AI-Informed Underfloor Heating Control with ESP32

Project Overview

This repository documents the development of a predictive AI model for a custom-built, underfloor heating system. The core innovation lies in moving away from reactive control loops to a proactive, data-driven model that can anticipate heating needs and optimise energy usage on a low-cost ESP32 microcontroller.

The project is built around a small-scale, benchtop model of a room, which allows for rapid and cost-effective experimentation. The model includes a printed Kapton heater mounted on a 6 mm aluminium block to provide realistic thermal inertia. All tests were conducted in a real-world environment (a conservatory) to capture diverse, fluctuating conditions.

The Problem: Why not Conventional PID?

Conventional heating systems often rely on PID (Proportional-Integral-Derivative) controllers. While effective, a PID controller is inherently **reactive**; it only adjusts the heater output *after* a temperature deviation has already occurred. This can lead to inefficient overshoot and oscillation, resulting in wasted energy and an unstable environment for the user.

The Solution: A Proactive, Data-Driven Model

Our approach leverages machine learning to build a **proactive** control model. By training on a diverse dataset, the model learns to predict the optimal heater output (PWM) based on a wide array of variables.

1. Data Collection & Augmentation

We collected two primary types of data to comprehensively model the system's behaviour:

- **Stabilised Sweeps:** Data collected after the system reached a steady state, used to model the long-term relationship between power and temperature.
- **Dynamic Sweeps:** Data collected during rapid, randomised power fluctuations, capturing the system's transient response and thermal inertia.

The raw sensor data (temperature, humidity, pressure) was then augmented with synthetic data points for occupied state and activity (Relaxing, Sleeping, etc.), enabling the model to learn human context and intent.

2. Model Benchmarking and Selection

We trained and benchmarked several models to find the most suitable one for the resource-constrained ESP32. The results were clear:

- **Neural Network (Validation MAE):** 5.91
- **Decision Tree Regressor (Test MAE):** 1.13

The **Decision Tree Regressor** was chosen as the final model not just for its superior accuracy, but also for its exceptional efficiency. It is lightweight, requires minimal RAM, and its logic can be converted directly into fast, native C code, making it perfect for embedded deployment.

Repository Structure

- data/: Raw CSV data files from the ESP32 sweeps.
- data_augmentation.py: Python script to merge raw data, add synthetic features, and preprocess for model training.
- model_training.ipynb: Jupyter Notebook documenting the full process from data loading and EDA to model benchmarking and C code generation.
- decision_tree_model.h: The C header file containing the lightweight Decision Tree model, ready for inclusion in an ESP32 sketch.
- esp32_firmware/: Directory for the final ESP32 sketch.

Getting Started

1. **Clone the repository:**
   git clone https://github.com/your-username/your-repo.git

2. Run the Python scripts:
   Use data_augmentation.py and the Jupyter Notebook to preprocess the data and train the model. The notebook will automatically generate decision_tree_model.h.
3. Compile and upload to ESP32:
   Include decision_tree_model.h in your ESP32 project. The sketch will read sensor values, format them into the required feature array, call the predict function from the header file, and set the heater PWM.

Author

K Seunarine