COMPUTING AND MATHEMATICS PROGRAMME AREA ASSIGNMENT SUBMISSION FORM (including group work)

Please complete all sections clearly in ball point pen

Do not write your name on your work unless your lecturer has explicitly told you to do so.

Student ID number	Title of degree studying	Level/Year
UP2200923	Bachelor of Science Data Science And Analytics	2022/2023

Short unit name:	IOT	**Due date:** 18/12/2023		**Deadline:** 18/12/2023	
Full unit name:	INTERNET OF THINGS				
Unit lecturer name:		**Group:** *(if applicable)*			
Additional items e.g. CD/disk/USB:	**Yes**		**No**		**Details:**

All additional items should be clearly labelled with ID number and unit name and securely attached to your work.

Candidates are reminded that the following are defined as Assessment Offences and will be dealt with in accordance with the University's Code of Student Discipline:

a) Any attempt to complete an assessment by means considered to be unfair;
b) Plagiarism, which the University defines as the incorporation by a student in work for assessment of material which is not their own, in the sense that all or a substantial part of the work has been copied without any adequate attempt at attribution or has been incorporated as if it were the student's own work when in fact it is wholly or substantially the work of another person or persons.

Please note: **Group coursework will be filed under the first Student ID number at the top of the list. Ensure you know all group member's ID numbers.**

NB: Coursework not collected will be disposed of six months after the hand-in date.

FOR OFFICIAL USE ONLY

Administration Office	Academic Staff Member
Date received/Office stamp	Provisional mark % / Comments

2023

Automated Fan Control

UP223923

RENATO TAN

Contents

Introduction

In Singapore, the weather conditions are usually quite hot and humid; averaging temperatures of around 30°C. Naturally, people will be looking for different methods to cool their homes and a few methods come to mind: Fans; from standing to ceiling fans. Air conditioning: Usually in the form of wall mounted system and an exterior Inverter.

From my personal experience, I frequently use a ceiling fan for cooling my room. With that in mind I started brainstorming an idea for an automated fan control system.

Project Aims

This project aims to develop an automated fan control system by using the DHT-11 sensor as well as a L9110 Motor.

The system will respond dynamically to the surrounding temperature fluctuations, adjusting the fan speed based on pre-defined thresholds.

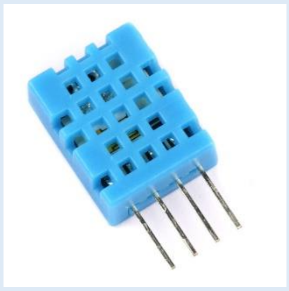
This should allow for efficient temperature regulation while minimizing fan noise and energy consumption.

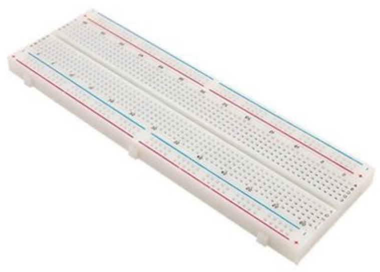
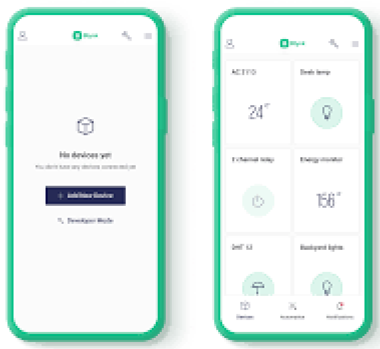
Using hysteresis to prevent rapid switching on/off, this will allow a smoother fan operation and preserve the motor from overwork.

Additionally, I plan to introduce a wireless card to allow for remote control of the device. This will use an external application (Blynk) to monitor and control the device.

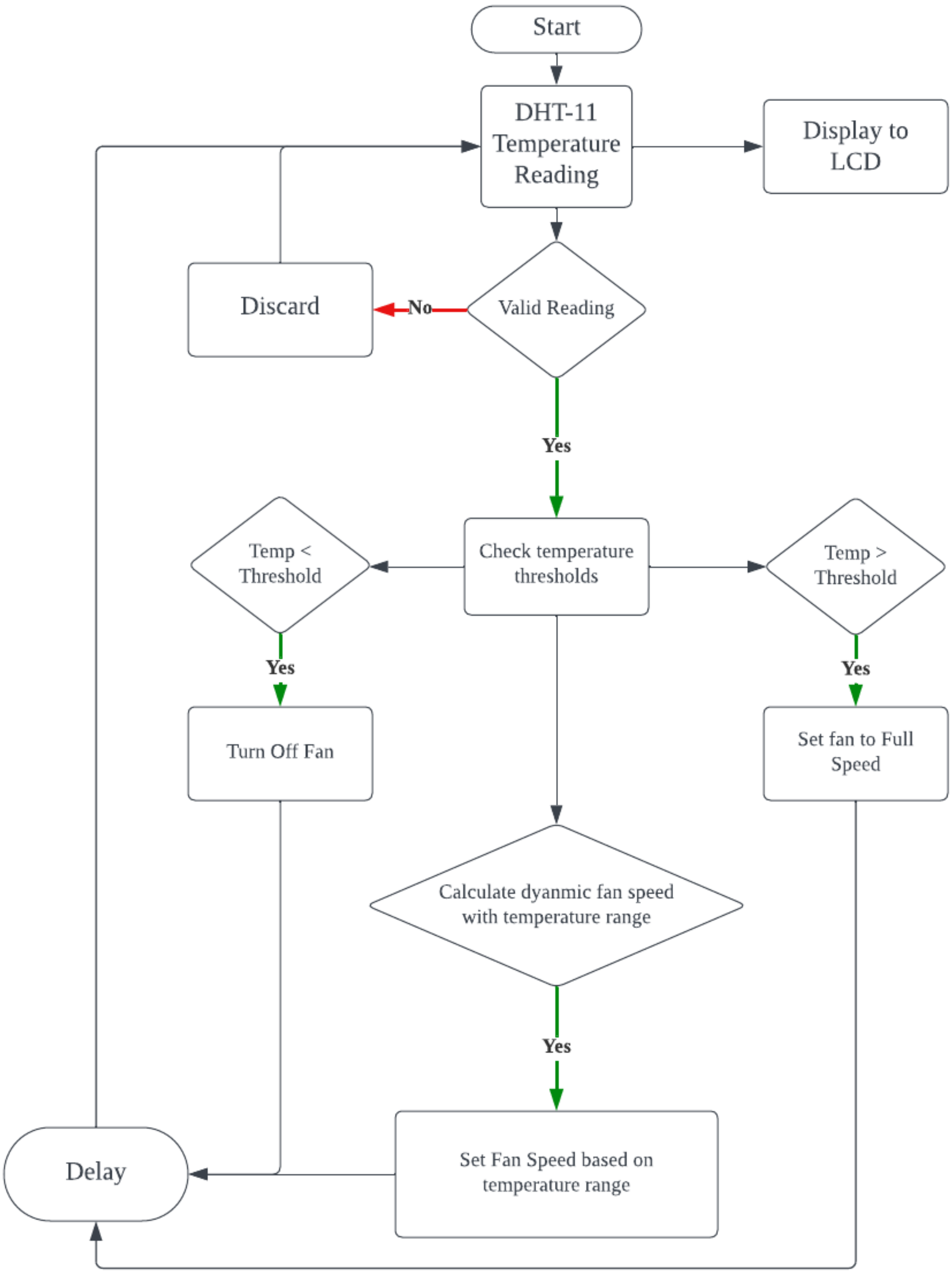
The other important aim of this project is to keep it simple and modular. The codes are kept straightforward and focused on core functionality (Fan control).

Hardware

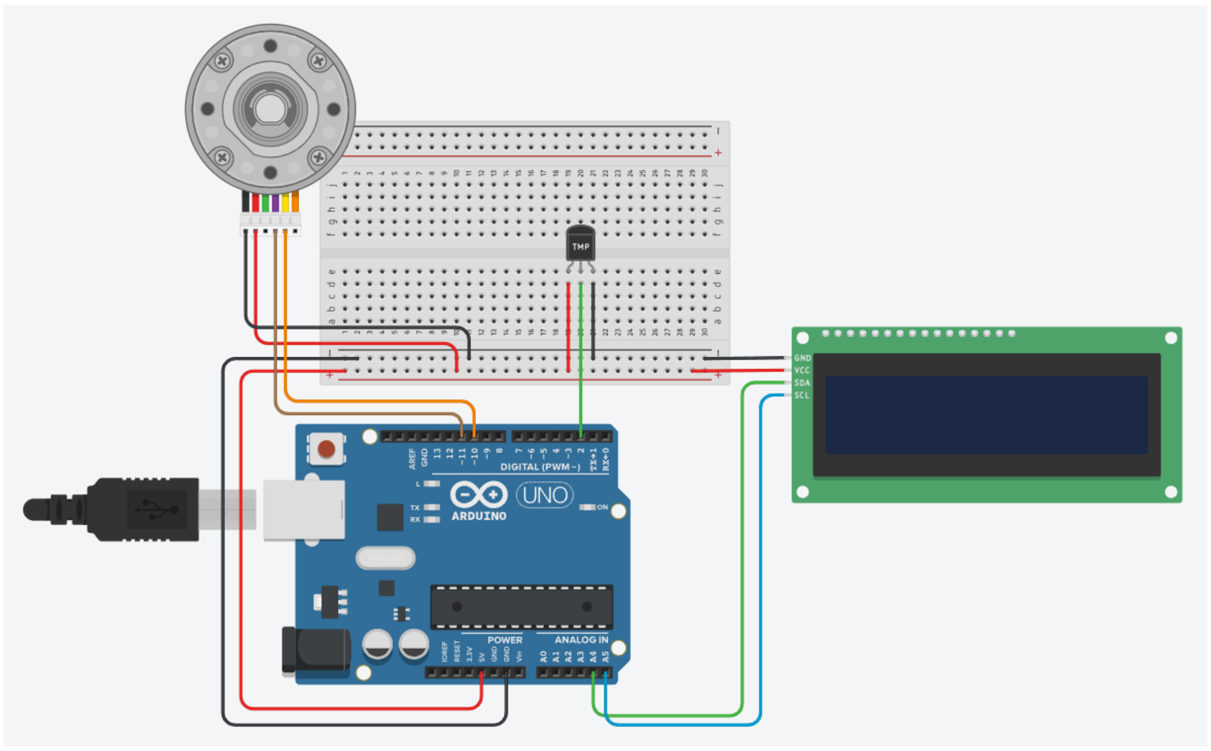
Name	Description	Cost
Arduino Uno	The main controller. All codes will be run through the Arduino IDE and passed into the board. It is also the main connection for all other parts.	SGD34.00
Jumper Wires	Cables used to connect the parts together.	SGD12.00
DHT-11 	Temperature Sensor used to record ambient temperature. Although not as precise as other temperature reading sensors, this will be enough for the fan controller to use.	SGD5.00
L9110 Motor with Fan blades	Motor used to power the spinning fan blades. This motor has specific pins that allows the control of the RPM of the motor; adjustable RPM will be the core feature of this project. Within this project, a Hysteresis value will be used to create a buffer zone to prevent rapid fan cycling.	SGD6.00

16x7 LCD Display w I2C	Screen used to show temperature and fan status. Mainly used for debugging and easy to glance during the runtime operations.	SGD10.00
Breadboard 	Used as an extension for cable connections. The breadboard is an inexpensive tool for extensive inputs and cable management.	SGD5.50
ESP8266 ESP-01 Wireless Card	Wireless card used for integrating the Blynk app. This wireless card allows users to connect to the local network which can be used for a wide array of functions.	SGD11.00
Blynk App 	App used for monitoring the automated fan system. Can be programmed to monitor temperature, fan speed, and remotely control the automated fan control.	Free/Paid if using enhanced features

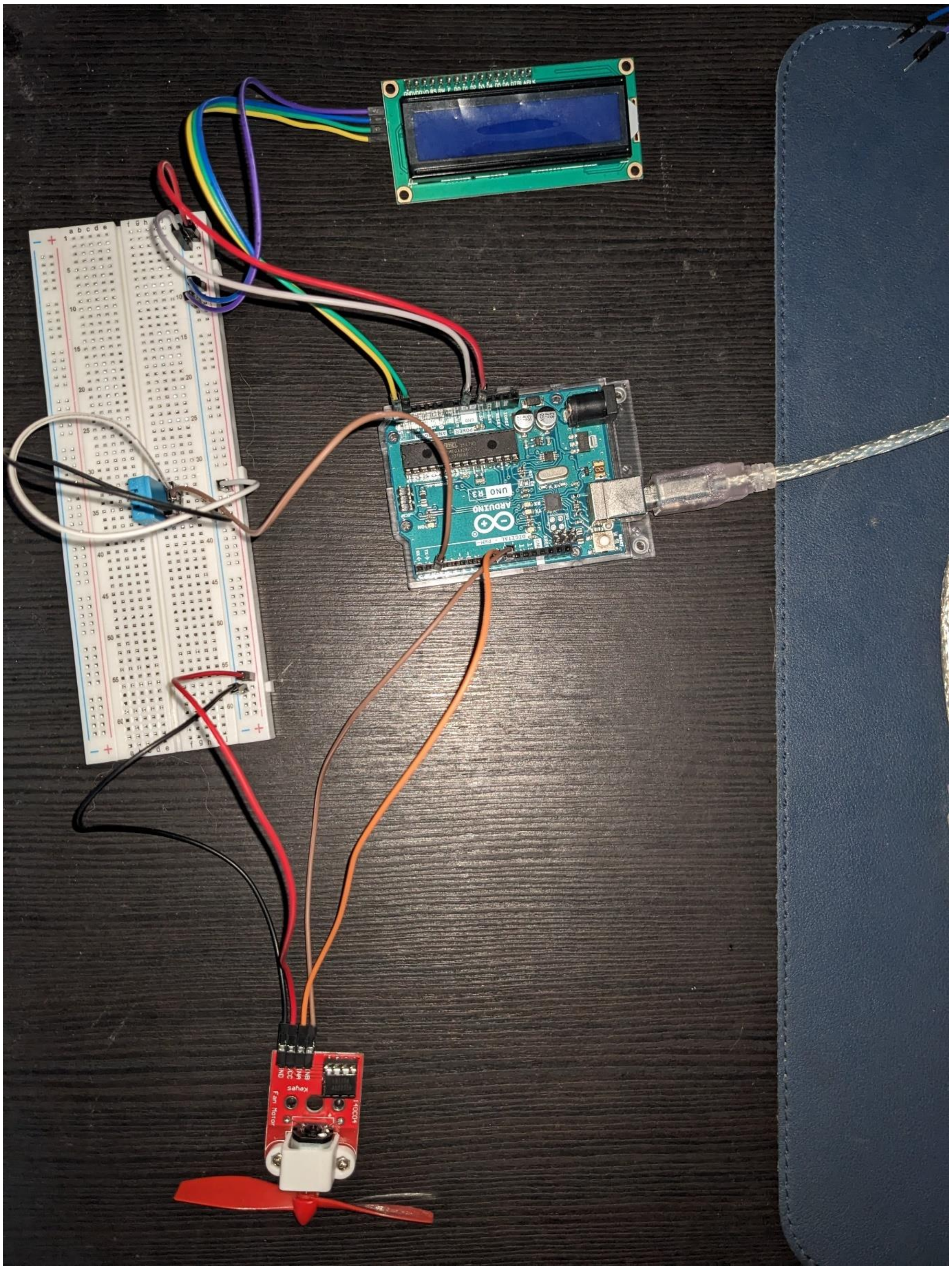
Flow Chart


Start
DHT-11 Temperature Reading
Display to LCD
Discard
No
Valid Reading
Yes
Temp < Threshold
Check temperature thresholds
Temp > Threshold
Yes
Yes
Turn Off Fan
Set fan to Full Speed
Calculate dyanmic fan speed with temperature range
Yes
Delay
Set Fan Speed based on temperature range

Circuit



In this virtual circuit Diagram, the motor depicted is not accurate to the current fan I was using.



1. DHT-11 is connected to the breadboard and controller via a 5V pin, GND and Input pin 2.
2. DHT-11 will read the ambient temperature in intervals of 5 seconds.
3. A LCD display with I2C welded on is used; It is connected by a 5V pin, GND, SDA and SCL. SDA and SCL are connected to A5 and A4 respectively.
4. The L9110 Motor fan is connected by a 5V pin, GND, INA and INB; INA and INB are connected to pin 9 and pin 10 respectively.

Arduino Codes

```
//Include libraries
#include <DHT11.h>
#include <LiquidCrystal_I2C.h>

//Define pins
const int sensePin = 2; //DHT sensor pin
const int INA = 9; //Fan control pin
const int INB = 10; //Fan control pin

//define constants
const float FAN_ON_TEMP = 31.0; //define turn on threshold
const float FAN_OFF_TEMP = 29.5; //define turn off threshold
const float HYS = 0.5; //Define hysteresis value
const float FAN_ON_TEMP_HYS = FAN_ON_TEMP + HYS; //define
hysteresis-adjusted on threshold
const int MIN_FAN_SPEED = 25; //minimum fan speed

//Define objects
DHT11 dht11(sensePin); //Define DHT11
LiquidCrystal_I2C lcd(0x27, 16, 2); //Define LCD display
```

Initialization:

1. Include libraries used in this project.
2. Define pins used on the physical connection.
3. Define constants for DHT-11 and Fan Speed
4. Define objects.

Thresholds for fan control and dead zone around the OFF thresholds are defined with 31 degrees Celsius for turning the fan ON, 29.5 degrees Celsius to turn off the fan and a 0.5 degrees Celsius buffer.

A hysteresis buffer is added to prevent rapid fan switching and improving motor life. This is done by adding a different threshold to turn the fan back on when compared to turning the fan off.

```
void setup() {
                          pinMode(INA, OUTPUT);
                          pinMode(INB, OUTPUT);
                          //setup lcd
                          lcd.init();
                          lcd.backlight();
                          lcd.begin(16, 2);
                          lcd.setCursor(0, 0);
                          lcd.print("Temperature: ");
}
```

Setup:

1. Set pin modes for INA and INB. This affects the fan speed and its output direction.
2. Set pin mode for the LCD, which is using an I2C module for ease of connection.
3. Prints temperature on the first line of the LCD.

```
void loop() {
	//read temperature
	int temperature = dht11.readTemperature();
	//Debug message
	if (isnan(temperature)) {
		Serial.println("Failed to read temperature");
		return;
	}
	//Printing to LCD and clearing second line
	Serial.print("Temperature: ");
	Serial.print(temperature);
	Serial.println(" °C");
	lcd.setCursor(0, 1); //Set to second line
	lcd.print("                  "); //Clear the line

	//Turn fan on at maximum speed
	if (temperature > FAN_ON_TEMP_HYS) {
		analogWrite(INA, 255); //Max speed, adjustable
		digitalWrite(INB, LOW);
		delay(2000); //Delay for 2 seconds
		lcd.setCursor(0, 1); //Set to second line
		lcd.print("Fan Speed: Max"); //print fan speed to LCD
	}
	//turn fan off
	else if (temperature < FAN_OFF_TEMP) {
		digitalWrite(INA, LOW);
		digitalWrite(INB, LOW);
		delay(5000); //Delay for 5 seconds
		lcd.setCursor(0, 1); //set to second line
		lcd.print("Fan Speed: OFF"); //print fan speed to LCD
	}
	//Set dynamic fan speed based on temperature
	else {
		int fanSpeed = map(temperature, FAN_OFF_TEMP,
FAN_ON_TEMP, MIN_FAN_SPEED, 255); //max fan speed is adjustable
		analogWrite(INA, fanSpeed); //Sets the voltage on INA pin
to control fan speed from map
		digitalWrite(INB, LOW); //sets INB to low, for motro
directions
		delay(2000);
		lcd.setCursor(0, 1);
		lcd.print("Fan Speed: ");
		lcd.print(fanSpeed); //prints fan speed value
	}
	//Display temperature and update LCD
```

```
      lcd.setCursor(12, 0);
      lcd.print(temperature);
      lcd.display();

      //Delay between readings 5s
      delay(5000);
}
```

During the loop operations:

The Temperature is read and stored as a float, a check for valid reading is used and prints error message if needed. The message will print to both the Serial Monitor and LCD display.

Fan ON: if temperature exceeds 31 degrees Celsius, INA is set to the maximum speed, INB is set to LOW for proper fan direction. The LCD will also display the max speed.

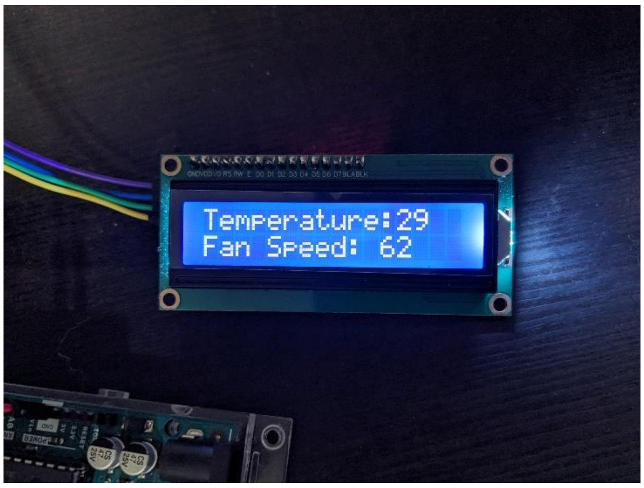
Fan OFF: When temperature falls below 29.5 degrees Celsius, both INA and INB are set to LOW; the fan will turn off. The LCD will display a Fan is OFF message.

If the temperature falls between the hysteresis thresholds: Mapping temperature to fan speed.

The map function maps the current temperature to a corresponding fan speed between the minimum and maximum values. This allows dynamic fan speed adjustments based on the current temperature. Lastly, the LCD will display the calculated speed.

The LCD will also display the current temperature readings, this will be updated every 5 seconds.

Findings



Playing around the temperature values, the fan speed is set according to temperature. In the screenshot above, I changed the FAN_ON_TEMP to 30 while setting FAN_OFF_TEMP to 28.5. I also lowered the fan max speed from 255 to 100 as an experiment.

By adding a Hysteresis value, I created a dead-zone around the actual turn on temperature to prevent constant switching on and off. If the hysteresis value was too large, it creates a dead zone that overlaps with the mapped fan speed range.

This resulted in the fan keeping its speed to the maximum and turning off when it reaches the turn off threshold. In this case, I kept the hysteresis value small at 0.5. This prevented unwanted rapid switching on/off of the fan.

```
const float HYS = 0.5; //Define hysteresis value
```

Further testing shows that if the mapping function was not properly defined, it could lead to unintended fan activation even when temperature is below the FAN_ON_TEMP_HYS.

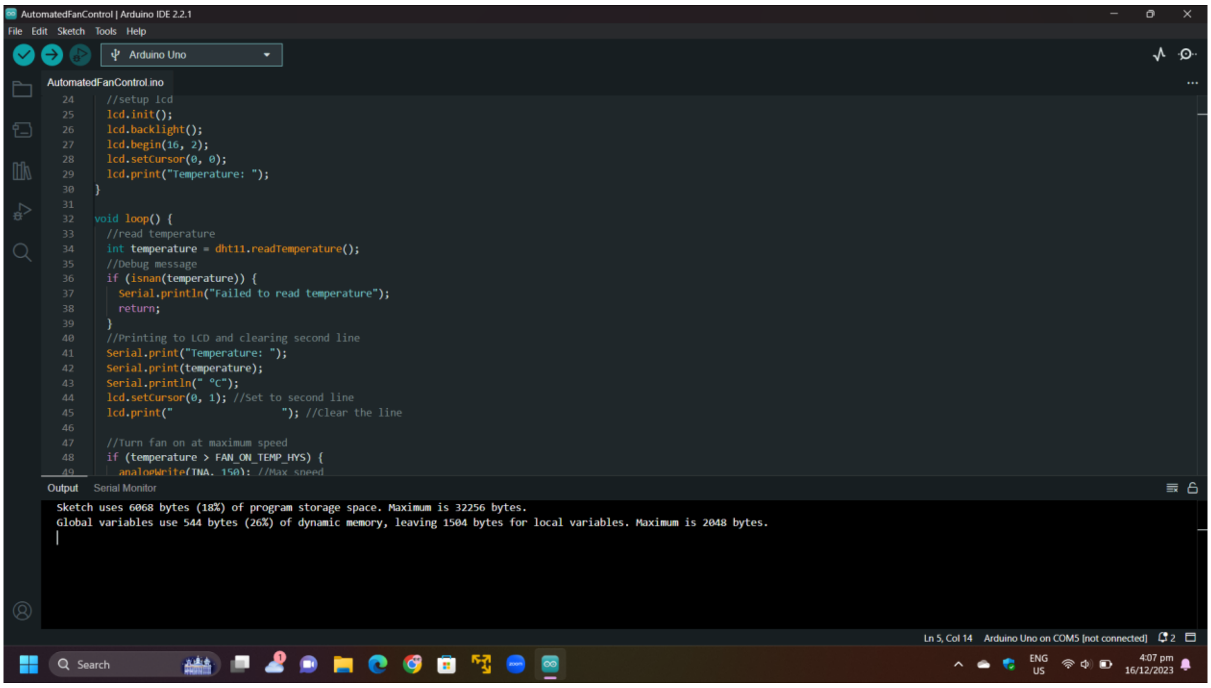
In this case, the map function is defined from 0 at FAN_OFF_TEMP and reaches 255 at FAN_ON_TEMP_HYS. This guarantees the fan remains off below the turn on threshold and reaches maximum speed at the adjusted threshold.

I was unable to implement the ESP8266 ESP-01 module into this project despite extensive searches and libraries. The current library used was incompatible with ESP8266 board management and I was unable to find a compatible alternative.

```
// Define Blynk virtual pin callbacks
  BLYNK_READ(vTemp) {
    int temperature = dht11.readTemperature();
    if (isnan(temperature)) {
      sensorError = true;
      Serial.println("Failed to read temperature");
    }
    else {
      Blynk.virtualWrite(vTemp, temperature); //Use temperature directly
         adjustFanSpeed(temperature, remoteFanSpeed); // Pass both temperature &
stored speed
     }
    }
 }
```

```
  BLYNK_READ(vPowerSwitch) {
    int state = param.asInt();
    if (state) {
      // Turn fan on remotely
      remoteFanSpeed = dht11.readTemperature(); //store current temperature
         analogWrite(INA, 100); // Max speed, adjustable max to 255
         digitalWrite(INB, LOW); // Fan direction
         Blynk.virtualWrite(vFanSpeed, 100);
         Serial.println("Fan turned on");
    }
       else {
      remoteFanSpeed = 0; //clear stored temperature
         adjustFanSpeed(dht11.readTemperature()); //Revert to dynamic fan control
    }
  }
}

void loop() {
  Blynk.run(); // Process messages and update virtual pins

  if(connectionError) {
         Serial.println("Connection error!");
  }
  if(sensorError) {
         Serial.println("DHT sensor error!");
  }
}

void dynamicfanspeed() {
       int temperature = dht11.readTemperature();
       if(isnan(temperature)) {
              adjustFanSpeed(temperature, remoteFanSpeed); //Use both temperature
and stored temperature
       }
}

void adjustFanSpeed(float temperature, float prevFanSpeed = 0) {
       const float onTemp = fanOnTemp + hysteresis;
       const float offTemp = fanOffTemp - hysteresis;

       if (temperature > onTemp) {
              analogWrite(INA, 100); //Max Speed
              digitalWrite(INB, LOW); //Fan direction
              Blynk.virtualWrite(vFanSpeed, 100);
       }
       else if (temperature < offTemp) {
              analogWrite(INA, 0); //Turn fan off
              digitalWrite(INB, LOW); //Fan direction
               Blynk.virtualWrite(vFanSpeed, 0);
       }
}
```

Attached a snippet of the codes unsuccessfully used. The codes conflicted with not only the i2c library but affected the Uno board as well. ESP8266 does not read UNO R3 and even after switching the board management, it will not run the codes.


AutomatedFanControl | Arduino IDE 2.2.1
File Edit Sketch Tools Help
Arduino Uno
AutomatedFanControl.ino
Output Serial Monitor
Ln 5, Col 14 Arduino Uno on COM5 [not connected]

```
  //setup lcd
  lcd.init();
  lcd.backlight();
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("Temperature: ");
}

void loop() {
  //read temperature
  int temperature = dht11.readTemperature();
  //Debug message
  if (isnan(temperature)) {
    Serial.println("Failed to read temperature");
    return;
  }
  //Printing to LCD and clearing second line
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");
  lcd.setCursor(0, 1); //Set to second line
  lcd.print("                "); //Clear the line

  //Turn fan on at maximum speed
  if (temperature > FAN_ON_TEMP_HYS) {
    analogWrite(INA, 150); //Max speed
```

```
Sketch uses 6068 bytes (18%) of program storage space. Maximum is 32256 bytes.
Global variables use 544 bytes (26%) of dynamic memory, leaving 1504 bytes for local variables. Maximum is 2048 bytes.
```

A screenshot of the IDE showing the resources used.

As seen above, the programme does not take much resources, keeping well below the maximum bytes for variables. This lightweight code can be used as a base to further improve/enhance its functionalities when required.

Pitfalls

The project is limited in scope.

It only consists of temperature monitoring and a basic fan control. It does not consider user's preferences and does not take other factors like time of day into account.

Potential overuse of the fan.

The dynamic fan speed might not be optimal in all situations, this may lead to unnecessary fan usage and unwanted noise pollution.

A simple solution includes time scheduling. With this time schedule set, the fan control system can be turned off after 6pm when users are leaving work. The fan system can then turn on at 6am the next day when users are arriving for the workday.

Limited Control

Users will require the use of Blynk to control the system. Currently, the system does not have additional input methods and require it to be connected to a network.

Temperature Sensor is not accurate.

DHT-11 sensor is not the most accurate or sensitive module. This might affect the constant reading during the operation runtime, it might not be updated correctly or read the temperature wrong. A simple solution is to upgrade to DHT22 or alternatives like LM35 or TMP36.

However, since this is a fan control system, the range of temperature does not need to be extreme and most commercial fans will not be able to change ambient temperatures much.

Advantages

Keeping the system Simple

This project allows users to add onto the codes to implement other functions like a motion sensor to automate turning on/off the fan. Another possible addition to this project is the inclusion of time-based monitoring; this can ensure the fan system is on and monitoring during certain periods of the day while turning off completely after working hours to conserve energy and preserve the parts conditions and longevity.

Minimal components

This project is affordable and allows other users to improve or adapt to their own conditions.

The core functions are standalone and requires no external services.

With a focus on keeping the codes simple and only on the core functions, users can add on to the codes instead of fiddling the basic functions.

Improvements

Implement temperature logging.
By introducing data logging, I can use it for analysis and optimizing the fan control system. With Blynk, I can easily store such data and by making use of Machine Learning; this can be further automated to improve the fan control system, reducing the energy usage, and improving the fan speed overtime.

Introduce time-based scheduling.

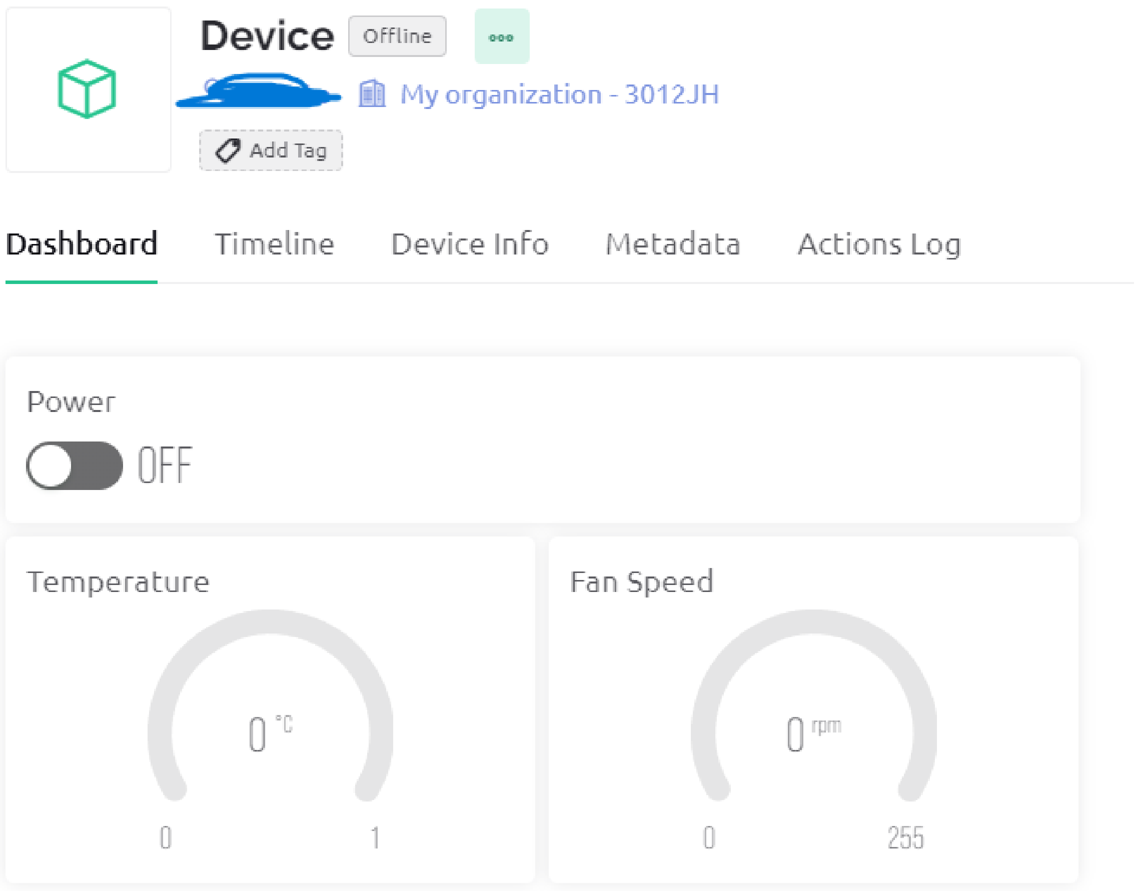
With a time-based schedule, I can adjust the fan behaviour based on specific times or events. Time could be set to after 6pm, when users are leaving the workplace; or during the afternoon, when most users are present.

Make functions modular.

By dividing the codes into separate functions or modules to handle specific tasks, it will make it easier to understand, maintain and modify in the future. It also allows me to easily add new features without modifying existing codes.

More user Inputs.

Currently, the user can only use Blynk application to monitor temperature and fan speed, in addition to turning the system on/off.



A definite solution by using Blynk cloud can improve this project massively. Blynk allows users to easily view and control the system remotely in an easy to configure user interface. Additional features include data logging, analysis in graphs and logs to view previous updates to the systems.

Conclusion

Although I was unable to integrate the wireless component into this project, I focused on the core functions of the automated fan control. The automated fan control can be used to enhance cooling solutions from smaller components like machine cooling solution to a smart home system where users can automate the ceiling fans.

This can help save users' energy consumption when the environment is cooler since the cooling system will take the temperature into account. It will improve the motor lifespan since it dynamically adjusts fan speeds instead of keeping the fan at a constant speed.

References

Keyestudio. (n.d.). Ks0168 Keyestudio L9110 fan control module. Retrieved from https://wiki.keyestudio.com/Ks0168_keyestudio_L9110_fan_control_module

Tinkercad. (n.d.). Tinkercad. Retrieved from

https://www.tinkercad.com

LiquidCrystal Library. (n.d.). Retrieved from https://reference.arduino.cc/reference/en/libraries/liquidcrystal-i2c/

DHT Sensor Library. (n.d.). Retrieved from

https://www.arduino.cc/reference/en/libraries/dht-sensor-library/

Arduino Forum. (2020, March 19). Hysteresis. [Forum post] Retrieved from https://forum.arduino.cc/t/hysteresis/506190

Blynk. (n.d.). Blynk dashboard. Retrieved from

https://blynk.cloud/dashboard/