Smart Heating System

by Shaun Dyson (220563357)

Overview

Background

I've always had trouble maintaining a comfortable temperature in my bedroom. While the heater has a dial and switches to control how hot it gets, the actual temperature it will reach is ambiguous and it these controls only affect how hot the heater gets, not how hot the room gets. This is especially bad at night, as I often turn the heater up before going to bed on a cold night but find it's too hot when I wake up the next morning. I can, in anticipation of this problem, keep the heater on a lower temperature overnight, but I often find it's too cold when I'm trying to sleep. An additional problem with my current heater is that I can forget to turn it off before leaving the house, wasting energy.

Problem Statement

There are two problems with conventional heaters. The first problem is that conventional heaters are inadequate in their ability to allow the user to control the temperature of the room. The second problem is that they can only be turned on or off manually and cannot automatically turn on or off based on whether someone is in the room.

A current solution is to use a smart heater, which allow users to set a desired temperature and to schedule times for when the heater should maintain that temperature or stay off. However, there are problems with this solution. First, smart heaters have the temperature sensor located on the heater itself, which means the sensor gives a temperature reading that is representative of the temperature of the air around the heater, not the overall temperature of the room. Second, this solution requires you to buy another heater, which may be costly and may result in your old heater going to waste. Thirdly, smart heaters only attempt to solve the first problem, controlling the temperature, and make no attempt to automatically turn the heater on or off based on activity in the room. Finally, it should be mentioned that, since smart heaters have a scheduling feature, any solution seeking to compete with smart heaters should also implement a similar feature.

Requirements

The goal of this project was to design and implement a heating system that fulfills the following requirements.

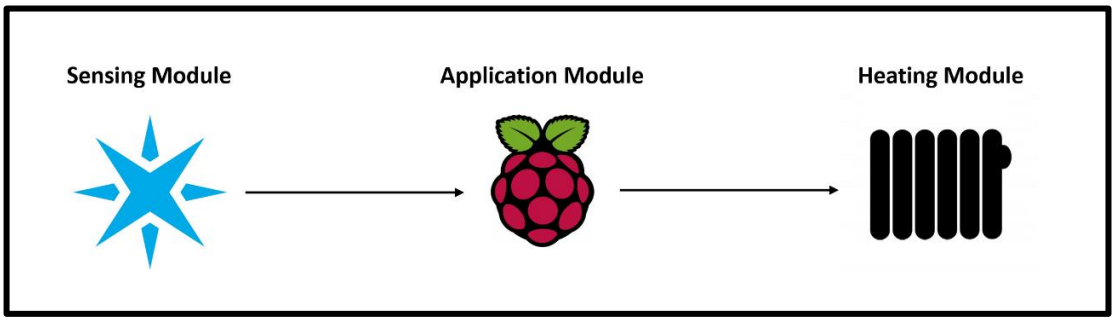
- The user should be able to set a target temperature for the heater to maintain in the room.
- The system should have the option to keep the heater turned off after a configurable period of room inactivity.
- The user should be able to set a schedule for when the heater should maintain the target temperature, regardless of activity in the room.
- The system should be able to incorporate a conventional heater as the heating element.

Design Principles

Overall Design

The system includes three modules: The Sensing Module, the Application Module, and the Heating Module. These modules are physically disconnected from each other and can be moved independently, provided they remain connected to the same computer network. The Sensing Module collects temperature and motion data from the environment and sends it to the Application Module via MQTT. The Application module runs a GUI application to allow the user to change settings, such as the target temperature, and uses these settings, plus the information received from the Sensing Module, to tell the Heating Module what to do through a IFTTT triggers. The Heating Module contains the heater itself and turns the heater on and off when instructed by the Application Module. Overall, this system follows the sense-think-act model, with the Sensing Module doing the sensing, the Application Module doing the thinking, and the Heating Module doing the acting.

Note that in this system, information flows in one direction; the Sensing Module sends information to the Application Module only, the Application Module sends information to the Heating Module only, and the Heating Module does not send information to any other module. This one-way flow of information keeps things simple and minimises dependencies between modules.



Fault Tolerance

Because the operation of the heater is dependent on data collected by the Sensing Module, this system cannot necessarily continue operating normally when a fault occurs. For example, if the connection between the Sensing Module and Application Module is lost, the Application Module no longer has the information it needs to operate the Heating Module correctly. If the connection between the Application Module and the Heating Module is lost, the Application Module can no longer tell the Heating Module what to do.

However, the system is still designed to be fault tolerant in the following ways: when a fault occurs, it will perform any necessary actions in response, will notify the user of the fault, and when the problem is resolved it will automatically resume normal operation. The system is tolerant of the following faults:

The Sensing Module loses the connection to the Application Module.
The Sensing Module will continuously check the connection to the Application Module and if this connection is lost, it will regularly attempt to reconnect. Once it has reconnected, it will resume normal operation and send sensing data to the Application Module. If the Application Module has not received any sensing information for a certain period of time, it will instruct the Heating Module to turn

off the heater (off is considered its 'default' state), pause further operation of the Heating Module, and will notify the user through an email (sent by triggering an IFTTT event) and an error on the GUI. When sensing information is received later, the Application Module will automatically clear the error and resume normal operation of the Heating Module.

A temperature of 0 is read.
The Sensing Module will occaisonally read the temperature as an erroneous 0 instead of the real temperature. It ignores temperatures lower than 1˚C and does not send these to the Application Module.

The temperature sensor malfunctions.
If the temperature is read as lower than 1˚C many consecutive times, the Sensing Module will no longer ignore the value and instead assumes the temperature sensor is malfunctioning (it has most likely become disconnected). It informs the Application Module which then instructs the Heating Module to turn off the heater, pauses further operation of the Heating Module, and notifies the user through an email (sent by triggering an IFTTT event) and an error on the GUI. When the temperature sensor stops malfunctioning, the Sensing Module will inform the Application Module, which will clear the error and resume normal operation.

The Application Module loses internet connectivity
If the Application Module loses internet connectivity, it can no longer control the Heating Module or send the user an email. However, it will still inform the user by showing an error on the GUI. When internet connectivity is restored, it will clear the error and continue normal operation.

Testability

The system is designed to be very testable and, in many cases, a module can be tested independently of the others. There are a variety of programs and scripts for testing different functions of the Sensing and Application Modules. These include:

- A Python script containing functions to trigger the different IFTTT events and print the responses. This can be used in a Python REPL.
- A C++ program for the Sensing Module to send simple data to the Application Module via MQTT, and a Python script for the Application Module to print that data.
- A C++ program for the Sensing Module to output its sensor data through a serial connection.
- A C++ program for the Sensing Module to send sensor data to the Application Module via MQTT, and a Python script for the Application Module to print that data.
- A Python script that contains different functions to connect to the Application Module via MQTT and send different events. This script can be used in a Python REPL to test how the Application Module responds to different events.
- The GUI application is divided into three modules of its own: core logic, MQTT, and GUI. The application simply combines these modules. Any of the Python scripts for these modules can be tested indepentently in a Python REPL.
- The Heating Module uses a smart plug to control the heater, which can easily be tested via the Kasa phone app.

Usability

The system is designed to be very easy to use once it's set up. There is a GUI application to display the current temperature, when motion was last detected, allow the user to change various settings, and to notify the user when a fault occurs. The settings the user has control over through the GUI application are:

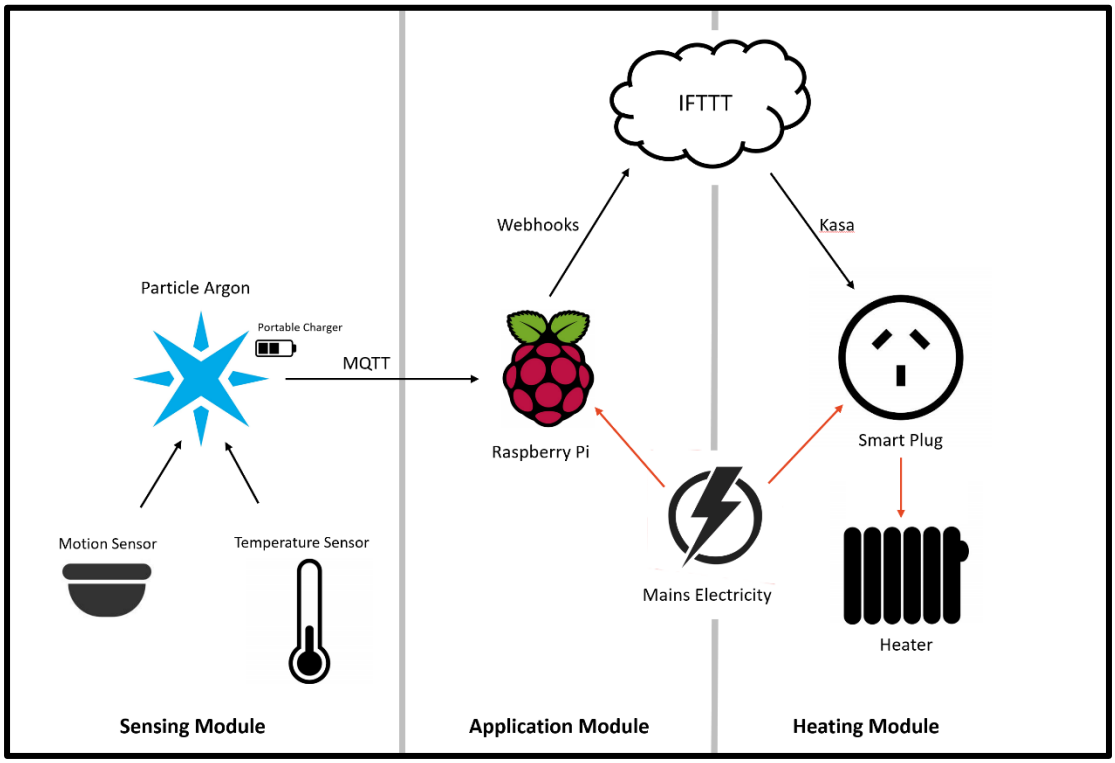
- Enable/disable the application's control of the heater.
- Set the target temperature.
- Enable/disable whether the heater should be turned off after a period of inactivity, and how long that period should be.
- Enable/disable whether the target temperature should be maintained during a specific time period, regardless of motion, and set what that time period is.

Weight

All components of the system, except the heater, are very light, and because the system is divided into three physically separate modules, each module can be moved indepently if needed. The Sensing Module must be light enough to mount on a wall, in the most covenient place for detecting motion.

Prototype Architecture

As mentioned before, the system is divided into the Sensing Module, the Application Module and the Heating Module. Below is a more detailed block diagram, showing the major components of each module.



Sensing Module

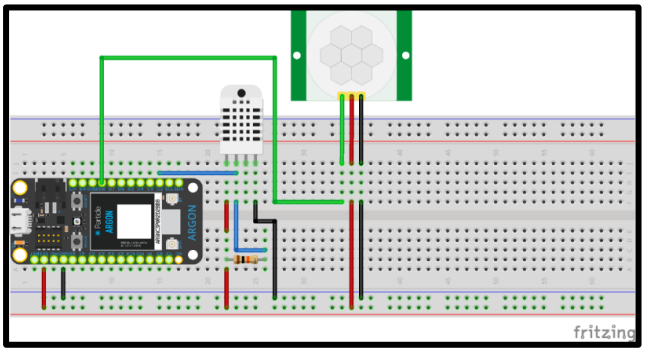
The main component of the sensing module is a Particle Argon. Connected to the Argon are a PIR motion sensor and a DHT22 temperature and relative humidity sensor. The Argon is powered with a portable charger, allowing the Sensing Module to be placed anywhere in the room. Some portable chargers automatically turn off when too little power is being drawn and these cannot be used to power the Argon, but any other portable charger will work.

Particle Argon
https://core-electronics.com.au/particle-argon-kit.html

PIR Motion Sensor
https://core-electronics.com.au/pir-motion-sensor-large-lens-version-seeed-studio.html

DHT22 Temperature and Relative Humidity Sensor
https://core-electronics.com.au/dht22-temperature-and-relative-humidity-sensor-module.html

The diagram below shows how the Argon, the PIR motion sensor and the DHT22 are connected. Note the 10kΩ resistor connecting the VCC and OUT pins of the DHT22.



The photo below shows the complete Sensing Module mounted on my bookshelf. The square board is mounted with hanging strips and is easy to remove and reattach. The portable charger is also attached to the board with a hanging strip and is easy to remove for recharging. I needed to use a right-angle header to connect the motion sensor while still having it face outwards (I took a header from an Arduino shield that wasn't working and bent the pins).

The program running on the Argon connects to the Application Module via MQTT. It reads the data from the PIR Motion Sensor and DHT22 and publishes events containing the data for the Application Module. To minimise network traffic, it generally only publishes an event

when motion is detected or the temperature has changed. However, it will publish an event at least once per minute regardless.

Application Module

The sole component of the Application Module is a Raspberry Pi 3 Model B+ running the NOOBS operating system.

> **Raspberry Pi 3 Model B+**
> https://core-electronics.com.au/raspberry-pi-3-starter-kit-34285.html

The Application Module contains Python scripts to run the GUI application, which receives sensing data from the Sensing Module, allows the user to change settings, and triggers IFTTT events to direct the Heating Module or send the user an email when a fault occurs. The Application Module will set the Heating Module to the desired state (on or off) at regular intervals, to maintain control of the heater even if people interfere. MQTT needed to be installed on the Raspberry Pi with the following terminal commands:

```
sudo apt update
sudo apt-get install mosquitto mosquitto-clients
sudo pip install paho-mqtt
```

Heating Module

The Heating Module consists of the heater itself and a TP-Link HS100 smart plug. The heater is plugged into the smart plug, which is plugged into a power point. The smart plug was added to a Kasa account so it can be turned on and off with IFTTT triggers.

> **TP-Link HS100 Smart Plug**
> https://www.harveynorman.com.au/tp-link-smart-wifi-plug.html

IFTTT Applets

In addition to the modules, three IFTTT applets are needed. To create these applets, IFTTT needs to be verified to use the Kasa account containing the smart plug.

First Applet

- Trigger: Webhook – Receive a web request, Event name = 'Turn On Heater'.
- Action: TP-Link Kasa – Turn on, Device = 'SmartPlug[HS100(AU)]'

Second Applet

- Trigger: Webhook – Receive a web request, Event name = 'Turn Off Heater'.
- Action: TP-Link Kasa – Turn off, Device = 'SmartPlug[HS100(AU)]'

Third Applet

- Trigger: Webhook – Receive a web request, Event name = 'Heating System Error'
- Action: Email – Send me an email, Subject = 'Heating System Error', Body = 'What: {{EventName}}
  When: {{OccurredAt}}
  The following error has occurred in the Heating System: {{Value1}}'

Protype Code

The tests and prototype code can be found at the GitHub repository linked below:
https://github.com/ShaunDyson/SIT210-FinalProject

Testing Approach

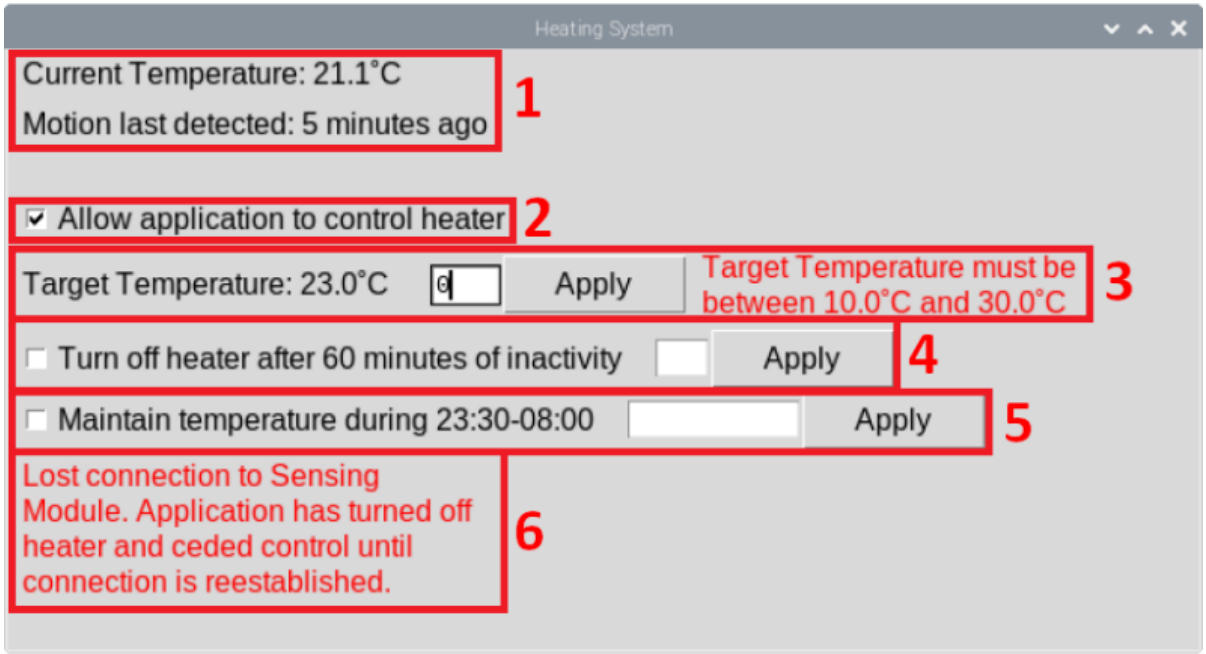
My testing approach was to implement and test one small piece of the project at a time. I wrote a Python script purely to test the IFTTT triggers after I had created them. Then I wrote simple C++ code for the Argon and a corresponding Python script for the Raspberry Pi to test MQTT communication between the two. After that, I expanded the MQTT test to read sensor data and send that instead of sample data. I built up the project in small, easily testable steps.

I developed the final Argon code and GUI application in tandem. To a large extent, these combined features that I had already implemented and tested. When developing these, I would build one small piece at a time and test it. For example, I first built the part of the GUI app that would show the current temperature and created a Python test script to publish MQTT sensing events for the GUI application to receive. I was able to use this test script to test this single GUI application feature without having yet written the Argon code. Once this was working, I wrote and tested the Argon code to send the temperature data. I continued developing the rest of my project in this way, implementing a single feature of the GUI application, testing it by expanding the test script and publishing a mock event, and then writing/modifying the Argon code to publish real events and testing those. I also made sure to test previous features as I developed new features, to make sure I hadn't broken anything.

When the project was finished, I tested every feature again, with different combinations of settings. I used both the test script and the Argon to test the final code. The test code I created is included in the GitHub repository linked above.

User Manual

Once the system is set up, the user just needs to run the 'app.py' file to use the GUI application.



1
This section simply shows the current temperature and when motion was last detected

2
This option toggles whether the app should control the heater at all. Instead of closing the app to regain control of your heater, you can simply disable this option and reenable it when you want the app to resume control.

3
This shows the temperature the app will try to maintain. You can change this with the text field and 'Apply' button. This value can be between 10.0˚C and 30.0˚C. If you provide an invalid input to any of the text fields, an error will be shown to the right of the corresponding 'Apply' button.

4
Enabling this option will cause the app to keep the heater turned off after the specified period of inactivity. You can use the text field and the 'Apply' button to change how long that period is in minutes. This can be between 1 and 999.

5
Enabling this option will cause the app to try and maintain the target temperature during the specified period, regardless of motion detected and the previous option. You can use the text field and the 'Apply' button to change the start and end times of this period. You must enter text in the format, '00:00-00:00'. If the end time is earlier than the start time, for example 23:30-08:00, the app will assume the period goes from the start time on one day to the end time of the next day, from 23:30 on 6th May 2021 to 08:00 7th May 2021, for example.

6
Any errors will be shown here, below the other options.

Conclusion

Designing, implementing, and testing this system was, for the most part, an enjoyable challenge. The biggest issue I faced was the motion sensor not working. I spent too much time and effort trying to get it to work and eventually had to accept that I had to continue the project regardless of the motion sensor not working. As such, the motion sensor still doesn't work, but the GUI application was still able to be fully tested with motion data through the tests that publish mock events.

Despite the motion sensor not working, I think this project was a great success overall. Everything has been implemented and tested and, even without a working motion sensor, the system is still useful for maintaining the room temperature. Replacing the motion sensor with a working one (and, if necessary, modifying code on the Argon to support the new sensor) is all that's needed to make this system fully functional.