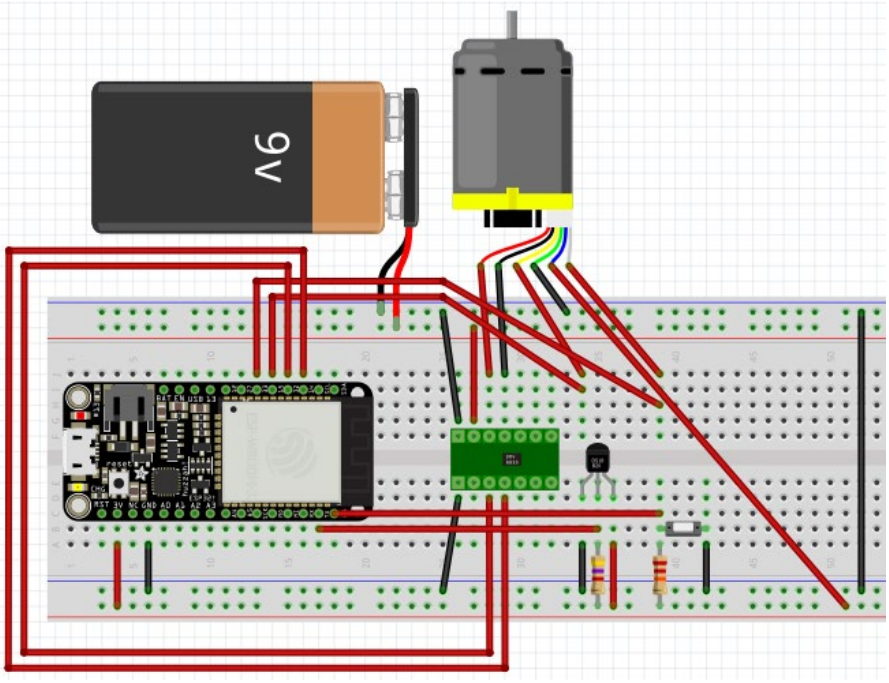

***Figure 5**: Fritzing circuit diagram for my Temperature Controlled Fan*

Finite State Machine:

I designed the behavior of the machine to be as simple as possible. Given that the output of the machine is the fan either being “ON” or “OFF”, the machine lent itself well to using a bang-bang control scheme. Since we don’t really care about the speed of the fan, its rise time, or settling time, a bang-bang controller was

appropriate with this machine. One common complaint about bang-bang control is that that it can cause the motor to turn on and off very frequently. This however was not of concern in this case as the fan (motor) is being used in a thermal control setting. Temperature control is one of the few good cases for using bang-bang control as the system is resistant to change In this case, it takes a considerable amount of time for the temperature to rise above its set max value and fall below its set min value.

To reduce the frequency of the fan turning "ON" and "OFF", I introduced hysteresis to the system. To depict this hysteresis, the ambient surrounding temperature must be greater than 70°F for the fan to turn on automatically and be below 68°F for the fan to turn off automatically. I also included a mechanism to turn off the fan even if the lower temperature threshold was not met. This allows for the user to turn off the fan in the case of an incident or if they are simply feeling cold. Additionally, I limited the maximum duty cycle of the motor to be 80% of the maximum 255. As a result, the motor is run at a PWM duty cycle of 51 on the "ON" state and at a PWM duty cycle of 255 on the "OFF" state. NOTE: the PWM duty cycles are such due to the motor being run in brake mode.

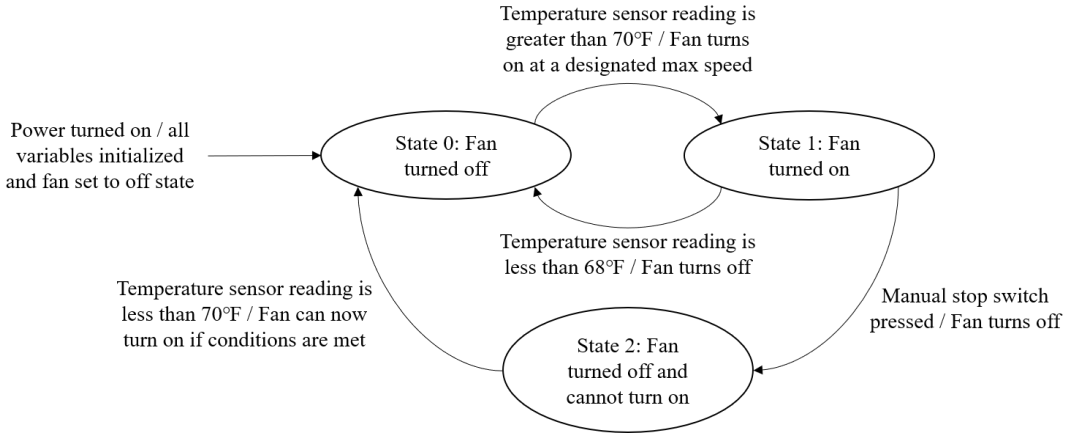

***Figure 6**: Finite State Machine Drawing for my device of my machine (state # refers to Arduino code) code)*

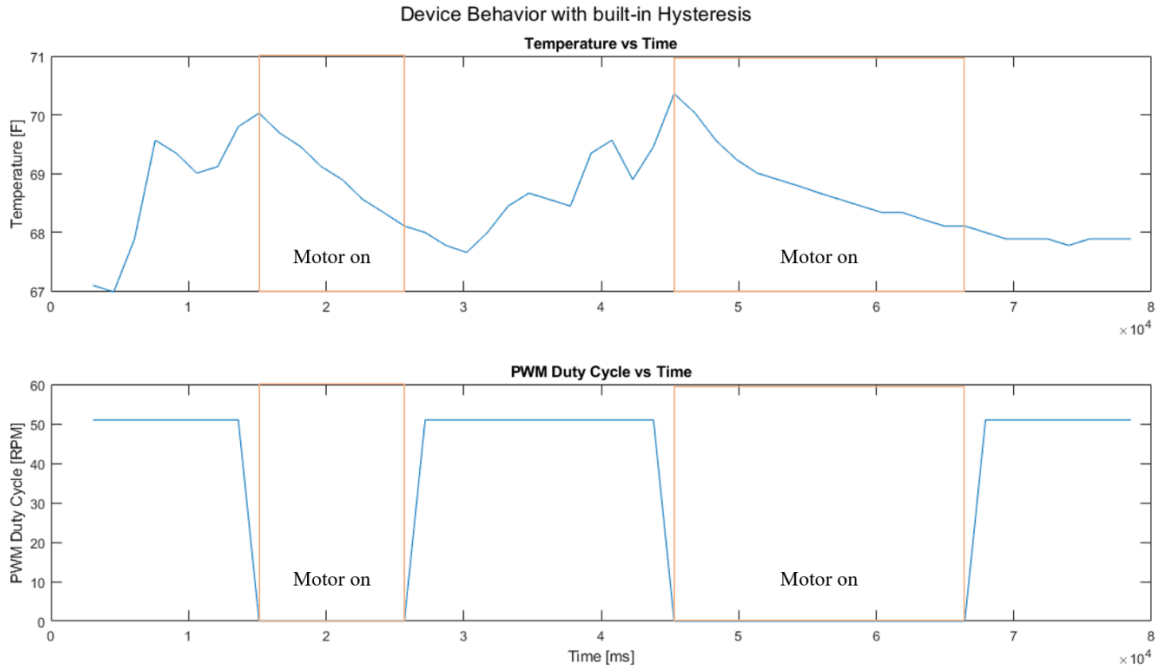

***Figure 7**: Graph of device behavior showcasing Hysteresis. Motor turns on only when temperature exceeds 70°F and turns off when temperature is less than 68°F. The motor is off for the areas of the graph not marked as "Motor on"*

Appendix:

Appendix 1: Arduino Code

```
//Project Code - Rishabh Roy

// Including libraries
#include <OneWire.h>
#include <DallasTemperature.h>

// Temperature bounds
const float max_temp = 70.00;
const float min_temp = 68.00;
float temp = 0.00;
int fan_state = 0;

// Setting up Temp sensor
#define TEMP_SENSOR 17
OneWire oneWire(TEMP_SENSOR);
DallasTemperature sensors(&oneWire);

// Setting up SWITCH
// Setting the Button pin to pin 14 of micro
#define PIN_BUTTON 21
// Creating a volatile Boolean to control our interrupts
volatile bool interrupt_bool_switch = false;
// The current time passage
unsigned long last_time = 0;
// Setting up var to store millis
unsigned long interrupt_time;

// Setting up MOTOR
#define PIN_MOTOR_1 32
#define PIN_MOTOR_2 14

// Setting up PWM
// Setting PWM properties
const int freq = 20000;
const int motor_channel_1 = 0;
const int motor_channel_2 = 1;
const int resolution = 8;
// Setting max PWM Duty Cycle
const int max_duty = 51;

// Defining our SWITCH interrupt function
void IRAM_ATTR hard_stop() {
  if (interrupt_time - last_time > 100) {
    interrupt_bool_switch = true;
  }
  last_time = interrupt_time;
}

void setup() {
  // Setup for MOTOR
  // configure MOTOR PWM functionalities
  ledcSetup(motor_channel_1, freq, resolution);
  // attach the channel to the GPIO to be controlled
  ledcAttachPin(PIN_MOTOR_1, motor_channel_1);
```

```
  // configure MOTOR PWM functionalities
  ledcSetup(motor_channel_2, freq, resolution);
  // attach the channel to the GPIO to be controlled
  ledcAttachPin(PIN_MOTOR_2, motor_channel_2);

  // Setup for SWITCH
  // Defining the Button pin as an input
  pinMode(PIN_BUTTON, INPUT);
  // Setting up interrupt based on button
  attachInterrupt(digitalPinToInterrupt(PIN_BUTTON), hard_stop, RISING);

  // Turn off MOTOR at the start
  service_false();

  // Setup Temperature Sensor
  sensors.begin();

  // Setup SERIAL MONITOR
  Serial.begin(250000);
}

void loop() {
  // Setting up MILLIS function - deals with debounce for switch
  interrupt_time = millis();

  // The loop runs every 1 second
  delay(1000);
  sensors.requestTemperatures();
  temp = sensors.getTempFByIndex(0);

  // Control the state of the machine
  switch (fan_state) {
    // Fan is off
    case 0:
      if (temp >= max_temp) {
        // Run the fan
        service_true();
        fan_state = 1;
        break;
      }
      break;
    // Fan is on
    case 1:
      if (temp <= min_temp) {
        // Do not run the fan
        service_false();
        fan_state = 0;
        break;
      }
      if (interrupt_bool_switch) {
        // Do not run the fan
        service_false();
        fan_state = 2;
        break;
      }
      break;
    // Fan is off and does not turn on if temp > max_temp
    case 2:
```

```
      if (temp < max_temp) {
        // Resets the interrupt bool
        interrupt_bool_switch = false;
        fan_state = 0;
        break;
      }
      break;
    default:
      break;
  }

  // (Debugging) Print out temp
  // Serial.print(interrupt_time);
  // Serial.print(", ");
  // Serial.print(temp);
  // if (fan_state == 0) {
  //  Serial.print(max_duty);
  // } else {
  //  Serial.print("0");
  // }
  // Serial.println(";");
}

// Service routine for fan on
void service_true() {
  ledcWrite(motor_channel_1, 255);
  ledcWrite(motor_channel_2, max_duty);
}

// Service routine for fan off
void service_false() {
  ledcWrite(motor_channel_1, 255);
  ledcWrite(motor_channel_2, 255);
}
```

Appendix 2: MATLAB Code

```
% Data to be plotted
data = [
    3056, 67.10, 51;
    4565, 66.99, 51;
    6074, 67.89, 51;
    7583, 69.57, 51;
    9092, 69.35, 51;
    10601, 69.01, 51;
    12110, 69.12, 51;
    13619, 69.80, 51;
    15128, 70.03, 0;
    16637, 69.69, 0;
    18146, 69.46, 0;
    19655, 69.12, 0;
    21164, 68.90, 0;
    22673, 68.56, 0;
    24182, 68.34, 0;
    25691, 68.11, 0;
    27200, 68.00, 51;
    28709, 67.78, 51;
    30218, 67.66, 51;
```

```
    31727, 68.00, 51;
    33236, 68.45, 51;
    34745, 68.67, 51;
    36254, 68.56, 51;
    37763, 68.45, 51;
    39272, 69.35, 51;
    40781, 69.57, 51;
    42290, 68.90, 51;
    43799, 69.46, 51;
    45308, 70.36, 0;
    46817, 70.03, 0;
    48326, 69.57, 0;
    49835, 69.24, 0;
    51344, 69.01, 0;
    52853, 68.90, 0;
    54362, 68.79, 0;
    55871, 68.67, 0;
    57380, 68.56, 0;
    58889, 68.45, 0;
    60398, 68.34, 0;
    61907, 68.34, 0;
    63416, 68.22, 0;
    64925, 68.11, 0;
    66434, 68.11, 0;
    67943, 68.00, 51;
    69452, 67.89, 51;
    70961, 67.89, 51;
    72470, 67.89, 51;
    73979, 67.78, 51;
    75488, 67.89, 51;
    76997, 67.89, 51;
    78506, 67.89, 51;
    ];

% Extracting data series
time = data(:, 1);
temp = data(:, 2);
duty_cycle = data(:, 3);

% Plotting
figure(1)
subplot(2, 1, 1)
plot(time, temp);
ylabel("Temperature [F]");
title("Temperature vs Time");
subplot(2, 1, 2)
plot(time, duty_cycle);
ylabel("PWM Duty Cycle [RPM]");
title("PWM Duty Cycle vs Time");
xlabel("Time [ms]");
sgtitle("Device Behavior with built-in Hysteresis");
```