Fall 12-14-2018

Integration of a Commercial Smart Thermostat to the Aggregated Load Control Simulation Framework

Jee Won Choi
University of New Mexico

Jee Won Choi

Candidate

Mechanical Engineering

Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Andrea Mammoli , Chairperson

Peter Vorobieff

Ali Bidram

Integration of a Commercial Smart Thermostat to the Aggregated Load Control Simulation Framework

by

Jee Won Choi

B.S., Mechanical Automotive Engineering, Keimyung University, 2015

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Mechanical Engineering

The University of New Mexico

Albuquerque, New Mexico

December, 2018

Dedication

사랑하는 외할아버지께.

To my dearly missed grandpa.

Acknowledgments

I would like to thank my advisor, Professor Andrea Mammoli, for his guidance, patience, and encouragement throughout the whole study. I would also like to thank my colleage, Matt, for his willingness to help, and for sharing his carrots.

Special thanks to the Fellowship Christian Reformed Church family, for their kindest encouragement, and strongest support with prayers. Many love and thanks to my family; grand parents, uncles, aunties, cousins, and my only sister.

Last, but not least, I would like to thank my parents, who define unconditional love, with my greatest and deepest gratitude. I cannot express enough how blessed I am to receive their love and prayers. Thank you, and I love you a ton.

Integration of a Commercial Smart Thermostat to the Aggregated Load Control Simulation Framework

by

Jee Won Choi

B.S., Mechanical Automotive Engineering, Keimyung University, 2015

M.S., Mechanical Engineering, University of New Mexico, 2018

Abstract

As part of an effort to achieve a better balance between the power demand and supply, load (demand) control simulation framework is previously developed. In the framework, HVAC load is used as a control resource, driven by the thermostat logic modeled within the framework. However, it has not been proved, whether a commercial thermostat is able to perform modeled thermostat's features. Therefore, it is desired to integrate a commercial thermostat to the framework, to verify its capability of participating and performing the Demand Response (DR) scheme developed. A 'Nest Learning Thermostat' is selected as a commercial thermostat. Nest Application Programming Interface (API) is used to import response of the Nest thermostat, as well as to input desired settings on the thermostat. A PID control system is implemented to regulate the temperature of a physical chamber, where the Nest thermostat is installed. As a result, the Nest thermostat is verified to be capable of responding to the DR signal. Also, simulation of the Nest thermostat is obtained by modeling its inertial behavior and switching logics empirically observed. A few limitations of the current study is introduced, and future work to overcome the limitations is proposed.

Contents

List of Figures

List of Figures

List of Tables

Chapter 1

Introduction

In order to reduce carbon emissions, as to mitigate climate change, end-uses of energy at the residential, commercial, industrial and transportation sectors are being electrified, leading to the increase in the overall electric power demand. All the while, the power demand at the residential sector, was responsible for 37% of the electricity delievered by the U.S. electric grid in 2017 [1]. While the contribution of the residential sector to the electric load is great, only 6% of households with broadband are currently participating in a demand response (DR) program—DR indicates managing the demand side instead of the supply, in order to balance the two—, according to Parks Associates [2]. One of the reason is because the existing DR strategies are one-size-fit-all and prioritize load reduction over user comfort [2]. However, studies on the DR control schemes, seeking the accomodation of both the load reduction and the user comfort, are actively being conducted with the advance of machine intelligence technology. Moreover, overall advanced technologies, in such areas as microprocessors and networks, are accelerating the pervasion of the smart home appliances. In fact, the U.S. department of Energy (DOE) reported that of the 40 million thermostats sold in 2015, 40% were smart thermostats [3].

Many reviewed various DR schemes available. In a study by Shariatzadeh et al. [4], the existing DR schemes are classfied into the two groups, dispatchable (incentive-based) DR, and non-dispatchable (time or price-based) DR. The former includes direct load control (DLC), and interruptible/curtailable service (I/C), the latter includes time of use (TOU), critical peak pricing (CPP), and real time pricing (RTP). The home energy management (HEM) technologies currently available on the market, are reviewed by Ford et al. [5]. In the review, categorization of the smart home products available and technologies applied, and their potential for delivering energy savings and demand shifting, are discussed in detail. Providing reviews on a total of 61 smart thermostats, the authors suggest that the energy savings potential in DR associated with smart thermostats of all the smart home technologies, is perhaps the most obvious. The fact that 46% of residential energy usage is from the heating and cooling [6], supports the suggestion.

With the potential that HVAC systems possess, and the technologies available, many utility companies are actively conducting residential DR programs via various smart thermostats. In general, DR programs seek the residential demand reduction, by pre-cooling / heating and relaxing the setpoints during the peak demand. Larson et al. [7] introduce DR demonstration projects conducted by National Grid. Two different DR programs ConnectedSolutions (CS) and Rush Hour Rewards (RHR) are offered, details such as DR event frequency and duration varying in thermostat model (ecobee, Honeywell, and Nest). Over $2,000$ thermostats were enrolled in the program in Messachusetts and Rhode Island. The study has revealed important findings, regarding how the various program design features affect customer acceptance of the DR program offerings, such as whether to agree on changing the setpoint or not. Midstate Electric (MEC) designed and piloted the Peak Hour Rewards program, to reduce peak demand across Central Oregon [6]. Lux / Geo-Wifi programmable thermostat is granted to the participants. Participants are informed

prior to a peak demand event, and MEC adjusts the setpoints by two to three degrees during the peak event. A total of 191 thermostats were enrolled in the program, and $300 - 400$kW of load reduction per event on average is reported. Farmers Electric Cooperative (Farmers EC) partnered with Nest, offering the participants Rush Hour Rewards (RHR) in Northeast Texas, summer of 2017. Farmers EC notifies Nest upcoming peak event. Nest then alerts the participants and conducts pre-cooling unique to each home. Kansas City Power & Light (KCP&L) also offers RHR across eastern Kansas and western Missouri. KCP&L accomplished close to $6,000$ thermostats enrolled [8]. They also reported achieving a 55% cooling load reduction and an annual reduction of 462kWh. KCP&L estimates that customers participating in the RHR program have attained 1.2kW reduction of power consumption on average per thermostat. This equates to 15% savings on cooling bills, $10 - 12\%$ savings on heating bills, and an average annual savings of $\$131 - 145$ on their utility bill [9]. Nest Labs also reported that the load was reduced an average of 55.1% for an average of 1.18kW per device, from the RHR participants across Austin Energy, Southern California Edison, and Reliant [2].

While the introduced DR programs, associating with the smart thermostats, are proved to reduce the HVAC load in some amount, they are not yet able to match the reduction with an exact amount. DR schemes pursuing precise shaping of a demand, at the substation or grid level, are currently being conducted by many researchers. However, their application to the market so far, is not prevailing. Growing power generation through renewable energy sources (RES), such as solar photovoltaics (PV's) and wind turbines, raises the need for those DR programs, due to the intermittency they bring. Their intermittent power generation leads to fast and irregular change in power supply, therefore degrading a reliability of a grid system. Mathieu *et al.* [10] suggest a development of control framework that enables nondisruptive control of thermostatically controlled loads (TCLs), such as HVAC. The load is controlled by

both decrasing and increasing power use over short time scales, in order to follow a desired value. This paper also proposes the level of sensing and communications through a thermostat, that are required to enable fast DR. Bashash *et al.* [11] introduced a modeling and control of aggregated air conditioning. The authors applied a sliding mode controller for the real-time management of thermostatic air conditioning loads, responding to the grid's need for balancing power supply and demand in real-time. The control of the aggregated air conditioning, is simulated with using the real wind power data collected by the National Renewable Energy Laboratory (NREL). A stochatic (probabilistic) DR control strategy, also as to control the power load, is introduced by Mammoli *et al.* [12]. Applying the stochastic DR scheme, the aggregated load control in real-time, specifically at the distribution level, using the air conditioning load as a controllable resource, is introduced by Yasaei *et al.* [13]. One of the key features of this work is that the effect of the HVAC deadtime (the minimum time required between the HAVC switching), which has not been addressed in many studies so far, is introduced and reflected in the simulation. This work, is then further developed into a total framework, that simulates the load aggregated from a residential community, as well as the control of the load, using the stochastic DR scheme [14].

In the framework introduced by Mammoli *et al.* [14], the thermostat modeled for the residential houses is assumed to be capable of, first, reporting its status to outside world, second, receiving external signals, and third, performing simple calculation. However, it is not yet verified whether these features, modeled for the simulated thermostat, are performable by commercially available thermostats or not. Therefore, the present study's main contributions are, first, to integrate a commercial smart thermostat to the existing aggregated load control simulation framework, and second, to verify the usability of a commercial smart thermostat in the framework, and finally, to study characteristics of its response.

The 3rd generation 'Nest Learning Thermostat' is selected as a commercial smart thermostat to be integrated and studied, which will be referred to as the Nest thermostat throughout the study. The Nest thermostat is attractive in that it provides an Application Programming Inteface (API) to the developers, which enables data acquisition and switching of the thermostat through the third party platform. Also, as of 2016, it was reported that more than 30% of homes in the U.S. have access to a Nest thermostat rebate or RHR program, with the addition of more than 50 utility partners, including Commonwealth Edison, Enbridge Gas Distribution, and Vectren [15]. In addition, the number of customers enrolled in Nest's RHR programs was found to be more than doubled from 2015 [15]. Considering Nest's success in the DR program, considerable market share, and growth in the business, conducting studies on the Nest thermostat is expected to bring better understandings of the current DR program, as well as the novel ideas of how to improve the program.

In the present thesis, the aggregated load control simulation framework which is developed previously, is introduced in Chapter. 2. The implementation of a physical chamber, where the Nest thermostat is installed, follows in Chapter. 3. The chamber serves as a residential house for the Nest thermostat, and its temperature is regulated by a PID controller. After experimental setups are established, the integration of the Nest thermostat to the framework through Nest API, is demonstrated in Chapter. 4. Results of aggregated load control simulation with the Nest thermostat synthesized, an unexpected observation made through experiments, and the limitations of the current study, are discussed in Chapter. 5. Lastly, the present study is concluded in Chapter 6, and a possible future work is also proposed in the same chapter.

Chapter 2

Aggregated Load Control Simulation Framework

In the present chapter, the Aggregated load control simulation framework previously developed [14], is introduced. The framework consists of the two major components, the load generator and the aggregator. From the generator, six representative residential loads, namely water heaters, HVAC, refrigerators, clothes washer / dryers, electric ranges, and lights, are statistically produced. The six energy uses are generated for each of multiple homes in a residential community, which represent the DR program participants. A probabilistic control signal for the load aggregated from the community, is yielded in the aggregator, with the HVACs serving as control resources.

2.1 Residential Load Generator

For the purpose of the present study, details are given only to the part associating with the HVAC, among the six residential loads. A schematic illustration of the

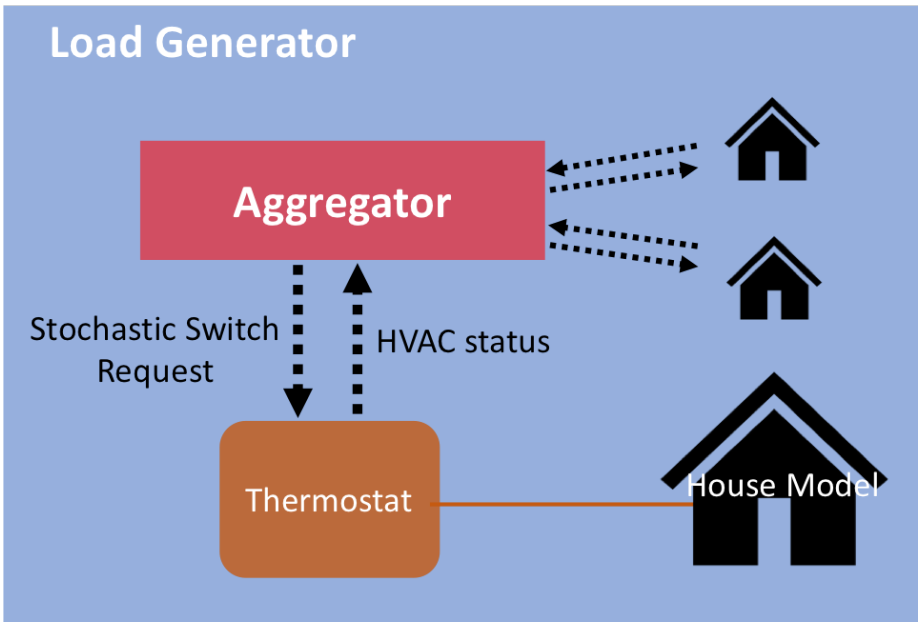


Figure 2.1: Statistically drawn house occupancy schedule, setpoint/deadband, and current house model temperature are sent to the load aggregator. Based on its control error, the load aggregator decides whether to activate or deactivate the HVAC loads. The simulated thermostat logic responds to the signal from the load aggregator, and its response is sent to the model. These steps are repeated in a time loop.

HVAC load simulation is shown in Fig. 2.1. The number of load occurrences per day, the load start time and the load duration for each house are characterized by the associated probability density functions (PDFs). Sample PDFs for a residential HVAC load are shown in Fig. 2.2. At the beginning of each day, a schedule for each meter and each load is drawn from PDFs. For the case of an HVAC event, the activation in the schedule represents the change in the setpoint / deadband. The deadband in the field of HVAC, means the temperature range where neither heating nor cooling is required. However, in the present study, it is used to indicate the temperature range where the HVAC switching does not occur, or the range desired by the user. The HVAC schedule activation also may be referred to as the space occupancy, since the deadband is set to a narrower (comfort-oriented) range during schedule activation, assuming that the user is present in the space. For example, as shown in Fig. 2.3, on

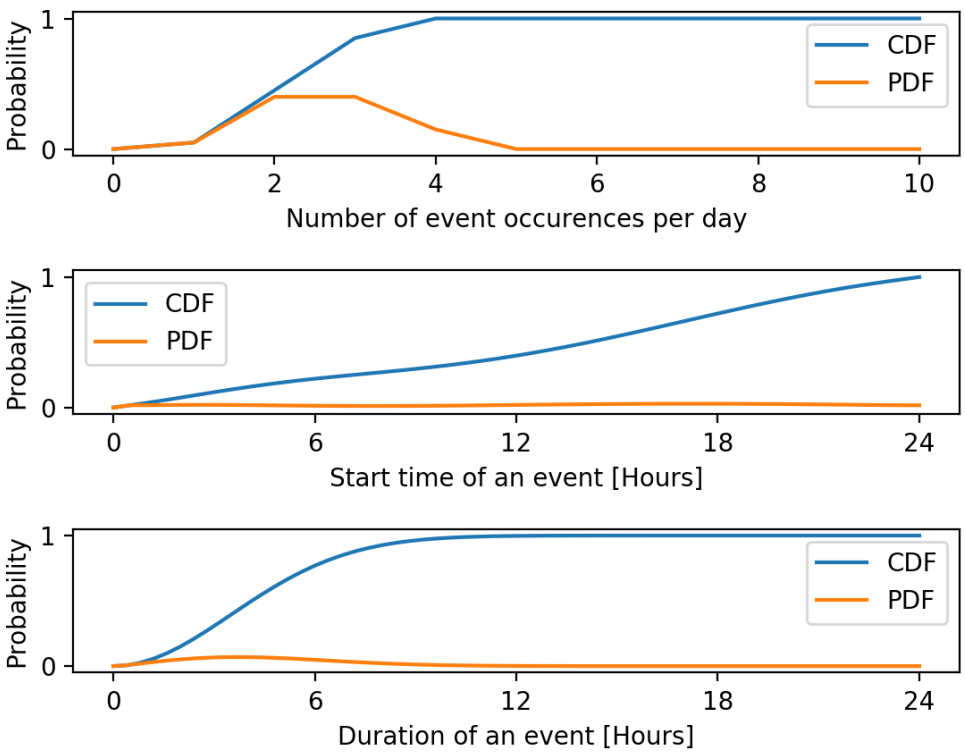


Figure 2.2: PDFs and CDFs of the number of residential HVAC load occurrences per day, the start time of the load occurence and the duration of the load. The residential HVAC load schedule in particular indicates the residential space occupancy.

a particular day, a specific house can be occupied twice, from 4:30 for 4.5 hours, and from 14:30 for 2 hours. A temperature setpoint and associated deadband are generally set to the value desired by user, although in some cases these could be modified for DR control purposes. DR control can take various forms, such as a change in the setpoint / deadband or by switching the HVAC operation back and forth. In the previous work, the deadband was simply set wider when unoccupied to reduce the HVAC load, as a normal smart thermostat would do, while the state of the HVAC compressor is switched depending not only on the thermal response model, that is followed in the next paragraph, but also on whether higher or lower load is required

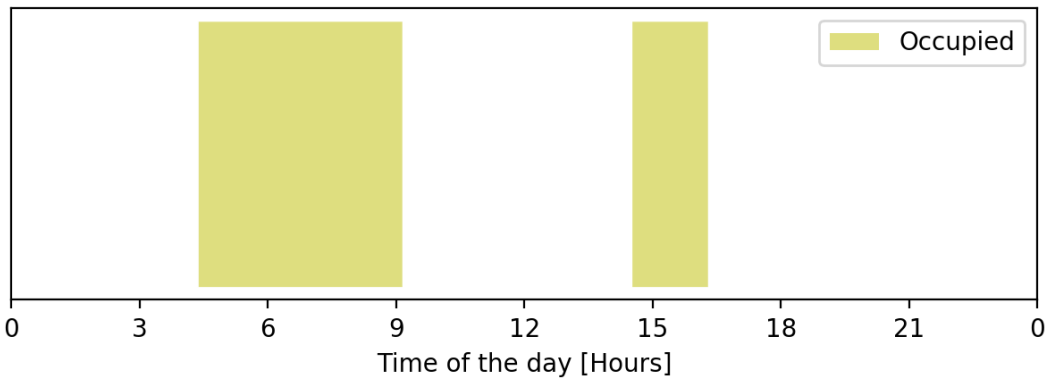


Figure 2.3: An example of the statistically drawn house occupancy schedule for a specific meter on a specific day.

by the distribution system (e.g. substation), that will be introduced in Section. 2.2.

For this study, it is assumed that all houses are equipped with an HVAC system based on an electrically powered heat pump for space temperature regulation. This setup is widely used in areas with moderate climate, and will become increasingly prevalent with increasing electrification of energy services. The thermal response of an air conditioned space can be described by

$$M_{\mathrm{s}}\frac{dT_{\mathrm{s}}(t)}{dt} = \dot{Q}_{\mathrm{L}} - \dot{Q}_{\mathrm{R}}, \tag{2.1}$$

$$\dot{Q}_{\mathrm{L}} = K_{\mathrm{s}}[T_{\mathrm{a}} - T_{\mathrm{s}}(t)], \tag{2.2}$$

$$\dot{Q}_{\mathrm{R}} = \Pi \times \Lambda \times \mathrm{COP} \times P_{\mathrm{AC}}. \tag{2.3}$$

In this model, derived from the physical nature of the system, M_{s} is the effective heat capacity of the conditioned space, $T_{\mathrm{s}}(t)$ is the temperature of the conditioned space, t is the time, $\dot{Q}_{\mathrm{L}}$ is the thermal energy exchange between the conditioned space and its surrounding, T_a is the ambient temperature outside of the building, K_{s} models the heat exchange between the building and the external air, $\dot{Q}_{\mathrm{R}}$ is the rate

of energy outflow or inflow from the space via the HVAC, COP is the coefficient of performance of the HVAC system, and P_{AC} is the electric power consumed by the compressor. It is assumed that the heat pump utilizes a fixed-speed compressor, so that P_{AC} stays constant. Π indicates whether the HVAC mode is cooling or heating, and Λ is a state function that indicates whether the compressor is on or off. The HVAC mode is determined by

$$\Pi = \begin{cases} 1, & \text{if } T_{\mathrm{a}} - T_{\mathrm{s}}(t) > 0 \\ -1, & \text{if } T_{\mathrm{a}} - T_{\mathrm{s}}(t) < 0, \end{cases} \tag{2.4}$$

where the HVAC is in the cooling mode when $\Pi = 1$, or in the heating mode when $\Pi = -1$. The switching logic for the simulated thermostat, when the HVAC is in the cooling mode, is described by

$$\Lambda = \begin{cases} 0, & \text{if } T_{\mathrm{s}}(t) < T_{\mathrm{L}} \\ 1, & \text{if } T_{\mathrm{s}}(t) > T_{\mathrm{U}}, \end{cases} \tag{2.5}$$

where T_{L} and T_{U} are the lower and upper deadband limits for the space temperature control. When the temperature is within the deadband, the state function Λ at a particular time step remains the same as its previous value, i.e. switching only occurs when the temperature reaches the deadband limits. In similar manner, when HVAC is in heating mode,

$$\Lambda = \begin{cases} 1, & \text{if } T_{\mathrm{s}}(t) < T_{\mathrm{L}} \\ 0, & \text{if } T_{\mathrm{s}}(t) > T_{\mathrm{U}}. \end{cases} \tag{2.6}$$

The HVAC load for an individual house, at a particular time step, is then calculated by

$$L_{\mathrm{HVAC}} = \Lambda \times P_{\mathrm{AC}}. \tag{2.7}$$

2.2 Aggregator

During operation, the information from every house model whether the HVAC compressors are on or off, and whether they are switchable or not (if it has passed the deadtime since the last switching), are reported to the aggregator. The aggregator, then makes a decision on whether and how much to collectively activate or deactivate HVAC loads. DR program participating thermostats are informed with this decision (DR signal), then finalizes HVAC state following the stochastic switching process. The sequence of the DR control process is :

1. At timestep k the substation reports its control error $\mathcal{E}$ - namely the difference between a desired load and the current aggregated load.

2. The aggregator calculates the maximum positive and negative response capacities (i.e. the maximum load increase or reduction obtainable from the HVACs of the community) $\mathcal{P}$ and $\mathcal{N}$, respectively, from information reported by each smart thermostat (HVAC on / off status, and HVAC switchability) participating in the control program.

3. The aggregator calculates the fraction $\mathcal{F}$ that represents the probability of the HVAC switching, i.e., higher $|\mathcal{F}|$ leads to more switching of HVACs. $\mathcal{F}$ is defined by

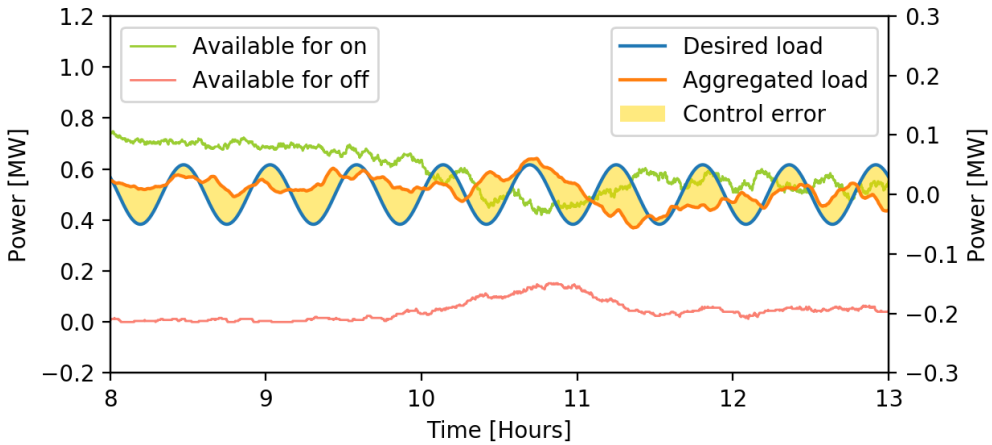
$$\mathcal{F} = \begin{cases} \frac{\min(\mathcal{R},\mathcal{E})}{\mathcal{P}}, & \text{if } \mathcal{E} > 0 \\ \frac{\max(-\mathcal{R},\mathcal{E})}{\mathcal{N}}, & \text{if } \mathcal{E} < 0, \end{cases} \tag{2.8}$$

   where $\mathcal{R}$ is the maximum ramping rate of the system. The aggregator broadcasts $\mathcal{F}$ to all thermostats.

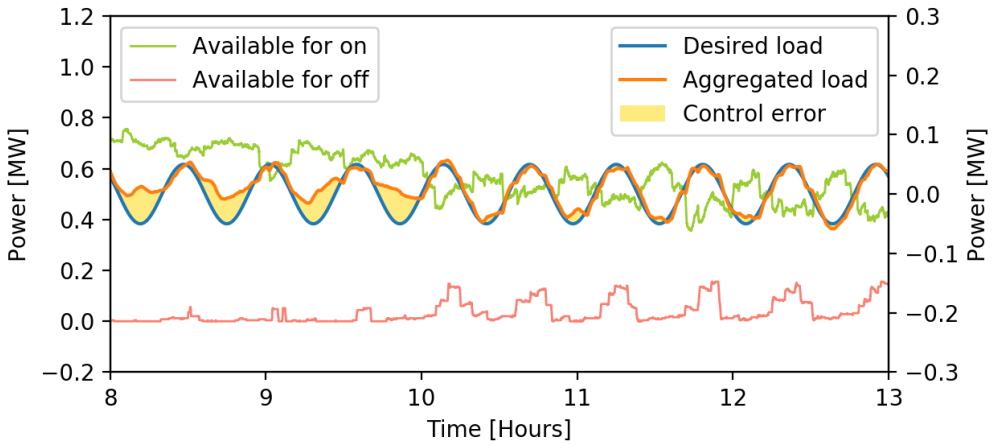
4. Each thermostat draws a random number $\mathcal{I}$ between 0 and 1, from a uniform distribution. If $\mathcal{I} < |\mathcal{F}|$, the HVAC is switched from off to on at timestep $k+1$,

if the error is positive ($\mathcal{F} > 0$), or from off to on, otherwise ($\mathcal{F} < 0$). Switching occurs only within the temperature deadband.

An example of the aggregated load control from the two hundred house models with stochastic switching algorithm, is shown in Fig. 2.4. Here, the sinusoidal signal as a desired load, represents a possible high frequency component of a power supply at a substation. The HVAC load is expected to track the highly frequent signal, since it provides fast reponse due to its high power draw. Note that components that are too high in capacity or frequency, are filtered from the aggregated load reported from the substation, since it is not reasonable to expect an HVAC to generate neither an infinite amount of load nor an infinite frequency load. In Fig. 2.4b, it is shown that the aggregated load tracks desired signal very well from 10.5 to 13 hours, while there exists some greater control error from 8 to 10.5 hours. It is observed that this negative control error occurs due to the lack of a resource, that is available for turning off, in contrast to barely no error when the system has enough resources. The thermal response of a random house model from the community, is shown in Fig. 2.5. When the aggregated load control is not activated, the simulated thermostat sets the HVAC to the cooling mode following Eqn. 2.4, then the house model behaves exactly as demonstrated in Eqn. 2.5. As shown in Fig. 2.5a, the HVAC turns off when the space temperature is lower than the lower deadband, turns on when higher than the upper deadband, and keeps the previous state within the deadband, i.e., switching does not occur within the deadband. On the other hand, when the aggregated control is activated, the HVAC switching occurs also within the deadband, responding to the DR signal broadcasted by the aggregator.

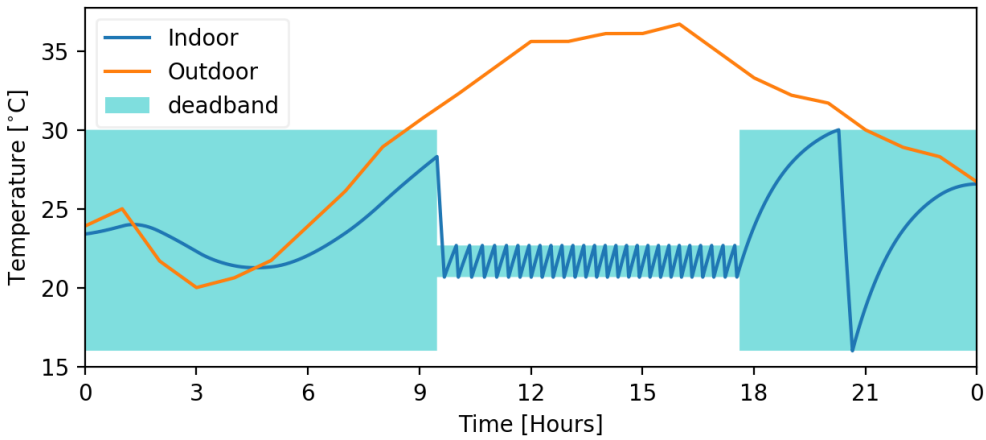


(a) Aggregated load control not applied

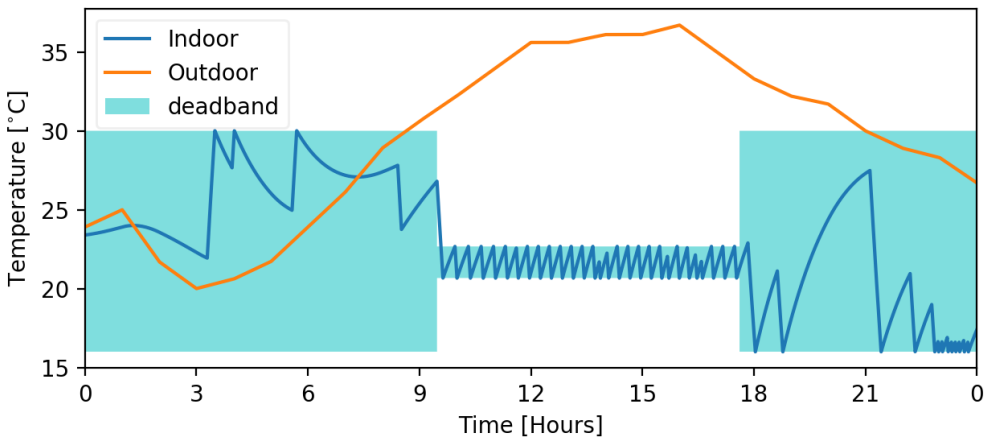


(b) Aggregated load control applied

Figure 2.4: Load aggregated from 200 house models, is controlled with HVAC load as a control resource. The sinusoidal signal represents a possible high frequency component of a desired demand. Components that are too high in capacity or frequency, are filtered from the aggregated load. The aggregated load is capable of tracking the desired signal, when there exists enough resources (available for on and off).



(a) Aggregated load control not applied



(b) Aggregated load control applied

Figure 2.5: When the aggregated load is not controlled, the HVAC switching occurs only when the space temperature reaches the boundaries of the deadband. On the other hand, switching occurs also within the deadband, when control on the load is activated.

Chapter 3

Experimental Setup

3.1 Chamber Temperature PID controller

The Nest thermostat is installed in a physical chamber. This chamber mimics the real-time conditions of a house model, whose temperature is modeled by Eqn's. 2.1-2.3. The Nest thermostat yields the actual response, that is desired to be utilized within the HVAC load simulation. The temperature inside the chamber is controlled by using two fan / heat exchanger combinations. The cooling and heating fan, pushes air through a chilled-water and a heated-water heat exchanger, respectively, then into the chamber. A visual illustration of the physical setup is shown in Fig. 3.1 and detailed descriptions follow.

3.1.1 Temperature Measurement

Chamber temperature is measured from type T thermocouples, that are located in strategic locations inside the chamber, as shown in Fig. 3.1. Analog signals from thermocouples are converted to digital signals by the amplifier 'MAX31856'. The

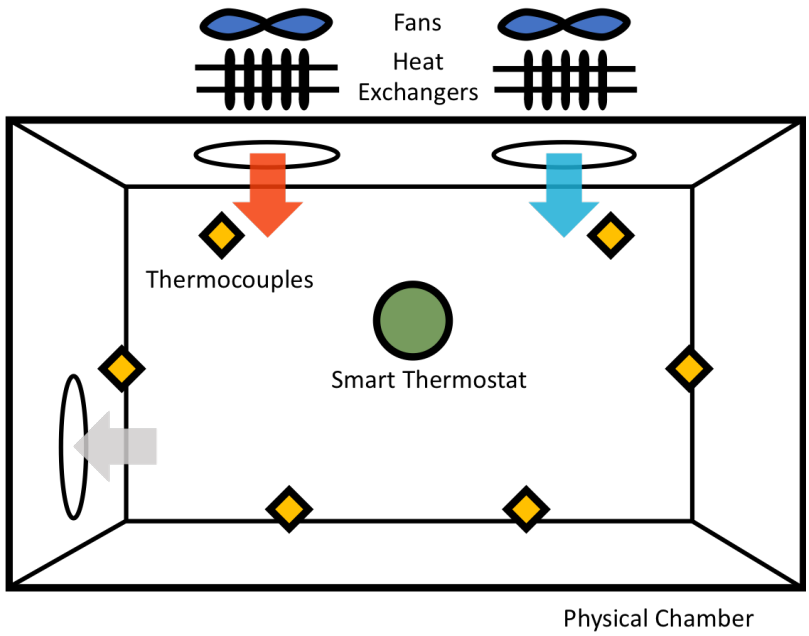


Figure 3.1: Illustration of the environmental chamber that simulates the temperature of a certain house model. The chamber temperature is controlled by two fans, providing the physical environment for the Nest thermostat.

amplifier is connected to the Raspberry Pi GPIO (General-Purpose Input/Output) pins, then they communicate through the SPI (Serial Peripheral Interface). Converted signals are obtainable in Python script using the library 'Adafruit_MAX31856', distributed by John Robinson [16]. An example of reading temperatures from two thermocouples, in a Python script using 'Adafruit_MAX31856', is given in List. 3.1. In SPI communication, four ports CLK(Serial Clock), DI(Master Output Slave Input), DO(Master Input Slave Output), and CS(Slave Select), should be designated. Lines 5-8 show how GPIO pins are assigned to these ports. Line 5 indicates that GPIO pin number 6, 17, 5 and 4 in BCM (Broadcom) mode, that can be seen in Fig. 3.2 [17], are assigned as CLK, CS, DO and DI, respectively. When using multiple amplifiers (thermocouples), ports CLK, DO and DI can be shared, but an individual CS should be assigned per thermocouple as shown in Line 6. The object `TempRead56` is initialized,

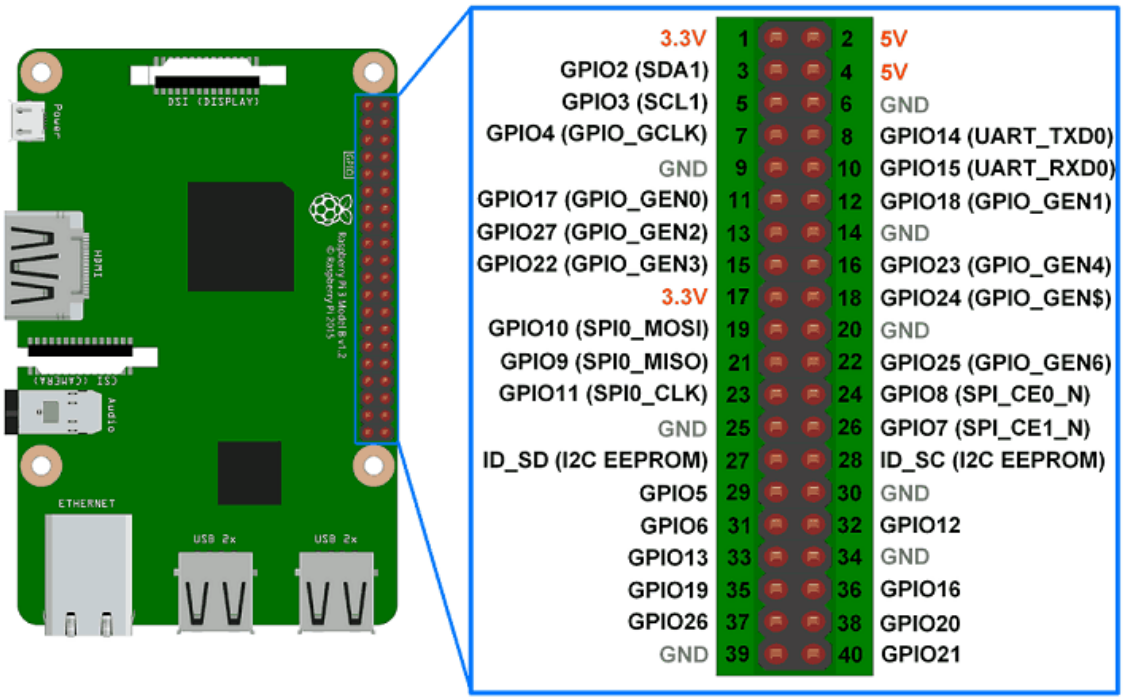


Figure 3.2: Raspberry Pi GPIO pin numbers. Numbers 1 through 40 in the middle, specify GPIO pin numbers in Board mode. Numbers on the outter left and right side, specify pin numbers in BCM mode. For example, GPIO pin 19 in Board mode is pin 10 in BCM mode.

i.e., `def __init__(self)` is executed, when it is first called as shown in Line 13. Line 14 executes `def TempC(self)`, then temperatures measured from each thermocouple are saved in variables `tr.Tc1` and `tr.Tc2`, and obtainable as shown in Line 15-16.

```
1  from Adafruit_MAX31856 import MAX31856 as MAX31856
2
3  class TempRead56: # MAX31856 : higher resolution amplifier
4      def __init__(self):
5          software_spi1 = {"clk": 6, "cs": 17, "do": 5, "di": 4}
6          software_spi2 = {"clk": 6, "cs": 27, "do": 5, "di": 4}
7          self.sensor1 = MAX31856(software_spi=software_spi1)
8          self.sensor2 = MAX31856(software_spi=software_spi2)
9      def TempC(self):
10         self.Tc1 = self.sensor1.read_temp_c()
```

```
11          self.Tc2 = self.sensor2.read_temp_c()
14
15 >>> tr=TempRead56()
16 >>> tr.TempC()
17 >>> tr.Tc1, tr.Tc2
18  (22.890625, 22.828125)
19 >>> tr.TempC()
20 >>> tr.Tc1, tr.Tc2
21  (22.921875, 22.8671875)
```

Listing 3.1: Temperature reading by using Python library 'Adafruit_MAX31856'

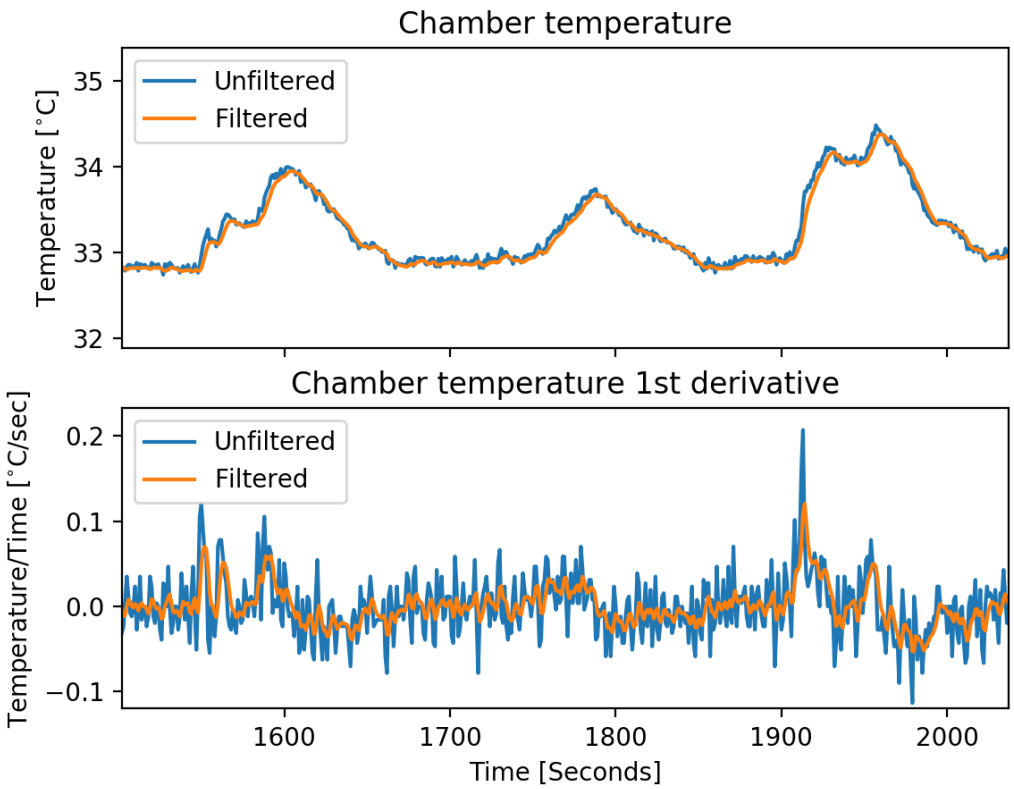


Figure 3.3: An analog temperature is converted to the digital signal with noises. The figure shows a signal, filtered by the linear recursive bandpass time-series digital filter, with coefficients $a = 0$ and $b = 0.3$.

Fig. 3.3 shows the temperature curve of the chamber, measured with the amplifier 'MAX31856', and its first derivative.The amplifier provides fairly precise resolution of 0.008°C, however as shown in the figure, the gradient of the measured temperature chatters between positive and negative values, with maximum magnitude of 0.2°C per second, i.e., the temperature could increase 0.2°C in a second then decrease 0.2°C at the next time step, then keep oscillating. This behavior is not physically realistic, therefore it is considered that there exists noise interruption within the amplifier. This noise can be filtered via linear bandpass time-series digital filter. The most general linear filter takes a form of

$$y_n = \sum_{k=0}^{M} c_k x_{n-k} + \sum_{j=1}^{N} d_j y_{n-j}, \tag{3.1}$$

where x is an unfiltered value, and y is a filtered value. In the case of IIR (Infinite Impulse Response) recursive filtering, when $M = 2$ and $N = 2$, coefficients c_k and d_k are determined as below

$$\begin{aligned} c_0 &= \frac{b}{(1+a)(1+b)} \\ c_1 &= 0 \\ c_2 &= \frac{-b}{(1+a)(1+b)} \\ d_1 &= \frac{(1+a)(1-b)+(1-a)(1+b)}{(1+a)(1+b)} \\ d_2 &= -\frac{(1-a)(1-b)}{(1+a)(1+b)}, \end{aligned} \tag{3.2}$$

where positive numbers a and b, correspond to lower and upper cutoff frequency, respectively, i.e., from the unfiltered signal, components with frequency lower than a and higher than b will be filtered [18]. The digital filter with coefficients $a = 0$ and

$b = 0.3$, gives a much smoother temperature curve with less magnitude in chattering as shown in Fig. 3.3.

3.1.2 Fan Control

Two 12V fans are controlled by a motor driver IC (Integrated Circuit) 'L293D'. The motor driver is connected to the Raspberry Pi GPIO, and is controllable in a Python script using the library 'RPi.GPIO'. The fan speed is modified by PWM (Pulse-width modulation), by adjusting the duty cycle. An example of controlling two motors in a Python script using 'RPi.GPIO' is shown in List. 3.2.

```
1  import RPi.GPIO as GPIO
2
3  class ChamberHvac:
4      def __init__(self):
5          GPIO.setmode(GPIO.BCM)
6          GPIO.setup(18,GPIO.OUT,initial=GPIO.LOW) # assign GPIO pin
               numbers
7          GPIO.setup(20,GPIO.OUT,initial=GPIO.LOW)
8          GPIO.setup(19,GPIO.OUT,initial=GPIO.LOW)
9          GPIO.setup(21,GPIO.OUT,initial=GPIO.LOW)
10         self.fc=GPIO.PWM(18,100)
11         self.fh=GPIO.PWM(19,100)
12         self.fc.start(0)
13         self.fh.start(0)
14     def AC(self,sign):
15         GPIO.output(18,GPIO.HIGH)
16         GPIO.output(20,GPIO.HIGH)
17         GPIO.output(21,GPIO.LOW)
18         self.fc.ChangeDutyCycle(sign)
19     def HT(self,sign):
20         GPIO.output(19,GPIO.HIGH)
21         GPIO.output(21,GPIO.HIGH)
22         GPIO.output(20,GPIO.LOW)
23         self.fh.ChangeDutyCycle(sign)
24     def Quit(self):
25         self.fc.stop()
```

```
26          self.fh.stop()
29          GPIO.cleanup()
30
31  >>> ch = ChamberHvac() # Call the object and initialize
32  >>> ch.AC(100) # runs cooling fan at pwm 100
33  >>> ch.HT(50) # stops cooling fan and operates heating fan at pwm 50
34  >>> ch.Quit() # stops fans in operation and clean up GPIO pins
```

Listing 3.2: Fan operation by using Python library 'RPi.GPIO'

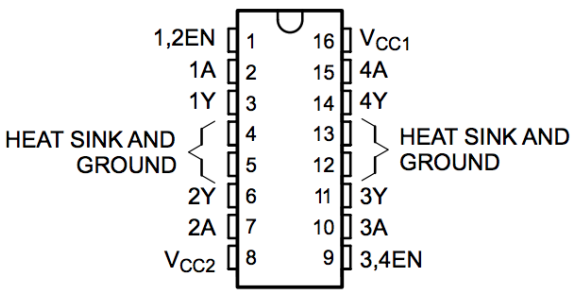


Figure 3.4: The motor driver 'L293D' is connected to the GPIO, then controls the fan motors.

In the function `def __init__(self)` as shown in Lines 4-13, GPIO pins are assigned to 'L293D'. GPIO 18, 20 and GPIO 19, 21, associate with the cooling fan and the heating fan, respectively. Connections between 'L293D' and GPIO pins are shown in Fig. 3.4. Fucntion `def AC(self,sign)` operates the cooling fan, at the duty cycle taken as an argument (`sign`). This `sign` must be an integer between 0 and 100. Line 17 indicates that the function stops the heating fan, if is operating, so that only either the cooling fan or the heating fan operates. Line 31 calls the object `Chamber Hvac` and executes `def __init__(self)`. Line 32 and 33 runs cooling fan at a duty cycle of 100%, and heating fan at a duty cycle of 50%, respectively. Line 34 stops any fan that is running, and cleans up the GPIO pins, so that GPIO pins assigned to 'L293D' are now emptied. The entire circuit connections of the thermocouples, 'MAX31856', motors, 'L293D' and Raspberry Pi GPIO, is shown in Fig. 3.5.

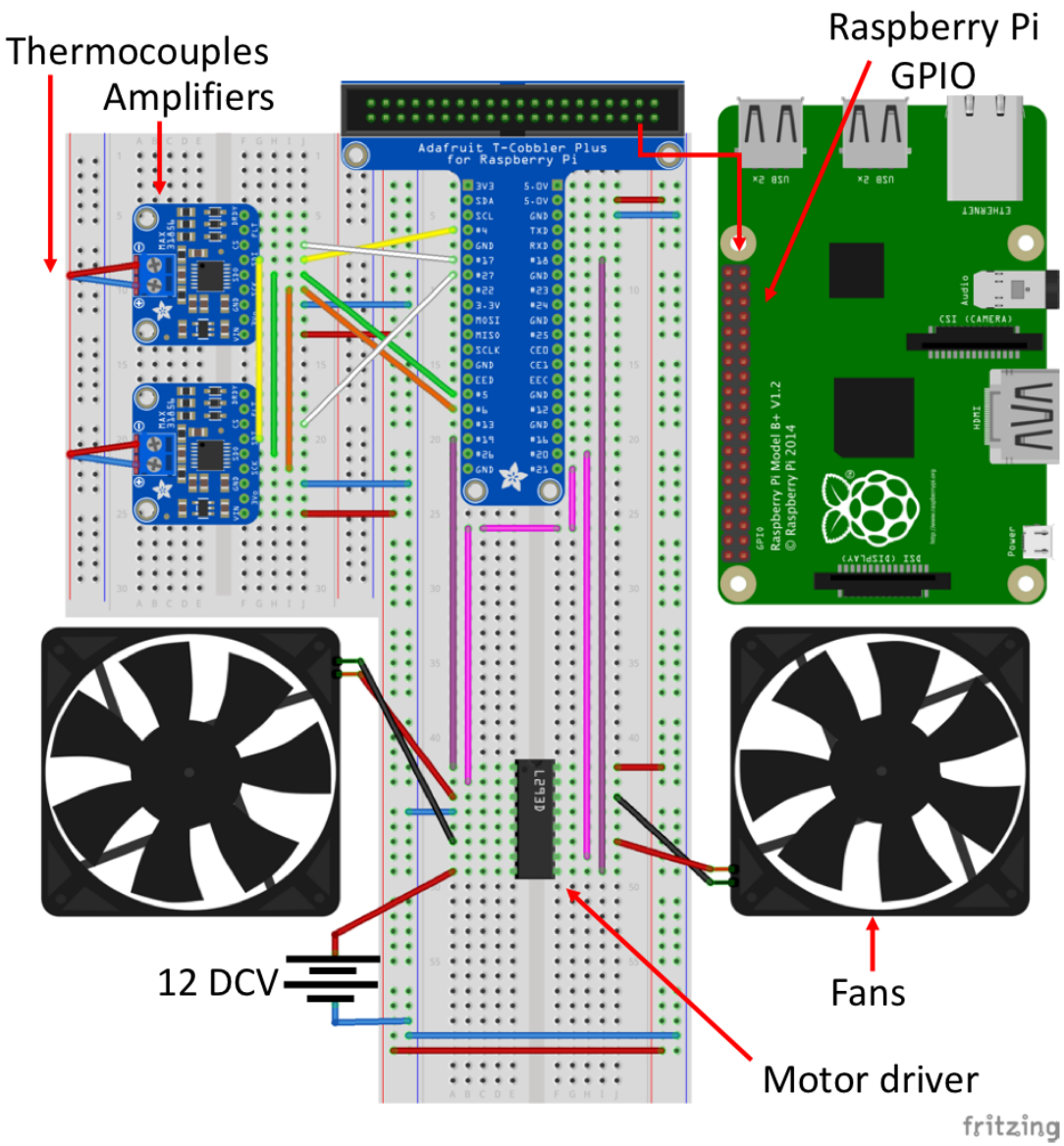


Figure 3.5: An illustration of the temperature measurement and fan control associating with the Raspberry Pi GPIO. Interaction with the thermocouple amplifiers and the motor driver, is available through Python libraries such as 'RPi.GPIO' and 'Adafruit_MAX31856'

3.1.3 Plant Design

Now, with the given hardware setup, it is possible to modify the temperature inside the chamber. First, On-Off control method is applied and the result is shown in Fig. 3.6. Here, the chamber temperature is trying to track the desired temperature at 27°C. When the error is positive or negative, the cooling or heating fan operates at a constant duty cycle. When the fans run with a duty cycle of 50%, the chamber temperature oscillates around the desired temperature resulting in a maximum error of 4 °C. With the duty cycle of 10%, it oscillates with a lower frequency but results in a higher maximum error of 5 °C. The result shows that the desired chamber temperature is unobtainable through the On-Off controller. Therefore it is desired to design a temperature control plant that outputs the variable fan speeds corresponding to the error.

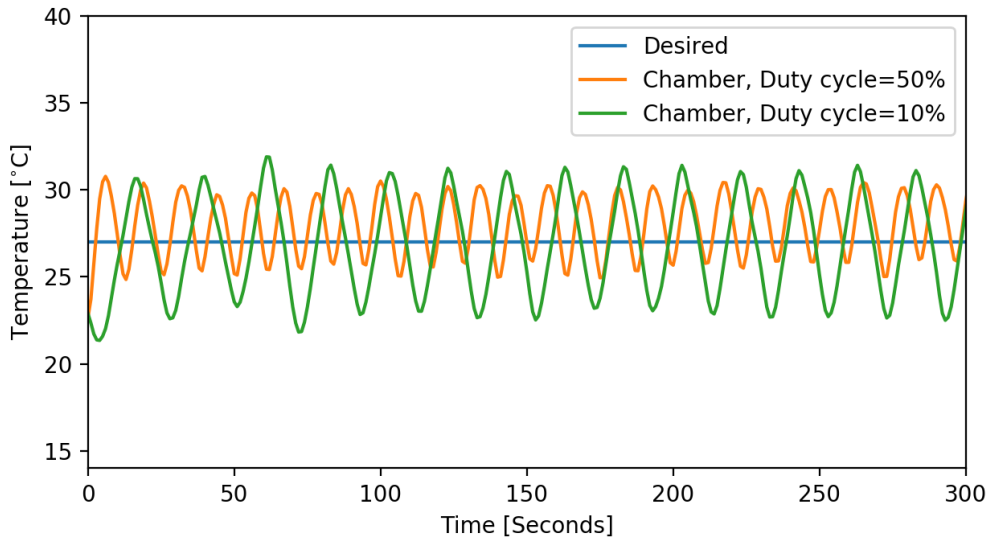


Figure 3.6: On-Off controller trying to track the desired temperature signal. The cooling fan operates when the error is positive, heating fan when negative. In both cases, fans run at a steady duty cycle. The value of the duty cycle affects only the amplitude of oscillation, leading to an unsettling of the chamber temperature.

In the experimental setup shown in Fig. 3.1, temperatures of water running through the heat exchangers remain steady, therefore, the speed of the fan determines the rate of temperature change of the chamber. The relation between the fan speed and the rate of temperature change, can be found from the heat transfer model of the system. By the first law of thermodynamics, the energy balance in the control volume is expressed as

$$\frac{dE_{\text{cv}}}{dt} = \dot{E}_{\text{in}} - \dot{E}_{\text{out}}. \tag{3.3}$$

With the chamber defined as a control volume as shown in Fig. 3.7, Eqn. 3.3 can be written as

$$\begin{aligned} M_{\text{ch}} c_p \frac{dT_{\text{ch}}}{dt} &= \dot{m}_{\text{ch,in}} c_{\text{p}} T_{\text{ch,in}} - \dot{m}_{\text{ch,out}} c_{\text{p}} T_{\text{ch,out}} \\ &= \dot{m}_{\text{ch}} c_p (T_{\text{ch,in}} - T_{\text{ch,out}}), \end{aligned} \tag{3.4}$$

where M_{ch} is the mass of the air inside the chamber, c_{p} is the specific heat capacity of the air, T_{ch} is the chamber temperature, $\dot{m}_{\text{ch,in}}$ and $\dot{m}_{\text{ch,out}}$ are mass flow rate coming in and out of the chamber, respectively, $T_{\text{ch,in}}$ and $T_{\text{ch,out}}$ are the temperature at the inlet and outlet, respectively, and it is assumed to be $\dot{m}_{\text{ch,in}} = \dot{m}_{\text{ch,out}}$. If the heat exchanger can be considered as a group of tubes, the heat transfer occuring inside each tube, can be defined as an internal forced convection. It is assumed that the surface temperature of the heat exchanger is constant, then the relation between temperature at the inlet and outlet of the tube is expressed as

$$T_{\text{hx,out}} = T_{\text{surf}} - (T_{\text{surf}} - T_{\text{hx,in}}) exp\Big(- \frac{hA_{\text{surf}}}{\dot{m} c_{\text{p}}} \Big), \tag{3.5}$$

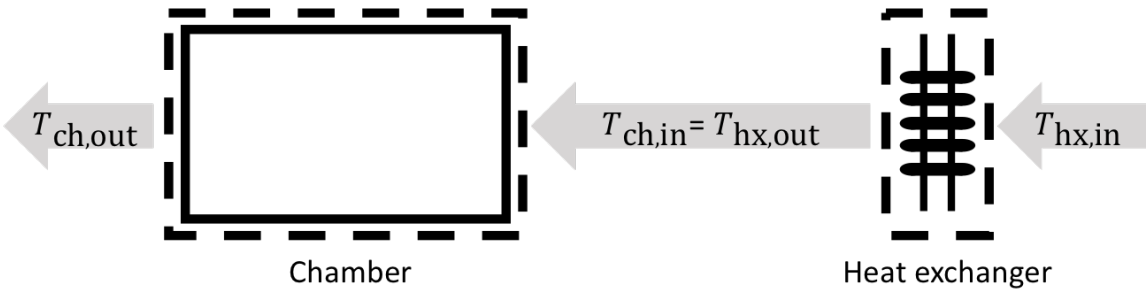


Figure 3.7: Chamber control volume. Mass flow rate $\dot{m}$ through the heat exchanger and the chamber, is assumed to be constant. $T_{\text{hx,in}}$ is steady at the indoor temperature, $T_{\text{hx,out}} = T_{\text{ch,in}}$ is obtainable by modeling heat transfer at the heat exchanger as an internal forced convection flow, and $T_{\text{ch,out}}$ is measured experimentally.

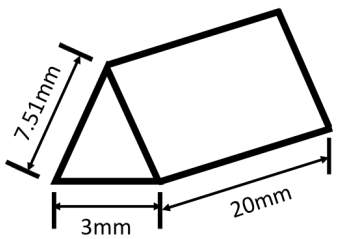


Figure 3.8: Specification of an isosceles triangular tube. The heat exchanger consists of 880 tubes.

where $T_{\text{hx,in}}$ and $T_{\text{hx,out}}$ are temperatures at the inlet and outlet of the heat exchanger, respectively, T_{surf} is the surface temperature of the heat exchanger, h is the convective heat transfer coefficient, A_{surf} is the surface area of the tube, $\dot{m}$ is the mass flow rate through the tube, and c_p is the specific heat capacity of the air since the air is pushed through the heat exchanger. The heat exchanger selected, consists of isosceles triangular tubes shown in Fig. 3.8. For the flow through noncircular tubes, the Reynolds number is defined as

$$\text{Re} = V_{\text{avg}} D_h \nu^{-1} \tag{3.6}$$

where V_{avg} is the average speed of the flow across the cross sectional area, ν is the kinematic viscosity of the air, hydraulic diameter $D_h = 4A_c p^{-1}$, A_c is the cross sectional area, and p is the perimeter of the tube. In the case of internal flow in a tube, flow is laminar for Re < 2300, or fully turbulent for Re > 10000, or transient in between. To define whether the flow is laminar or turbulent, Reynolds numbers, within the range of fan operation, is calculated by using Eqn. 3.6, and results are tabulated in Table. 3.1. Note that the relation $\rho_{\text{air}}\dot{V}_{\text{hx}} = \dot{m}_{\text{hx}} = 880\dot{m}_{\text{tube}}$ holds where ρ_{air} is the air density, $\dot{V}_{\text{hx}}$ is the total volumetric flow rate coming through the heat exchanger from the fan, $\dot{m}_{\text{hx}}$ is the total mass flow rate, and $\dot{m}_{\text{tube}}$ is the mass flow rate of a single tube. Various mass flow rate is represented in terms of duty cycle (fan speed). From the table, it is shown that Reynolds numbers are always smaller than 2300 within the given fan speed range, therefore the flow is laminar. When the

Air properties						**Tube properties**				
ρ_{air} $[kg/m^3]$	c_p $[J/kg\dot{K}]$	ν_{air} $[m^2/s]$	k_{air} $[W/m\dot{K}]$	Pr	p $[m]$	A_c $[m^2]$	D_h $[m]$	L $[m]$	L_h $[m]$	L_t $[m]$
1.2047	1.006[3]	1.5111×10^{-5}	0.025596	0.71559	18.02×10^{-3}	11.04×10^{-6}	2.45×10^{-3}	0.02	$0.012 - 0.12$	$0.0086 - 0.086$
Duty cycle	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
$\dot{V}_{\text{hx}}$ $[m^3/s]$	0.0059	0.0118	0.0177	0.0236	0.0295	0.0354	0.0413	0.0472	0.0531	0.059
V_{avg} $[m/s]$	0.607	1.215	1.822	2.429	3.036	3.644	4.251	4.858	5.466	6.073
$\dot{m}_{\text{hx}}$ $[\times 10^{-2} kg/s]$	0.711	1.422	2.132	2.843	3.554	4.265	4.975	5.686	6.397	7.108
$\dot{m}_{\text{tube}}$ $[\times 10^{-5} kg/s]$	0.808	1.615	2.423	3.231	4.038	4.846	5.654	6.462	7.269	8.077
Re $[-]$	98.5	197.0	295.5	394.0	492.4	590.9	689.4	787.9	886.4	984.9
Nu $[-]$	7.79	8.05	8.30	8.56	8.81	9.07	9.32	9.58	9.83	10.09
h $[W/m\dot{K}]$	81.42	84.08	86.74	89.41	92.07	94.73	97.40	100.06	102.72	105.39
NTU $[-]$	3.611	1.865	1.282	0.991	0.817	0.700	0.617	0.555	0.506	0.467

Table 3.1: System properties associated with the heat transfer, with varying mass flow rate (duty cycle).

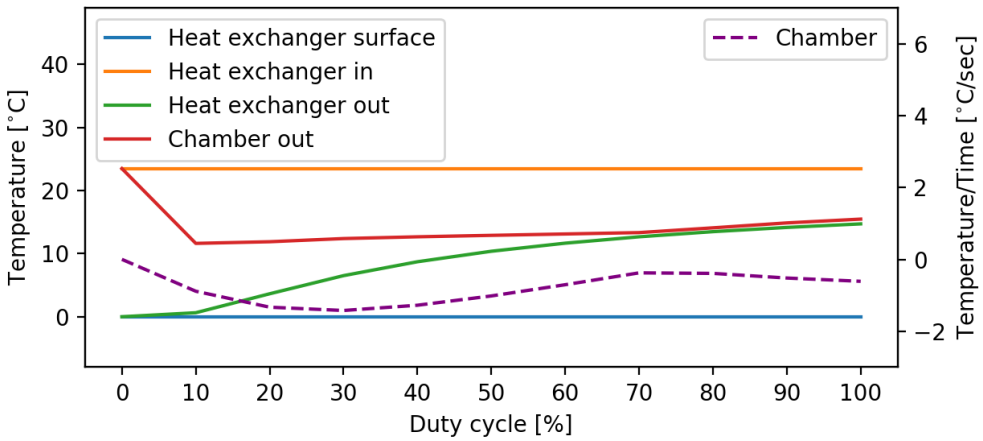
flow is laminar, hydrodynamic and thermal entry lengths are $L_h \approx 0.005\text{Re}D_h$ and $L_t \approx 0.005\text{RePr}D_h$, respectively, where Pr is the Prandtl number of the air. The flow is determined to be fully developed, if both the hydrodynamic and the thermal entry lengths are shorter than the tube length. Otherwise, if either the hydrodynamic or

the thermal entry lengths is longer than the tube length, then the flow is determined to be still developing. From Table. 3.1, L_h ranges from $0.012 - 0.12m$, and L_t ranges from $0.0086 - 0.086m$, within the range of fan operation. It is not provided in the table, however, except at the duty cycle of 10%, L_h and L_t are greater than $0.02m$, which is the length of the tube. Therefore for convenience, the flow is considered as the developing flow. When the flow is laminar and developing, Nusselt number is defined by

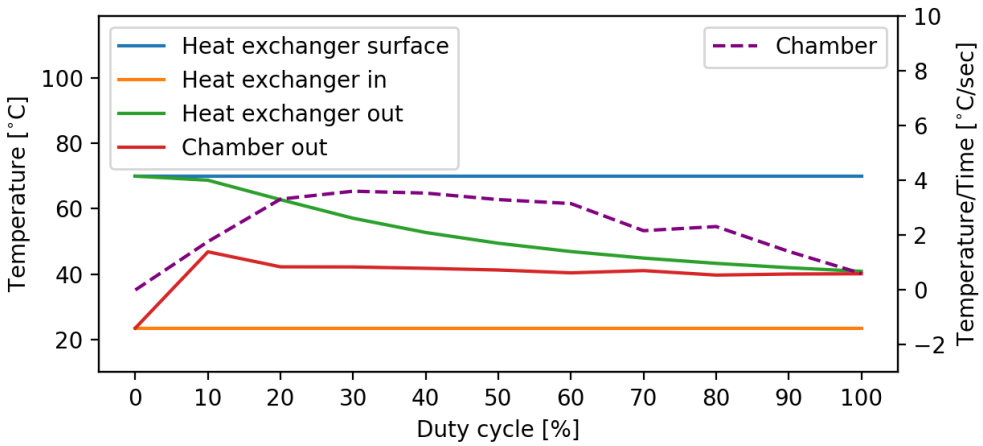
$$\mathrm{Nu} = 7.54 + \frac{0.03(d_h/L)\mathrm{RePr}}{1 + 0.016[(D_h/L)\mathrm{RePr}]^{2/3}}, \tag{3.7}$$

where $h = k\mathrm{Nu}{D_h}^{-1}$, and k is the thermal conductivity of the air. Using Eqn. 3.5, $T_{\mathrm{hx,out}}$ can be calculated, where $T_{\mathrm{hx,in}}$ is constant at an indoor temperature. Once $T_{\mathrm{hx,out}}$ is found, the chamber temperature change rate dT_{ch}/dt, at a certain mass flow rate (fan speed), can be obtained from Eqn. 3.4. Here, $T_{\mathrm{ch,in}} = T_{\mathrm{hx,out}}$ and $T_{\mathrm{ch,out}}$ is measured experimentally. The relation between dT_{ch}/dt and the fan speed, can be established by repeating the process at various fan speeds. The result is shown in Fig. 3.9a for cooling, and in Fig. 3.9b for heating. As shown in the figure, in the case of cooling, dT_{ch}/dt increases as the duty cycle increases, however with duty cycle greater than 30%, dT_{ch}/dt starts to decrease as the difference between $T_{\mathrm{hx,out}}$ and $T_{\mathrm{ch,out}}$ decreases. Similar behavior is observed for heating. The magnitude of dT_{ch}/dt increases until the duty cycle is 30%, then decreases with duty cycle greater than 30%. dT_{ch}/dt is also found experimentally, and the observation is given in Fig. 3.10. The chamber temperature and its first derivative (dT_{ch}/dt) is measured by time, with the fan speed fixed. In the case of cooling, dT_{ch}/dt increases as the fan speed increases, until the duty cycle is $40 - 60\%$, then decreases with the higher duty cycle. In the case of heating, dT_{ch}/dt increases until the duty cycle increases to 100%. The chamber temperature behaves as expected from the heat transfer model, with some degree of disagreement in values given in Fig. 3.10. A linear plant is designed based

on the observation, and is shown in Fig. 3.11. Considering that dT_{ch}/dt does not show big difference when the duty cycle is $40 - 60\%$ in the case of cooling, and $40 - 80\%$ when heating, the maximum duty cycle for both cases are set to 40%. Performance of the control with the plant designed, is shown in Fig. 3.12. Even as the plant is capable of adjusting the fan speed corresponding to the error, it still results in the oscillating chamber temperature. Therefore, the proportional control is applied, and it is shown in Fig. 3.12 that the oscillation reduces and settles with smaller proportional gain. However, it still is not capable of reducing the steady state error, thus the PID controller is applied to overcome the problem.

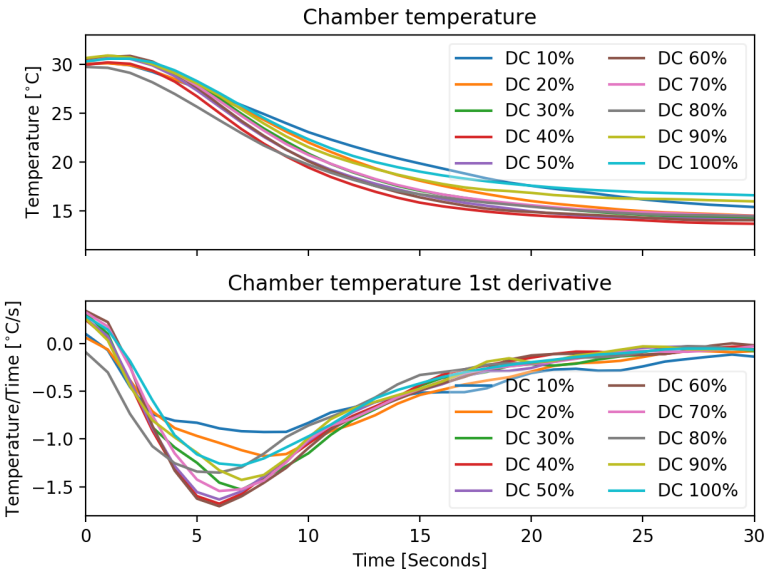


(a) Cooling fan

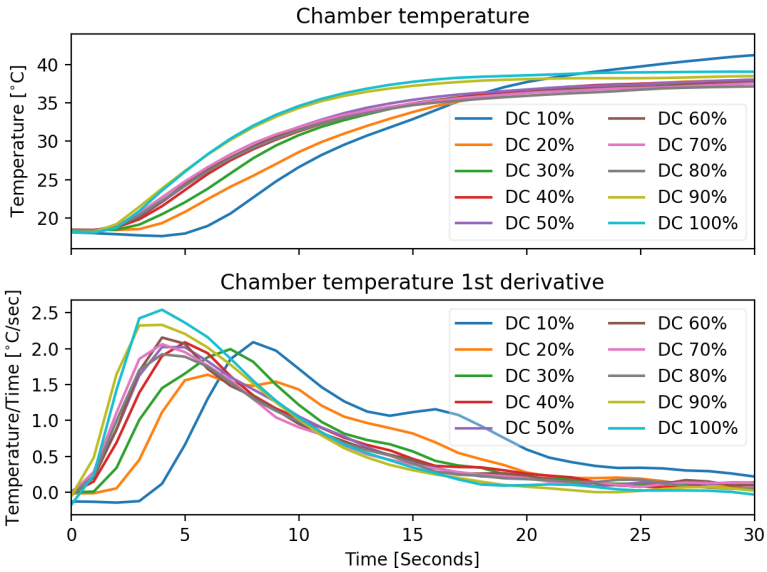


(b) Heating fan

Figure 3.9: A relation between the mass flow rate in terms of fan speed, and the chamber temperature change rate. For the cooling fan, $|dT_{\mathrm{ch}}/dt|$ increases until the duty cycle increases up to 30%, and decreases with the higher duty cycle. The same result is observed for the heating as well.



(a) Cooling fan



(b) Heating fan

Figure 3.10: dT_{ch}/dt corresponding to various fan speeds. The chamber temperature responds as expected from the theory. When cooling, dT_{ch}/dt increases until the duty cycle is 50%. When heating, dT_{ch}/dt increases until the duty cycle is 100%.

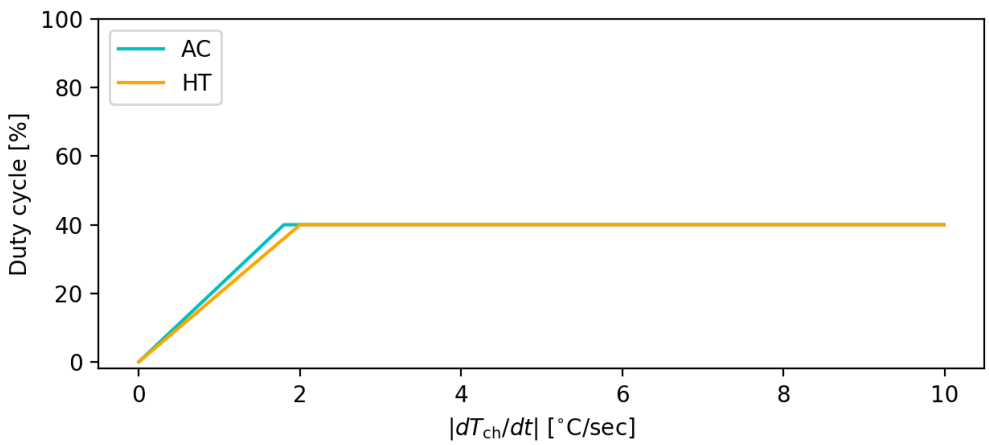


Figure 3.11: The plant is capable of adjusting the fan speed corresponding to an error. The plant is designed linearly, and based on the experimental observation, the maximum cooling and heating fan speed is set to 40% duty cycle.

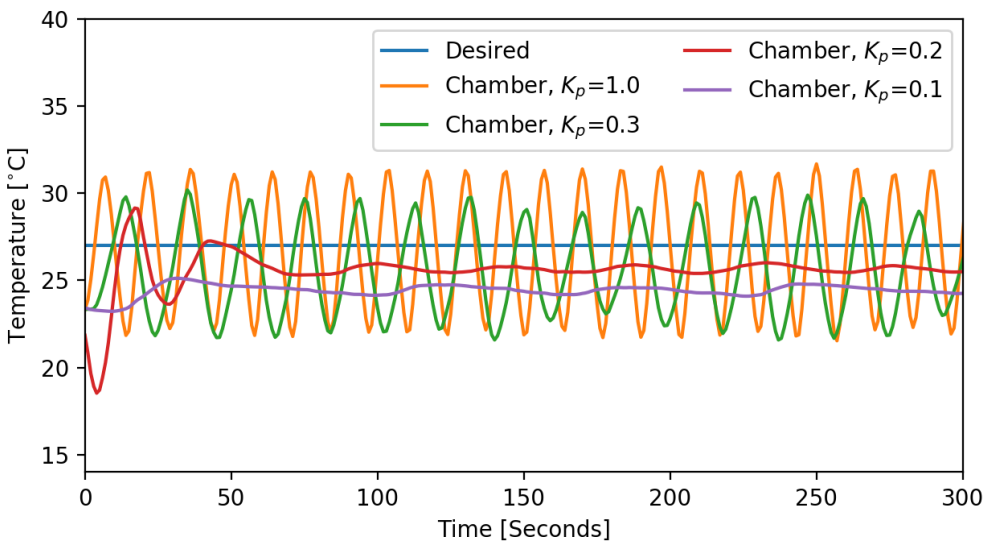


Figure 3.12: The chamber temperature can be settled to the desired signal with the proportional control, however it is not capable of eliminating the steady state error.

3.1.4 PID control system

Earlier, it is shown that the proportional control is not adequate to achieve desired temperature response. To obtain reasonable control on the chamber temperature, PID control method is applied. The PID controller implemented, simply solves the following equation,

$$u(t) = K_{\mathrm{P}} e(t) + K_{\mathrm{i}} \int_{t-\Delta t}^{t} e(\tau) d\tau + K_{\mathrm{d}} \frac{de(t)}{dt}, \tag{3.8}$$

where $u(t)$ is a new input to the control plant, $e(t)$ is the temperature error, and Δt is a time constant. The PID gains K_{p}, K_{i} and K_{d} are tuned experimentally, so that the chamber temperature can precisely track the house model temperature. The chamber temperature is the average temperature from a set of T-type thermocouples, placed at strategic locations within the chamber, as shown in Fig. 3.1. The performance of a properly tuned PID controller, with $K_{\mathrm{p}} = 0.1, K_{\mathrm{i}} = 0.004, K_{\mathrm{d}} = 0.7$, is shown in Fig. 3.13. Then it is also tested whether the PID controller with tuned gains, is capable of implementing the temperature profile from the house model, and the result is shown in Fig. 3.14. From the figure, it is shown that the PID controller with tuned gains tracks the house model reasonably.

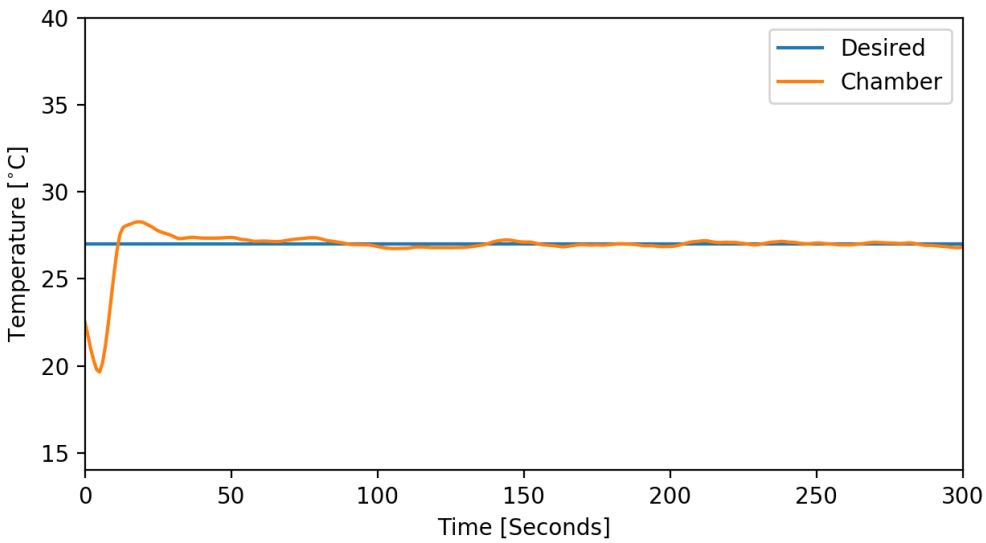


Figure 3.13: With gains of $K_\mathrm{p} = 0.1$, $K_\mathrm{i} = 0.004$, $K_\mathrm{d} = 0.7$, the PID control system tracks the constant desired temperature reasonably.

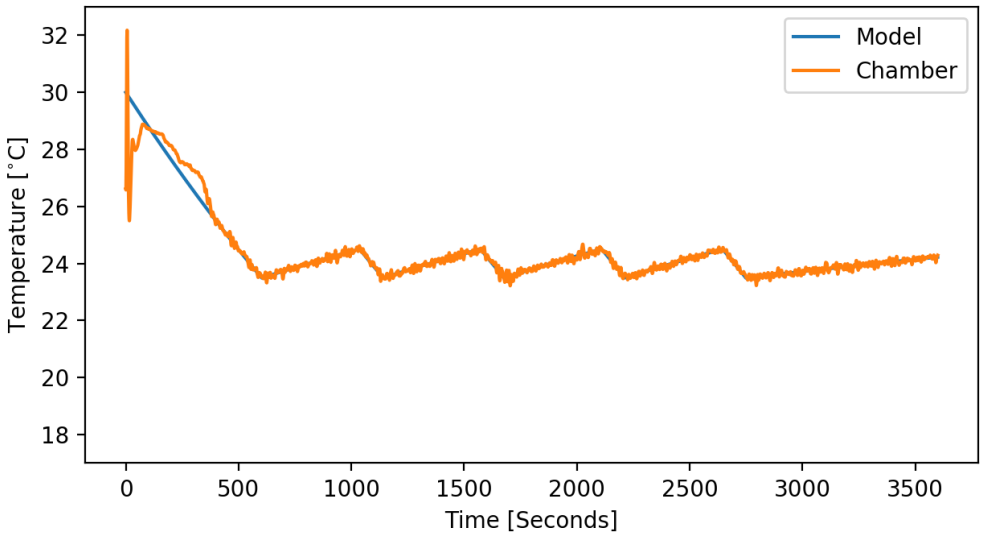
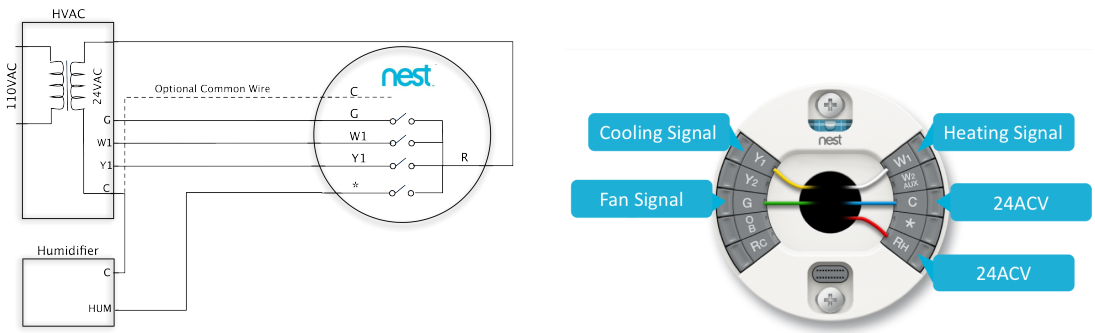


Figure 3.14: With gains of $K_\mathrm{p} = 0.1$, $K_\mathrm{i} = 0.004$, $K_\mathrm{d} = 0.7$, the PID control system is also capable of implementing the house model temperature inside the chamber.

3.2 Nest settings

A 3rd generation Nest Learning Thermostat is selected as a physical commercial thermostat. An electrical circuit of the HVAC system with a heat pump in an actual residential space, and the wiring of the Nest thermostat to the HVAC, is shown in Fig. 3.15. The 24V AC power supply from the transformer, runs through terminals R and C, which are used to power the thermostat. Terminal W1, Y1 and G are used for sending signals to the heating element relay, to the compressor relay and the fan relay of the heat pump system, respectively. In the present experiment, the Nest thermostat terminals are not connected to an actual HVAC, but the Nest thermostat still generates HVAC control signals in response to its environment. These signals are obtainable through the voltage measurement in relevant terminals, or through the utilization of the Nest API. The latter method is used in this work, with details covered in the following section. By detecting connections in its terminals, the Nest thermostat knows what kind of HVAC system it is controlling. In order to let the Nest thermostat know that it is associating with a heat pump system, open-ended wires are connected to terminal W1, Y1 and G, as shown in Fig. 3.15 [19]-[20].



(a) (b)

Figure 3.15: The Nest thermostat wiring for an actual HVAC system with a heat pump and a fan. Its response is also reachable through the Nest API.

Chapter 4

Nest Synthesis to Simulation Framework

Given this physical environment for the Nest thermostat, it is now necessary to integrate the response of the Nest thermostat to the overall framework. The Nest API is the key tool utilized in the integration process. Steps taken for this purpose are described in the present chapter.

4.1 Nest API

The Nest thermostat supports the API development through the REpresentational State Transfer (REST) network HyperText Transfer Protocol (HTTP), which allows a user to read or write on the Nest thermostat without physically contacting it. Through REST HTTP, a client (the thermostat itself, or a third-party agent such as an aggregator) can make requests to specific server URLs. The data can be manipulated on the server itself, or transfered back to the client [21]. Nest provides

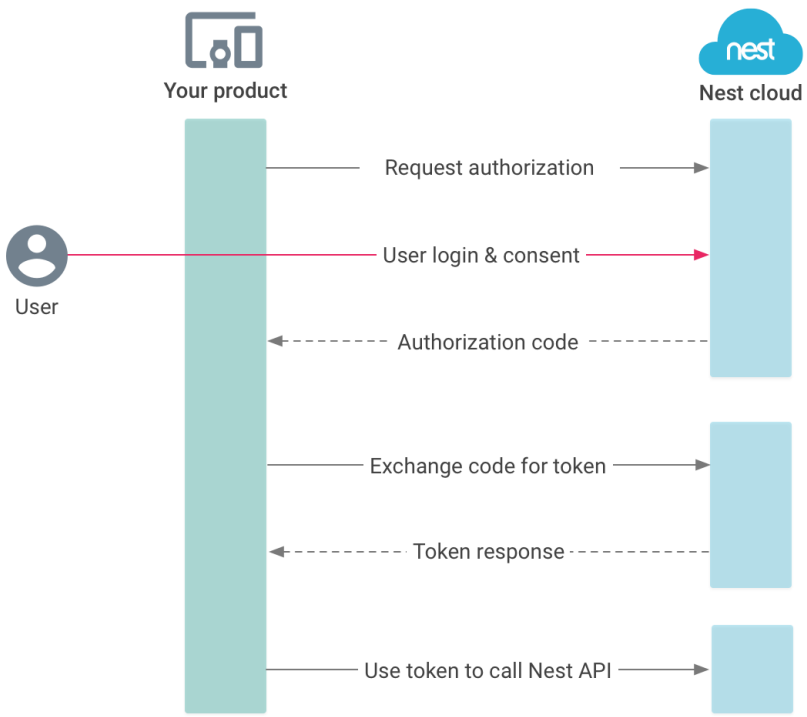


Figure 4.1: Before making requests to the data, an authorization must be made. An access token yielded from the authorization enables requests to the Nest API.

relevant URLs to API developers. While HTTP requests can be made via various platforms, in the present work, HTTP requests have been made in Python through the 'requests' library [21]. A client is created in the Nest Developers website, that is assigned with a unique client ID, a client secret and an authorization URL. Before access to private data of the Nest thermostat is available through the API, an access token must be obtained [22]. The Nest API uses the OAuth 2.0 protocol for the authorization process. With the client ID, the client secret ID and the authorization code, an access token is granted by making requests. The authorization code is obtained by getting into the authorization URL, after providing the user consent. An example of requesting an access token in a Python script is given in List. 4.1. The request made with the client ID, client secret and authorization code, returns in an access token as shown in Lines 14-15. Once the access token is obtained, it is

possible to make requests to the Nest API, then access private data, such as setpoint or HVAC state, of a specific Nest thermostat as illustrated in Fig. 4.1. An example of reading data from the Nest thermostat, in a Python script is given in List. 4.2. Data are pulled in a JavaScript Object Notation (JSON) object, through the 'GET' method of the REST API. An example of the private data in JSON object, read from the Nest thermostat, is shown in List. 4.3. Here, a device ID and a structure ID of the Nest thermostat can be obtained, that are required to write on the Nest thermostat. Device ID indicates the corresponding thermostat, and the structure ID indicates the space where the device is installed. With the ID's obtained, data can be pushed in through the Nest thermostat, using the 'PUT' command of the REST API. An example of writing the setpoint and the away status on the thermostat, in a Python script, is shown in List. 4.4. The error handling can be done by checking the request status code, as shown in Lines 14-15 of List. 4.2, and 15-16 of List. 4.4. The status code returns a value of 200 if the request is successfully made; the value other than 200 indicates that there has been an error during the execution.

```
1  import requests
2  # own info
3  client_id = "your client id"
4  client_secret = "your client secret"
5  auth_url = "your authorization url"
6  # go to auth_url then get auth_code
7  auth_code = "your authorization code"
8  # requesting access token (response is access token)
9  url = "https://api.home.nest.com/oauth2/access_token"
10 payload = "code=%(AUTH_CODE)s&client_id=%(CLIENT_ID)s&client_secret=%(
       CLIENT_SECRET)s&grant_type=authorization_code" %{'AUTH_CODE':
       auth_code,'CLIENT_ID':client_id,'CLIENT_SECRET':client_secret}
11 headers = {'content-type': 'application/x-www-form-urlencoded'}
12 response = requests.request("POST", url, data=payload, headers=headers)
13
14 >>>print(response.text)
15 >>>{"access_token":"your access token","expires_in":315360000}
```

Listing 4.1: The process of making a request to obtain an access token in Python script [23]. The client ID, client secret and authorization code is required. The authorization code is obtained from the authorization URL. With the access token granted, access to private data of the Nest thermostat is available.

```
1  import json
2  import requests
3
4  url = "https://developer-api.nest.com/"
5  token = "your access token"
6  headers = {'Authorization': 'Bearer {0}'.format(token), 'content-Type':
       'application/json'}
7  initial_response = requests.get(url, headers = headers, allow_redirects
       = False)
8  testdata = initial_response.text
9  testdata = json.loads(testdata) # into dic
10
11 device = testdata['devices']['thermostats']['your device id']
12 structure = testdata['structures']['your structure id']
13
14 >>> print('initial_response.status_code')
15 200
16 >>> device['hvac_mode'],device['hvac_state'],structure['away']
17 'eco','off','away'
```

Listing 4.2: Reading private data from the Nest thermostat in a Python script [24]. The device ID and the structure ID, that are required to write on the thermostat, can be obtained through this process. The status code of 200 indicates that the process was successful.

```
{
    "devices": {
        "thermostats": {
            "your device id": {
                "ambient_temperature_c": 21.0,
                "device_id": "your device id",
                "eco_temperature_high_c": 30.0,
                "eco_temperature_low_c": 16.0,
                "hvac_mode": "eco",
                "hvac_state": "off",
                "is_online": true,
                "previous_hvac_mode": "cool",
                "structure_id": "your device id",
                "target_temperature_c": 24.5,
                "target_temperature_high_c": 26.0,
                "target_temperature_low_c": 23.5,
                "temperature_scale": "C"
            }
        }
    },
    "structures": {
        "your structure id": {
            "away": "away",
            "postal_code": "87106",
            "structure_id": "your structure id"
            }
        }
    }
}
```

Listing 4.3: Private data of the Nest thermostat, read in JSON object.

```
1  import json
2  import requests
3
4  url = "https://developer-api.nest.com"
5  device_id = "your device id"
6  structure_id = "your structure id"
7  token = "your access token"
8  setpoint = 20.0
9  away ='home'
10
11 payload = "{\"devices\":{\"thermostats\":{\"%(device_id)s\":{\"
       target_temperature_c\": %(setpoint)f}}},\"structures\":{\"%(
       structure_id)s\":{\"away\": \"%(away)s\"}}}" %{'device_id':device_id
       ,'structure_id':structure_id, 'away':away, 'setpoint':setpoint}
12 headers = {'Authorization': 'Bearer {0}'.format(token), 'content-Type':
        'application/json'}
13 initial_response = requests.put(url, headers = headers, data=payload,
       allow_redirects = False)
14
15 >>> print(initial_response.status_code)
16 200
17 >>> print(initial_response.text)
18 {"devices":{"thermostats":{"your device id":{"target_temperature_c":
       20.000000}}},"structures":{"your structure id":{"away": "home"}}}
```

Listing 4.4: Writing data on the Nest thermostat in a Python script [25]. With the example given, the setpoint of the device and the away status of the structure, will be set to 20°C and to 'home', respectively. The device ID and the structure ID that are required to write on the thermostat, are obtainable from the reading process. The status code of 200 indicates that the process was successful.

4.2 Nest Data

Nest thermostat data associated, are tabulated in Table. 4.1. The 'hvac_mode' and the 'hvac_state' correspond to Π and Λ, from Eqn. 2.4 and Eqn's. 2.5-2.6, respectively. A total of five HVAC modes, are offered in the Nest thermostat. When the HVAC mode is set to 'Heat' or 'Cool', the Nest thermostat utilizes a single setpoint (labeled as 'target_temperature_c', value in °C), activating only heating or cooling, respectively. In 'Heat-Cool' or 'Eco' mode, the setpoints for both heating and cooling are utilitzed, therefore the HVAC system can either heat or cool the space. Users may want to set the setpoints to the less-comfort oriented values, in 'Eco' mode. Setpoints in the 'Eco' mode are non-writable through the API, but are changeable directly at the physical thermostat or the mobile application provided by the Nest. Note that the Nest thermostat does not support the user set deadband, but only the setpoints. The three HVAC states indicate whether the HVAC compressor is on (heating / cooling) or off. Also note here, that the HVAC state cannot be designated by the user, since it is driven by logics inside the Nest thermostat. The 'ambient_temperature_c' is the space temperature measured by the thermostat. The Nest thermostat senses the human occupancy via near-field activity, far-field activity and ambient light sensor, then sets away status of the structure, where it is installed, to 'Home' when occupied, or 'Away' otherwise. However the away status can be set manually by the user as well. In the present work, the thermostat is not allowed to make decision on the occupancy, but the occupancy schedule from the load generator will be pushed to the thermostat through the API. When the away status has turned to 'Away' from 'Home', the Nest thermostat automatically switches HVAC mode to 'Eco' regardless of the previous mode. However, the user can deactivate this automatic mode switching, and it is deactivated in the present study. By the way, 'Eco' mode is independent of the away status, meaning that the user can set the HVAC mode to non-'Eco' mode, even when the space is unoccupied, if it is desired.

Device Thermostat	**Returns/Range**	**Access**	**Note**
hvac_mode	‘heat’,‘cool’,‘heat-cool’,‘eco’,‘off’	read/write	
hvac_state	‘heating’,‘cooling’,‘off’	read-only	
ambient_temperature_c	number	read-only	
target_temperature_c	number/9-32	read/write	hvac_mode=‘heat’ or ‘cool’
target_temperature_low_c /target_temperature_high_c	number/9-32	read/write	hvac_mode=‘heat-cool’ min gap between low/high = 1.5
eco_temperature_low_c /eco_temperature_high_c	number/4-21/24-32	read-only	hvac_mode=‘eco’
Structure	**Returns/Range**	**Access**	**Note**
away	‘home’,‘away’	read/write	

Table 4.1: Data utilized in the integration. Reading or writing the data is available through the Nest API.

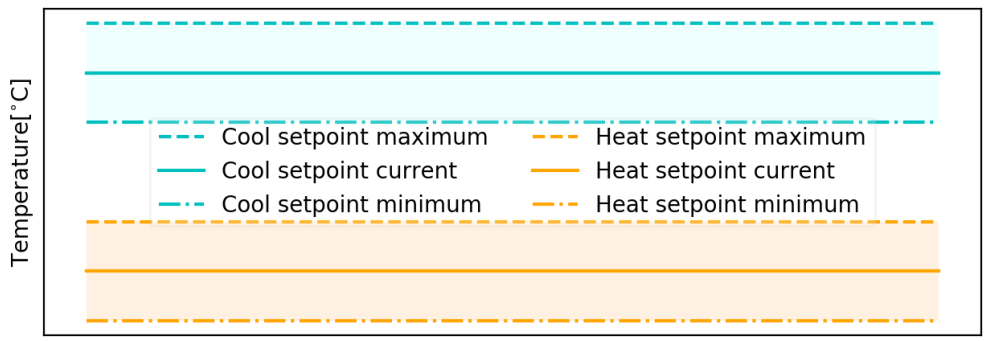


Figure 4.2: The aggregated load control is applicable to the Nest thermostat, by modifying the setpoints based on the probability of swithing, which is broadcasted by the aggregator. To do so, bounds to the setpoints need to be given.

4.3 Stochastic switching model for the Nest thermostat

The Nest thermostat has its own features, that are different from the thermostat simulated in the framework. Therefore it is neccessary to model a stochastic switching logic suitable for the Nest thermostat. As mentioned earlier, a direct HVAC state designation is not obtainable. Instead, the setpoint can be modified, to achieve the HVAC state switching. First, a random number $\mathcal{I}$ can be drawn outside of the thermostat – since an individual developer is not yet allowed to plant any custom algorithms within the Nest thermostat. If the HVAC state is decided to be switched, i.e., if $\mathcal{I} < |\mathcal{F}|$, then the new setpoint is determined, following

$$T_{\text{set,new}} = \begin{cases} T_{\text{set,min}}, & \text{if } \mathcal{F} > 0 \\ T_{\text{set,max}}, & \text{if } \mathcal{F} < 0, \end{cases} \tag{4.1}$$

in the case of cooling. $T_{\text{set,min}}$ and $T_{\text{set,max}}$ are the arbitrary marginal setpoints that

bound the space temperature, functioning similar to the deadband (Fig. 4.2). In the case of heating, the new setpoint will be selected by

$$T_{\text{set,new}} = \begin{cases} T_{\text{set,max}}, & \text{if } \mathcal{F} > 0 \\ T_{\text{set,min}}, & \text{if } \mathcal{F} < 0. \end{cases} \tag{4.2}$$

Alternatively, the setpoint can be dynamically adjusted, instead of using $\mathcal{I}$ draw process. Considering that $\mathcal{F}$ indicates the probability of switching, and also assuming that the larger change in the setpoint would result in the higher probability of state switching, the increment of the setpoint can be determined as

$$\Delta T_{\text{set}} = \frac{|T_{\text{reference}} - T_{\text{set,current}}|}{|\mathcal{F}|_{\max} - |\mathcal{F}|_{\min}} \times \mathcal{F}, \tag{4.3}$$

where $-1 \leq \mathcal{F} \leq 1$, $T_{\text{reference}}$ is the marginal reference value of the setpoint, and $T_{\text{set,current}}$ is the current setpoint before responding to the aggregated load control. Following this idea, in the case of cooling, a new setpoint can be determined as

$$T_{\text{set,new}} = \begin{cases} T_{\text{set,current}} - (T_{\text{set,current}} - T_{\text{set,min}}) \times \mathcal{F}, & \text{if } \mathcal{F} > 0 \\ T_{\text{set,current}} - (T_{\text{set,max}} - T_{\text{set,current}}) \times \mathcal{F}, & \text{if } \mathcal{F} < 0. \end{cases} \tag{4.4}$$

If the HVAC is desired to be turned on or off, the setpoint will be set to the lower or higher value, respectively, following Eqn. 4.4. In the case of heating, the new setpoint will be calculated by

$$T_{\text{set,new}} = \begin{cases} T_{\text{set,current}} + (T_{\text{set,upper}} - T_{\text{set,current}}) \times \mathcal{F}, & \text{if } \mathcal{F} > 0 \\ T_{\text{set,current}} + (T_{\text{set,current}} - T_{\text{set,lower}}) \times \mathcal{F}, & \text{if } \mathcal{F} < 0. \end{cases} \tag{4.5}$$

In the meanwhile, the information whether the Nest thermostat is switchable or not, is defined manually outside of the thermostat. The switchability of the thermostat is certainly considered inside the Nest thermostat, however it is not always available. For example, if the user moves the setpoint and the HVAC switching is desired, but it has not passed the deadtime since the last HVAC switching, then the thermostat indicates on its screen, that the HVAC operation has been delayed for some amount of time. In other words, the switchability is unavailable, unless some action (e.g. changing the setpoint) is done. Even worse, this information is not obtainable through the API. Therefore, in the current study, the time between the state changes is manually counted, with the deadtime set to five minutes, as in the simulated thermostat.

4.4 Integration

Having established that data can be pushed in to or pulled from the Nest thermostat using the Nest API, the integration of the Nest thermostat to the simulation framework, can now proceed. In this process, the thermostat logic in the load simulation for one of the house models, is replaced with the physical Nest thermostat. Note that 200 meters (house models) are used in the simulation, and the thermostat of Meter 23 in particular, is substituted with the Nest thermostat. A schematic illustration is shown in Fig. 4.3, and the sequence of the integration is:

1. An initial temperature of the house model and the initial thermostat status are sent to the environmental chamber, and to the Nest thermostat, respectively. Then the initialization is done accordingly (Table. 4.2).

2. The daily occupancy schedule is drawn from the statistical information in the load generator.

3. The sepoint / deadband, and the occupancy status, are sent to Meter 23, and is ready to be written in the Nest thermostat (Table.4.3).

4. Meter 23 receives a stochastic switch request ($\mathcal{F}$) from the aggregator, and calculates the new setpoint / deadband accordingly, then writes the information on the Nest thermostat (Table. 4.6).

5. The response (HVAC mode / HVAC state) is read from the Nest thermostat, and sent to the thermal house model of Meter 23, then the space temperature at the next time step is calculatd.

6. Chamber tracks the temperature of the house model through the PID controller.

7. Meanwhile, HVAC state of the Nest thermostat and its switchability is reported to the aggregator (Table. 4.4).

8. The aggregator at the same time, receives the aggregated load, and calculates its control error. It is also reported with the HVAC information (HVAC state / HVAC switchability) from 200 house models (Table. 4.5), then creates and broadcasts the stochastic switch request ($\mathcal{F}$) to the thermostats of house models (Table. 4.6).

9. Repeat steps 3 to 8.

To simulate the physical separation of the load generator (200 houses), the aggregator, and the HVAC of Meter 23, each model is installed on individual Raspberry Pi's [26] as shown in Fig. 4.4. Communication between models are done via Secure Copy Protocol (SCP), which supports file transfers between remote hosts on a network [27]. Examples of data transferred are tabulated in Table. 4.2- 4.6. The load generator and the aggregator are modeled in Fortran scripts, and Meter 23 is written in a Python script. It is possible to execute a SCP command line inside the Fortran or Python script, however it requires a password of the remote host, every time it is executed. In order to handle this problem, an Expect script is executed inside the Fortran script. Expect is a program to automate interactions with programs that expose a text terminal interface [28], and it automatically provides a password when the script is executed. An example is given in List. 4.5. An example of secure copying files in the Python script using libraries 'paramiko' and 'scp' is shown in List. 4.6 as well. The outdoor temperature is obtained by the weather API provided by the Nest, and its usage is shown in List. 4.7. To run the simulation in real-time, the time trigger implemented by Ayon [26], which is given in List. 4.8, is utilized. The time trigger independently runs outside the simulation, outputing file 'timehascome.csv', each time the seconds digit of the computer clock changes. Each Raspberry Pi's has its own trigger running in the background, and will start its loop when the file

Meter	Space temperature	Setpoint	R ins	A HX	T mass
23	23.3929138	21.6705360	2.18746376	310.662842	838285.125

Table 4.2: Randomly generated initial values for Meter 23, is passed to Meter 23 from the load generator, in a file 'meter23_init.csv'.

Occupancy	Setpoint	Lower deadband	Upper deadband
F	21.2895031	16.0000000	26.0000000

Table 4.3: The house occupancy, setpoint and the deadband are transferred from the load generator to Meter 23, in a file 'meter23_occup.csv'.

'timehascome.csv' is found. The file, then will be deleted, after being found in each Raspberry Pi's.

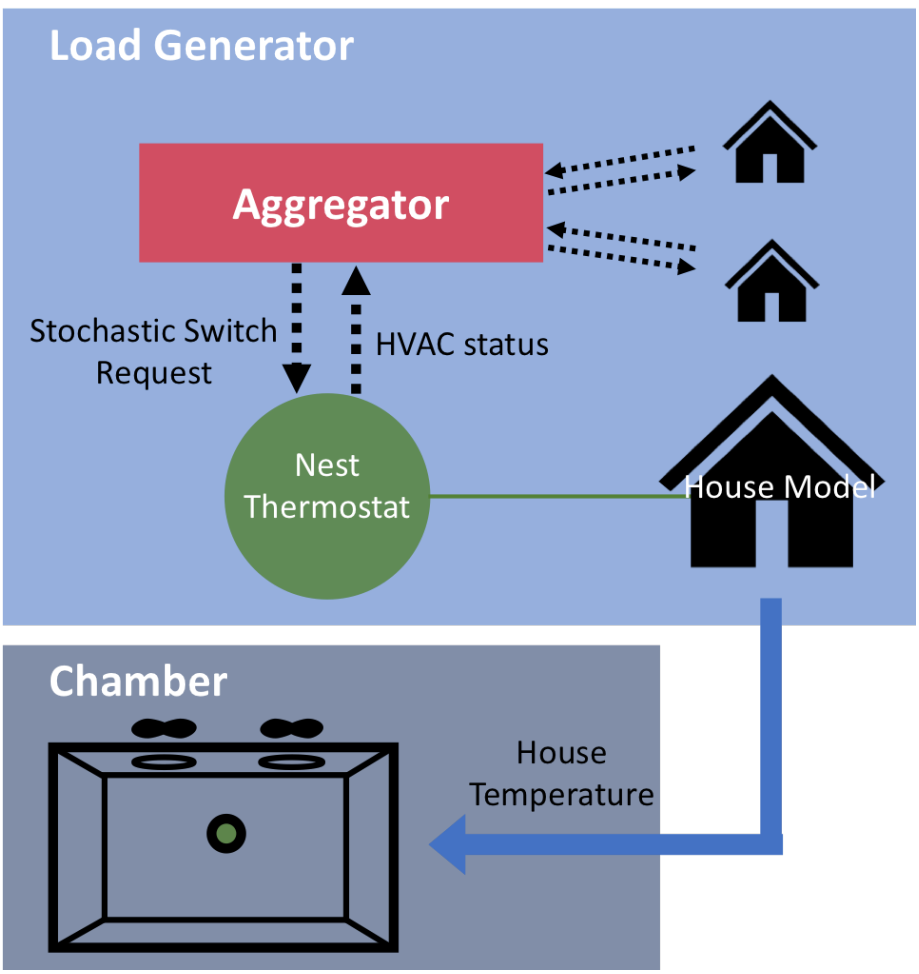


Figure 4.3: In the load control framework, the simulated thermostat logic is replaced with the physical Nest thermostat. The Nest thermostat receives or sends data through the Nest API. The model temperature is implemented in a physical chamber via the PID controller, providing the Nest thermostat an environment to yield an actual response.

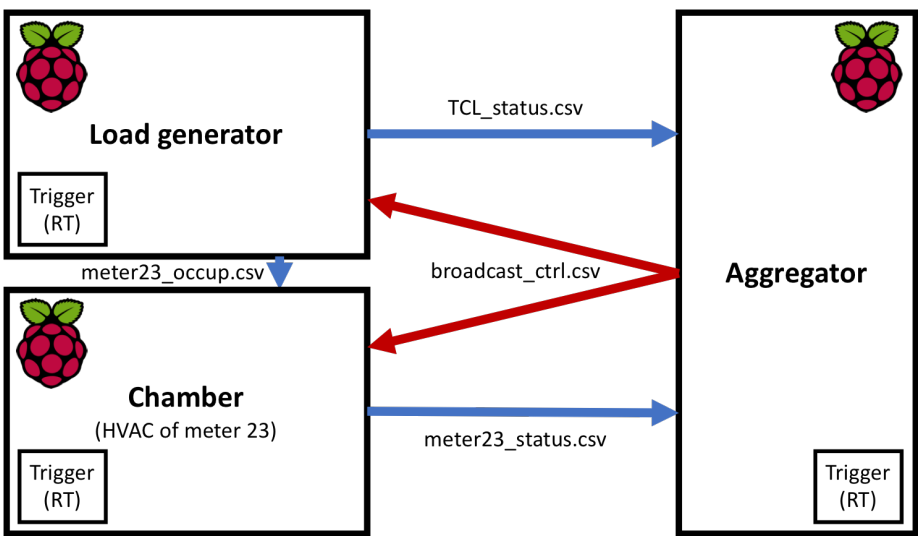


Figure 4.4: The load generator, the aggregator and the HVAC system of Meter 23, are installed in individual Raspberry Pi's. Pi's communicate by transferring files via SCP. The figure demonstrates the flow of files in real time. Each component is triggered every second.

Meter	**AC on**	**AC switchability**
23	1	0

Table 4.4: HVAC state and HVAC switchability, are transferred from the Nest thermostat of Meter 23 to the aggregator, in a file 'meter23_status.csv'.

Meter	**AC on**	**AC switchability**
1	0	0
2	1	0
⋮	⋮	⋮
199	0	1
200	0	0
Total power		
191265.812		

Table 4.5: HVAC state, HVAC switchability of 200 house models (except Meter 23), and the total aggregated load, are transferred from the load generator to the aggregator, in a file 'TCL_status.csv'. HVAC state and switchability are data reported from smart thermostats in the model to the aggregator, and the total load is reported from the substation to the aggregator.

$\mathcal{F}$
0.0286

Table 4.6: Stochastic switch request broadcasted by the aggregator, in a file 'broadcast_ctrl.csv'.

```
#!/usr/bin/expect
spawn scp ./tcl_status.csv username@server:target directory
set pass "your password"
expect {
password: {send "$pass\r"; exp_continue}
          }
```

Listing 4.5: Files can be secure copied inside a Fortran script by executing expect script.

```
import paramiko
import scp

class SCP:
    def __init__(self):
        self.my_ssh=paramiko.SSHClient()
        self.my_ssh.load_system_host_keys()
        self.my_ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy
            ())
        self.my_ssh.connect('your server',port=22,username='your
            username',password='your password')
        self.my_scp=scp.SCPClient(self.my_ssh.get_transport())
    def SCPFile(self):
        status=[[23,abs(self.IO),self.switchable]]
        np.savetxt("meter23_status.csv",status,delimiter=",",fmt=['%01d
            ','%01d','%01d'])
        self.my_scp.put('meter23_status.csv',recursive=True,remote_path
            ='~/aggregator')

>>> copy = SCP()
>>> copy.SCPFile()
```

Listing 4.6: Secure copying files in a Python script using the library ‘paramiko’ and ‘scp’.

```
1  import requests
2  import json
3
4  class Weather:
5      def __init__(self):
6          self.url = "https://home.nest.com/api/0.1/weather/forecast/your
               ZIP Code"
7      def current_temperature(self):
8          initial_response = requests.get(self.url)
9          data = initial_response.text
10         data = json.loads(data)
11         self.To = data['now']['current_temperature']
12         with open('./Tout.csv','w') as handle :
13             json.dump(self.To,handle)
14
15 >>> w=Weather()
16 >>> w.current_temperature()
17 >>> print(w.To)
18  16.1
```

Listing 4.7: The outdoor temperature based on the ZIP code is obatainable through the API provided by the Nest.

```
1  import datetime as dt
2
3  nxttime=0
4  while True:
5      currenttime=dt.datetime.now().strftime('%H:%M:%S')
6      if currenttime == nxttime:
7          pass
8      else:
9          open('timehascome.csv','w')
10         sys.stdout.flush()
11         nxttime=currenttime
```

Listing 4.8: The time trigger triggers the aggregator, the load generator and Meter 23, every second.

Chapter 5

Results and discussion

The Nest thermostat learns about the HVAC system it is controlling, such as how fast the space warms and cools, and how efficient the system is [29]. It also learns the HVAC schedules from intractions with its human users. In the present study, the Nest thermostat is trained in real-time, by inputing the statistical occupancy schedule and the setpoint to the thermostat, and also by implementing the house model response in the chamber over several days. Therefore the Nest thermostat is expected to have learned the HVAC characteristic of the house model. These characteristics, are also expected to be reflected in the thermostat responses, that are observed through the experiments. In this chapter, the Nest thermostat response in various modes, its response to the aggregated load control DR signal, and analysis and modeling of the responses observed, are presented.

5.1 Nest response in various modes

5.1.1 Cool Mode

Fig. 5.1 shows the response of the Nest thermostat in Cool mode, with a setpoint of 24°C. In Cool mode, it is observed that the Nest thermostat activates cooling, when the space temperature is at the setpoint or above. Also, the thermostat deactivates cooling a short time after the temperature goes under the setpoint. Note, for example, that at around 4000 seconds, the cooling is found to be still activated while the

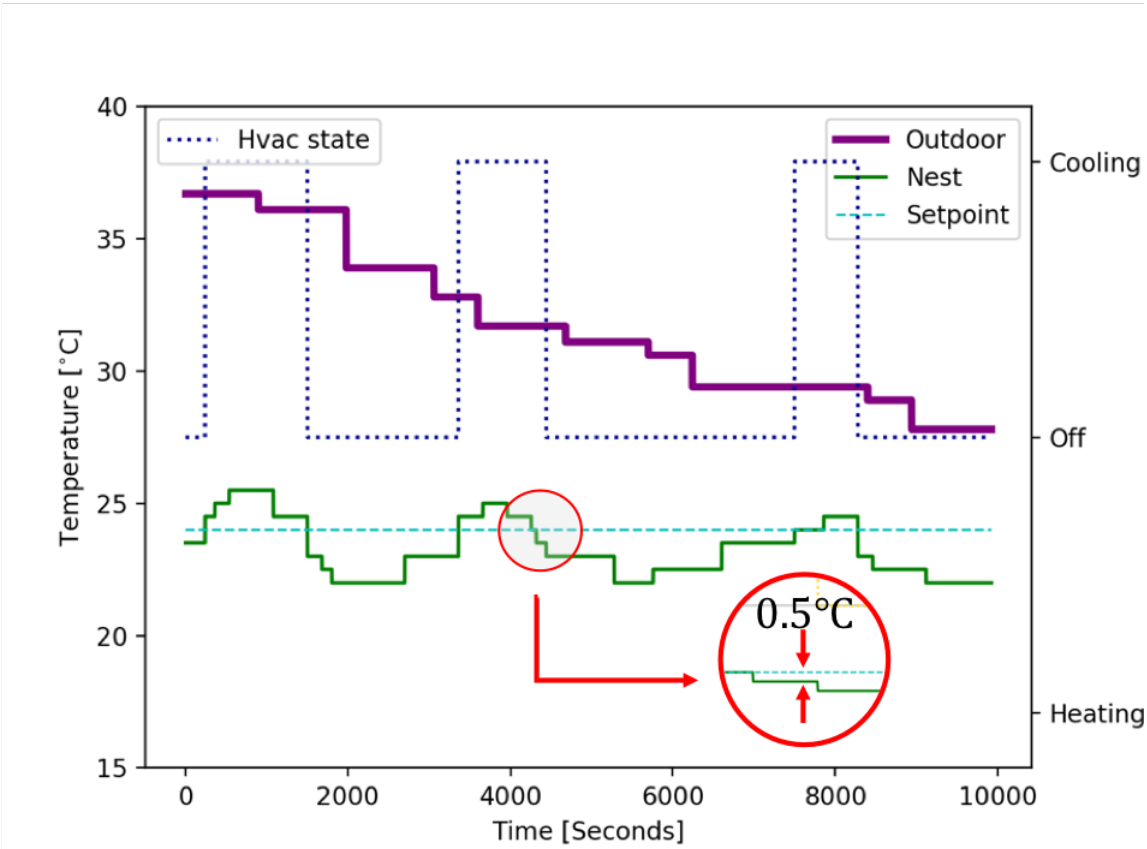


Figure 5.1: Nest thermostat response in Cool mode. Cooling is activated when the space temperature reaches the setpoint. An internally-set lower deadband of the Nest thermostat, is observed to be 0.5°C.

space temperature is measured at 23.5 °C, slightly below the setpoint, but is deactivated shortly afterward. While the Nest thermostat could use logic based on an internal deadband, this does not necessarily correspond to the actual space temperature deadband implemented in the load control framework. However, a deadband-like behavior is also observed in the space temperature response. The connection between the internal Nest deadband and the space temperature deadband is discussed later.

5.1.2 Heat-Cool Mode

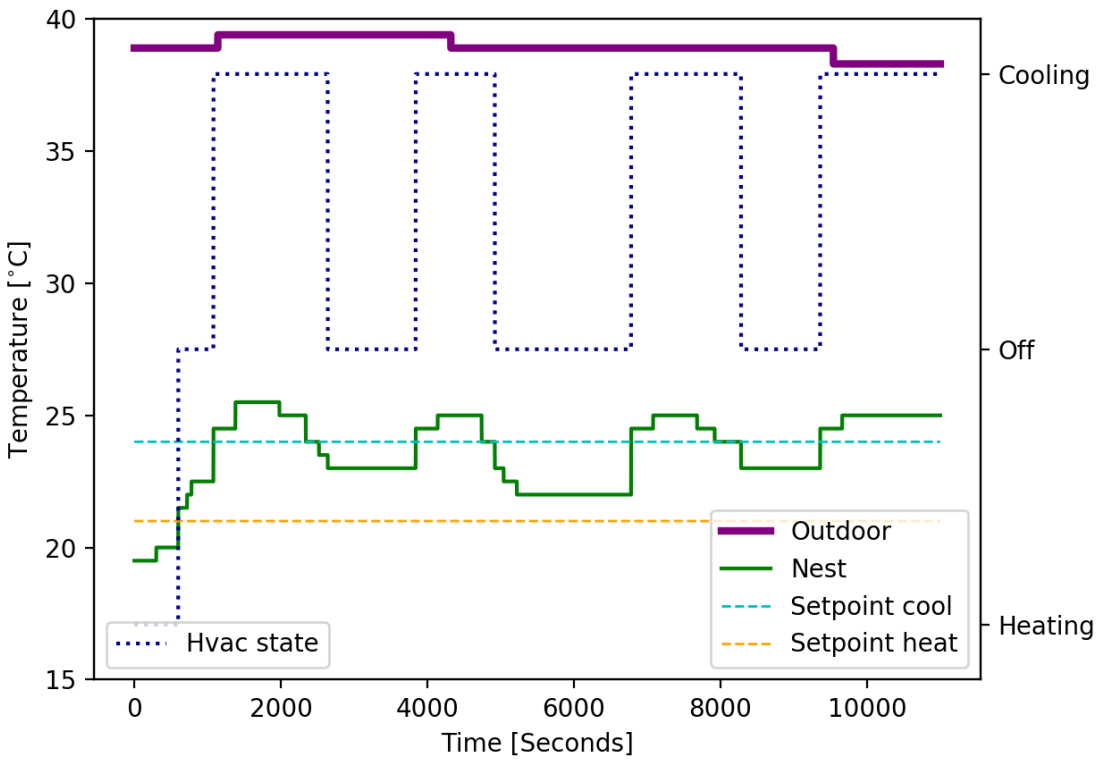


Figure 5.2: The Nest thermostat response in Heat-Cool mode. Both cooling and heating are available in this mode. Cooling is on when the temperature is above the cooling setpoint, heating is on when the temperature is below the heating setpoint. The lower deadband is also observed in this mode.

Figs. 5.2-5.3 show the Nest thermostat response in the Heat-Cool mode. Both cooling and heating are supported on the Heat-Cool mode, with setpoints of 24°C and 21°C taken at each mode. The Nest thermostat controls the HVAC the same as in the Cool mode with the involvement of the lower internally set deadband observed. From Fig. 5.3 at around 9000 seconds, it is found that the cooling is not activated when the space temperature is 24°C. Therefore it appears that a higher internally set upper deadband is used in the Heat-Cool mode. The activation of the heating is observed in Fig. 5.2, when the temperature is below the heating setpoint.

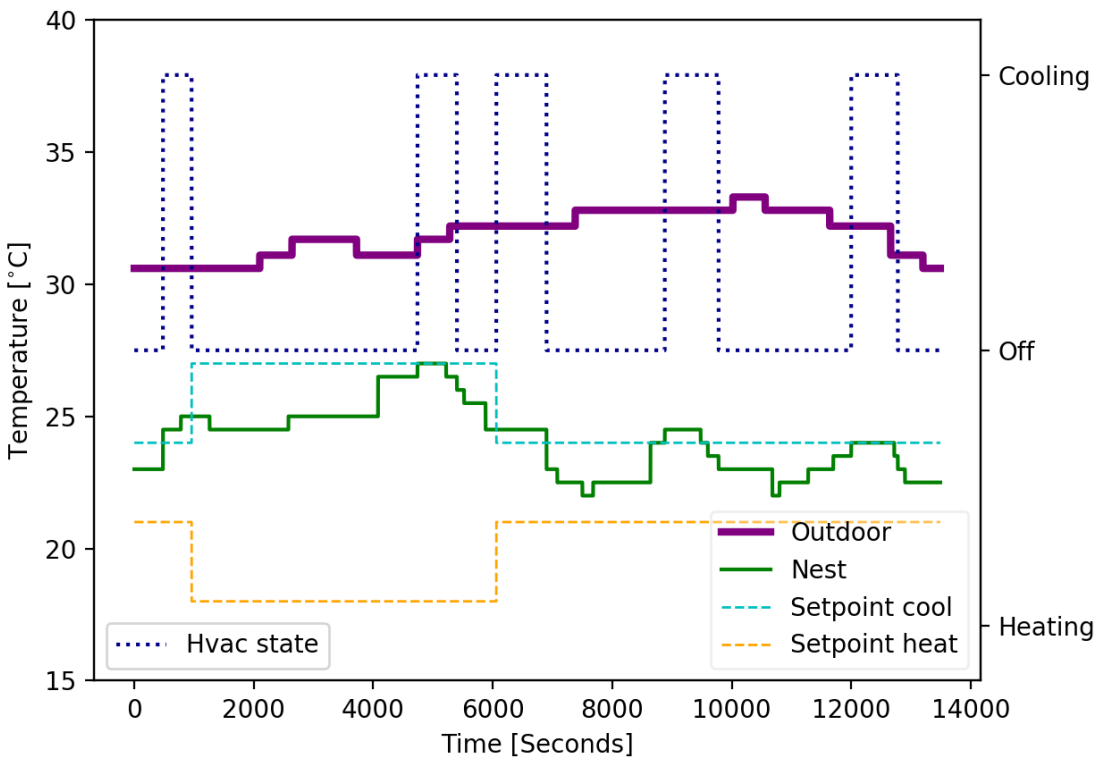


Figure 5.3: The HVAC load can be forcedly increased or decreased by moving the setpoint. Cooling is deactivated when the cooling setpoint is moved to a higher value, at around 1000 seconds, activated when the setpoint moved to a lower value, at around 6000 seconds.

5.1.3 Eco Mode

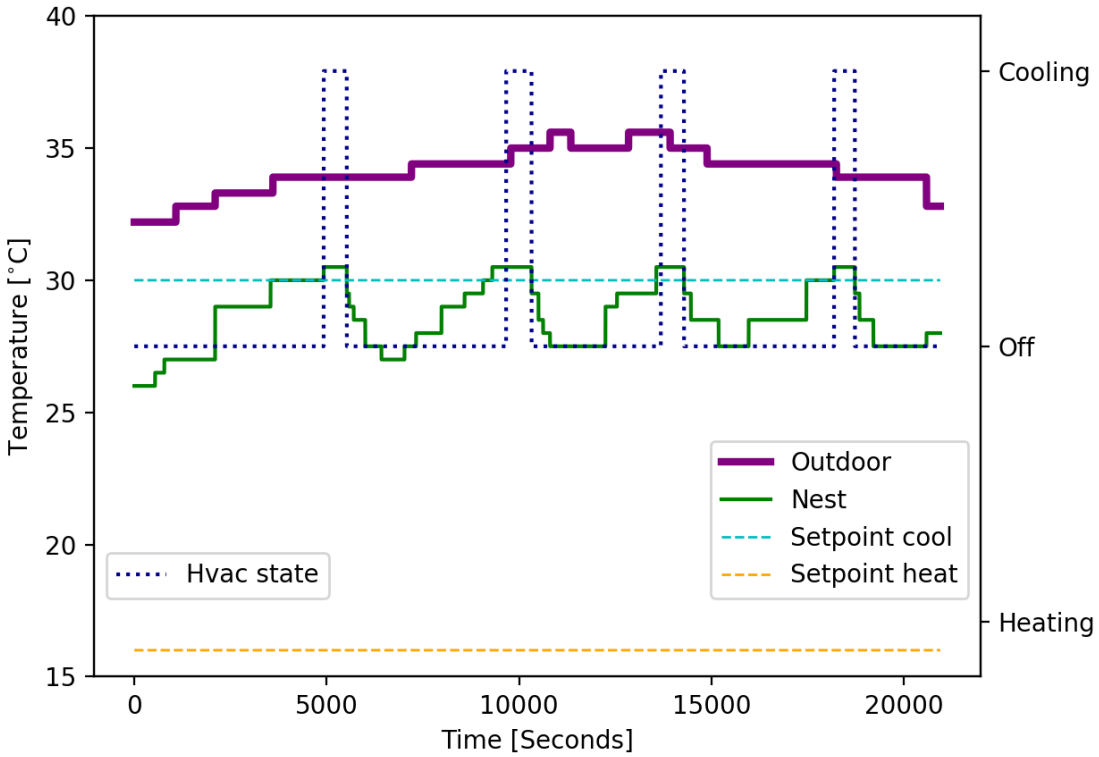


Figure 5.4: Nest thermostat response in Eco mode. Cooling is not always activated immediately when the space temperature is above the cooling setpoint. No lower deadband set for the cooling setpoint in the Eco mode. As soon as the space temperature comes under the cooling setpoint, the Nest thermostat turns off the cooling.

The response of the Nest thermostat in Eco mode is shown in Fig. 5.4. In similar manner to the Heat-Cool mode, the thermostat uses two setpoints for cooling and heating. The utilization of the setpoint in the control algorithm appears to be almost the same as in the Heat-Cool mode. However, it is observed that the HVAC is activated with less strict rules in the Eco mode. In the cooling case, the upper deadband for the cooling setpoint is detected, however it is found that the lower deadband for the cooling setpoint is not utilized. As soon as the space temperature reaches the

cooling setpoint, the Nest thermostat turns off the cooling. The same response is presumed in the heating mode. The cooling activation at each mode, found throughout the experiments, is tabulated in Table. 5.1.

Mode	**Cool**	**Heat-Cool**	**Eco**
Temperature			
$> T_{\text{set}} + 0.5°\text{C}$	◯	◯	◯
$T_{\text{set}} + 0.5°\text{C}$	◯	△	△
T_{set}	◯	△	×
$T_{\text{set}} - 0.5°\text{C}$	△	△	×
$< T_{\text{set}} - 0.5°\text{C}$	×	×	×

Table 5.1: Cooling activation of the Nest thermostat. ◯ : cooling activated, △ : cooling sometimes activated and sometimes not, × : cooling deactivated.

5.2 Nest response to the aggregated DR signal

The Nest thermostat state switching driven by adjusting the setpoint, is proved to be obatainable. In Fig. 5.3, at around 1000 seconds, it is observed that the state is switched to off from cooling on, as soon as the cooling setpoint is moved from 24°C to 27°C. The HVAC switching to cooling on from off, is also shown at 6000 seconds, as the cooling setpoint is set to lower value. The former and latter, each leading to reduction and increase of the HVAC load, respectively.

The Nest thermostat responding to the aggregated DR control signal, following Eqn's. 4.4-4.5, is shown in Fig. 5.5. The heating setpoint is moved to a higher value

at around 11 hours, resulting in the HVAC switching from off to on, as well as the increase in the aggregated load. However, a slight delay in the switching from the setpoint change, is observed. This is possibly due to the system deadtime, which is arbitrarily set to five minutes, indicating that the HVAC system has the deadtime longer than five minutes.

Meanwhile, Fig. 5.6 shows the performance of the aggregated load control, that is happening simultaneously. It is shown that the control error is positive when the switching in the Nest thermostat occurs (11 hours), meaning that the increase in the aggregated load is desired. Therefore, the Nest response to the DR signal, is verified to be appropriate. Fig. 5.7 illustrates the HVAC switching of a random Meter 77 of the community, also happened at the same time. The switching also has occured in a way to increase the total load, responding to the positive control error. However, it is shown that the control error has been obviously positive, from 9 hours. Also, the HVAC of Meter 77 has been responding to it from 9 hours. Therefore, it is reasonable to question about why the HVAC switching had not happened earlier in Meter 23. This is possibly due to the probabilistic property of the control process, i.e., switching does not occur, if $\mathcal{I} > \mathcal{F}$. Also, the API rate limit may have caused this phenomenon. The rate of making request, is generally limited by the API providers, in order to protect the system. For instance, too many switching may harm the compressor, even if it is protected by the internal algorithm, since the user can possibly forcingly shut down the whole system. Therefore, in the present study, the request is made every minute, whereas the simulated thermostats respond to the DR siganl every second.

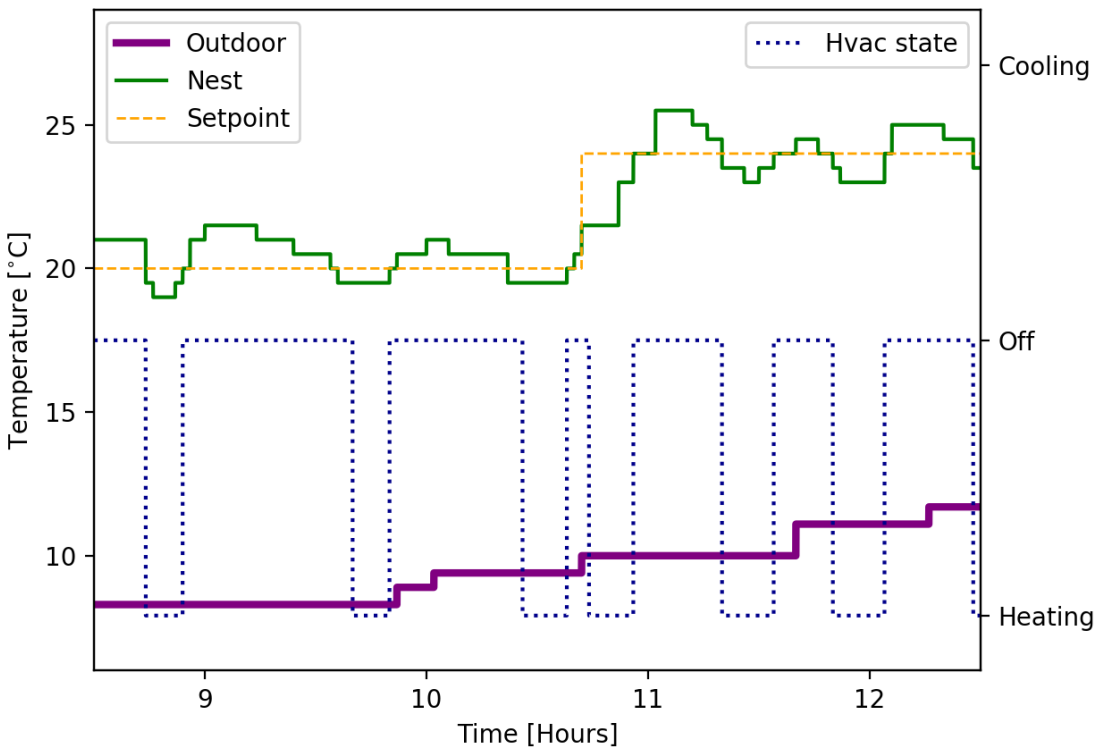


Figure 5.5: The heating setpoint is moved to a higher value, responding to a probabilistic control signal. A slight delay in switching is detected, possibly due to the HVAC system deadtime.

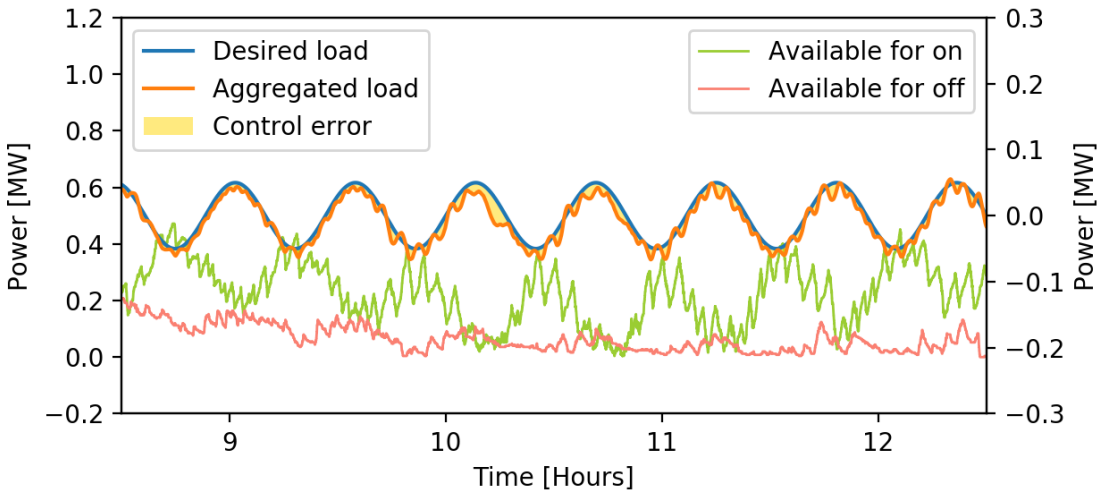


Figure 5.6: When the switching of the Nest thermostat has occured at around 11 hours, the control error is positive.

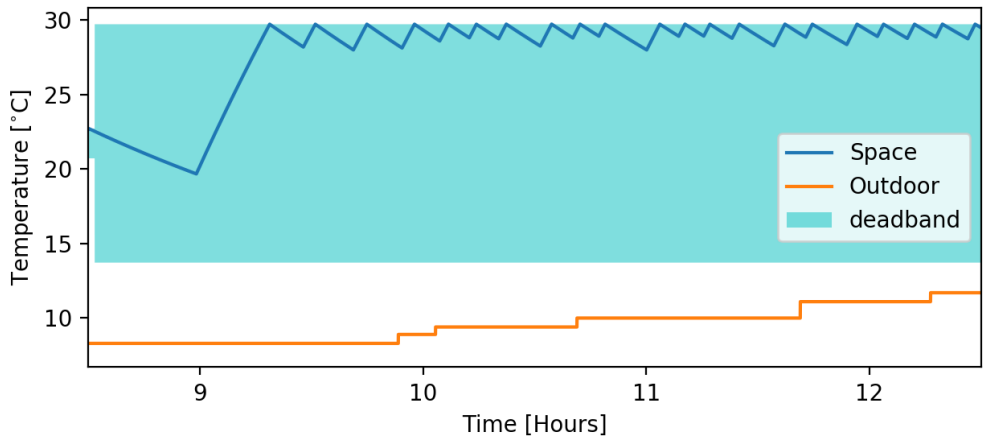


Figure 5.7: HVAC switching of a random Meter 77. The switching has occured in a way to increase the load, as it is desired at the substation.

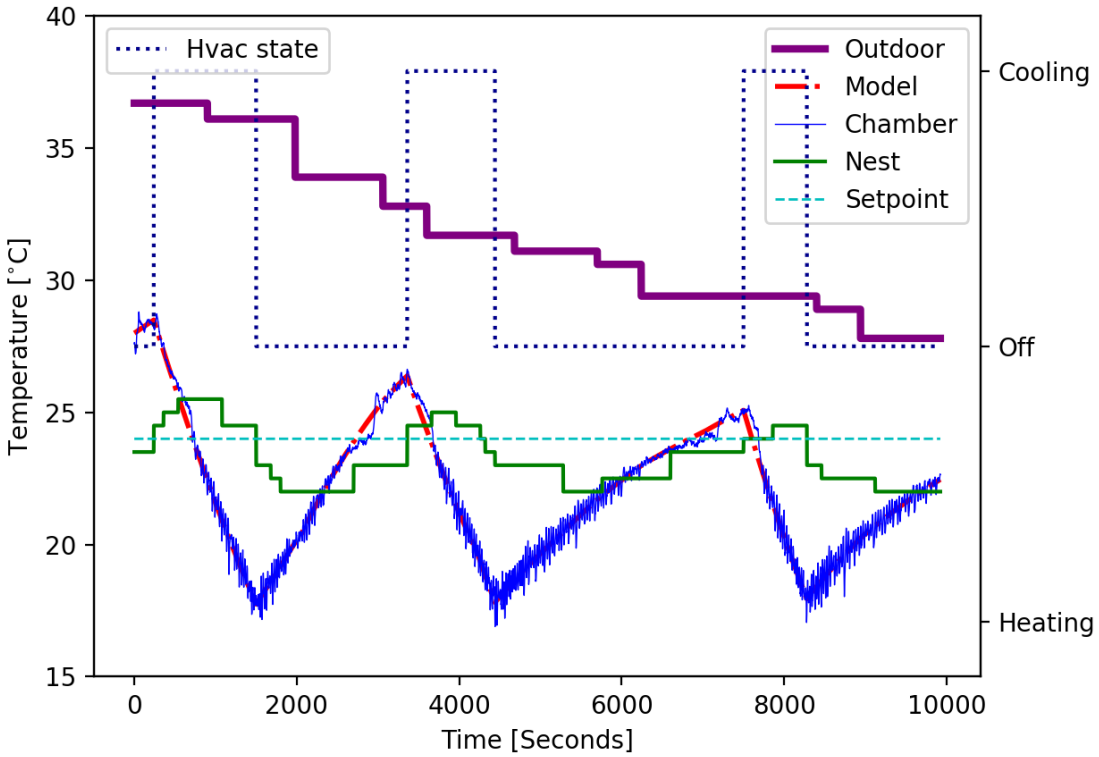


Figure 5.8: A disagreement between the temperature measured by the Nest thermostat, and thermocouples is observed, due to the heat capacity the Nest thermostat possesses.

5.3 Nest thermostat response analysis

Before the Nest thermostat can be used for implementing DR schemes, it is necessary to fully understand its dynamic response to temperature changes. The experimental apparatus of this study allows such a measurement, since the response of the thermocouples used in the environmental chamber is essentially instantaneous, and accurate, due to careful calibration of the thermocouples. Fig. 5.8 shows temperatures as indicated by the thermocouples, and by the Nest thermostat inside the environmental chamber. A discrepancy is observed. For example, when the chamber temperature

reaches the upper deadband limit and starts decreasing, the Nest thermostat continues to increase. Also, the Nest thermostat measures the temperature with 0.5°C resolution, while thermocouples have better than 0.01°C of resolution. Moreover, since the rate of data transfer through the Nest API is limited, the Nest thermostat temperature and its response are updated in one-minute frequency. While all these factors contribute to the discrepancy observed, the principal mechanism responsible for the behavior observed, is likely to be the effective heat capacity embedded in the thermostat, and its temperture measurement apparatus, whose detailed implementation is not well-documented. It is hypothesized that there is a heat transfer process between the air surrounding the thermostat and the Nest thermostat temperature measurement system, which can be modeled as a convective heat transfer to a thermal mass. This behavior can be modeled by

$$M_{\mathrm{n}}\frac{dT_{\mathrm{n}}(t)}{dt} = K_{\mathrm{n}}[T_{\mathrm{s}}(t) - T_{\mathrm{n}}(t)], \tag{5.1}$$

where M_{n} is the effective heat capacity of the Nest thermostat, $T_{\mathrm{n}}(t)$ is the modeled Nest Temperature, and K_{n} is a constant that models the convective heat exchange between the conditioned space and the Nest thermostat temperature sensors. Then the control logic mimicking the logic observed in the real Nest thermostat, can be modeled. The control logic in Cool mode in particular, could be described by

$$\Lambda = \begin{cases} 0, & \text{if } T_{\mathrm{n}}(t) < T_{\mathrm{set}} - 0.5°\mathrm{C} \\ 1, & \text{if } T_{\mathrm{n}}(t) \geq T_{\mathrm{set}} \\ \text{keeps previous state}, & \text{otherwise}, \end{cases} \tag{5.2}$$

replacing Eqn.2.5. The response of the Nest thermostat modeled from Eqn.5.1, reflecting its temperature resolution and the API rate limit, is shown in Fig. 5.9. The

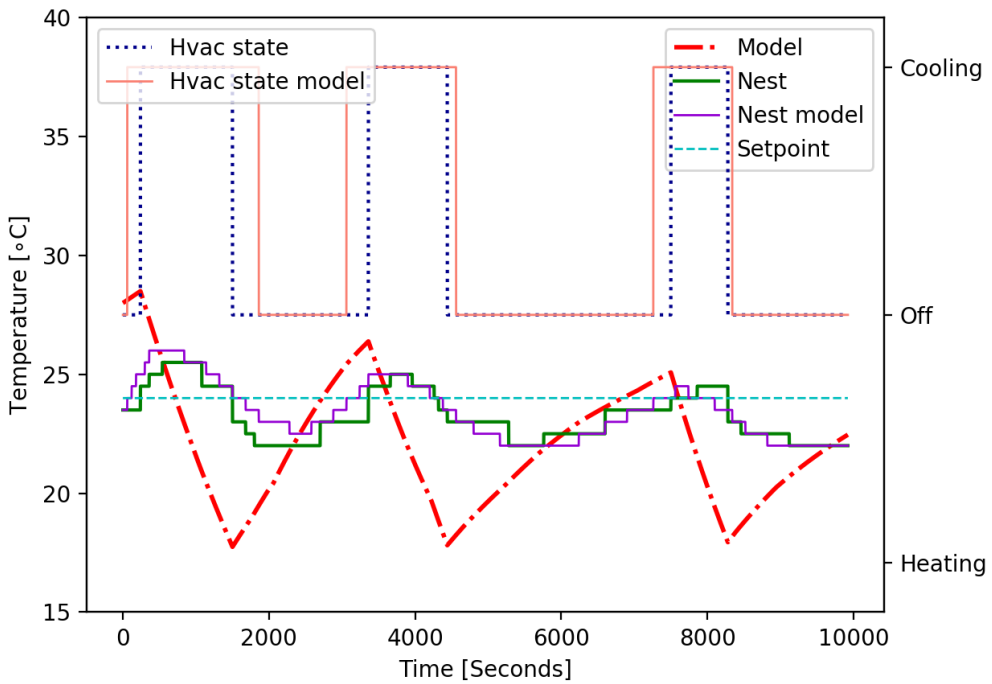


Figure 5.9: An actual and a modeled version of the Nest thermostat control, in the cooling mode. The Nest thermostat is modeled considering the heat transfer between the chamber and the Nest thermostat, the temperature measurement resolution, and the API rate limit.

suggested model of the Nest shows reasonably equivalent response to the actual Nest thermostat response.

5.4 Estimating space temperature from the Nest thermostat temperature

As found in the previous section, due to the effective heat capacity that the Nest thermostat possesses, the Nest thermostat conditions space based on somewhat wrong temperature information. Assuming that there are not additional temperature sensors other than the Nest thermostat, it is desired to estimate the actual space temperature based on the thermostat measurement. The actual space temperature $T_{\mathrm{s}}(t)$, can be calculated from Eqn. 5.1. However, since $T_{\mathrm{n}}(t)$ is non-differentiable, it needs to be fitted into a differentiable continuous curve. Then Eqn. 5.1 can be modified to

$$M_{\mathrm{n}}\frac{dT_{\mathrm{n,fit}}(t)}{dt} = K_{\mathrm{n}}[T_{\mathrm{s,est}}(t) - T_{\mathrm{n,fit}}(t)], \tag{5.3}$$

where $T_{\mathrm{n,fit}}(t)$ is the curve fitted Nest temperature, and $T_{\mathrm{s,est}}(t)$ is the estimated space temperature. The Nest thermostat temperature is fitted into a sinusoidal curve, through the least squres method, using Python library 'scipy.optimize_leastsquares'. Here, $dT_{\mathrm{n,fit}}(t)/dt$ is replaced with $D_{\mathrm{n,fit}}(t)$ that is

$$D_{\mathrm{n,fit}}(t) = \alpha\dot{T}_{\mathrm{n,fit}}(t-1) + (1-\alpha)\dot{T}_{\mathrm{n,fit}}(t) \tag{5.4}$$

where $0 \leq \alpha \leq 1$ that gives a weight on the derivative of the fitted curve at the previous time step, in order to alleviate the effect of the sharp change in $dT_{\mathrm{n,fit}}(t)/dt$. An example of the estimation is shown in Fig. 5.10. In the example, the Nest temperature curve is fitted with 3000 previous points and updated every 1500 seconds, with $\alpha = 0.99$. The estimated space temperature follows the phase of the space temperature curve, however with the considerable amount of errors.

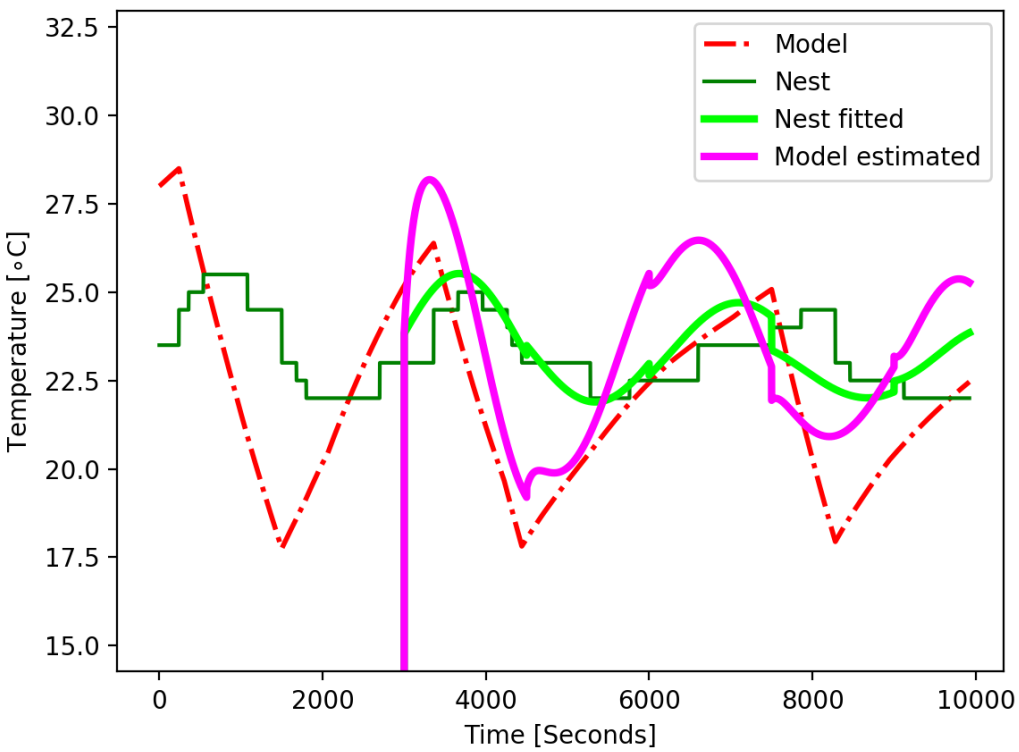


Figure 5.10: Estimation of the actual space temperature is desried for the future use, pursuing the user comfort. An estimation, using the current and previous Nest temperatureacture history, can be done by fitting the Nest temperature measurements to a differentiable curve.

Chapter 6

Conclusion

6.1 Conclusion

The purpose of the present work was to integrate an actual response of the Nest thermostat, to the aggregated load control simulation framework previously developed by Mammoli *et al.* [14]. The physical chamber, where its temperature is regulated through the PID controller, is implemented, providing the Nest thermostat with the thermal responses of the house model. With the help of the Nest API, the Nest thermostat could easily report its status to the outside world, and receive external signals such as DR signals from the aggregator. The reception of the external signal in the Nest thermostat does not occur internally, since the access to the internal logic of the thermostat is not granted to an individual. However, indirect response to the DR signal is available. The setpoint is modified responding to the stochastic switch request from the aggregator, then the Nest thermostat responded to the modified setpoint. As a result, the Nest thermostat is successfully integrated to the framework, and proved to be able to participate in the aggregated load control DR program, that is previously developed. The ultimate goal of the present study is to

fully understand the interaction between the physical response of the Nest thermostat and its environment. The goal of the present work is achieved, resulting in more ways to add realism to the simulation that will be discussed in the following section.

6.2 Future Work

While it is proved that the Nest thermostat can be integrated to the aggregated load control simulation framework, and that it is able to respond to the DR control siganl, possible future work, that is expected to bring improvements to the simulation framework, is discovered. Firstly, the chamber temperature control system was capable of implementing the space temperature ranging from 15 to 45°C, due to the physical limitation of heat exchangers, fluid running through heat exchangers, and fans selected. Therefore, the setpoints beyond that range were intentionally avoided. Establishing the more powerful chamber temperature implementing system, that will provide a wider range of the temperature in the chamber, will give opportunities to test the simulation in various conditions. Secondly, the existing thermostat logic in the load control framework, can be replaced with the modeled Nest thermostat discussed in Section 5.3, in order to simulate the DR program performance, derived by multiple commercially available smart thermostats, which is expected to better reflect the reality. Thirdly, an improvement to the space temperature estimation suggested in Chapter 5, is desired. Considering that the space temperature conditioning is solely dependent on the temperature measured from the Nest thermostat, the discrepency between the thermostat measurement and the actual temperature can lead to user discomfort. An accurate estimation of the space temperature, is expected to prevent the predicted user inconvenience. The result presented in the current study, is not yet perfect, since it is not very flexible with different data, i.e., a lot of manual adjustment of parameters is required when curve fitting. Application of more pow-

erful curve fitting method, or predicting through machine learning, are anticipated to be possible solutions for the problem. Lastly, getting a higher level of access to the Nest thermostat system, is desired. For instance, knowing the system deadtime, will not cause the delay in HVAC swithing as discussed in Section. 5.2. Also, having more direct access to the Nest thermostat, not through the API, will enable a higher communication frequency, therefore leading to a sophisticated DR performance.

Appendices

Appendix A

Model codes running in Real-time

Models of the load generator, the aggregator and Meter 23 are provided in the following github repository. Data files related are also included.

`https://github.com/chatchi923/master_thesis`

References

[1] Lawrence Livermore National Laboratory, "Estimated u.s. energy consumption in 2017: 97.7 quads." Available: `https://flowcharts.llnl.gov/commodities/energy`.

[2] Nest Labs, "Rush hour rewards : Results from summer 2013," tech. rep., May 2014.

[3] J. Rotondo, R. Johnson, N. Gonzalez, A. Waranowski, C. Badger, N. Lange, E. Goldman, and R. Foster, "Overview of existing and future residential use cases for connected thermostats," tech. rep., Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States). Building Technologies Research and Integration Center (BTRIC), 2017.

[4] F. Shariatzadeh, P. Mandal, and A. K. Srivastava, "Demand response for sustainable energy systems: A review, application and implementation strategy," *Renewable and Sustainable Energy Reviews*, vol. 45, pp. 343–350, 2015.

[5] R. Ford, M. Pritoni, A. Sanguinetti, and B. Karlin, "Categories and functionality of smart home technology for energy management," *Building and Environment*, vol. 123, pp. 543 – 554, 2017.

[6] M. H. Adam Wirtshafter, Jane Peters, "Do smart thermostats make for smart demand response programs?," July 2018.

[7] T. Larson, M. Chandra, K. Ward, D. Brannan, and S. Tobias, "Cutting peak demand - two competing paths and their effectiveness," in *Proc. IEPEC International Energy Program Evaluation Conference*, (Baltimore, MD, US), Aug. 2017.

[8] CLEAResult, "Kansas city power & light thermostat program : Case study." Available: `https://assets.ctfassets.net/wolv2nw8xwej/`

5f3fdwxO5iamIYoKGKKCSA/bb69b6b80c00d4d8f639286abc756b44/1116-CLR-MKTG-579028-KCPL-Nest-CaseStudy-R3_WEB.pdf, 2016.

[9] T. Brown, "Upgrading a thermostat dlc program to enhance utility and customer value." Available: https://www.peakload.org/assets/17thSpring/5c.Brown.KCPL.pdf, apr 2016.

[10] J. L. Mathieu, S. Koch, and D. S. Callaway, "State estimation and control of electric loads to manage real-time energy imbalance," *IEEE Transactions on Power Systems*, vol. 28, no. 1, pp. 430–440, 2013.

[11] S. Bashash and H. K. Fathy, "Modeling and control of aggregate air conditioning loads for robust renewable power management," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 4, pp. 1318–1327, 2013.

[12] A. Mammoli, C. B. Jones, H. Barsun, D. Dreisigmeyer, G. Goddard, and O. Lavrova, "Distributed control strategies for high-penetration commercial-building-scale thermal storage," in *Transmission and Distribution Conference and Exposition (T&D), 2012 IEEE PES*, pp. 1–7, IEEE, 2012.

[13] Y. Yasaei and A. A. Mammoli, "A novel framework for characterizing the aggregated response of thermostatically controlled loads in distribution networks," in *Power and Energy Society General Meeting (PESGM), 2016*, pp. 1–5, IEEE, 2016.

[14] A. Mammoli, M. Robinson, V. Ayon, M. A. Hombrados-Herrera, and M. Martinez-Ramon, "A simulation framework to develop control and forecasting tools for aggregated residential energy resources." preprint submitted to Energy and Buildings, 2017.

[15] "Nest expands energy business to more than 50 energy providers; nest thermostat rebates and rewards available to 30 percent of u.s. homes," Apr. 2016.

[16] J. Robinson, "Adafruit python max31856." Available: https://github.com/johnrbnsn/Adafruit_Python_MAX31856.

[17] Electronic Wings, "Raspberry pi gpio access." Available: http://www.electronicwings.com/raspberry-pi/raspberry-pi-gpio-access.

[18] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, "Numerical recipes in fortran 77: The art of scientific computing. cambridge univ," *Press, Cambridge*, 1992.

[19] Nest, "Nest learning thermostat advanced installation and setup help for professional installers." Available: https://nest.com/support/pro/article/Help-with-installation-and-set-up.

[20] Nest, "How to install your nest thermostat." Available: https://nest.com/support/article/How-to-install-your-Nest-Learning-Thermostat#works/?mode=guide.

[21] Real Python, "Api integration in python - part1." Available: https://realpython.com/api-integration-in-python.

[22] Nest Developers, "Oauth 2.0 authentication and authorization." Available: https://developers.nest.com/documentation/cloud/how-to-auth.

[23] Nest Developers, "Sample code for authorization." Available: https://developers.nest.com/guides/samples/sample-code-auth.

[24] Nest Developers, "Api read examples." Available: https://developers.nest.com/guides/api/how-to-read-data.

[25] Nest Developers, "Api write examples." Available: https://developers.nest.com/guides/api/how-to-write-data.

[26] V. H. Ayon, "Co-simulation framework for power distribution system analysis with humans in the loop," Master's thesis, The University of New Mexico, Albuquerque, NM, U.S., Dec. 2017.

[27] Wikipedia contributors, "Secure copy — Wikipedia, the free encyclopedia." Available: https://en.wikipedia.org/wiki/Secure_copy#cite_note-1. [Online; accessed 09-October-2018].

[28] Wikipedia contributors, "Expect — Wikipedia, the free encyclopedia." Available: https://en.wikipedia.org/wiki/Expect. [Online; accessed 17-August-2018].

[29] Nest, "Learn about early-on and how to change settings." Available: https://nest.com/support/article/What-is-Early-On.