International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 3, May-June 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Low-Cost Design & Development of Auto Cooling Electronic System

Jiya Upadhyay*, Hunny Panchal*, Jagdish M Rathod**
*Second Year Student, Electronics Engineering
** Faculty of Electronics Engineering Department
BVM Engineering College-V V Nagar
Email: jiyahu2611@gmail.com

----------**********----------

Abstract:

In this work, we describe how we built a fan system that adjusts its speed on its own based on the surrounding temperature, using an Arduino UNO board paired with a DHT11 sensor. Rather than running at a constant rate, the fan responds to real-time temperature readings and varies its speed through PWM signals. The microcontroller reads incoming sensor data and decides which speed level to apply, based on thresholds we set during the design phase.

Keywords — **Arduino UNO, DHT11 Sensor, PWM, DC Fan, Temperature Control, LCD Display.**

----------**********----------

I. INTRODUCTION

In recent years, there has been growing interest in building smarter, more responsive systems for homes and industries alike. One area where this matters a great deal is thermal management — keeping temperatures under control to avoid damage to machines, electronics, and even the surrounding environment. Conventional fan setups run at a single fixed speed no matter what the temperature is, which wastes energy and often fails to provide adequate cooling when it's actually needed.

In this project, we put together a temperature-sensing fan controller using an Arduino UNO and a DHT11 sensor, which picks up both temperature and humidity from the environment. The fan speed changes dynamically as the temperature shifts, and this control is achieved through PWM — a technique that adjusts how much power the motor receives at any given moment. All readings and the current fan status are shown on a 16x2 LCD screen, so the user always knows what's happening without needing to guess.

What we ended up with is a low-cost yet practical setup that handles cooling intelligently, without any need for manual input. Since the fan only runs as fast as the situation demands, power consumption drops noticeably compared to fixed-speed alternatives. We found this approach well-suited for environments like server rooms, plant nurseries, enclosed electronics, and general household use where temperature management is important but constant human oversight isn't practical.

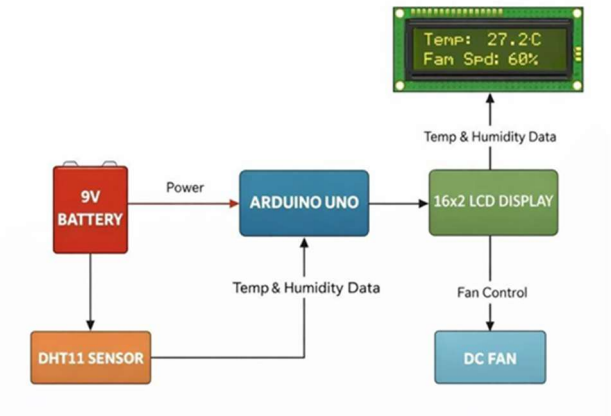

Fig. 1. Block Diagram of Auto Cooling System

II. OBJECTIVES

We set out with a few clear goals in mind when designing this system: (1) Reliably capture real-

time temperature data from the DHT11 sensor through the Arduino UNO. (2) Make the DC fan respond automatically to shifting temperatures by applying the right PWM duty cycle at each stage. (3) Keep the user informed at all times by showing current temperature and fan speed on the 16x2 LCD. (4) Cut down on wasted electricity by running the fan only as fast as cooling actually requires. (5) Build something that is affordable, easy to replicate, and genuinely useful across different real-world scenarios.

III. COMPONENTS USED

For building this system, we gathered the following components: an Arduino UNO R3 board (built around the ATmega328P chip running at 16 MHz, with 14 digital I/O pins and 6 PWM-capable outputs), a DHT11 sensor for measuring temperature (range: 0–50°C, accuracy: ±2°C), a 5V brushless DC fan with PWM compatibility drawing around 0.2A, a 2N2222 NPN transistor to act as a switch for driving the fan, a standard 16x2 LCD module (HD44780 compatible), a 1KΩ resistor for the transistor base, a 10KΩ potentiometer to set LCD contrast, a 1N4007 diode to handle back-EMF from the fan motor, a 9V battery with its connector, and the usual jumper wires and breadboard.

On the software side, we used Arduino IDE version 1.8.19 to write and upload the code. Sensor communication was handled through Adafruit's DHT library, and the display was driven using the standard Liquid Crystal library.

IV. SYSTEM DESCRIPTION

At the heart of this system sits the Arduino UNO, which takes care of all the processing and decision-making. The DHT11 is wired to one of the Arduino's digital pins and uses a single-wire interface to send data. Every second or so, the sensor takes a fresh temperature measurement and passes it along to the microcontroller for evaluation.

Once the Arduino has the temperature value, it checks it against our defined thresholds and produces a PWM signal accordingly. This signal goes to the base of the 2N2222 transistor, which we're using here as a variable current switch rather than a simple on/off device. The fan motor connects to the transistor's collector side, and since PWM effectively varies the average voltage, the motor speed tracks the duty cycle quite closely.

We placed a 1N4007 diode across the fan terminals to absorb any voltage spikes the motor might generate when switching — this is critical to prevent transistor damage over time. The LCD connects to the Arduino via a 4-bit parallel configuration, and it refreshes continuously to show the latest temperature reading alongside a plain-text label for the fan's current operating state: OFF, LOW, MEDIUM, or HIGH.

V. WORKING PRINCIPLE

The whole thing works as a closed-loop feedback system: the sensor feeds data in, and the fan adjusts its output based on what it reads. When you first power on the device, the Arduino sets up both the DHT11 and the LCD before entering its main loop. From that point, it reads the temperature once per second and runs it through our threshold checks to decide what to do next.

Below 25°C, the fan stays off completely — there's no point running it when it isn't needed. Between 25°C and 30°C, the system kicks in at a low setting, applying roughly 40% duty cycle to keep things cool without overdoing it. From 30°C up to 35°C, the duty cycle rises to about 70%, offering more airflow for moderate heat conditions. Once the temperature climbs past 35°C, the fan goes to full power with a 100% duty cycle to tackle the heat head-on.

Changing the PWM duty cycle directly influences how often the transistor switches, which in turn changes the average voltage the fan motor sees — and so the speed changes proportionally. Meanwhile, the LCD refreshes to show the most current temperature value and a status label, giving whoever is watching a clear, immediate sense of where the system stands.

VI. CIRCUIT DIAGRAM AND IMPLEMENTATION

We built the prototype on a standard breadboard to keep things accessible and easy to modify. The

DHT11's data line went to pin 2 on the Arduino UNO, with a 10K pull-up resistor between that line and the supply voltage as the sensor datasheet recommends. The PWM output came from pin 9 — one of the Arduino's dedicated hardware PWM pins — and we ran it through a 1K resistor before connecting it to the transistor base.

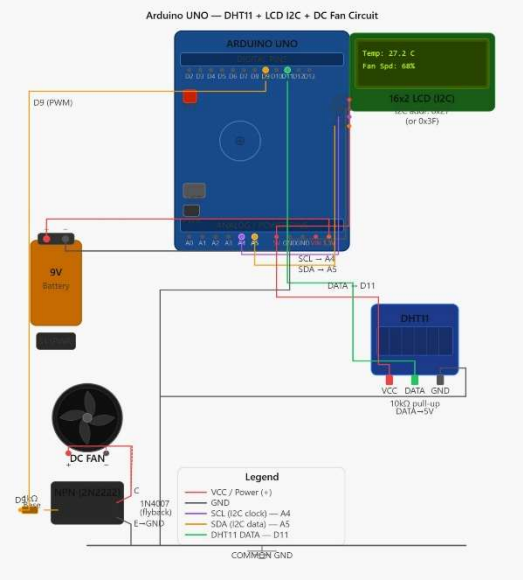

Fig. 2. Circuit Diagram of Auto Cooling System

The transistor's emitter was tied to ground, while its collector fed into the negative terminal of the DC fan. The fan's positive side connected to the 9V supply rail. We fitted the 1N4007 diode across the fan — cathode toward the positive terminal — to clamp any inductive spikes. For the LCD, we used pins 7, 8, 9, 10, 11, and 12 for the data and control lines, with the potentiometer wired to the V0 pin to let us dial in the right contrast.

VII. RESULTS AND DISCUSSION

We ran the system through a range of temperature conditions to see how it held up in practice. The DHT11 gave us consistent readings that matched what we expected from its specifications, with no noticeable drift during the tests. Whenever the temperature crossed a threshold, the fan responded quickly — usually within a second or two — and the speed transition felt smooth rather than abrupt.

When the room stayed below 25°C, the fan sat idle and drew nothing from the supply. In the 25–30°C band, it ran at the 40% duty cycle setting, which noticeably cut down power use compared to running the motor flat out. Above 35°C, the fan ramped up to full speed as designed, and the LCD stayed accurate and in sync with what the sensor was reporting throughout every test we ran.

Over longer test runs, the system held up without any false triggering or sensor glitches. Putting together the full bill of materials came to under INR 500, which makes this a very accessible build for students picking up embedded systems, hobbyists experimenting at home, or anyone needing a simple thermal control solution on a tight budget.

VIII. APPLICATIONS

This kind of system turns out to be useful in more places than you might expect. In a typical home, it can keep a room comfortable without anyone needing to manually adjust a fan. For computers, servers, or any electronics that generate heat during operation, tying cooling directly to actual thermal conditions — rather than just running fans constantly — can meaningfully extend component life.

Growers working in greenhouses benefit from stable, automatically maintained temperatures that support healthy plant development. In factory settings, the same approach protects motors and control panels from heat damage by activating airflow only when the temperature actually demands it. Beyond these, the design could work well for battery pack cooling, early-stage automotive climate experiments, and as a practical teaching tool for students picking up embedded systems or IoT development.

IX. CONCLUSION

Through this project, we built a working fan controller that thinks for itself — sensing the room temperature and deciding how hard to run the fan without any manual input. The PWM approach proved to be a practical and efficient way to achieve graduated speed control, meaning the fan consumes only the power actually needed at any given temperature. Having the LCD show live readings made the whole system feel transparent

and trustworthy to anyone watching it operate. We tested it across different temperature ranges and found that the sensor stayed consistent, the speed changes were smooth, and the overall behavior was stable throughout. and we didn't encounter any unexpected failures or erratic readings during those tests. Looking ahead, there's room to take this further — adding Wi-Fi or Bluetooth connectivity for remote monitoring, incorporating humidity into the control logic, or switching to a PID-based controller for finer thermal regulation would all be worthwhile directions. As a starting point for temperature-aware embedded systems, this project holds up well and could scale to fit a variety of home, industrial, or agricultural needs.

X. ACKNOWLEDGMENT

We would like to thank Dr. J. M. Rathod and the entire Electronics Engineering faculty at Birla Vishvakarma Mahavidyalaya for their continued guidance and the support they offered throughout this work.

XI. REFERENCES

[1] Arduino Official Website. [Online]. Available: https://www.arduino.cc/
[2] DHT11 Sensor Datasheet. [Online]. Available: https://components101.com/sensors/dht11-temperature-sensor
[3] Adafruit DHT Sensor Library. [Online]. Available: https://github.com/adafruit/DHT-sensor-library
[4] Arduino Liquid Crystal Library. [Online]. Available: https://www.arduino.cc/en/Reference/LiquidCrystal

