IoT Smart Fan for Personal Comfort Hub

Andrew Politz, Eric Du, Miles Nash
andrewpolitz@berkeley.edu , ericdu@berkeley.edu, miles.r.nash@berkeley.edu

1 Motivation

This project aims to create a "smart fan" that adjusts its cooling based on room occupancy and environmental conditions, allowing for more efficient climate control. HVAC systems are notably inefficient with the end result being only a slight majority of workers in an office space being comfortable which is not only energy inefficient but also harmful to worker productivity [1]. The HVAC system of a commercial office building works on the kiloWatt scale and constant dynamic adjustments to temperature are not as efficient as single settings. By utilizing smaller module climate comfort devices like our fan, the office space can make climate adjustments on an individual scale with energy consumption on the Watt scale. Additionally by including wifi and network connection capabilities with a microprocessor such as the ESP32, these modules can communicate with the larger network and inform the building wide control of individual preferences. An example of this aggregation would be through a thermostat system such as HomeAssistant [2] or NEST.

This project was conducted in partnership with the Center for the Built Environment (CBE) and Electrical Engineering and Computer Science department at the University of California, Berkeley as part of their ongoing research into smarter HVAC control systems and personal comfort devices. Previous research has developed a suite of personal comfort devices (PCDs) which interface with Home Assistant, an open source home automation platform, to remotely monitor workstation temperature conditions and enable building scale automations.

This next-generation HVAC control system aims to reduce energy use and improve overall comfort by aggregating multiple data streams from individual workstations to make more informed decisions on the HVAC temperature. The system helps to modernize existing HVAC systems, as the aggregator (home assistant in this case) can integrate wirelessly with PCDs and other modern inputs, analyze them, and present a single control temperature to legacy HVAC systems without the need to upgrade the entire system. In the case of HomeAssistant, a single Raspberry Pi can perform the needed computations and can be scaled up for additional computation needs. The Smart Fan PCD system aims to empower users to improve their personal comfort through either manual or automatic control of a 5V desk fan to achieve their desired temperature, while capturing and transmitting ambient temperature, body temperature (through thermal imaging), and occupancy data, over MQTT for integration into a building automation system such as Home Assistant.

2 Design

The design of this project can be divided into essentially two categories, the circuit components and the state machine software logic. The circuit design began with the ideal goal to be able to control any USB fan with a PWM signal from our microcontroller. However, the design space for this task is large as fans on the market run at many varying voltage levels 5V, 9V, 12V, etc. and incorporating these varying voltages into our design would be a difficult task. The decision was made to focus on a 5V USB fan. This implementation includes an Adafruit HUZZAH32 ESP32 Feather, a NPN transistor, MPC9808 temperature sensor, AMG8833 8x8 thermal camera, rotary encoder, 12 LED NeoPixel ring, Passive Infrared (PIR) sensor, a 12V to 5V converter, a 5V to 3.3V converter, and a 12V power supply with a barrel jack.

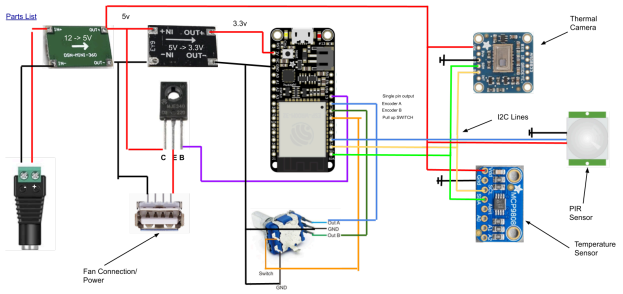

Figure 1. Circuit schematic for ESP32 and peripheral connections for fan control.

The ESP32 requires a 3.3V input which is handled by the 5V to 3.3V converter, while the fan input is connected to the 5V line via the NPN transistor which is modulated by a ESP32 signal via the base of the

transistor and outputs an effective voltage based on the PWM duty cycle. The temperature sensor and thermal camera are both read by the I²C lines of the ESP32 with the thermal sensor at address 0x18 and the thermal camera at 0x69. The PIR sensor is declared as an input pin and placed in trigger mode, which then reads high whenever a presence is detected.

The logic portion of the ESP32 and the automation of the fan is controlled by a program with state machine logic in the Arduino IDE with libraries for both the AMG8833 and the MPC9808 imported to help read values from the sensors.

The program logic is organized into three states, off, manual, and automatic with the default state being off. In off mode, the fan remains inactive, awaiting user input to switch to a different state. A press of the rotary encoder prompts the finite state machine (FSM) to switch states.

When in manual mode, turning the rotary encoder in manual mode adjusts the speed of the fan on a scale from 0% to 100%, with each click yielding 10% change in speed. The speed is then translated to the PWM duty cycle of the fan. The state machine begins polling the sensors to check the desired power input and whether the person is still present. The present signal is composed with a combination of the PIR sensor and the thermal camera. The PIR camera acts as an initial present signal which then flags the thermal camera to start capturing images and check the max temperature. If the max temperature is greater than the background temperature, the user is considered still present. Otherwise, the camera flag turns off and the PIR sensor is once again the main determination of presence. An edge case arises when the user's temperature is cooler than the background, potentially causing the thermal camera to fail in detecting presence. In this scenario, the system defaults to relying on the PIR sensor, which provides robust detection of motion. For enhanced reliability, a future implementation could integrate an edge detection algorithm for the 8x8 thermal pixel grid to improve presence verification.

When in the automatic state, the state machine continues to poll for presence with the same system logic, and then the fan's power is determined by the distance from the user's preferred temperature for proportionality control, stepping up or down as seen in Fig. 2.

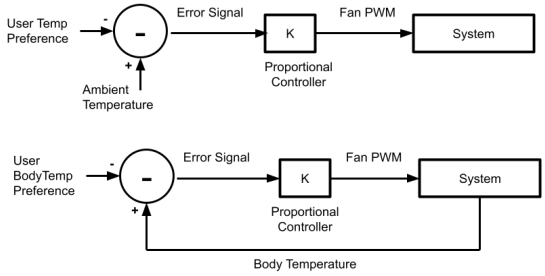

Figure 2. Demonstration of open and closed loop control logic for the fan with the PWM signal.

In both the manual and automatic states, the system actively gathers and publishes critical environmental and status data via the MQTT protocol. This includes ambient room temperature, the maximum temperature detected by the IR camera, and user's presence status. These data points are transmitted wirelessly to a designated MQTT broker, enabling seamless integration with the advanced home automation systems like HomeAssistant or Google NEST. By subscribing to the ESP32's MQTT feed, these systems continuously monitor updates in real time. This ensures they can make informed decisions or trigger automated responses, such as adjusting room-wide HVCA settings or sending notifications to the user. The use of MQTT as the communication backbone provides a robust, efficient, and scalable mechanism for sharing key data across the smart home ecosystem, enhancing functionality and user experience.

Finally when re-entering the off state, all of the sensor polling timers are turned off and only the button is checked for additional inputs.

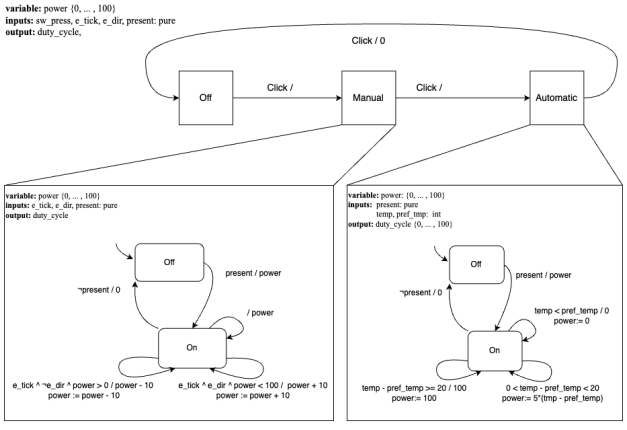

Figure 3: Hierarchical State Machine design for the ESP32 logic execution.

3 Implementation

This project initially aimed to create automations strictly in the ESPHome framework to control a fan with a temperature sensor, thermal camera, and rotary encoder. However, defining state and adding more specific automations for unsupported sensors in the .yaml file that ESPHome uses became difficult and as a result the project was transferred into the Arduino IDE and largely written in C++. The sensors in the system are periodically polled using timers, while the fan's operation can be controlled wirelessly through MQTT-triggered interrupts. This design allows for efficient sensor data collection and real-time remote control of the fan, ensuring seamless integration with the smart home ecosystem and responsive adjustments based on the user commands or environmental changes. The main framework revolves around a setup() and loop() function with initializations of the libraries used and variables handled in setup() such as ensuring the I^2C connections for the sensors are found. The main structure of the program focuses on void functions which alter the main global variables of the program which are power, temperature, presence, and the camera max temperature.

The circuit was first developed on a breadboard and smaller programs written for each component to test the functionality. Notably the rotary encoder and NeoPixel light ring posed problems, even with demonstration programs provided by the Arduino libraries. Once each component demonstrated proper functionality, each component was added to the larger state machine logic.

First the encoder logic was built to control the PWM with the input from the user which mapped an encoder increase or decrease to an increase or decrease in the duty cycle of the PWM. Next the PIR sensor was placed in trigger mode with the PIR reading in a high signal whenever motion was detected. The MCP9808 temperature sensor uses a library to read in the temperature data as well as the AMG8833 thermal camera which were both implemented with the help of these libraries. The PIR sensor was calibrated manually by physically turning the potentiometers to adjust the trigger timer duration and sensitivity. The encoder was digitally debounced by setting a minimum interval between button presses and additionally the range was set to {0,100} and step size was set to 10. The temperature sensor calibration and logic was handled by its own on board controller and was read in by the MCP9808.

The thermal camera calibration was the most involved. The camera was calibrated by first reading in the 8x8 pixel array and capturing pictures in bursts of three. From there, we checked each picture by plotting it as an 8x8 temperature gradient and then observing if the fan had any noticeable effect on the user and the environment when turned on, otherwise a closed loop feedback design would not be possible or effective. To test the theory of observing a person with the thermal camera, we created a small program that took three sets of three pictures and modulated the fan on and off which resulted in the images seen in Fig. 4.

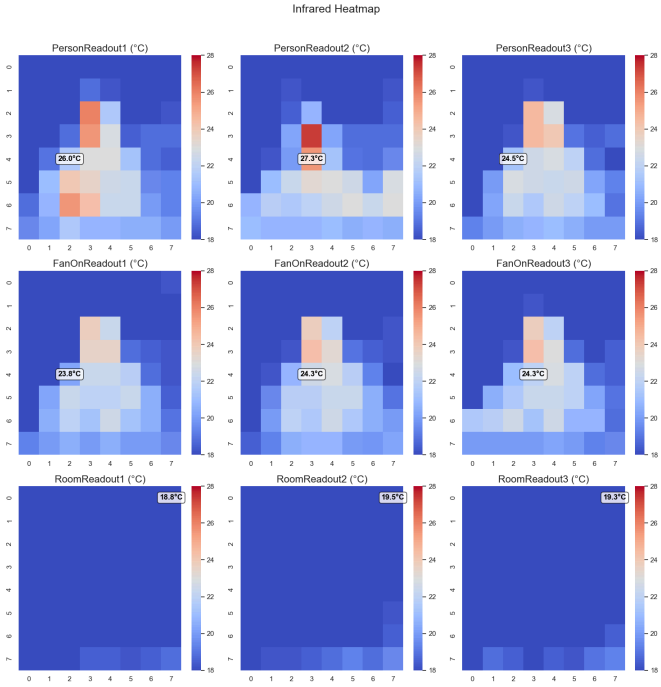

Figure 4. Images taken with the thermal camera with a person present and no fan (top), a person present and the fan on (middle), and no person present (bottom).

The difference between background and a person present is noticeable with a clear person shaped blob showing up in the image. Additionally, the fan appears to make a noticeable change in the user's temperature with the few degree difference in the hottest point of the camera. A more robust method for object detection would need to be set up to get a formally significant result, likely taking a distribution of images over different temperature profiles or a static test object to calibrate the temperature. However for this project, we simplified the control to take the max temperature pixel on the screen for basic controls as this indicates the presence of a hot object and likely person, prompting the fan to turn

on if motion is jointly detected. More involved methods for calculation were considered such as upsampling the image and then downsampling to an odd index matrix to be able to take the center point an better averaged value, or using a linear interpolation or nearest neighbor approach to attempt to increase camera resolution. However, this ultimately was ruled out due to the limiting storage on the ESP32 chip thus a simpler method was favored.

All the components were then placed in a 3D printed PLA housing, including the temperature sensor and thermal camera. While the temperature sensor is not largely affected by this change, the thermal camera encounters challenges. The enclosure, which featured a small slit for the camera's field of view but limited airflow, caused the PLA housing to heat up during operation. This heating altered the camera's readings, resulting in skewed data within the frame of interest. To address this issue, the camera was then calibrated to understand the effects of the PLA housing seen in Fig. 5. The same calibration method mentioned above was used.

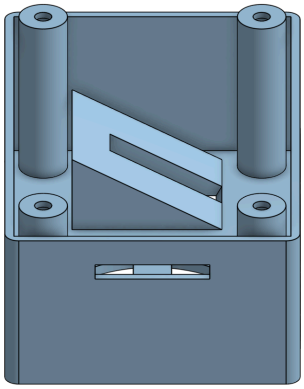

Figure 5: side view of smartfan enclosure with 30° wedge for AMG8833 IR camera mounting.

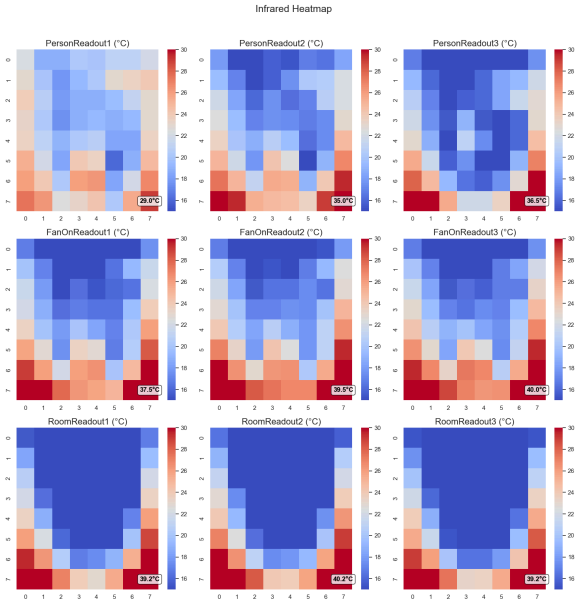

Figure 6. Images taken with the effects of thermal housing in the same calibration scheme as Fig. 4.

Following this calibration with temperature effects from the thermal housing, the max temperature reading was reduced to a 4 pixel array in rows 5 and 6 and columns 3 and 4. The effect is still similar and there is a noticeable change in temperature in the 4 pixels identified.

Following the implementation of all of the sensors to be properly read into the ESP32, we then implemented an MQTT protocol to publish the state of our device to an identified topic in this case IoTFan/esp32 and published to the MQTT broker. The MQTT broker for HomeAssistant is Mosquitto, but for the purpose of readability and demonstration, we decided to use MQTTX to visualize the MQTT publications made by our device. To fully integrate our design back into HomeAssistant, the HomeAssistant library can be imported onto our device and the main host node can subscribe to the desired topics such as temperature and presence.

Finally, all of the sensor polling was then set at desired intervals through digital timers from the Ticker library. The system state and temperature was published by MQTT and read in every 10 seconds with the encoder turn input polled every 100 microseconds and presence checked every 5 seconds. For a production level program, these values could likely be made significantly higher as the state of the system is likely not as dynamic and an interrupt set up to handle reading in the PIR sensor

could minimize operations on the ESP32. However, for demonstration purposes the polling provides a more reliable interval.

4 Evaluation

Over the course of this multi-week project, our team encountered a number of challenges, some of which may need to be further addressed in a future iteration of this system.

Our initial software development took place using ESPHome, a tool for integrating ESP32 devices with Home Assistant. The tool automatically generates firmware from YAML text based instructions to allow those unfamiliar with programming to add smart home devices to the system. Unfortunately we found working with this system to be limiting when attempting to implement our state machine, integrate with more complex sensors (AMG8833 IR Camera), and build more complex automations such as our open loop fan control. For this reason, we switched over to Arduino based C++ halfway through the development process.

As this was the first time building embedded systems projects for a few of us, we ran into some development process challenges which will act as lessons learned for future projects. Unlike software development, embedded systems projects can have long iteration lead times. When faced with burnt-out LEDs, ill-fitting enclosures, and a BJT rated for the wrong operating voltage, replacement parts typically took at least a day to arrive. This bottlenecked software bring-up and added cascading schedule pressure to complete this project by its intended deadline.

Our team also faced some difficulties with prioritization of features. Due to scheduling pressure and conflicts with other obligations, we were forced to do a lot of work in parallel. Most of us were very eager to get working on the more complicated parts of the project: implementing the state machine, sending MQTT signals, and presence detection logic, which unfortunately left some of the core functionalities (PWM control of our fan using encoder input) non-functional on demo day. We plan to take a more strict bottom up approach in future projects, where priority doesn't shift away from core features until they are fully implemented and thoroughly tested.

A final general lesson would be to allocate as much time for iteration as possible. Some specific unexpected roadblocks we came up against included two days worth of encoder debugging, an enclosure which interfered with our IR camera output, hotspot issues on demo day, and unanticipated cold weather which interfered with our control algorithm. More time would've allowed us ample time to respond to these roadblocks and do some more testing in our target setting (Cory courtyard) to find issues earlier.

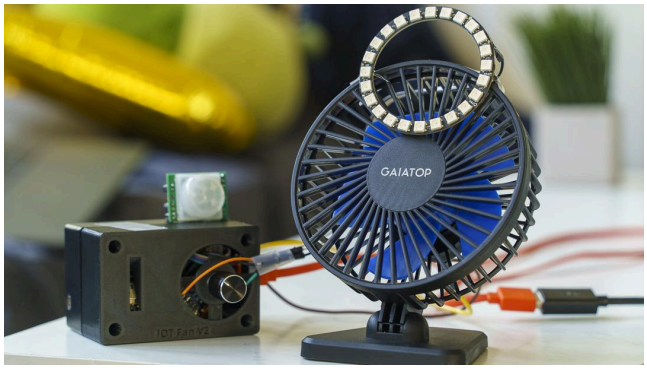

Figure 7. Image of the finished IoT fan with a protoboard placed in its enclosure.

ACKNOWLEDGMENTS

Sponsored by the U.C. Berkeley Center for the Built Environment, California Institute for Energy and Environment, California Energy Commision, and the Citrus and the Banatao Institute. Special thanks to Dr. Tobias Kramer and Professor Prabal Dutta for their help with this project.

https://uc-ciee.org/projects/mainstreaming-personal-comfort-devices-with-modular-controls/

REFERENCES

[1] "The never-ending battle over the best office temperature," BBC, 2016. https://www.bbc.com/worklife/article/20160617-the-never-ending-battle-over-the-best-office-temperature.
[2] Home Assistant. (2024) https://www.home-assistant.io
[3] Christensen, J. (2021) MPC9808 (V1.2.0) [Library]. https://docs.arduino.cc/libraries/mcp9808
[4] Adafruit. (2022) Adafruit AMG88xx Library (V1.3.2) [Library]. https://docs.arduino.cc/libraries/adafruit-amg88xx-library
[5] O'Leary, N. (2020) PubSubClient (V2.8.0) [Library]. https://docs.arduino.cc/libraries/pubsubclient
[6] Adafruit. (2024) NeoPixel (V1.12.3) [Library]. https://docs.arduino.cc/libraries/adafruit-neopixel
[7] Staub, S. (2021) Ticker (V4.4.0) [Library]. https://docs.arduino.cc/libraries/ticker