Deep Reinforcement Learning for Real-Time Residential HVAC Control

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Evan Michael McKee

December 2019

Acknowledgements

I wish to thank all those who supported me during the completion of this thesis. I wish to thank my major professor, Dr. Fangxing Li, for being a continual source of guidance and inspiration, as well as the University of Tennessee faculty, for their oversight. Every class in my University of Tennessee career contributed in some way to this thesis.

I would also like to give a special thank you to Helia Zandi and the staff of Oak Ridge National Laboratory, for providing equipment and the means to perform all the tests shown in this thesis. Without the contributions of my colleagues, in particular Kuldeep Kurte, Jeffrey Munk, Travis Johnston, Olivera Kotevska, and Yan Du, who collaborated on this project with me, this work would not have been possible. I would like to thank the staff of CURENT and the DOE who act as sponsors of valuable and necessary research. This work was funded by the Department of Energy, Energy Efficiency and Renewable Energy Office under the Buildings Technologies Program.

Finally, I wish to thank my friends and family for their patience and encouragement.

Abstract

The Artificial Intelligence (AI) development described herein uses model-free Deep Reinforcement Learning (DRL) to minimize energy cost during residential heating, ventilation, and air conditioning (HVAC) operation. HVAC is difficult to accurately model and is unique for every home, so machine learning is used to allow for on-line readjustment in performance. Energy costs for the multi-zone cooling unit shown in this work are minimized by scheduling on/off commands around dynamic prices. By taking advantage of precooling events that take place when the price is low, the agent is able to reduce operational cost without violating user comfort. The AI was tested in simulation where the learner achieved a 33.5% cost reduction when compared to fixed-setpoint operation. The system is now ready for the next phase of testing in a live, real-time home environment.

Table of Contents

List of Figures

List of Tables

List of Equations

List of Useful or Unique Terminology

To avoid confusion, terms with multiple meanings, such as “setpoint,” are defined in this section and used consistently throughout the text. This section is meant only as a supplementary resource, and each term is redefined when it appears again.

Setpoint – Here, refers to the thermostat target, in degrees Fahrenheit, toward which the physical Air Conditioning (AC) unit is operating. The AI learner will convert a chosen action into a setpoint and pass it along to the physical AC unit. To avoid confusion, "setpoint" in this work never refers to a customer's indoor temperature preference. "Customer High" and "Customer Low" are used in place of "setpoint" for this purpose.

Customer High - Indoor temperature, in degrees Celsius, that the customer has chosen to be the upper limit on comfort; the indoor temperature that the customer does not want their house to exceed during a specified time interval. A traditional, fixed-setpoint thermostat would be set to Customer High.

Customer Low - Indoor temperature that the customer has chosen to be the lowest comfortable temperature.

Comfort Zone – The area of tolerable temperatures between Customer High and Customer Low.

Setpoint Governance - When the AI is prohibited from generating a setpoint that is above Customer High or below Customer Low, setpoint governance is said to be enforced. See Section 3.2.2.

Precooling – Precooling occurs when the AI injects an excess of cool air into the home during low price, thereby avoiding high price AC operation later.

Thermal Mass – Here, the effect that outdoor temperature has on indoor temperature. Used in this work to broadly describe all the physical objects, people, and materials within a home that contribute to heat retention in a home.

Minimum Cycle Time (MCT) - The minimum set length of time, in minutes, that the HVAC must commit to its decision, whether “off” or “on”. The amount of energy wasted from over-cycling, and the physical constraints of the HVAC used, did not permit for our test case a Minimum Cycle Time under 5 minutes.

Baseline Model - Or “naive” model, a model that imitates a fixed-setpoint thermostat found in most homes. The baseline model does not perform any learning during operation.

Yarnell Station House – The physical 2,400 sq. ft home on Yarnell Station Road in East Tennessee in which live testing was performed.

List of Abbreviations

AI - Artificial Intelligence

ANN - Artificial Neural Network

CBR - Case Based Reasoning

DDPG - Deep Deterministic Policy Gradient

DQL - Double Q Learning

DQN - Deep Q Neural Network

DP - Dynamic Programming

DRL - Deep Reinforcement Learning

DRE - Demand Response pricing Environment

GA - Genetic Algorithm

HERS - Home Energy Rating Score

HVAC - Heating, Ventilation, and Air Conditioning

IECC - International Energy Conservation Code

IHL - Internal Heat Load

KBS - Knowledge-Based System

MAE - Mean Average Error

MAS - Multi-Agent System

MC - Monte Carlo method

MCT - Minimum Cycle Time (in minutes)

MDP - Markov Decision Process

MOC - Minutes Outside Comfort

MPC - Model-based Predictive Control

ORNL - Oak Ridge National Laboratory

PAPA - Price And Price Alone model

PG - Policy Gradient

PID - Proportional-Integral-Differential

PIR - Passive Infrared Sensors

QRL - Q-Learning or Q-Reinforcement Learning

RC - Resistance-Capacitance building model

RL - Reinforcement Learning

RMSE - Root Mean Squared Error

RNN - Recurrent Neural Network

SARSA - State-Action-Reward-State-Action

SHEMS - Smart Home Energy Management System

TD - Temporal Difference learning

VAV - Variable Air flow Volume HVAC system

CHAPTER I: INTRODUCTION

The Artificial Intelligence (AI) development described in this thesis was the collaborative work of seven researchers: Helia Zandi, Jeffrey Munk, Travis Johnston, Kuldeep Kurte, Olivera Kotevska, Yan Du, and myself, Evan McKee. I performed testing, quality assurance, and parameterization, and the results of my individual contributions are presented in detail in Section 3.2. However, to omit background information about the AI development would deprive the reader of important context and diminish the work done by the team as a whole. Therefore, in addition to validating my own effort, this thesis acts as a high-level summary of the entire work and contains descriptive information about each part of the AI development. The following section provides background information explaining the purpose of this research.

1.1 Demand Response Load Scheduling

The genesis point of this research is the advent of demand response pricing environments (DRE) throughout the U.S. In a fixed energy pricing environment, there are no financial incentives to strategically load scheduling. The introduction of dynamic pricing allows for strategic choices that can result in reduced energy cost. For example, a homeowner can use more power when the price of electricity is low, and less at high price. This motivation is the driving force behind demand response pricing environments (DRE), in which utilities attempt to use dynamic pricing to influence consumer behavior with the goal of reducing peak demand [1]. In an ideal DRE, the homeowner saves money on their electric bill, and the utility company avoids unpredictable peaks and valleys in demand. For a single home, an inhabitant could determine the optimal scheduling themselves, but only with a significant commitment of time and calculation.

Automation presents a more favorable alternative. Smart Home Energy Management Systems (SHEMS) allow for the automatic activation and deactivation of devices throughout a home in accordance with some schedule. The Heating, Ventilation, and Air Conditioning (HVAC) appliance is an ideal candidate for automation due to its intermittent use and hands-free operation. Energy consumed by the HVAC system of a home accounts for approximately 50% of total energy usage [2]. Other large appliances, like dishwashers, washing machines, and electric vehicle chargers, are more difficult to pre-schedule because homeowners tend to use them whenever needed, regardless of energy price. In these cases, the necessity of their immediate use

outweighs the cost. HVAC units, on the other hand, run intermittently and are absent from a consumer's mind as long as comfort is maintained. Attempts to automate air conditioner use through pre-optimized automation have met with some success – Even just passively shifting the timing of air conditioners to precool a room has resulted in a reduction in electricity bills [3].

1.2 HVAC Modeling Challenges

HVAC automation in DRE's represents an opportunity for homeowners to save money. However, traditional automation systems use Model-Based Predictive Control (MPC), which requires an accurate model of the building containing the HVAC. Researchers have struggled with thermal modeling over the years because of its nonlinearity and strong specificity of application [4]. Google Sketchup was used to create the infographic in Figure 1-1, which gives only a partial list of the numerous co-dependent variables that contribute to the thermal profile of a single room.

The dimensions of the interior of the room play an important role, as well as the materials that make up the wall, floor, and ceilings [5]. A room full of objects will retain more heat than an empty one. Leather furniture will retain more heat than cloth furniture. Human occupants contribute a significant amount of heat to a room, and several works in the literature have found savings just by timing HVAC operation around occupancy [6] [7]. For an interior wall, the heat transfer between rooms must be addressed. If a door connecting the rooms is open, the air flow must be considered. If the door is closed, the materials and thickness of the door come into play. The same principle applies to windows in an exterior wall - windows that could be single pane, double pane, open or closed. Exterior walls have a heat transfer interaction with the outdoor temperature and the sun, an interaction that depends on the position, orientation, and latitude of the home, as well as the amount of available sunlight [5]. A room's elevation in a building is also a factor, as the tendency of heat to rise in building makes attics naturally warmer than basements.

All of the aforementioned attributes apply only to a single room. When modeling a house, each room has its own thermal profile that interacts with adjacent rooms and the outside, and air flow between rooms must be addressed [8]. These processes create a tangled web of nonlinear interactions.

Within this work, we use "Thermal Mass", a term that broadly refers to the amount that ambient temperature has an effect on indoor home temperatures, as a catch-all for the black

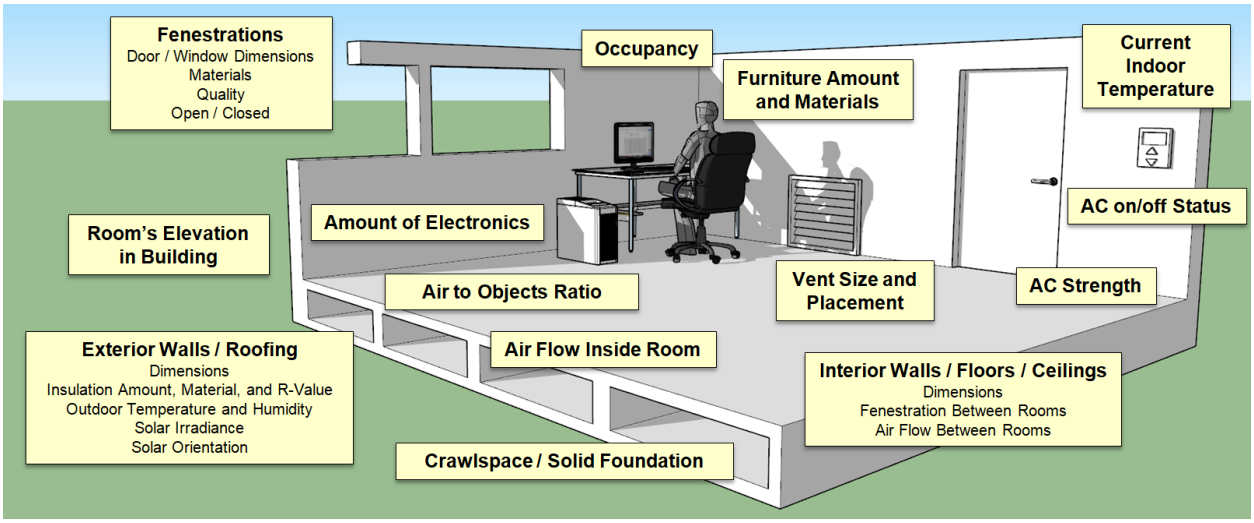


Figure 1-1: Partial list of contributions to future indoor temperature in a single room.

box of partially observable home attributes that influence the indoor temperature. Accounting for the presence of Thermal Mass during a temperature prediction is the primary challenge of HVAC automation. A perfectly simulated model might yield an optimal lowest cost of energy solution, but we expect the thermal profile of a home to change over time. Occupancy will change daily as inhabitants enter and leave. A homeowner might upgrade their windows or wall insulation, or they may take objects from one room to fill another. An MPC model fine-tuned for a home built today could be unusable one year later. Even the most thorough and accurate simulations in literature have high specificity of application. As a final snapshot of the complex state of modern HVAC modeling, consider that the 2017 paper acting as the starting point for this project specifically recommended that their algorithm *not* be used in a real home, citing the inherent complexity of HVAC modeling [2]. Researchers studied the weaknesses of MPC and searched for another answer to this problem. They found it in machine learning.

1.3 Machine Learning in HVAC

Machine learning, specifically Reinforcement Learning (RL), presents an alternative to model-based systems by testing the environment experientially and learning iteratively. In this method, no foreknowledge of values such as insulation coefficients or internal heat load is necessary. The learner simply makes decisions, updates its knowledge, and attempts to maximize return. Instead of a simulated model which contains every measured variable, those attributes that are cost effective to measure will be included in the state, and every other relationship will be accounted for by leveraging Deep Learning. An algorithm which combines the optimization of RL with the pattern recognition of a Deep Neural Network is said to employ Deep Reinforcement Learning (DRL). An AI controller equipped with such an algorithm could be configured to learn indefinitely inside a home and take self-corrective actions until it has approximated the lowest cost HVAC operation. Thermal Mass would still influence the result, but DRL could compensate for its ambiguous nature. Oak Ridge National Laboratory (ORNL) aspired to create such an AI controller, and the resulting development is presented within this work.

1.4 ORNL Development and Precooling

ORNL performed extensive prior work with a simulated home model which approximates, within an average error of one degree Celsius, the thermal behavior of a test home on Yarnell Station Road in Knoxville, Tennessee [8]. Beginning in March 2019, work began on a three-year project to design and implement a DRL controller that could perform training for HVAC usage in the multiple zones of the Yarnell Station House.

The timeline in Figure 1-2 shows the progress and future milestones of this project. Now that the AI has successfully learned the simulated model, the AI is using the actual Yarnell Station Home as a testbed. Eventually, the AI will be placed into a smart home neighborhood where the habits of different users will challenge the adaptability of the learner trained in the Yarnell Station House. The project is only considered a success if the AI can consistently demonstrate a 20% improvement over the baseline model, which is the traditional fixed-setpoint AC available in most homes.

The AI presented in this work is physically restricted from using setpoints outside of the user-defined comfort zone, so the learning that takes place is primarily focused on how to best manage incoming price increases to take advantage of precooling opportunities. Any discussion of the intelligence of the learner is a discussion of precooling events. Figure 1-3 zooms in on two days of behavior from a trained learner. Gray lines have been added at the low to high edges of the price signal, which is the bottom green line. For the experiments shown throughout this thesis, a square wave alternating between $0.05/kWh and $0.25/kWh is used as the price signal. The blue line, which represents Zone 1 indoor temperature, is what the learner actually controls. Note that the behavior of this blue line coincides with the vertical gray lines, and that precooling events take place before each price increase. The comfort zone of the user, which is the span of temperatures below the upper temperature preference (Customer High) but above the lower temperature preference (Customer Low) is marked as a green shaded region.

From point A to point B, the learner is cycling and precooling as necessary, influenced mainly by the pull of the outdoor temperature. At point B, the beginning of a precooling event, the learner observes an incoming price increase and a corresponding opportunity for cost

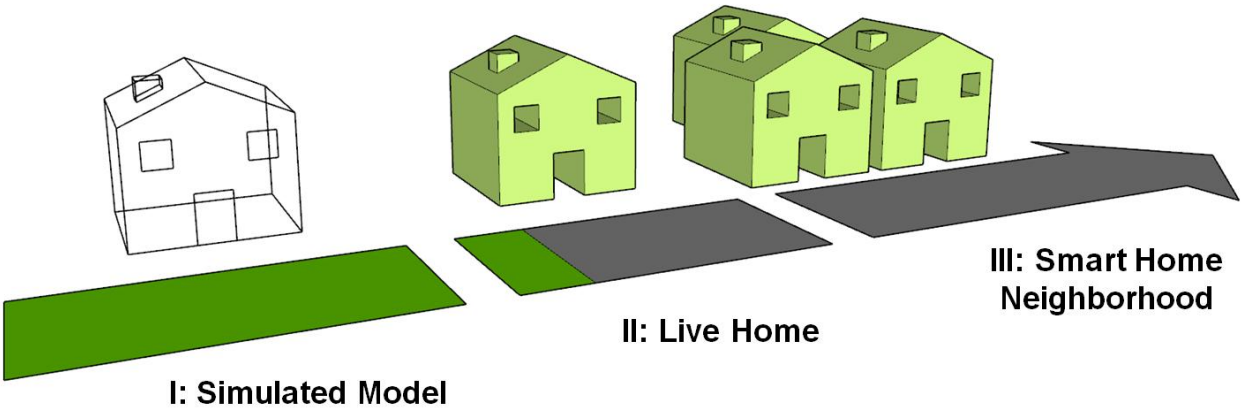


Figure 1-2: Project three-year timeline.

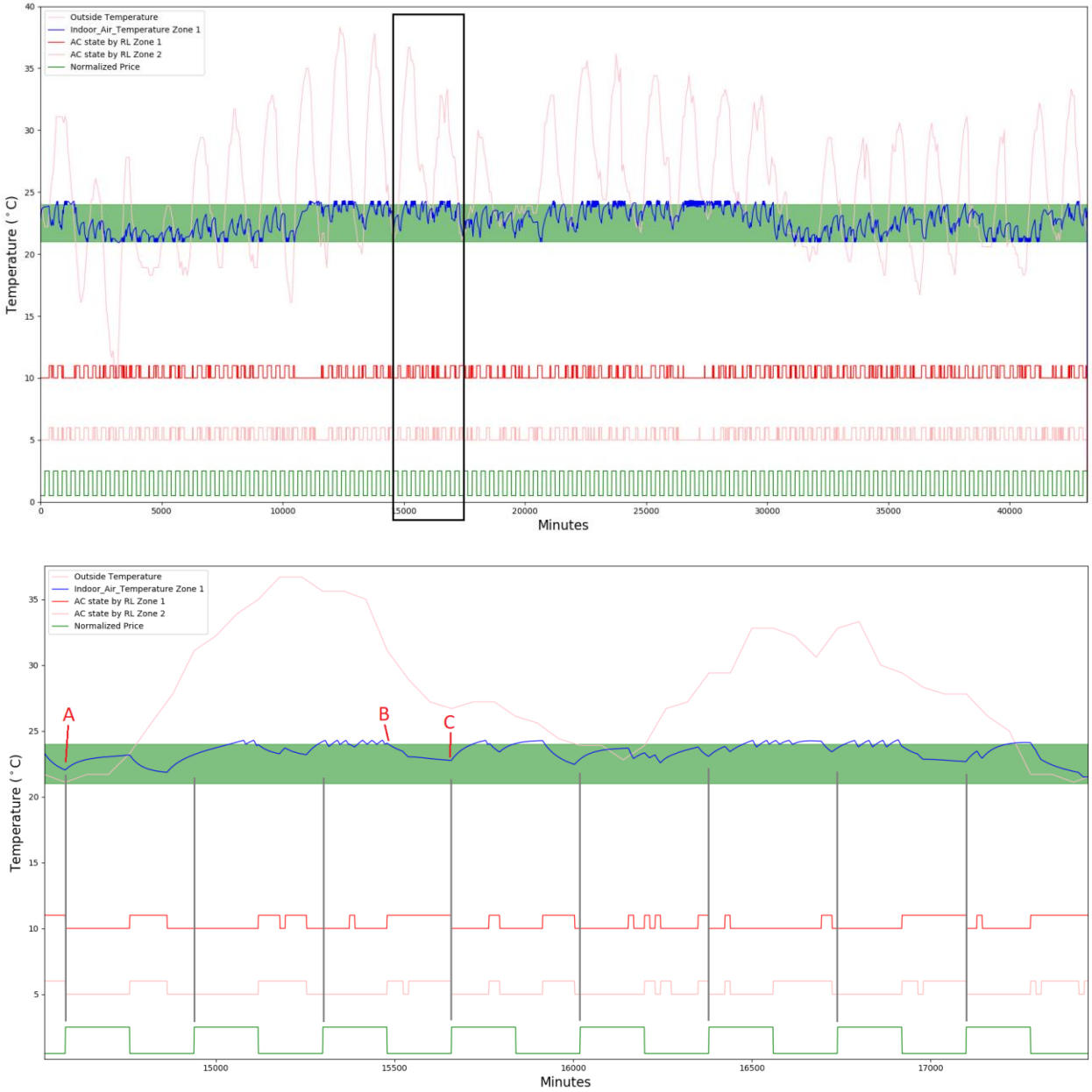


Figure 1-3: Precooling events coincident with price changes.

savings. The timing of this event is dependent on the lookahead length of the learner (how far into the future the learner can make observations) and the time at which the learner has determined a precooling event should be triggered. An increase in price is, by itself, not enough for the AI to trigger precooling. Certain conditions must be met for a precooling event to be deemed cost effective. At point C, the price increases and the AC is free to let the indoor temperature rise until it is forced to resume its natural cycle. In this way, the AI has saved money by moving an inevitable cycling event from high to low cost. A precooling event is nearly always associated with a price increase from low to high and an outdoor temperature that is greater than Customer High.

1.5 Statement of Problem and Purpose

Having established that cost savings are possible using an RL-guided HVAC controller, the objectives and purpose of such a controller are presented in the following section. The overall problem can be modeled as a constrained optimization problem: Minimize energy cost with minimal violations of user comfort.

Energy cost is calculated using

$$cost = \sum_{t=0}^{n} P_t * C_t \quad ,$$

Equation 1-1: Total cost over interval t = [0,n].

where P_t is the instantaneous price of energy at time *t*, C_t is the power consumption of HVAC over minute *t*, and *n* is the last minute tested in given time interval. User comfort is deemed satisfied if two objectives are fulfilled: a) The AC never runs while indoor temperature is 0.5 degrees below Customer Low, and b) The AC runs continually if indoor temperature is 0.5 degrees above Customer High. A 0.5 degree tolerance is applied so that the AC can cycle naturally without violating comfort. Note that comfort is not violated simply because the temperature rises above Customer High. In the case of a weak air conditioner on a hot summer day, the AI might not have the capability to cool the house, even if it is willing. As another example of an unavoidable comfort violation, if the customer changes their temperature preferences, the indoor temperature might stray outside the comfort zone until the AI can

readjust. Additionally, the AI presented in this work is not punished for falling below Customer Low, as it only has the ability to lower the indoor temperature through cooling, and never to raise the indoor temperature.

The zones outside of comfort are considered areas in which learning is unnecessary. There is no optimization problem to solve, only rule-based desired behavior: The AC must always be running when it is above comfort, and must always be off when it is below.

If the AI development can save a residential customer 20% over the fixed-setpoint baseline, it is considered a success. The AI that achieves this benchmark would be ready to move from the Yarnell Station House into other homes for testing. Although the primary objective of this project is to reduce consumer energy cost in a residential environment, additional objectives include discovery of the configuration of states, features, and rewards that yields the lowest long term cost results, quantification of the differences between simulated and real-time behavior, and expansion of the AI's usefulness to include other homes.

1.6 Reinforcement Learning

The following section is not meant to serve as a comprehensive explanation of the extensive field of Reinforcement Learning (RL). Instead, we review the main tenets of RL, with precedent given to those subjects that relate to the problem at hand. First, Section 1.6.1 will review RL in broad terms. Then, Section 1.6.2 will demonstrate why environment changes are problematic in the presence of limited information. Section 1.6.3 lists examples of environment changes in the present application that will test the flexibility of our AI development. Finally, Section 1.6.4 will address other unique challenges which arise as a result of live, real-time testing of RL in HVAC.

1.6.1 Reinforcement Learning Introduction

RL is a branch of machine learning that studies the conditioning of a learning agent towards accomplishing some goal through rewards and punishments [9]. At every iteration, the learner (agent) takes an action and is given a positive or negative reward. The agent is not told which action to take, but must discover the maximum long-term reward yielding actions through trial and error. RL works best when a problem can be modeled as a Markov Decision Process (MDP), which brings the problem into the scope of the Bellman Optimality Equation.

The Bellman Equation gives the value of taking action *a* from state *s* as

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a'),$$

Equation 1-2: The Bellman Optimality Equation.

where *Q(s,a)* is the expected return, which depends on the reward recieved for entering this state and the Q value of the next state *s'*. This equation links the states of an MDP together and gives the agent a roadmap for deciding the next action from its current state. The state values do not depend solely on instantaneous reward, but on the expected return of the entire trajectory, which is the cumulative reward from the current state to the goal state. Evaluating the next state using expected return improves the chance that the learner will prioritize long-term over short term reward. In an MDP, the next state is dependent on the current state and action, but independent of all previous state-action pairs.

After visiting a state and taking an action, the agent calculates an updated value for that state-action pair in accordance with the chosen algorithm. The flowchart in Figure 1-4 shows the cyclical interaction between state, action, and reward in a typical RL problem. After a sufficient period of training has elapsed, the agent intends to converge to an optimal policy that indicates what the agent should do at each state for maximum expected return. The program may then output an action value function *Q* which shows each state and the value of taking each possible action from that state, and use it to find a policy function π. Table 1-1 gives the characteristics of common RL methods, including their strengths and weaknesses.

Three variables are common to RL algorithms: 1) the step-size parameter α influences the training time by prioritizing recently-learned information over old data, 2) the probability ε guarantees exploration in the commonly used ε-greedy approach by granting the agent a probability ε of taking a randomly selected action, and 3) a discount rate γ which must be applied to the rewards so that their sum will approach a number other than infinity, if the task to be accomplished is continuous [9].

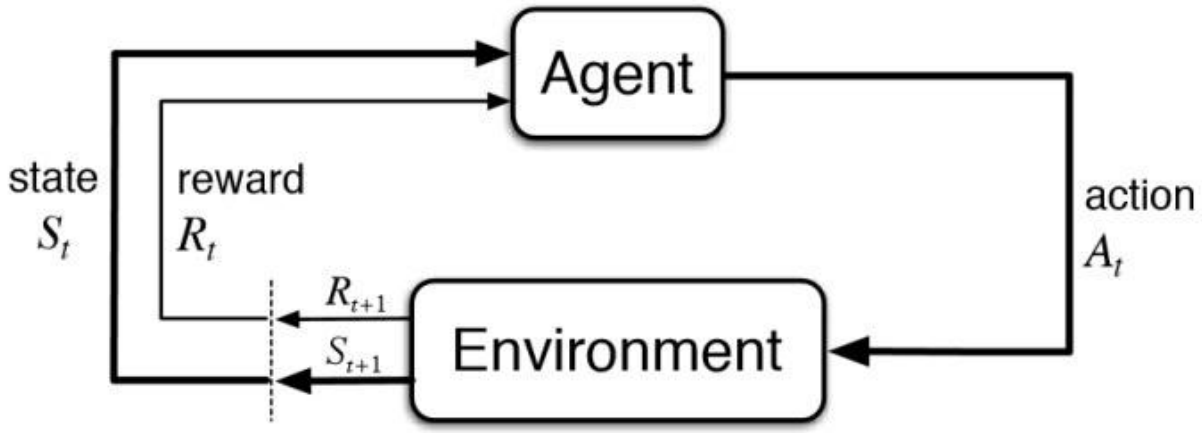


Figure 1-4: Interaction between state, action, and reward in a RL problem.

Table 1-1: Common Reinforcement Learning methods [9] [10].

Method	Description	Model Knowledge	Strengths	Weaknesses
Dynamic Programming (DP)	Iterates through a known environment until convergence.	Model-Based	Accurate results; can be used to bench test other algorithms	Requires a fully known model with fixed transition probabilities
Monte Carlo (MC)	Methods which sample average returns experientially after each complete episode.	Model-Free	Can be applied with incomplete knowledge of environment	Can take longer to converge than methods that practice bootstrapping, because MC only updates after episode termination
Temporal Difference (TD)	Combines sampling of MC with the bootstrapping of DP.	Model-Free	Converges faster than MC due to bootstrapping	Requires tweaking to properly hybridize the strengths of MC and DP
Q-Learning (QRL)	An off-policy TD control in which the learned action-value function Q approximates the optimal action-value function, independent of the policy followed.	Model-Free	Very common; converges faster than standard TD with better returns	Suffers from maximization bias because the same sample determines both the maximizing action and its estimated value
Double Q Learning (DQL)	Uses one estimate to determine the maximizing action and another to estimate its value.	Model-Free	Immune to maximization bias; converges faster than traditional Q-Learning	Does not generalize; each state-action pair must be visited to estimate its value.
Policy Gradient (PG)	Trains by making reward-producing actions more likely and reward-costing actions less so.	Model-Free	Can use continuous action space	Has high variance that must be minimized, difficult to select learning rate
Deep Q RL Network (DQN)	Uses deep learning to estimate the value function given experiential data.	Model-Free	Generalizes; using a neural network is usually better at model-free learning	More computationally expensive than RL without deep learning

1.6.2 Environment Changes

The “unknowns” that constitute incomplete knowledge in this HVAC environment must be addressed if RL is to be effectively applied. Understanding the challenge brought on by incomplete knowledge requires an understanding of the difference between an environment change and a state transition.

Suppose an AI is learning the behavior of a 6-sided die tossed once per turn. We set the state to consist of only the number showing on the outside of the die. After a sufficient amount of time has passed, the state transition probabilities from any state are shown in Table 1-2

Now suppose the 6-sided die is suddenly replaced by a 10-sided die. We expect every value in the table to change, to reflect the new probabilities shown in Table 1-3. Note that since the number on the outside of the die is the AI’s only interaction with the environment, its knowledge is *overwritten*, rather than added to. This is an example of an environment change. An environment change carries some permanence. The AI cannot store both tables, so it must re-learn state transitions every time the die is switched from 6-sided to 10-sided and back again.

If we expect the number of sides to change often, we should include this information when recording the state. Suppose a second learner has, as its state, the number of sides on the die as well as the number showing in the toss. As before, it rolls a 6-sided die for a sufficient number of turns and learns the left part of Table 1-4. Here, *P(6)* is the probability of the next roll being a 6-sided die, and *P(10)* the probability of the next roll being 10-sided. This time,
switching the number of sides from 6 to 10 registers as a state transition instead of an environment change. The learner still has to learn the new behavior, but it keeps the old, and this learner can be trusted to go back and forth between 6 and 10 sides. For frequent changes that are measurable, a state transition is preferable to an environment change.

The decision of which variables to include in the state, and how many, is important in any RL problem, but especially when the environment could change. Consider the variables presented previously in Figure 1-1 in Section 1.2. They are a partial list of all the factors expected to influence indoor temperature in a room. Some have positive correlation, such as solar irradiance and outdoor temperature. These have the potential to be combined if included in a state observation. Others are not likely to change, like wall thickness, and can be omitted from the observation. Several others, like the ones that make up Thermal Mass, contribute heavily to the result but are not cost

Table 1-2: State transition probabilities for the tossing of a 6-sided die.

#	**Probability of next state**
1	1/6
2	1/6
3	1/6
4	1/6
5	1/6
6	1/6
7	0
8	0
9	0
10	0

Table 1-3: State transition probabilities for the tossing of a 10-sided die.

#	**Probability of next state**
1	1/10
2	1/10
3	1/10
4	1/10
5	1/10
6	1/10
7	1/10
8	1/10
9	1/10
10	1/10

Table 1-4: State transition probabilities table learned by a 2-feature AI.

#	**Probabilities for 6-sided**	**Probabilities for 10-sided**
1	1/6 * *P(6)*	1/10 * *P(10)*
2	1/6 * *P(6)*	1/10 * *P(10)*
3	1/6 * *P(6)*	1/10 * *P(10)*
4	1/6 * *P(6)*	1/10 * *P(10)*
5	1/6 * *P(6)*	1/10 * *P(10)*
6	1/6 * *P(6)*	1/10 * *P(10)*
7	0	1/10 * *P(10)*
8	0	1/10 * *P(10)*
9	0	1/10 * *P(10)*
10	0	1/10 * *P(10)*

effective to measure. These dynamics will register as environment changes if not included in the state. Ultimately, this problem can be modeled as a partially observable Markov Decision Process (MDP) because each transition probability is dependent not only on the present state, but Thermal Mass is hidden from the observation. Therefore, RL can be applied to this problem.

1.6.3 HVAC Environment Changes

Since one of the goals of the AI development is generalization over multiple homes, environment changes are inevitable. However, identifying and anticipating them is the first step in mitigating their effects. The major environment changes that will test the AI's ability to relearn its environment are listed in this section.

1.6.3.1 Transition Between Houses

All of the models tested will be re-homed, whether from one house into another or from simulation to a house. If this environment change occurs, we expect the model to learn the new thermal profile of the new home and overwrite the old. One might ask, why bother keeping any of the previously trained information if we expect it to be overwritten? There are two answers: One, the beginnings of an RL session are associated with "flailing," random movements while the learner gets its bearings. Great care is taken so that these movements happen in simulation, and not inside the home of an actual customer. The second answer highlights one of the strongest advantages of the experiential learning performed by RL. Because the learning is based on experience, probability acts as a safety net to bias the learner towards experiences that are more likely – not just future states that are more likely, but future environments. If we expected to re-home the model into a completely unknown, stochastic environment, then pre-training would not be useful. Instead, we consider it likely that AC operation in one home carries many of the same qualities as AC operation in another. In other words, the pre-trained AI learner should already know most of the rules of the game when it enters the new house, and will then be free to focus on adjusting to the HVAC characteristics of the new environment.

1.6.3.2 Thermal Upgrade

A homeowner could install new insulation or upgrade their HVAC unit. Again, the properties of heating and cooling in the home would undergo a permanent change, and the AI would have to learn new state transition probabilities.

1.6.3.3 Changes in Occupancy

Probability will be able to catch some of the time-based comings and goings of people in the house. However, the cost savings achieved by other projects which have accounted for occupancy suggest that an effort to estimate occupancy could be profitable. Results are shown in Section 3.2.5.5 that show how the training time of the AI is affected when time is added as a feature, thereby accounting for the daily occupancy routine of a homeowner. Bear in mind, however, that in our smart home neighborhood case, the homeowner schedules their Customer High and Customer Low preferences partly around when they expect to be present in their homes.

1.6.4 Other RL HVAC Considerations

In past developments, researchers were able to run RL experiments in simulation. Their work was designed and optimized for simulated testing. However, one of the goals of the ORNL development described in this work is a successful transition from simulation to live testing. In this section, problems associated with real-time learning and installation into an occupied home are discussed.

1.6.4.1 Online Operation

The AI will learn and run in an occupied home. Since a live human being will be on the receiving end of any hardware or software malfunctions, care must be taken to account for things like lost remote connection or power loss. We are mindful that an unexpected hardware bug could cause loss of money or comfort. A default case is hardwired into the user's home that sets their AC setpoint to the scheduled Customer High if it does not receive a signal saying otherwise.

1.6.4.2 Real-Time Operation

The AI's training time is bound by the AC's Minimum Cycle Time (MCT), the amount the AC must commit to cycling "On" or "Off" after making a decision. For a 5 minute MCT, the learner will make 288 learning updates per day (i.e., one every five minutes). If the MCT is tripled to 15 minutes, the learner makes only 96 decisions per day, and that same AI's training time will be three times slower. Data and results will also take three times longer to collect. Gathering a month's worth of data in simulation takes seconds, but a month's worth of data in the physical Yarnell Station House is based on a real month's runtime. It was decided that a) we would not install an AI that had not undergone some pre-training for this application, and b) a premium is placed on a fast training time. For example, if a method were discovered that arrived at the empirical lowest cost path for a month, but the convergence rate on this method were 10,000 iterations, we would abandon it. A homeowner should not spend excess money running their AC for months while the AI adjusts to an environment change.

1.6.4.3 Tangible Exploration Cost

The AI must spend money to explore. The baseline against which the AI competes is a fixed-setpoint model, called the "naive" model elsewhere in this work. It functions like an ordinary household thermostat, as shown in the bottom half of Figure 1-5. Whenever the indoor temperature is above a specified point, the AC turns on. The AI, on the other hand, has more opportunities to cycle incorrectly while it is learning and exploring. The AI operating in the top half of Figure 1-5 is an untrained learner at the beginning of its experience and is subject to the random "flailing" common to the first few RL iterations. Each wasted cycling operation, though informative to the learner, represents a loss in the customer's money. This characteristic, which shapes the goal of saving 20% over the baseline, means that any monetary loss associated with exploration must be recovered during precooling.

The potential savings are more apparent during warmer months, but the cost associated with exploration is consistent throughout the year. This means that an AI could be cost effective only for July, but could spent money learning during the other eleven months of the year. One

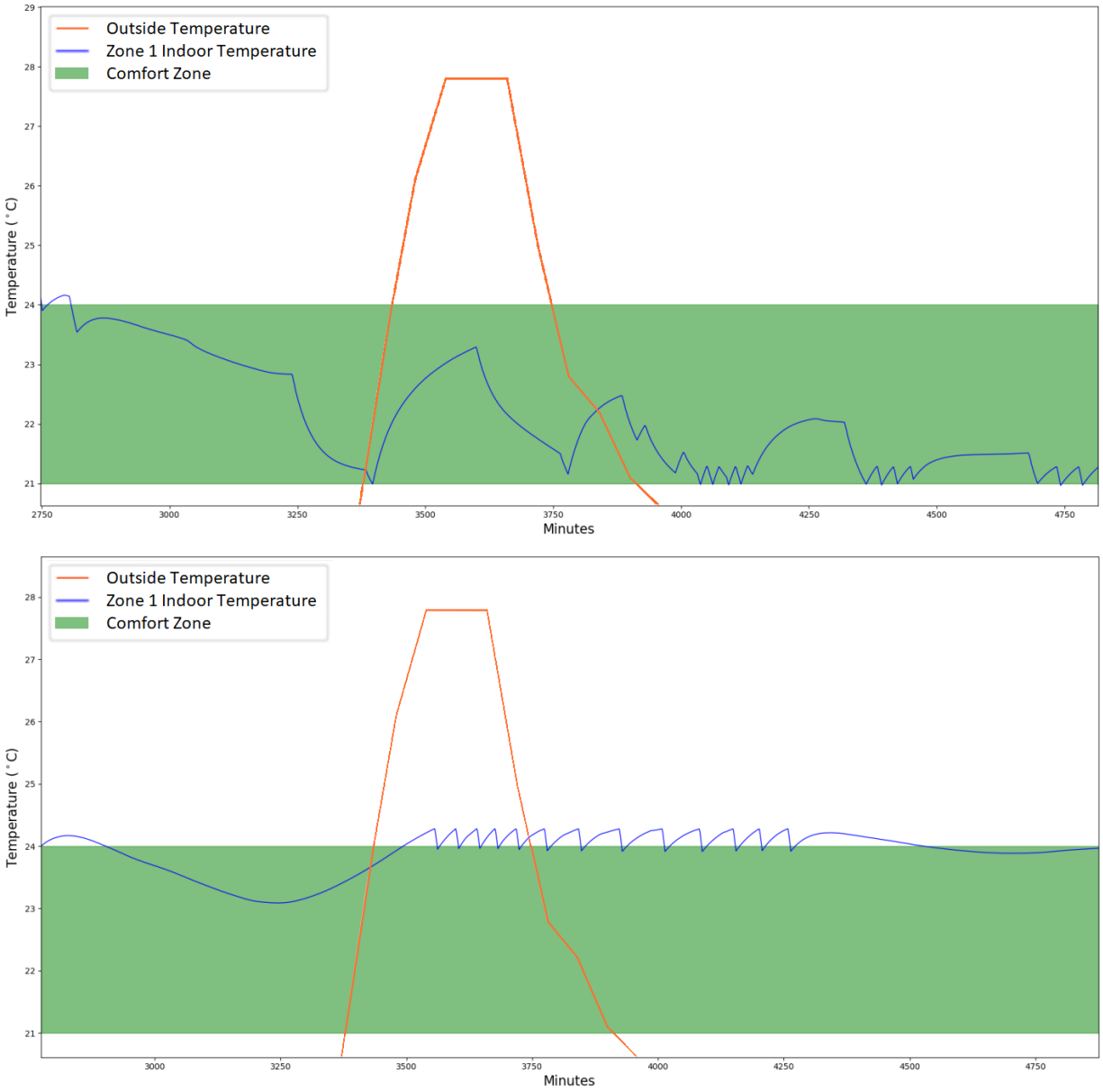


Figure 1-5: Untrained experiential learner (Top) vs. the baseline (Bottom).

untested option is the introduction of software which disengages learning and leaves the setpoint at Customer High whenever the short-term weather forecast shows only temperatures below Customer High. This governance might be revisited when heating and “auto” mode are introduced.

1.6.4.4 Endless Runtime

The AI must be designed to run indefinitely. Some of the algorithms studied in literature, such as [6], had an exploration phase with an ε-greedy policy followed by a fully greedy exploitation phase. The project described here cannot fully disengage exploration after deployment, because we must account for environment changes that occur during in-home use. We anticipate loss of money as a result of allowing this exploration. Additionally, since the algorithm is designed for an infinite number of episodes, any algorithm parameter or exploration decay rate that depends on the total number of episodes must be recalibrated to account for an “out-of-home” training phase and an “in-home” training phase. Although the project has not advanced far enough into testing to make these divisions, our goal is to consider infinite runtime as early in the development as possible.

CHAPTER II: LITERATURE REVIEW

Since my personal contribution to this work revolves around parameterization, the part of academic literature most relevant to my work was the state, action, and reward combinations other researchers had chosen for their developments. It is nevertheless worthwhile to show the reader the contributions to HVAC automation that have preceded the work described, and to distinguish our work from theirs academically. The following chapter is divided into two sections: first, a broad survey of AI controlled HVAC units and energy management systems will be given. Then, three of the cases will be investigated in further detail - one using a Deep Q RL network in simulation, one using Deep Deterministic Policy Gradient (DDPG), and one using a holistic smart home approach to cool a physical office. The setup, state-action-reward system, and methodology of each will be presented. At the end of each case study, I will discuss differences between these methods and the work described here, and show where the ORNL project can provide academic novelty.

2.1 Survey of AI in Smart Home Energy Management

A number of organizations and researchers have designed automatic HVAC controls in an attempt to reduce energy costs. Some focused on HVAC specifically, while others used AI to manage SHEMS. The earliest notable effort is the 1997 case which used a precalculated optimized setting for control [11]. Table 2-1 and Table 2-2 show a survey of developments in AI-assisted HVAC control since 1997, pared down to those examples most relevant to our case.

The table represents roughly 20 years of effort on HVAC automation. There is no universal baseline by which the tests can be compared. Some were attempts to predict consumption given past events [12] [13] [14]. Others incorporated control logic in an attempt to reduce energy consumption or cost [11] [15] [16]. Still others attempted to quantify a thermal comfort level for their systems to maintain [6] [17] [18]. Although the amount of improvement varies widely from 3 to 60%, nearly all cases reported an improvement in results through their development.

Table 2-1: Key AI HVAC developments in literature. [19].

Year	System	Automation	Results	Ref.
1997	HVAC system for occupied comfort and efficient running costs	Knowledge-based System (KBS) for predictive control	20% electricity savings	[11]
1998	Expert system in commercial buildings	KBS for energy conservation	Up to 60% cost savings	[20]
2000	HVAC system with variable air volume and constant air volume coils	Genetic algorithm (GA) cost estimation	0.1%-1.9% simulated savings	[21]
2002	Smart Home demonstration at Massachusetts Institute of Technology	Data analysis for energy savings and thermal comfort	14% energy savings	[22]
2003	Fuzzy controller for indoor environment management	Fuzzy P controller	Up to 20.1% heating and cooling energy savings	[23]
2003	HVAC Optimization	Artificial Neural Network (ANN) for predicting optimal heating start times	Linear relationship between predicted and real, with R^2 value between 0.968 and 0.996	[24]
2005	Energy Forecast of Intelligent Buildings	Fuzzy multi-criteria decision making method	3% Cost savings	[25]
2005	Adaptive control of home environment	Distributed AI with sensors	Electrical consumption sensors adapt to inhabitants' habits	[26]
2006	Centralized HVAC system	Multi-agent system (MAS) for thermal comfort control	7.5%-11% prediction error rate with respect to thermal comfort	[12]
2006	Predictive control for building heating system	Fuzzy + proportional-integral-differential (PID) controller for improving control performance	For heater control, temperature increase times can be reduced.	[27]
2007	Achieving thermal comfort in two simulated buildings	Development of linear reinforcement learning controller	Over four years, energy consumption increased marginally, but dissatisfaction index decreased from 13.4% to 12.1%	[17]
2009	Control performance improvement of HVAC	Model-based predictive control (MPC) on time delay model	For 1200 sq. m area, predicted set point with error rate of 0.13 ° C	[28]
2010	Intelligent multi-player grid management	Evolutionary computation development	1 kWh of energy cost reduced by 62.4%	[15]
2011	Controller development for heating/cooling	GA-based fuzzy PID controller development	Equipment operating costs up to 20% lower	[29]

Table 2-2: Key AI HVAC developments in literature (cont.) [19].

Year	System	Automation	Results	Ref.
2012	Coordinating occupant's behaviors for building energy / comfort management	Distributed AI, multi-agent comfort management	Reduced energy consumption by 12% while maintaining < 0.5% comfort variation	[6]
2013	Optimization through load shifting	GA development for load shifting control	35% load shift possible with storage	[3]
2014	Energy consumption prediction of commercial office building	Case-based reasoning (CBR) model development using three hour weather lookaheads	CV-RMSE under 13.2%, RMSE under 14 kW	[13]
2014	Energy management optimization in a wooden building	Distributed AI development	Generated optimal setpoints save up to 39% energy	[30]
2015	Real-world energy savings in a smart building	Rule-based approach for scheduling control	Daily energy savings up to 4%	[31]
2016	Model-based predictive control	MPC development	Set point optimization saved up to 34.1% energy	[14]
2016	Multi-objective control for smart energy buildings	Hybrid multi-objective GA development	31.6% energy savings in a smart building	[32]
2017	Deep reinforcement learning for building HVAC control	DRL-based algorithm	11% energy savings	[2]
2017	Office HVAC system	RL and RNN	2.5% energy savings, comfort improved on average 15%	[18]
2018	Home air conditioner energy management under demand response	MPC for demand response	9.2% energy savings against conventional on/off control	[33]
2018	Enhanced HVAC system energy efficiency	MPC	Energy savings between 10-15%.	[16]
2018	HVAC systems at an office building	MAS and CBR for energy management	41% energy savings	[7]
2019	HVAC control for reducing energy consumption and maximizing thermal comfort	Deep Deterministic Policy Gradient (DDPG) development	Maintains thermal comfort within 0.5 ° C	[34]

2.1.1 Automation Techniques

The methods employed to automate HVAC in Table 2-1 and Table 2-2 vary. In a Knowledge-Based System (KBS), an AI attempts to use the knowledge of a human expert to support its decision making [11]. If the system encodes expert knowledge as conditional rules, it is a Rule-Based System. If the KBS imports a set of cases which have already been solved to support its decisions, it is performing Case-Based Reasoning (CBR)[13].

Load shifting is the broad term for any attempt to move power usage away from expensive demand response timings and toward cheap ones [34]. In HVAC, load shifting is the goal of all the models that incorporate demand response pricing and load shifting has demonstrated success by exhibiting precooling.

Artificial Neural Networks (ANN) and Recurrent Neural Networks (RNN) are said to imitate the workings of the human brain by linking neurons into input, output, and hidden layers [19]. Two tools provided extended control for ANN's: fuzzy logic controls and model based predictive control (MPC). In MPC, the results of a system's prediction use a feedback sensor to give the system "insight" on the next prediction [19]. Fuzzy logic control, in which outcomes are given grades rather than the traditional true/false Boolean dichotomy [23], is rarely applied to real-time control but can be used to analyze datasets [19].

Genetic Algorithm training (GA) is a machine learning algorithm based on evolutionary biology [21]. The best outcomes of each generation are kept to prime the next one, without the need for a mathematical model representing the system. Cheng and Lee [19] noted that this technique is computationally expensive and recommends avoiding its use in real-time application.

Lastly, distributed AI systems use multiple agents that execute in parallel to form a smarter control system than can be achieved by any of them acting alone [19]. Sometimes, the goal is to avoid computational bottlenecking, and sometimes the multiple systems have separate objectives altogether. It is likely that after adding support for heating control and other energy loads, the ORNL project described in this work will evolve to become a distributed AI system.

2.2 Case Studies

2.2.1 Deep Q RL Approach (2017)

The starting point for the attempt described in this work is found in a 2017 article titled "Deep Reinforcement Learning for Building HVAC Control" [2]. There, Wei, Wang, and Zhu simulated an environment using EnergyPlus software and used Deep RL to manage operation of two air conditioning zones. The experiment reported 11% energy savings over a rule-based baseline model which switched on when the indoor temperature rose above Customer High and ran continuously until it reached Customer Low.

Here are the state, action, and reward function as defined by their work:

2.2.1.1 2017 Deep Q RL State

In the 2017 Deep QRL paper, the state consisted of four features, shown in Table 2-3. Since a different learning algorithm was responsible for control of each zone, only the temperature of one zone is recorded in a single state.

2.2.1.2 2017 Deep Q RL Action

The VAV (variable air flow volume) HVAC system allowed an air flow rate chosen from discrete levels for each zone. The action space was made of all possible combination of these rates for each zone, with a total number of actions $n = m^z$.

2.2.1.3 2017 Deep Q RL Reward

The reward function was given by

$$r_t = -cost(a_{t-1}, s_{t-1}) - \lambda \sum_{i=1}^{z} \left(\left[T_t^i - \overline{T}_t^i \right]_+ + \left[\underline{T}_t^i - T_t^i \right]_+ \right)$$

Equation 2-1: Two-term reward function in the 2017 Deep QRL paper.

where $cost(a_{t-1}, s_{t-1})$ is the monetary cost from the previous state-action pair, λ is the weighting factor applied to the comfort term, and the comfort term is the amount that the indoor

Table 2-3: State definition in the 2017 Deep QRL paper.

Feature name	Description
t	Minute of the day
T_{zone}	Zone temperature
T_{out}	Outdoor temperature
Q_{sun}	Solar irradiance intensity

temperature has strayed outside of comfort. This is the first time we encounter the two-term reward system, a reward function that accounts for the monetary cost and comfort violations and has a weighting factor amplifying the effects of one term over the other. λ was used to weight the comfort penalty relative to the cost term. A λ of 100 was used in their experiment so that the comfort penalty would outweigh the cost term.

The results of single and four-zone training are shown in Figure 2-1, showing that the trained learner was able, in both cases, to maintain comfort.

The ORNL project described in this work is similar to the 2017 Deep QRL paper in scope and objective. However, some differences separate our research from theirs. The first, and largest, is that the system developed by our ORNL team is tested in an actual building. Wei, Wang, and Zhu stood by the accuracy of their EnergyPlus models, but recommended that this algorithm not be used in a real-time setting. The problems associated with real-time operation described in Section 1.6.4.2 shed some light on why so much of the literature was limited to simulation testing. We consider the ORNL work a field test for some of the claims that previous works have made in simulation.

2.2.2 Deep Deterministic Policy Gradient Approach (2019)

Instead of Deep Q RL, the February 2019 attempt by Gao, Li, and Wen utilized DDPG as their learning algorithm [34]. As zones and flow control options are added to the action-space in HVAC automation, the problem becomes complex, multidimensional, and computationally difficult. DDPG was chosen so that the development could use a continuous action space, setting the temperature and humidity setpoints of the HVAC to virtually any value. Their system was tested in TRNSYS, a thermal simulation software.

While DQN tries to use deep learning to approximate and generalize a Q-table, PG methods attempt to approximate the policy. DDPG adds an extra level of complexity by using an actor-critic framework. In an actor-critic network, two neural networks are connected as shown in Figure 2-2.

The actor network specifies a control action from a given state. The critic network evaluates the action and, in batches, uses a TD error to update the actor network with the sampled policy gradient. After sufficient training has elapsed, only the actor network is needed to control the system.

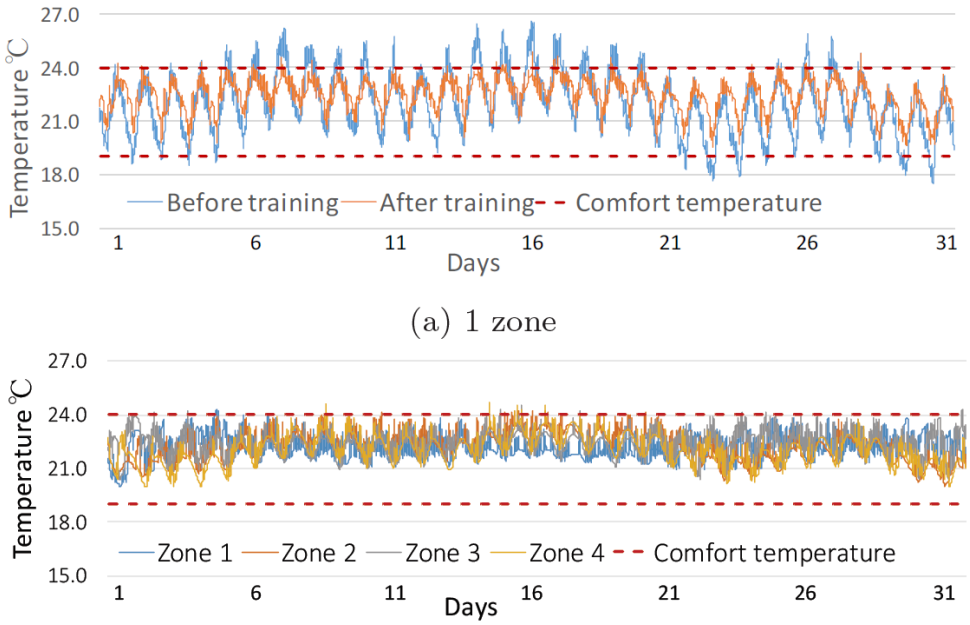


Figure 2-1: Results of a one-month trial in the 2017 Deep QRL paper.

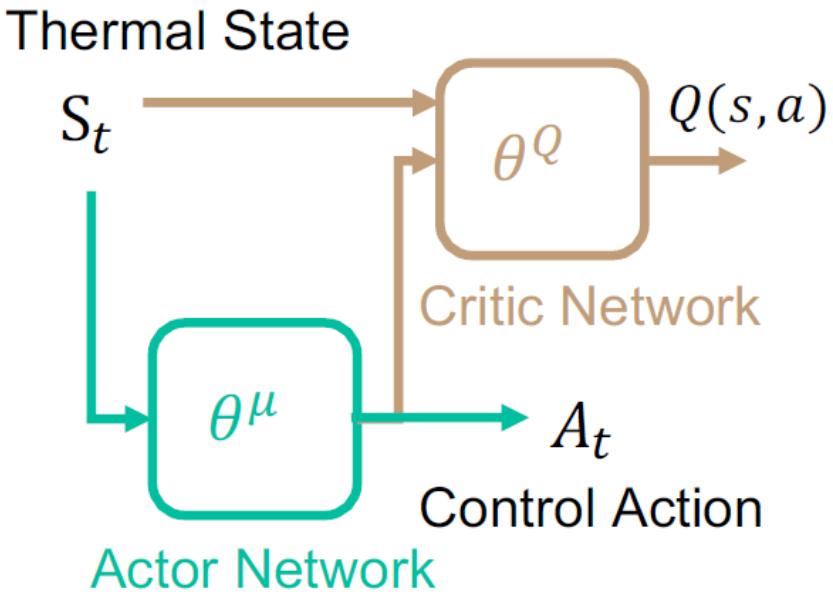


Figure 2-2: Actor-critic DDPG network [34].

2.2.2.1 2019 DDPG State

Table 2-4 shows the state definition used in the 2019 DDPG development. The state incorporated indoor and outdoor temperature, as well as indoor and outdoor humidity.

2.2.2.2 2019 DDPG Action

The 2019 DDPG development used a continuous action space that included a range of setpoints for indoor temperature and indoor humidity. The desire to use this action space prompted their selection of the DDPG algorithm.

2.2.2.3 2019 DDPG Reward

The reward system used in the 2019 DDPG development is shown in Equation 2-2.

$$R_t(S_t, A_t) = -\beta P_t - \begin{cases} 0 & -D < M_t < D \\ (M_t - D) & M_t > D \\ (-D - M_t) & M_t < -D \end{cases}$$

Equation 2-2: Equation showing the reward structure in the 2019 DDPG paper.

Although the equation looks different, this is the same two-term reward system used in the previous case study. *D* is the thermal comfort value threshold. The quantification of comfort was divided into zones from -3 to 3, with -3 being unbearably cold and 3 being unbearably hot. No penalty was incurred for an indoor temperature between -*D* and *D* (Inside the comfort zone). β acted as a weighting factor to control the relative contribution of the energy cost and comfort terms. The researchers experimented with different values of β and reported that a β of 0.075 (weighting comfort 13 times greater than cost) resulted in sufficiently low energy cost.

The researchers reported favorable results. In Figure 2-3, the authors show faster convergence and a higher overall reward than DQN, Q-learning, and SARSA (State-Action-Reward-State-Action, another learning algorithm). In Figure 2-4, they report less energy consumption than these methods as well.

As in the 2017 Deep Q RL case, the system was only tested in simulation. DDPG was one of the methods tested in the ORNL project, but we were unable to reproduce the results described here. Thier success could come from the fact that humidity control is better suited for

Table 2-4: State definition in the 2019 DDPG paper.

Feature name	Description
T^{in}_t	Indoor Temperature
H^{in}_t	Indoor Humidity
T^{out}_t	Outdoor Temperature
T^{out}_t	Outdoor Humidity

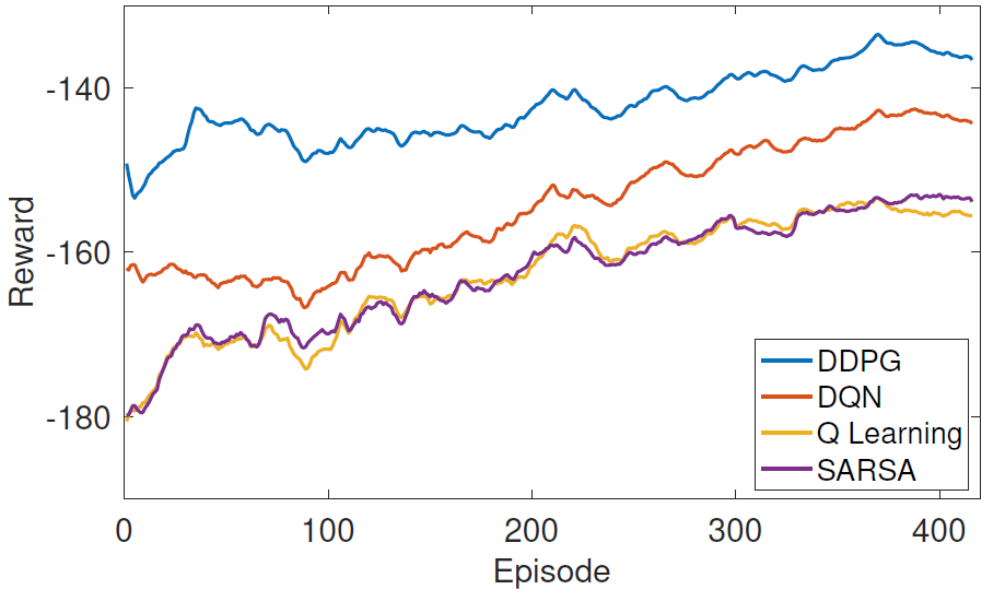


Figure 2-3: Algorithm convergence in the 2019 DDPG paper.

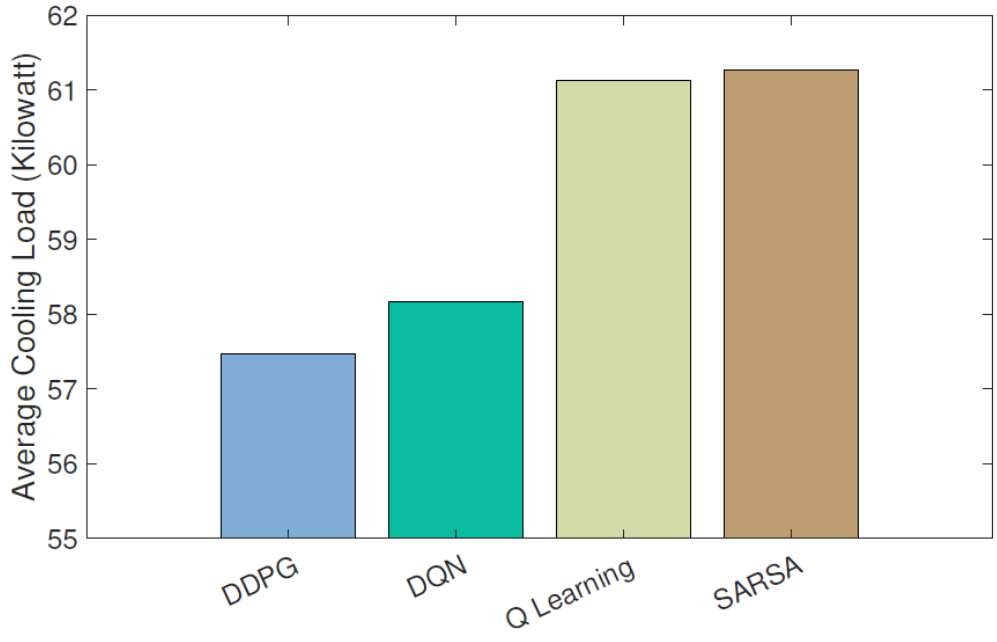


Figure 2-4: Average cooling load (cost) for algorithms in the 2019 DDPG paper.

continuous control. Our experiment does not include humidity as an input. Future iterations of the project will continue to experiment with this algorithm to test its viability.

2.2.3 HVAC Control in an Office Building (2018)

Unlike the previous cases, the 2018 development by Gonzalez-Briones et. al. employs a holistic SHEMS approach instead of an HVAC-focused one [7]. However, I wanted to present at least one development which tested in a real-world environment. The project incorporated data from sensors placed throughout an office and reported an average energy savings of 41%.

The framework, shown in Figure 2-5, is a multi-agent distributed AI, which was selected for its autonomy and extensibility. Temperature sensors collected indoor and outdoor temperature, while Passive Infrared Sensors (PIR) recorded occupancy data. To account for occupants not at their desks, and therefore outside of the PIR's range, pressure mats were placed at the entrances to rooms in the building. Weather forecasts and occupancy trends were analyzed by a separate agent which learned and coordinated scheduling patterns. A calendar agent was also introduced to account for scheduled vacations in the building.

The case-based reasoning (CBR) agent was responsible solely for learning the employees' occupancy comings and goings. It tracked variables such as whether employees were in the office at all, what time the first employee arrives, and the time that has elapsed since the last employee left. Since human body heat accounts for a significant portion of the indoor temperature of a room, factoring in occupancy contributed heavily to the cost savings of this project.

Another agent, the Manage Workflow agent, decided the order in which commands should be carried out to achieve the most favorable outcomes. Interestingly, one of the overall optimization targets of this development is gradual temperature change, because the researchers found that rapid temperature change is associated with high energy costs.

Although the MAS improved performance, consider the cost associated with implementing such a system in a residential setting. This experiment was conducted with four indoor temperature sensors in each office, as well as an outdoor temperature sensor attached to

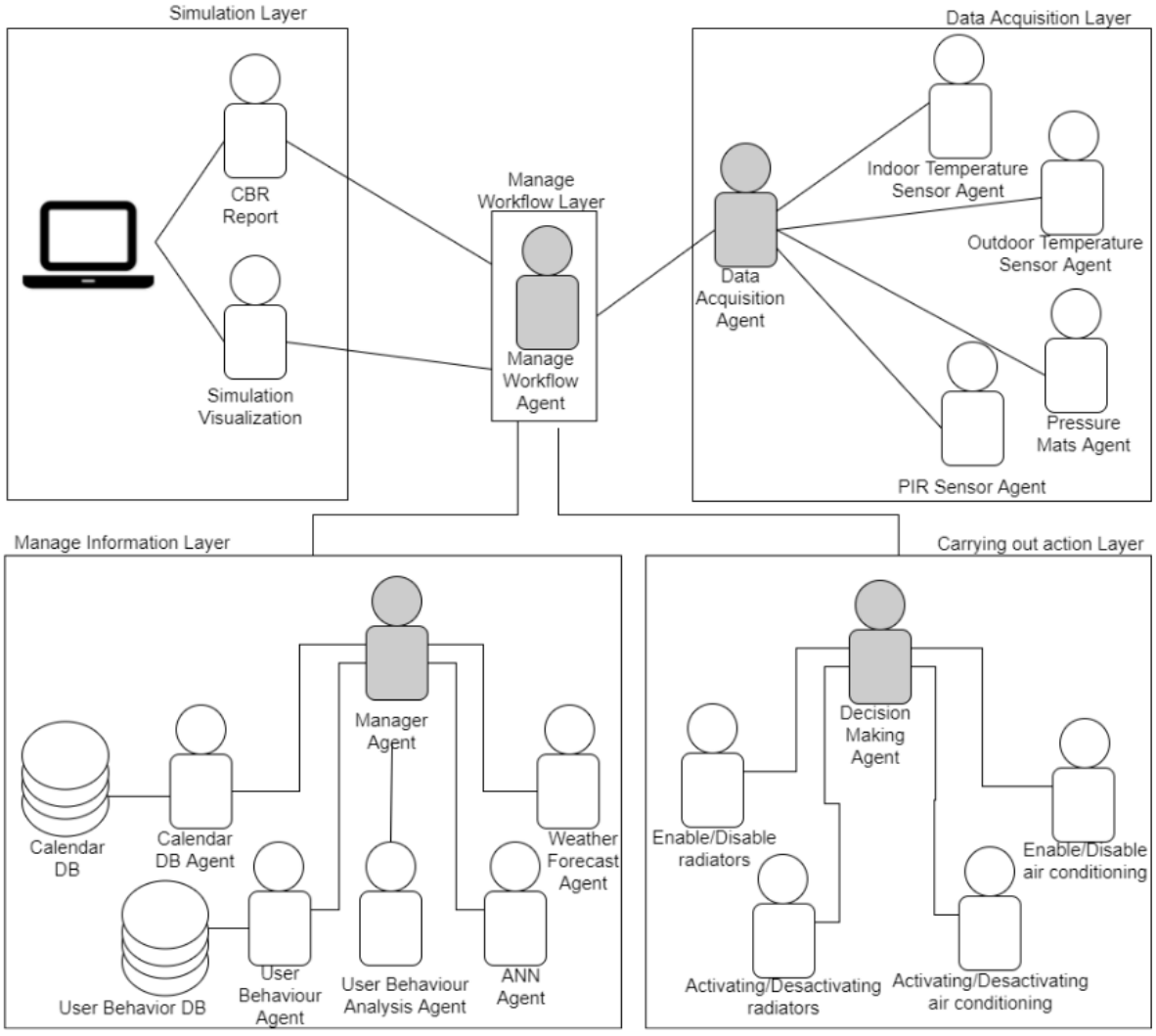


Figure 2-5: Responsibilities of the four agents in the 2018 office deployment.

each window. A PIR sensor was installed at each desk, for an average of 15 in each of seven rooms, as well as a pressure mat at each entrance. The cost to add each sensor, both in cost of materials and installation time, must be recovered in energy savings. Although future developments of the ORNL project described in this work could include a multi-agent system cooperating for energy savings, the experiments currently underway are an attempt to gauge the energy savings that are possible with a minimum of sensors, preferably ones readily available in the average home.

CHAPTER III: APPROACH

Having thoroughly examined the problems associated with HVAC automation as well as attempts by other researchers to address it, attention now turns to the ORNL development itself. The final architecture of the resulting AI HVAC controller is described in Section 3.1. In Section 3.2, the decision making and testing processes that I performed to justify this configuration are recounted. Finally, the simulated and real-time environments in which testing took place are described in Section 3.3.

3.1 Current RL Architecture

Shown below are the state-space, action-space, reward structure, and algorithm parameters chosen for this project which have yielded the most satisfactory results. All the graphs generated in this work use the features shown below, unless otherwise stated.

3.1.1 State

Table 3-1 shows the state used by the model, made up of 7 features. All features were normalized to the interval [0, 1] before an observation was recorded. The “min” and “max” values reported in Table 3-1 are the highest and lowest possible feature values sent to the state before normalization.

Normalization of price to a set of universal boundaries is problematic because a) The system should accept input from a price signal in any units, and b) The utility should have the freedom to raise prices indefinitely. Here, prices fluctuated between 0.05 $/kWh and 0.25 $/kWh, so the normalization interval for price was [0,1].

3.1.2 Actions

The action space used “On” and ”Off” commands for each zone, representing 2^z actions, where z is the number of zones. The two-zone action space is shown in Table 3-2.

“On” and “Off” actions are simplified versions of the actual commands interpreted by the HVAC. If the command given is “On”, the AI transmits a setpoint that is below the current

Table 3-1: The state used in this work's two-zone control development.

Feature	Title	Function	Min	Max
1	Zone 1 Temperature	Zone 1 thermostat temp(t) – Customer High 1 (t)	-15	10
2	Zone 2 Temperature	Zone 2 thermostat temp(t) – Customer High 2 (t)	-15	10
3	Outdoor Temperature	Outdoor thermostat temp(t)	-10	40
4	Price 1	Energy price(t), in $/kWh.	0	1
5	Price 2	The energy price in 5 minutes	0	1
6	Price 3	The energy price in 15 minutes	0	1
7	Price 4	The energy price in 30 minutes	0	1

Table 3-2: Action options in 2-zone testing.

	Zone 1	
Zone 2	**Off**	**On**
Off	0	2
On	1	3

indoor temperature. If the command is "Off", the setpoint delivered is one that is higher than the current indoor temperature. The AI selects from either Customer High or Customer Low when choosing these setpoints.

3.1.3 Reward

The reward structure employed by the AI is shown in Equation 3-1.

$$R_t = -100 * \text{Cost of previous cycle} - (pu1 + pl1 + pu2 + p12), \text{ where}$$

```
pu1 = Zone 1 temp - Customer High 1 if Zone 1 temp > Customer High 1, else 0
pl1 = Customer Low 1 - Zone 1 temp if Customer Low 1 > Zone 1 temp, else 0
pu2 = Zone 2 temp - Customer High 2 if Zone 2 temp > Customer High 2, else 0
pl2 = Customer Low 2 - Zone 2 temp if Customer Low 2 > Zone 2 temp, else 0.
```

Equation 3-1: Reward function used in the two-zone HVAC development.

Like the cases studied in Section 2.2.1 and 2.2.2, the reward structure contains an energy cost term and a comfort violation term.

3.1.4 Algorithm Structure

The algorithm behind the learning done in this development is a Deep Q Neural Network (DQN). Its framework was inspired by the 2015 work done by Mnih et al [10]. DQN attempts to combine RL with a deep convolutional neural network. Pseudocode is shown in Figure 3-1 to describe its behavior during training.

Neural networks need a representative set of samples to effectively train, but RL only collects one sample at a time. Therefore, no learning takes place until a set of initial iterations have been collected, here called a "mini-batch". We used a mini-batch of 200 iterations to prime the network. Table 3-3 shows some of the parameters used in this setup.

Two networks are utilized during training, one "evaluation," or Q, and one "target," or $\hat{Q}$. Mnih et. al. were able to show that the use of two networks could offer stability and reduce potential oscillations during training [10]. Figure 3-2 shows this architecture.

The agent's experience at each timestep is the set (s_{t-1}, a_{t-1}, r_t, s_t). This tuple is stored in replay memory and a uniform random mini-batch of tuples is chosen from this memory and used

$MB \leftarrow$ Reserve and Initialize replay memory
$Q(s,a;\theta) \leftarrow$ Initialize the estimation network with random weights θ
$\hat{Q}(s,a;\hat{\theta}) \leftarrow$ Initialize the target network with random weights $\hat{\theta}$
$N_{episodes} \leftarrow$ Maximum number of episodes
$TS_{max} \leftarrow$ Maximum time-steps per episodes
$k \leftarrow$ Decision time interval

for $episode = 1$ to $N_{episodes}$ do
    $Env \leftarrow$ Reset building environment
    $S_{pre} \leftarrow$ GetIntialObservations(Env)
    $a \leftarrow$ GetInitialAction(Env)
    for $t_s = 1$ to TS_{max} do
        if $t_s \mod k = 0$ then
            $S_{curr} \leftarrow$ GetCurrentObservations(Env, t_s)
            $r \leftarrow$ CalReward$(Env, S_{pre}, a, S_{curr})$
            $MB \leftarrow$ StoreTransition$((S_{pre}, a, r, S_{curr}))$
            Sample random mini-batch $(s_j, a_j, r_j, s_{j+1}) \leftarrow MB$
            Calculate $y_j \leftarrow \begin{cases} r_j \text{ if episode terminates at } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \hat{\theta})] \text{ Otherwise} \end{cases}$
            Calculate loss $L \leftarrow \sum_j (y_j - Q(s_j, a_j; \theta))^2$
            Update $\theta \leftarrow \theta - \alpha \frac{\partial L}{\partial \theta}$
            Every Δt_c time-steps copy $\hat{\theta} \leftarrow \theta$
            $S_{pre} \leftarrow S_{curr}$
        end if
        Execute simulation SimulateBuilding$(Env, t_s, action)$
    end for
end for

Figure 3-1: Algorithm pseudocode for the evaluation and target networks.

Table 3-3: DQN Parameters used.

NN Learn Rate	0.01
Input Layers	1x7, one input per feature
Hidden Layers	2x10 (2 layers with 10 neurons each)
Output Layers	1x4, one output per action
Reward Decay (γ)	0.9
Epsilon (ε)	0.1
Memory Size (Experience Replay Memory)	20,000
Batch Size	32
Initial Iterations	200
Δt_c	300
Optimizer	AdamOptimizer

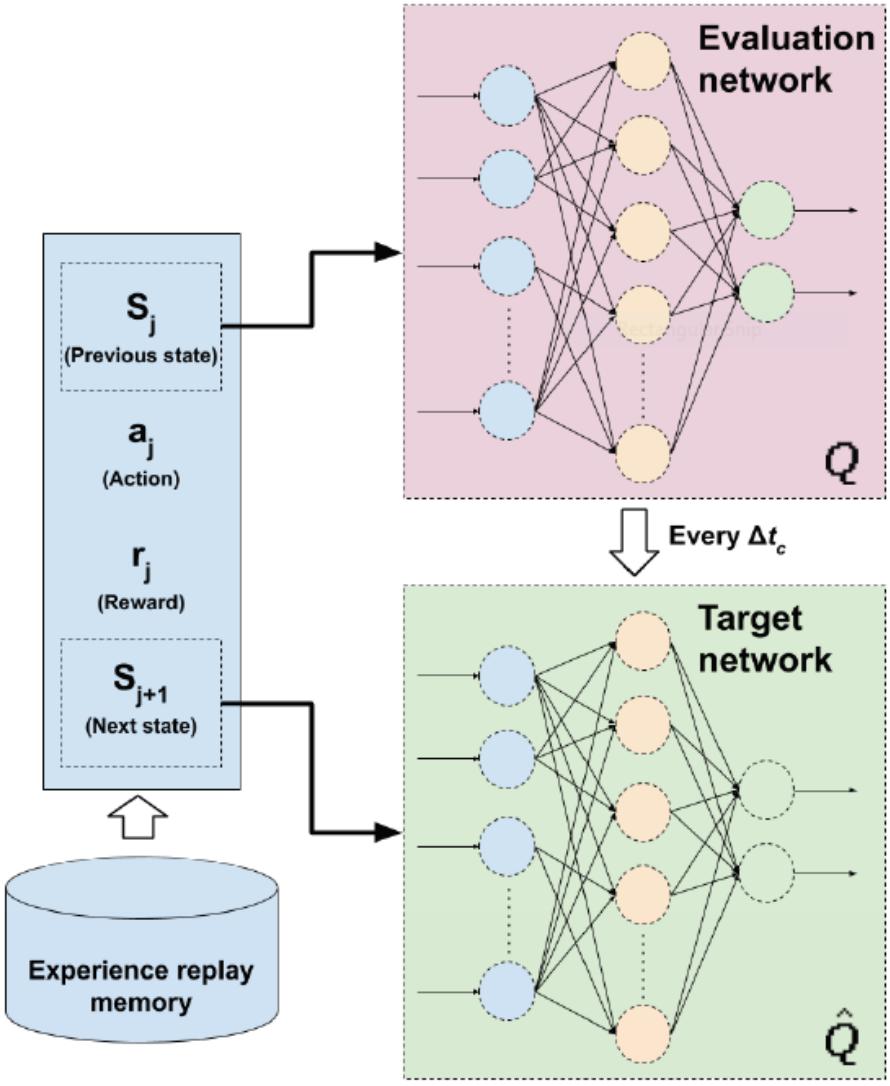


Figure 3-2: DQN neural network structure using evaluation and target networks.

for the mini-batch update during learning. A forward pass through the evaluation network estimates the Q Table. The loss calculated from 32 samples (one batch) is averaged and the set of weights θ is updated using back propagation. After every Δt_c updates of Q, the evaluation network is copied to the target network $\hat{Q}$. The mean squared error between target network $\hat{Q}$ and the output of the evaluation network Q represents the loss function that the algorithm is attempting to minimize. This function is given by

$$L = \frac{1}{2n}\sum_{i=1}^{n}[Q^*(s_t, a_t^i) - Q(s_t, a_t^i)]^2 \quad ,$$

Equation 3-2: DQN loss function.

where *n* represents the number of actions, and the Q tables from each network are being compared to calculate mean squared error. This two-network DQN method proved useful when applied to our system.

3.2 Parameterization

The following section highlights my personal contributions to the ORNL development. The inclusion of setpoint governance, the inclusion of the comfort penalty, and the relative temperatures function will all be described and justified. First, a baseline run will be executed which gives the reader a sense of the cost-saving ability of the algorithm before my modifications were made. Then, adjustments will be added one by one until we arrive at the optimized model as it is used in this project. In Section 3.2.5, I explore other potential improvements that were not integrated into the final project, but still warrant examination.

Note that for each figure, "AC state by Zone 1" does not refer to the physical "On" / "Off" behavior of the AC. Instead, it refers to the action that the learner selected, whether "On" or "Off", and sent to the HVAC in the form of a setpoint.

3.2.1 Baseline Comparison

The naïve, fixed-setpoint baseline model represents the traditional thermostat available in most homes. With no AI to guide its decisions, the AC activates only when the indoor air

temperature crosses a fixed threshold. Figure 3-3 shows the performance of a baseline model that has been set to 24° C throughout the month of July. The monthly cost to operate this model was $47.55. Figure 3-4 shows another baseline run, this one with a threshold of 22.5 °C. This 1.5 degree setpoint reduction led to a 16.6% cost increase, to $55.45. A lower setpoint consistently resulted in higher cost.

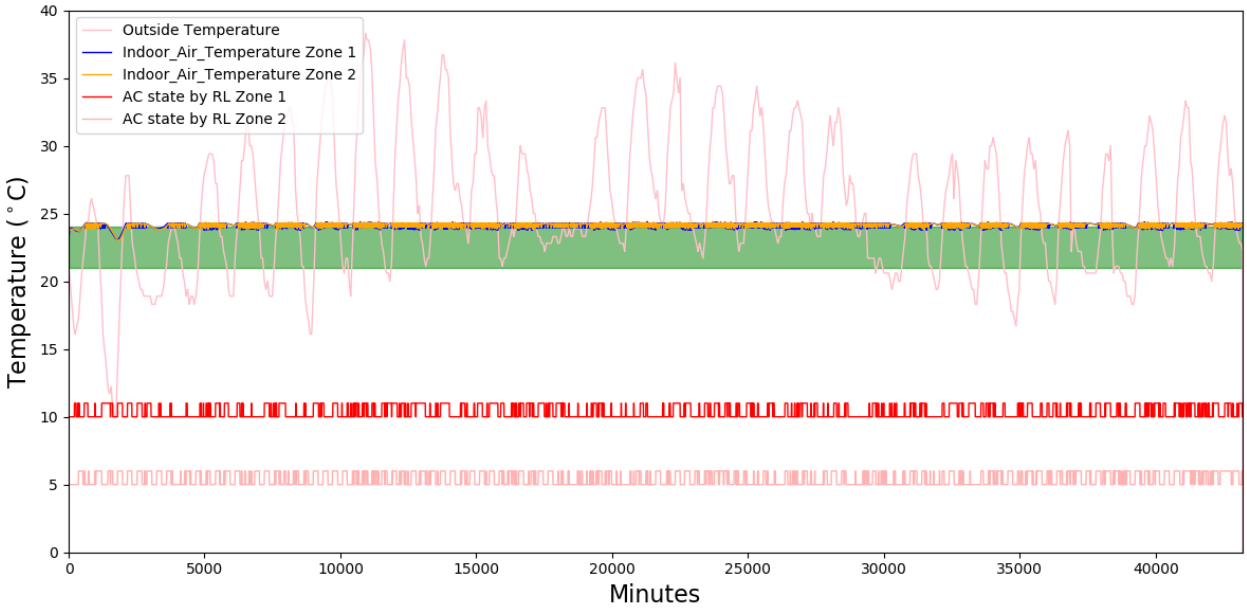
Each episode is a set of iterations that lasted a month. Since we used an MCT of 15 minutes, an episode is 30 days of data, or 2,880 steps. Our learning model with smart controls was trained for 20 episodes in the month of June, then tested on July. Its performance is shown in Figure 3-5, demonstrating a 30-day cost of $37.39. AI controls have saved this homeowner $10.16 for the month of July, or 21.4%.

3.2.2 Setpoint Governance

Having confirmed that energy costs can be reduced by using our RL model, the model is now subjected to a new metric, Minutes Outside Comfort (MOC). Whenever the indoor temperature goes over Customer High plus 0.5 degrees, the learner is said to have violated the comfort constraint and MOC increases by one minute. The learner in Figure 3-5 spent 215 minutes outside comfort, or 0.5% of its total 30-day run.

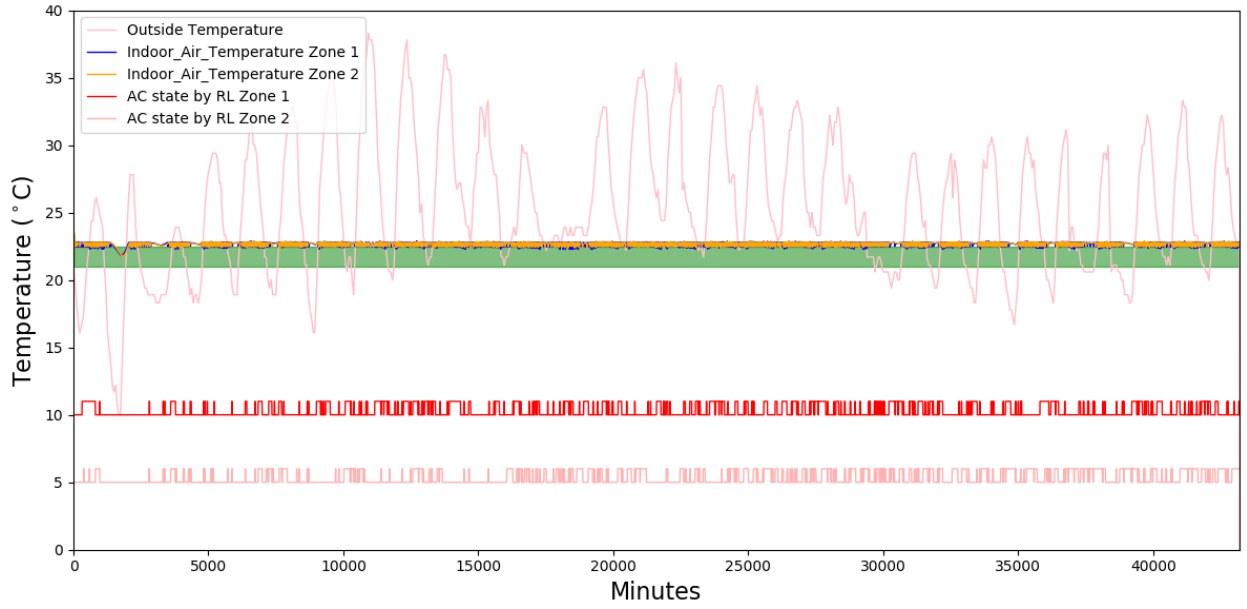
Since user comfort is one of the goals of the AI, testing was performed to see if the model could be improved by adding a hard setpoint constraint to operation, with the goal of reducing MOC. The model shown in Figure 3-6 is subject to setpoint governance, whereupon it is incapable of submitting a setpoint to the HVAC which is higher or lower than the customer's tolerable comfort range. No other changes were made to the model. After 20 June episodes of training, the July performance is plotted.

The model produces a significantly reduced cost of $22.40, and its MOC has been reduced to zero, thereby showing an improvement. This test, among others, showed that narrowing the range of states available to the learner is conducive to faster learning, and that setpoint governance through hard constraint warrants a permanent place within the model. Of all the modifications made over the course of the project, setpoint governance is the one which produced the greatest reduction in cost.



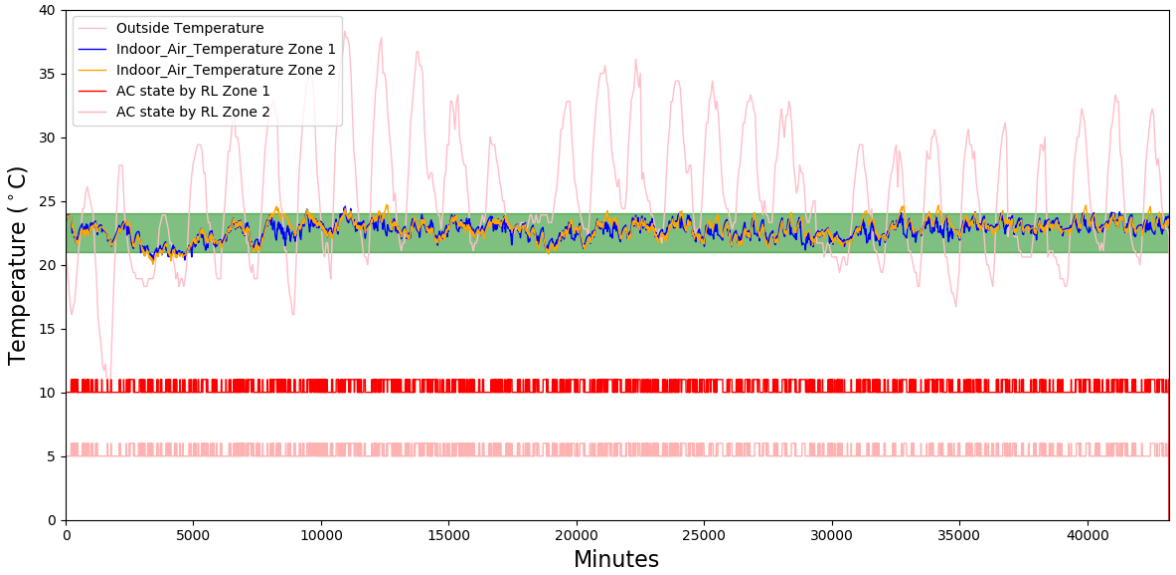
30 Day Cost: $47.55

Figure 3-3: Baseline model with a fixed setpoint of 24 °C.



30 Day Cost: $55.45

Figure 3-4: Baseline model with a fixed setpoint of 22.5 °C.

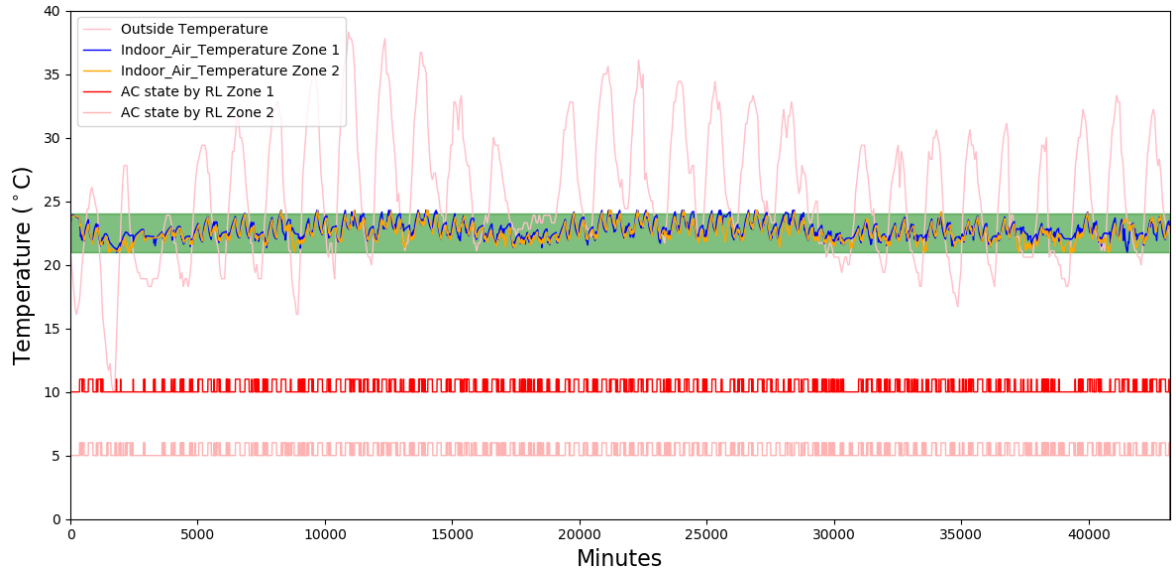


Training Schedule: **20 June, 1 July**

30 Day Cost: **$37.39**

Minutes Outside Comfort (MOC): **215**

Figure 3-5: RL model with smart controls.



Training Schedule: **20 June, 1 July**

30 Day Cost: **$22.40**

Minutes Outside Comfort (MOC): **0**

Figure 3-6: RL model with hard setpoint constraint.

3.2.3 Comfort Penalty

For further cost reduction, it was hypothesized that with comfort now governed by hard constraint, the reward function might reflect only the price of energy as an input, and the part of the reward function that encouraged comfort could be removed. The old and new reward functions are shown side by side in Table 3-4. The results of testing with this model were unexpected, and are shown in Figure 3-7.

The outcome is a higher cost of operation, $45.94. Apparently, without the comfort penalty, the AI is biased to cycle naturally at the top of the user's comfort range. It could be that with only the price to consider, the learner determined that activating the AC for a full cycle was never less expensive than cycling at the customer's upper bound, and never escaped this assumption to capture long-term rewards. I created a heavily discretized environment which mimicked our HVAC simulation and used dynamic programming (DP) to create a heatmap of the values of each indoor temperature at each timestep. In Figure 3-8, the light blue squares represent indoor temperature, the red outdoor temperature line is held constant at 31º C, and the price increases by 5x halfway through the run to incite a precooling event. The green boxes represent state values of indoor temperatures over time– darker shades represent higher valued states.

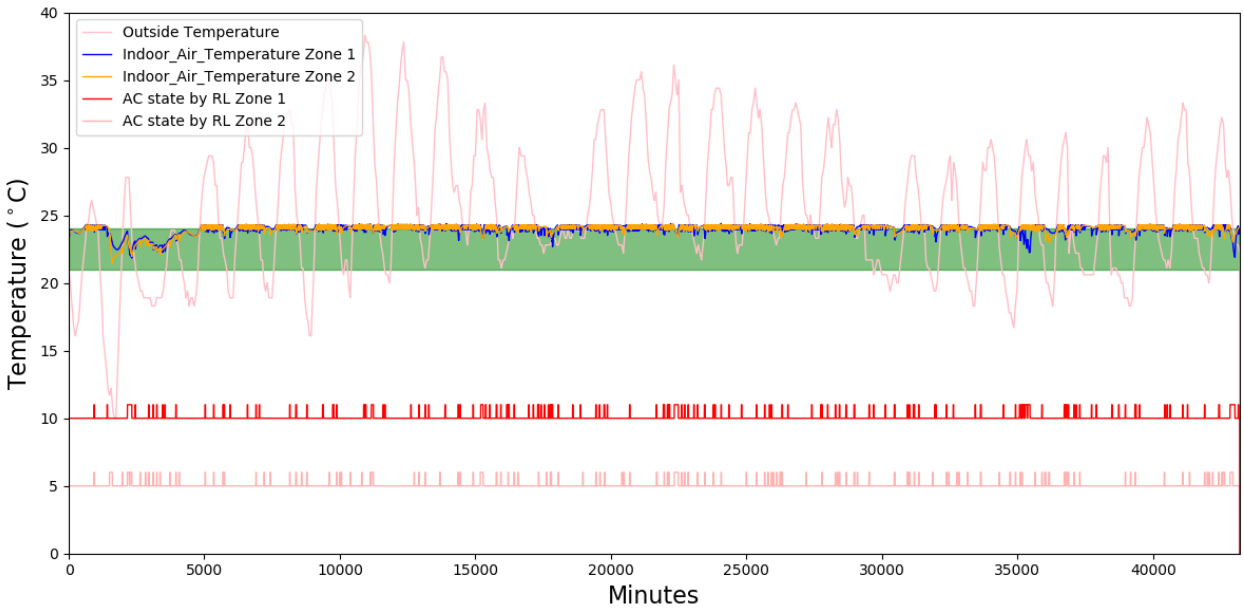
DP has drawn similar blue lines on both figures, showing that precooling is, in fact, the ideal path. This demonstrates that the behavior of our learner without comfort penalty (which does not precool) is aberrant, because the optimal cost-saving decision is to precool even without the comfort penalty. Thanks to the heatmap, we can see the logic behind this – the learner wants to avoid the area of low-value states immediately following the price increase and close to Customer High. I was unable to reproduce this result outside of DP. Whatever the reason for the higher cost, there is something in the comfort penalty portion of the reward function which gives the learner a more practical and useful clue as to the values of states within its reach. This phenomenon warrants more study, and the comfort penalty, as it is shown in Table 3-4, remains.

3.2.4 Relative Temperatures

To observe one problematic feature of the original RL model, consider Figure 3-9. Here, the customer preferences, shown as black lines in the figure, have been changed from 20-23 °C

Table 3-4: Reward functions with and without Comfort Penalty.

Reward with Comfort Penalty	Reward Without Comfort Penalty
*-1 * (100 * (Price * Consumption) + (Amount in degrees over comfort) + (Amount in degrees under comfort if AC is running))*	*-100 * (Price * Consumption)*



Training Schedule: **20 June, 1 July**

30 Day Cost: **$45.94**

Minutes Outside Comfort (MOC): **0**

Figure 3-7: RL model with setpoint governance and no comfort penalty.

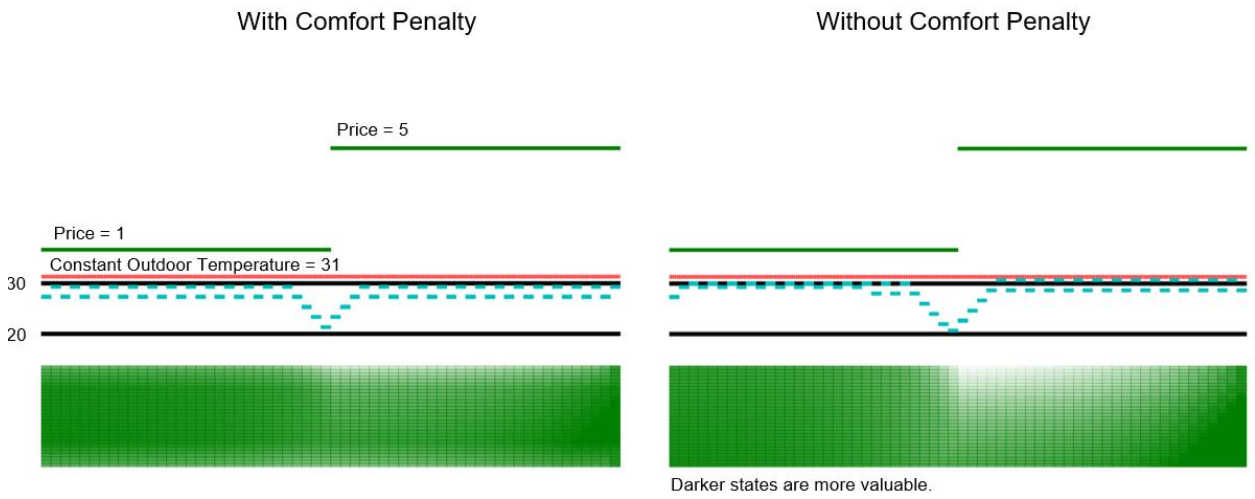
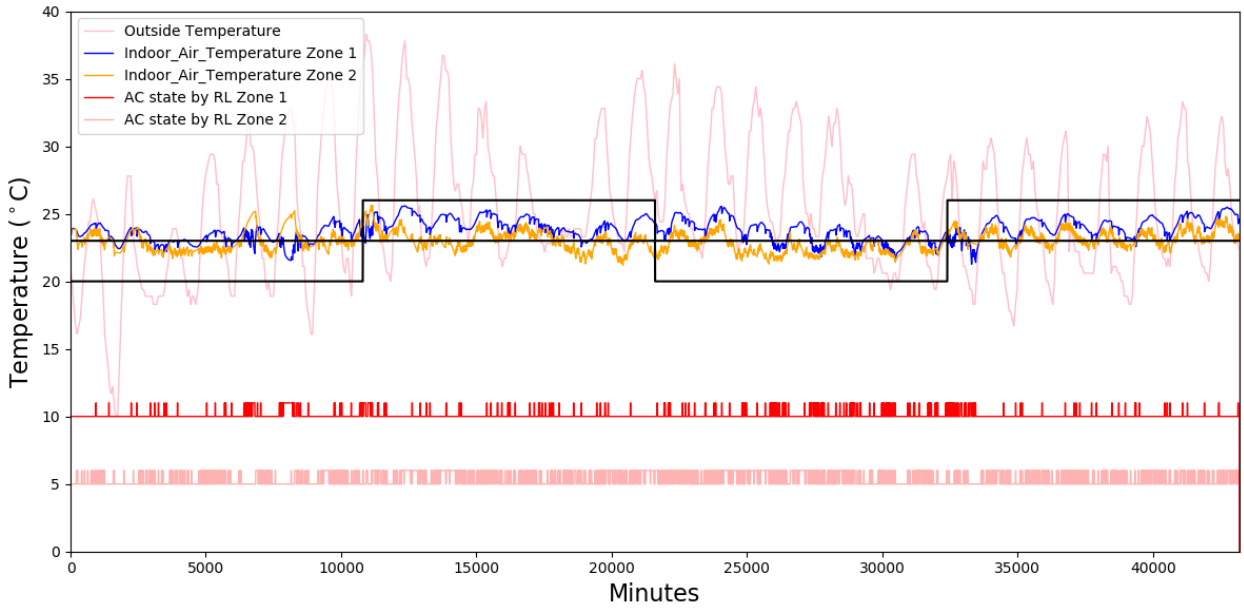


Figure 3-8: DP results with and without comfort penalty.



Training Schedule: **20 June, 1 July**

30 Day Cost: **$41.75**

Minutes Outside Comfort (MOC): **11,607**

Figure 3-9: RL absolute temperature model with changing customer preference zones.

to 23-26 °C and back four times over the course of a month. As before, training occurs in June and testing in July. Setpoint governance has been disengaged.

The results of this preference-changing experiment are poor: a cost of $41.75, and an MOC of 11,607 minutes. This learner spent 26.8% of its time during the month of July outside user comfort. Whenever the preferences change, the model suddenly encounters low reward states that used to be high, and high reward states that used to be low. The effect on the learner is similar to what happened when switching from a 6-sided to a 10-sided die during the example in Section 1.6.2. With indoor temperature as part of the observation, but not the customer's preference, the learner has to re-learn its state transition probabilities whenever the preferences change. Fortunately, there is a simple fix, and my solution came to be known as the relative temperatures model.

Figure 3-10 shows how a preference change of 3 °C is seen by the learner when absolute temperatures are recorded as a feature of the state. The values to the left of each colored bar correspond to how the learner encounters each state under absolute and relative configurations. It was hypothesized that the value of a state given indoor temperature was more closely tied to a function, namely the difference between the indoor temperature and Customer High. In the rightmost part of Figure 3-10, the preference shift is associated with an accompanying positional shift of the learner.

By recording the indoor temperature into the state as indoor temperature minus Customer High, the model in Figure 3-11 reports a lower cost ($28.43) and a greatly reduced MOC (426 minutes). Note the visual improvement: the path drawn by the indoor temperature of the learner is doing a much better job of staying within the range of user comfort.

The MOC can be reduced still further by using the hard setpoint constraint described in Section 3.2.2. The run in Figure 3-12 includes both the relative temperature modification and the hard setpoint governance. The result is a comparable cost to Figure 3-11 ($31.12) and a MOC that has been reduced even further (198). Here, any minutes spent outside comfort occur during preference changes, when indoor temperature is reducing or increasing to its new comfort zone. These preference changes are defined by a pre-set schedule chosen by the homeowner, and the changes may occur multiple times per day. For this reason, the indoor temperature should be recorded by the state as a relative, and not absolute, value.

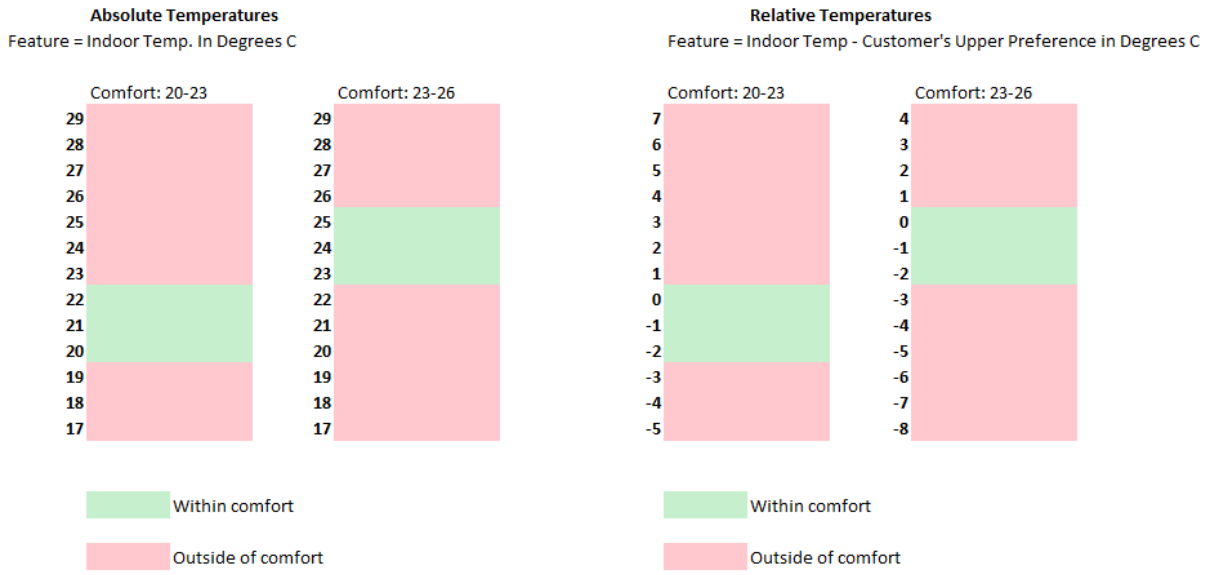
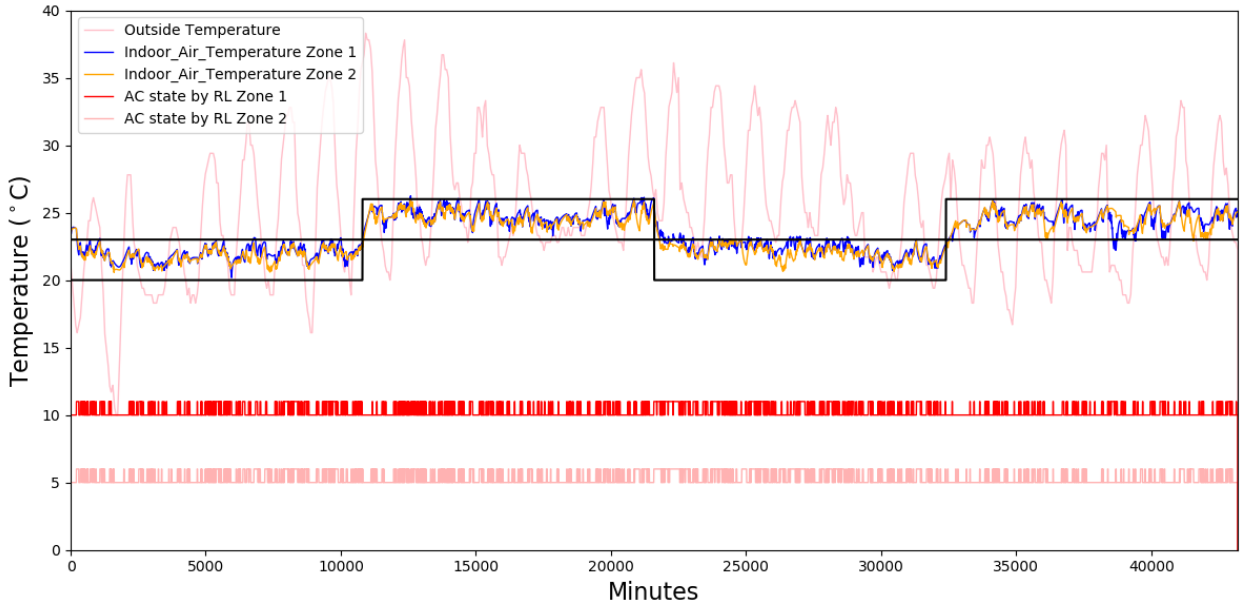


Figure 3-10: Absolute and Relative temperatures seen by learner.

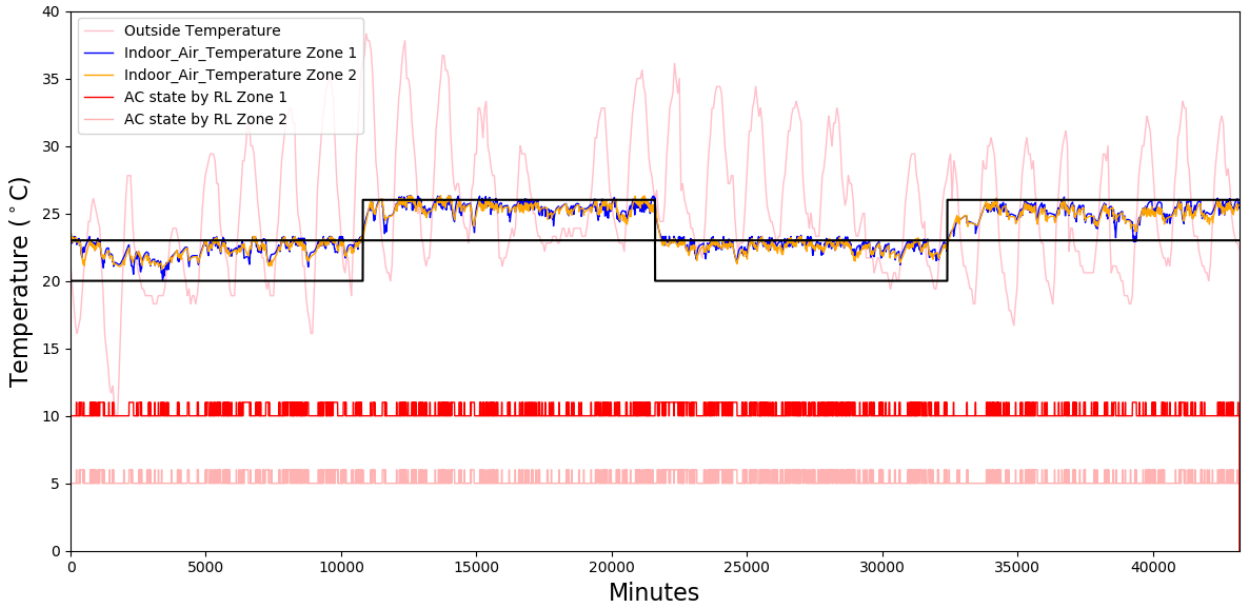


Training Schedule: 20 June, 1 July

30 Day Cost: $28.43

Minutes Outside Comfort (MOC): 426

Figure 3-11: RL model with relative temperature recordings.



Training Schedule: **20 June, 1 July**

30 Day Cost: **$31.12**

Minutes Outside Comfort (MOC): **198**

Figure 3-12: RL model with relative temperature and setpoint governance.

As part of continued quality assurance testing, I wanted to make sure that my method of relative temperature recording was the best of all possible options. I tested four single-zone models, each using a different approach to recording indoor temperature, and recorded the MOC present in each run. Each model was trained for 100 days and the user comfort preference was altered every 25 days, from 19-24 ° C to 16-21 ° C and back again. Table 3-5 gives the results of these tests. In order to give a true comparison between the models, setpoint governance is disabled for these models.

One particularly telling result of this test is how much the training time is hurt by increasing the number of features. Without explicitly telling it the relationship between Indoor Temperature and Customer High, the learner spends extra effort discovering this relationship on its own. The “2 Features” model shows the lowest MOC, but it takes an additional 20 days to converge. Since fast training time is a priority, the relative temperatures model was declared the winner of this experiment, and testing continued using the relative temperatures method.

3.2.5 Other Potential Improvements

In searching for the lowest monthly operating cost, several configurations were tested that did not become part of the final project. Some of them are listed in this section. The overarching goal of these tests was to find more or different information that we could give to the learner that would improve its post-convergence average reward.

3.2.5.1 Comfort Tolerance Model

One of the earliest models tested was the comfort tolerance model, which divided the comfort zone into tolerance zones that depended on price. Early on, it was observed that a lower cycling setpoint is associated with a higher price, due to the greater pull of the outdoor temperature. Figure 3-13 shows the cycling behavior of four learners, all of which use a Customer High of 25° C. Indoor temperature is shown in blue. As the constant outdoor temperature increases, the number of cycles increases as well as the cost. In other words, the AC must work harder to maintain indoor temperature as the outdoor temperature increases.

For the comfort tolerance model, Customer High is permitted to "relax" by two degrees whenever the price crosses from low to high over a predetermined price threshold. For the rule-based model shown in Figure 3-14, the blue indoor temperature line shows that the AC is cycling

Table 3-5: Relative Temperatures experimental configurations.

Model Name	**Temperature Features**	**Setpoint Governance**	**MOC**	**Approximate Apparent Convergence (Days)**
Absolute Temperatures	[Indoor Temperature]	OFF	15,639	30
2 Features	[Indoor Temperature, Customer High]	OFF	12,557	50
Relative Temperatures	[Indoor Temperature – Customer High]	OFF	14,083	30
3 Features	[Indoor Temperature, Customer High, Customer Low]	OFF	13,032	60

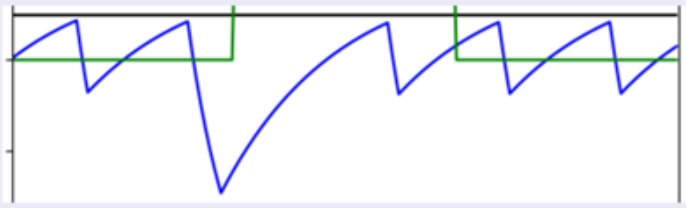
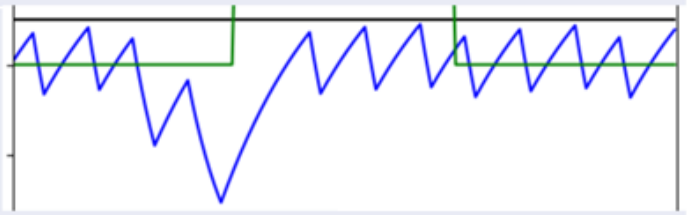
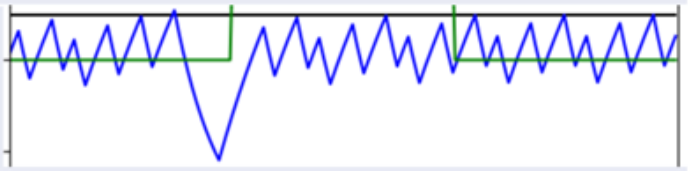
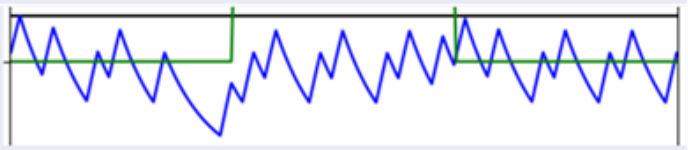
Constant Outdoor Temp(° C)	Indoor Temp Path
26	
28	
31	
35	

Figure 3-13: Increased cycling at higher temperatures.

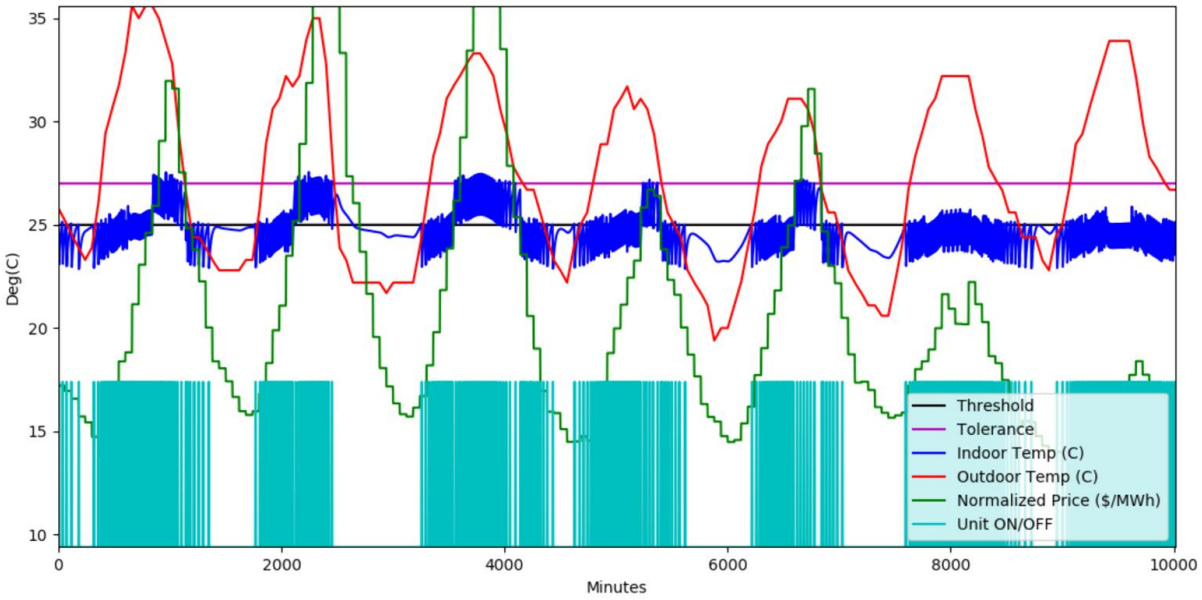


Figure 3-14: Comfort Tolerance Model results.

at 25° C when the price is low, and rising to cycle at 27° C whenever the normalized price (the green line) exceeds its threshold. The price has been normalized so that the black line is both the price threshold and Customer High.

Intuitively, if the comfort zone of the user were allowed to expand during periods of high price, the system could save costs. In practice, this design was problematic for three reasons: 1) a price threshold must be defined in this model that cannot be generalized over any price input, 2) the customer will observe temperatures and setpoints on their thermostat outside of the ones they select, and 3) it is more straightforward to analyze the behavior of the learner when it has the freedom of a single, uniform comfort zone to navigate. For these reasons, the comfort tolerance model was abandoned in favor of the fixed threshold comfort zone model previously described.

3.2.5.2 AC Status as a Feature

The blue indoor temperature line in Figure 3-15 is an aid to visualizing the effects of Thermal Mass. This rule-based AC unit is programmed to switch between "Off" and "On" every 160 minutes. The red line represents outdoor temperature. Choosing a different action as the previous cycle is associated with a large temperature change, while choosing the same action as the previous cycle is associated with a smaller change. This is because the wall thermostat only measures indoor air temperature in the house. A second temperature is present, the Thermal Mass made up of the combined temperatures of all the objects in the room. All else being equal, Thermal Mass will always lag behind indoor air temperature.

The fact that Thermal Mass cannot be measured leads to a unique problem when using an RL controller. In Figure 3-16, we imagine two learners that begin at the same indoor temperature point. Learner 1 cycles naturally in short bursts, while Learner 2 runs continually before turning "Off" at Customer Low. By our state definition, they arrive at the same indoor air temperature point. However, the true indoor temperatures of the learners are different. In order to make a "smart" decision, the learner must estimate with some confidence the probability of its next state. Unfortunately, there is a wide variance of possibilities in this case that that next state could be, and no clue contained in the state itself.

The first strategy to compensate for this was the inclusion of the action of the previous cycle as a feature, called "AC Status". The feature would only ever be 0 or 1, and would tell the learner if it was about to choose a different action or the same action as the previous cycle. To

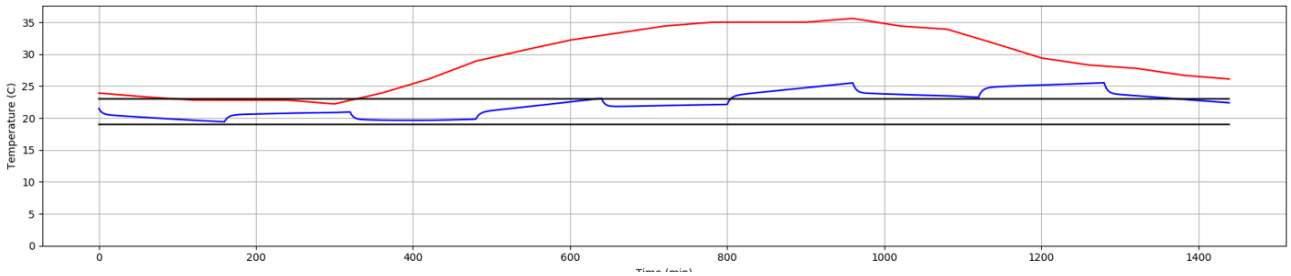


Figure 3-15: Rule-based model with 160 minute cycles.

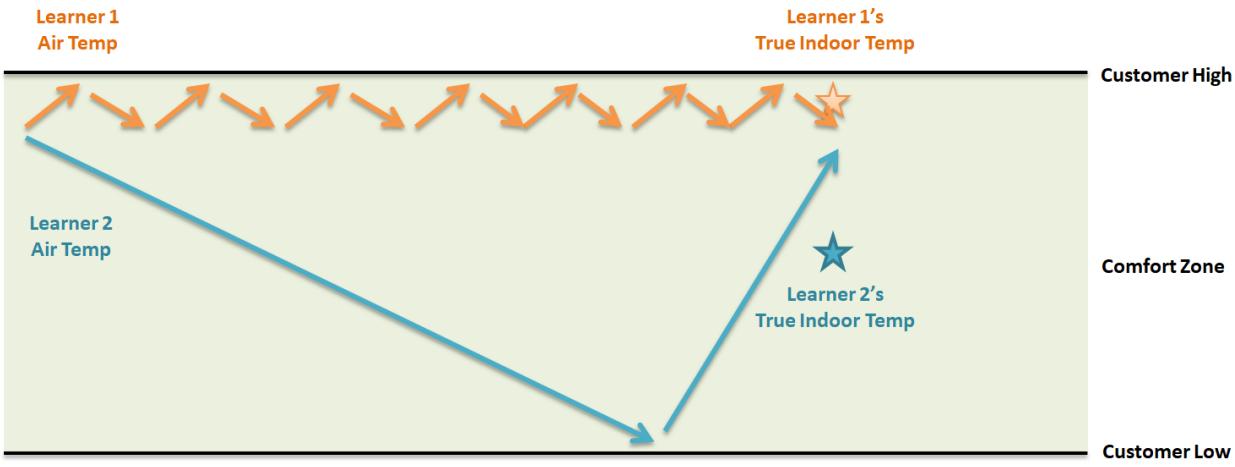


Figure 3-16: Effect of Thermal Mass on next state.

keep the number of features constant, I replaced the first and second price features with AC Status of the two zones. The results of incorporating this feature are shown in Table 3-6. I did not observe a significant improvement in performance, so I moved on to another strategy.

3.2.5.3 Point-Slope Method

A second option that I hoped would address Thermal Mass was the Point-Slope Method. As before, I hoped to modify the state so that it would include not just the current indoor temperature, but information about the last few indoor temperatures. This idea was inspired by the 2013 Atari 2600 RL experiment, where 4 historical frames of each game were combined to form an observation with some velocity information, rather than a static frame of the game [35]. In a similar fashion, the inclusion of only instantaneous values into our state was making the system less of an MDP. A set of data could be collected that spanned a length of time, then a linear regression line could be taken of these points and the slope of that line could be used as a feature. The advantage of this method is that any lookahead or lookback length could be used at a universal cost of two features. This method was used to form the set of features in Table 3-7. So that the number of features would not change, the slope of Zone 2 was not included in the set.

Table 3-8 gives information on how the model performed against the AI in a five month test. The method did not produce substantially improved results. It is apparent that whatever information the learning algorithm needs, it learns from either set of features.

3.2.5.4 PAPA Model

The PAPA (Price And Price Alone) Model was developed after a team member presented a rule-based model that resulted in a July cost of $16.21 for the month. To date, it represents the cheapest July run observed on any model and is shown in Figure 3-17.

The rule-based logic is simple: When the price is low, the setpoint is Customer Low. When the price is high, the setpoint is Customer High. Learning does not take place in this model. In effect, the model is enforcing the ideal precooling conditions by running continuously as long as the price is below a threshold. The setpoints take care of comfort violation. Since there is setpoint governance and only two options for setpoint, the AC is never operating out of comfort. The model was able to work for cold months, too, beating our average cost scores for all five test months. The results of those tests are shown in Table 3-9.

Table 3-6: Cost of AC Status inclusion vs. AI model.

2018 Month	AC Status Model Cost ($)	AI's Cost ($)
May	11.60	9.87
June	21.99	20.81
July	29.54	32.40
Aug	27.12	26.41
Sept	17.32	14.19

Table 3-7: Point-Slope model set of features.

Feature Number	Description	Parameters
0	Zone 1 temperature(t)	
1	Slope of linear regression line made from last *n* Zone 1 temperatures	n = 60
2	Zone 2 temperature(t)	
3	Outdoor temperature(t)	
4	Slope of linear regression line made from *n* forecasted outdoor temperatures	n = 60
5	Price(t)	
6	Slope of linear regression line made from *n* forecasted prices	n = 60

Table 3-8: Cost of Point-Slope method vs. AI model.

2018 Month	Point-Slope Cost ($)	AI's Cost ($)
May	9.84	9.16
June	19.23	20.26
July	27.92	30.46
Aug	24.61	23.42
Sept	15.68	15.82

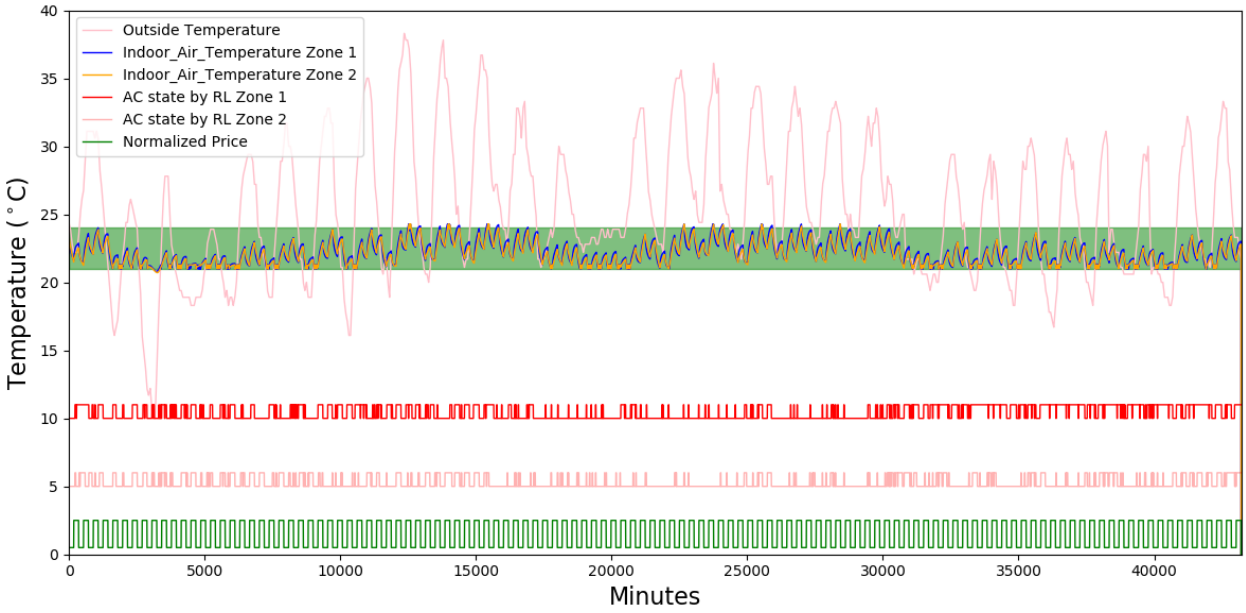


Figure 3-17: Rule-Based Hero Tuning for July.

Table 3-9: Cost of Hero Tuning vs. AI model.

2018 Month	Rule-Based Hero Cost ($)	AI's Cost ($)
May	6.14	9.48
June	12.81	19.78
July	16.21	29.51
Aug	15.18	26.92
Sept	10.47	14.55

Replacing our RL model with this rule-based model is not practical because a cost threshold would need to be specified that was not general over all price signals. However, using this precooling behavior as a target, I hypothesized that using setpoint governance had eliminated the goal of maintaining comfort from the learner's concern, and all that remained was price. An experimental model was developed, PAPA, which would incorporate only two features: The instantaneous price, and the slope of a linear regression line made from the next four cycles of price (See Section 3.2.5.1). Testing was repeated, and the results are shown in Table 3-10.

I observed that this model did well in hot months, but that leaving out information about Thermal Mass, indoor temperature, and outdoor temperature did not seem to help the learner. These results go on to show that the cost difference is not significant when omitting every input except price, confirming the theory that price is the variable contributing the most to monthly cost. Indoor and outdoor temperature effects on this result are small enough that the instantaneous outdoor temperature is a sufficient feature to include without incorporating any forecast values.

3.2.5.5 Time as a Feature

For this experiment, it was hypothesized that the learner might benefit from the predictive advantage of including time of day into the state, in the fashion of the 2017 work by Wei, Wang, and Zhu [2]. The outdoor temperature is expected to increase until noon, peak, then decrease until midnight. In theory, including information about the time of day could help the learner infer Thermal Mass information and would justify the cost of an extra feature. This turned out not to be the case, as the following results demonstrate.

A feature was added to the state which represented time of day rounded to the nearest slice *n*. Experiments were conducted in a single-zone environment discretizing the day into 0, 2, 4, 6, 12, 24, 48, 96, 144, and 288 slices. Each episode in this experiment is a 2-day set randomly selected from 150 days of data. Figure 3-18 shows cumulative reward over 15 episodes using these divisions.

Consistently, adding discretizations hurt the training time. Cumulative reward was highest, and convergence quickest, when zero discretizations were used. With these results, it is apparent that including time as a feature does not help the learner enough to justify the cost of an extra feature, and so testing proceeded without using time as a feature.

Table 3-10: Cost of PAPA vs. AI model.

2018 Month	Cost with PAPA ($)	AI's Cost ($)
May	9.99	9.48
June	24.26	19.78
July	27.61	29.51
Aug	24.88	26.92
Sept	14.43	14.55

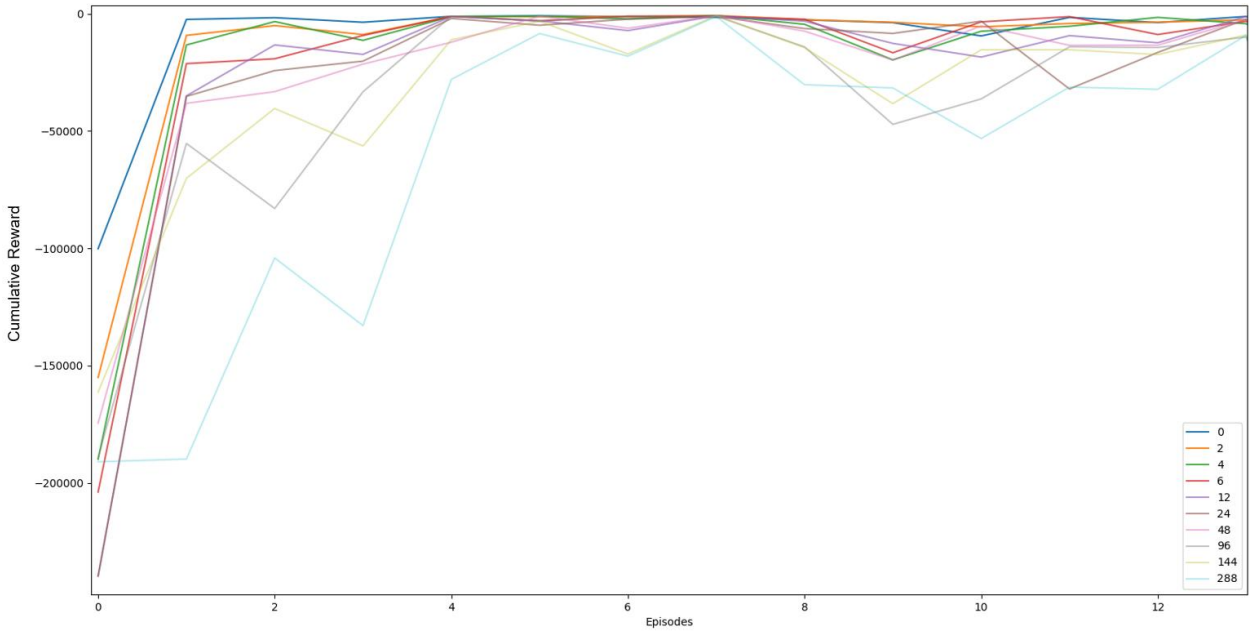


Figure 3-18: Cumulative Reward using varying time discretizations.

3.3 Environment

The Yarnell Station House is a 2400 sq. ft. single family home located on Yarnell Station Road in Knoxville, Tennessee [8]. A photograph of the house is shown in Figure 3-19, and its characteristics are listed in Table 3-11.

The house is divided into two zones, connected by an open stairwell. The zones influence each other with airflow through the upstairs floor and the stairwell, elucidated in Figure 3-20. The simulated building model is considered “gray-box” because the simulation is built on both physical principles and measured data. The model uses a technique called resistance-capacitance (RC) modeling, in which electric analog resistances represent thermal resistances, and electric analog capacitances represent thermal capacity. Figure 3-21 shows the RC model.

The validation results when comparing the temperature predicted by the RC model to the actual, measured zone temperatures are shown in Figure 3-22. With a Mean Average Error (MAE) of 0.499 ° C and a Root Mean Squared Error (RMSE) of 0.619 ° C, the simulated model was declared a sufficient estimate of the real-time house. The use of setpoint governance, that hard constraint that prevents the HVAC from using a setpoint outside of a user’s comfort zone, further increases the level of confidence that a model trained in simulation will suffice inside an actual home.

Note that in both simulation and the physical Yarnell Station House, occupancy is kept at zero. Consistent results must be obtained from real-time testing before we can confidently place the AI into a home with varying occupancy.

Figure 3-19: Photograph of Yarnell Station House.

Table 3-11: Yarnell Station House characteristics [1].

Year Built	2013
Stories	2
Size	223 m^2 (2400 ft^2)
Home Energy Rating Score (HERS)	92
Codes Met	International Energy Conservation Code (IECC)
Occupants	0

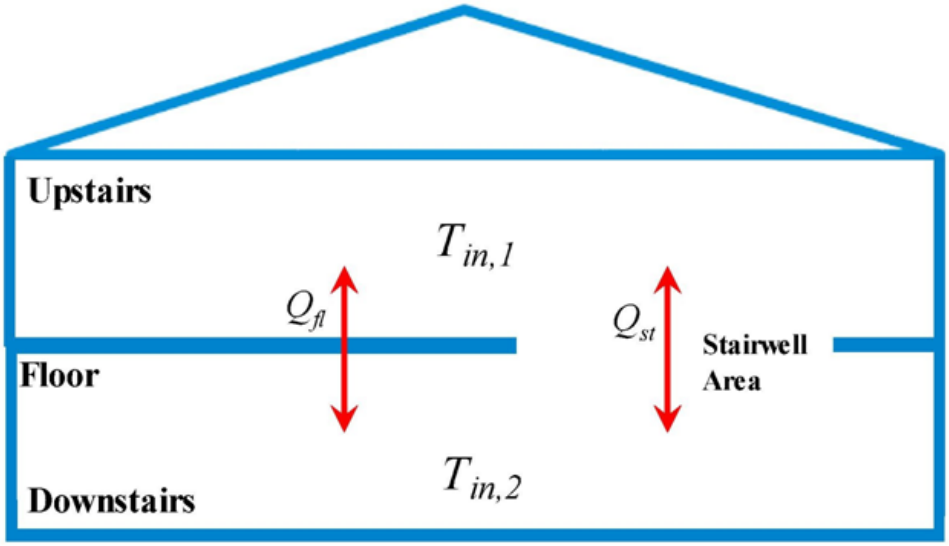


Figure 3-20: Two-zone division in the Yarnell Station House.

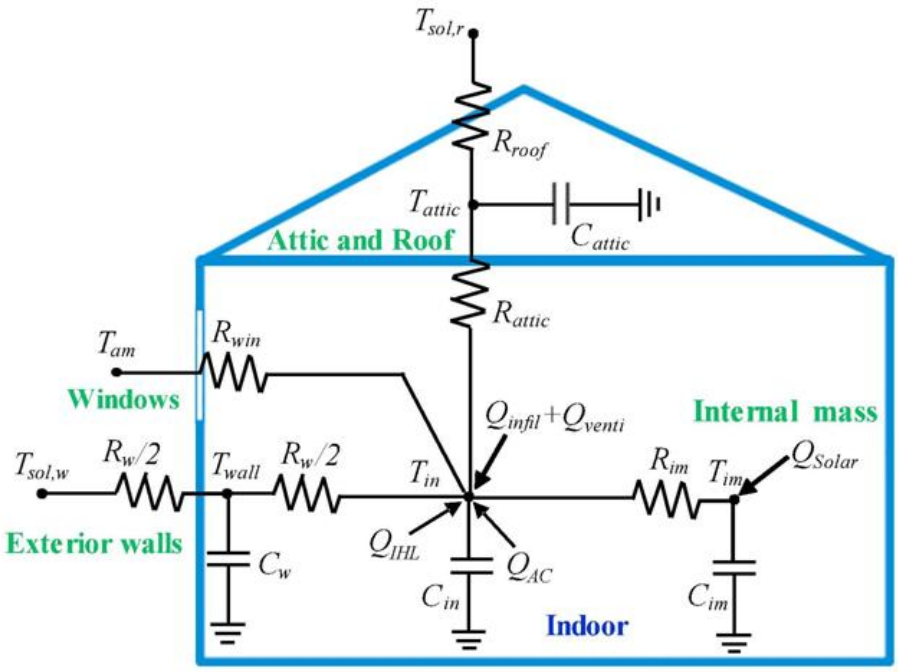
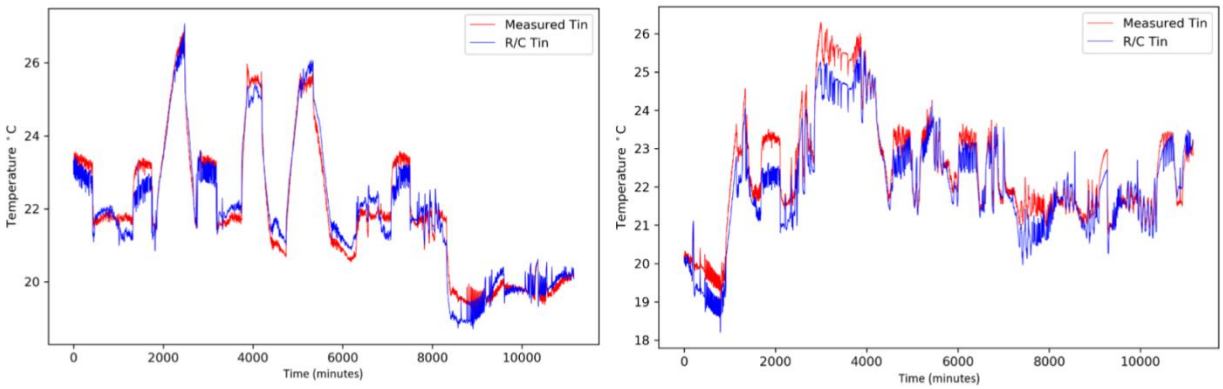


Figure 3-21: RC building model of the Yarnell Station House.



(a) Training RMSE of 0.36°C (96.8°F) **(b)** Validation RMSE of 0.53°C (127.4°F)

Figure 3-22: Yarnell Station House model validation [8].

CHAPTER IV: RESULTS

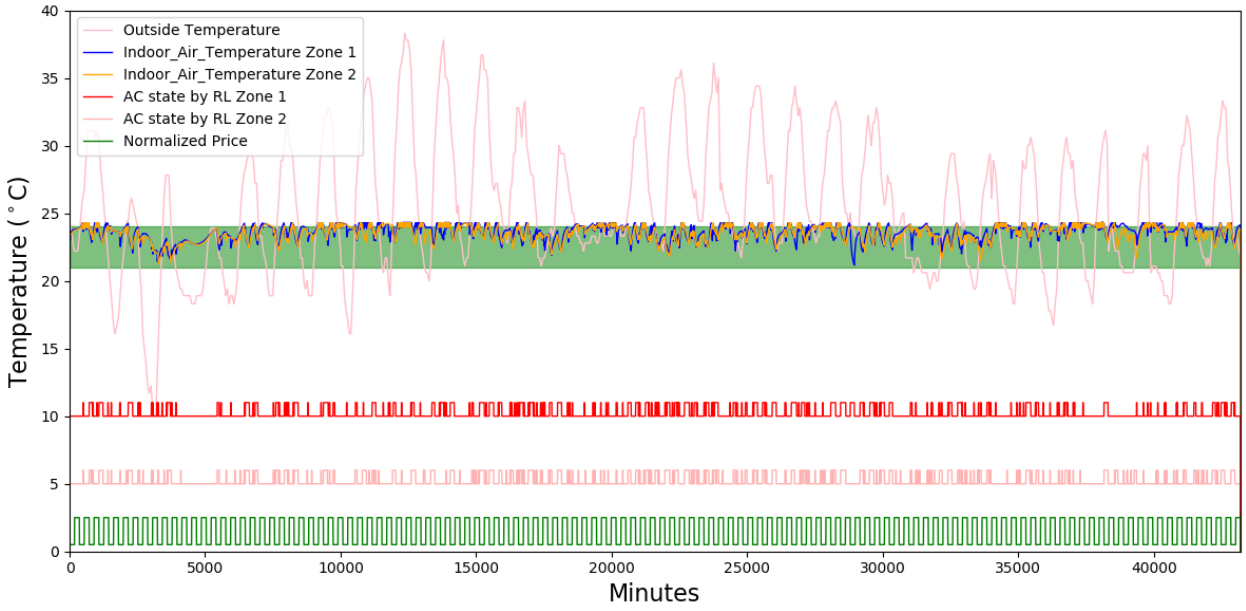
4.1 System Performance

In the following section, a brief summary of the current state of the ORNL development is collated. First, simulation results of a typical July run are presented, and compared to baseline results. Then, the convergence and training time of the algorithm are discussed. In the last section, conclusions about the state of this project and possible future developments are explored. At the time of this writing, testing is being conducted using the physical Yarnell Station House, but there are no real-time results to present.

Each episode represents 30 days of operation. The chosen cycle time *k* was 15 minutes, so each episode was 2,880 iterations of decision making for the learner. The whole dataset was 150 days of outdoor temperature data recorded each minute by the Yarnell Station House from May-September 2018. The price used was a square wave which alternated between $0.05/kWh and $0.25/kWh every three hours. Since July was our benchmark month, we ran twelve episodes of uniform randomly selected months that were not July and then one July episode at the end. Figure 4-1 shows the results of the July run. The cost, $31.62 represents a 33.5% savings over the baseline July run previously reported in Figure 3-3.

The "cost" plotted in Figure 4-2 is not monetary cost, but the output of the loss function calculated in the neural network described in Section 3.1.4. This cost is plotted for the first thirty days of the experiment shown in Figure 4-1. The figure indicates a convergence at approximately ten days, a typical training time for this model.

We prefer to chart training time using a plot like Figure 4-2, instead of a reward vs. episode graph. Convergence almost always occurred within the first thirty days of training, so a single value for a single episode would not give us a useful approximation. Also, reward in our case was heavily dependent on the overall ambient temperature of the selected month. A decrease in reward was not always indicative of a lack of learner knowledge, but often as simple as moving from a colder to a warmer month. Extending the amount of available data might help us in this during future tests.



Training Schedule: **12 Eps non-July, 1 July**

30 Day Cost: **$31.62**

Minutes Outside Comfort (MOC): **0**

Figure 4-1: July indoor temperature results after 12 months of training outside July.

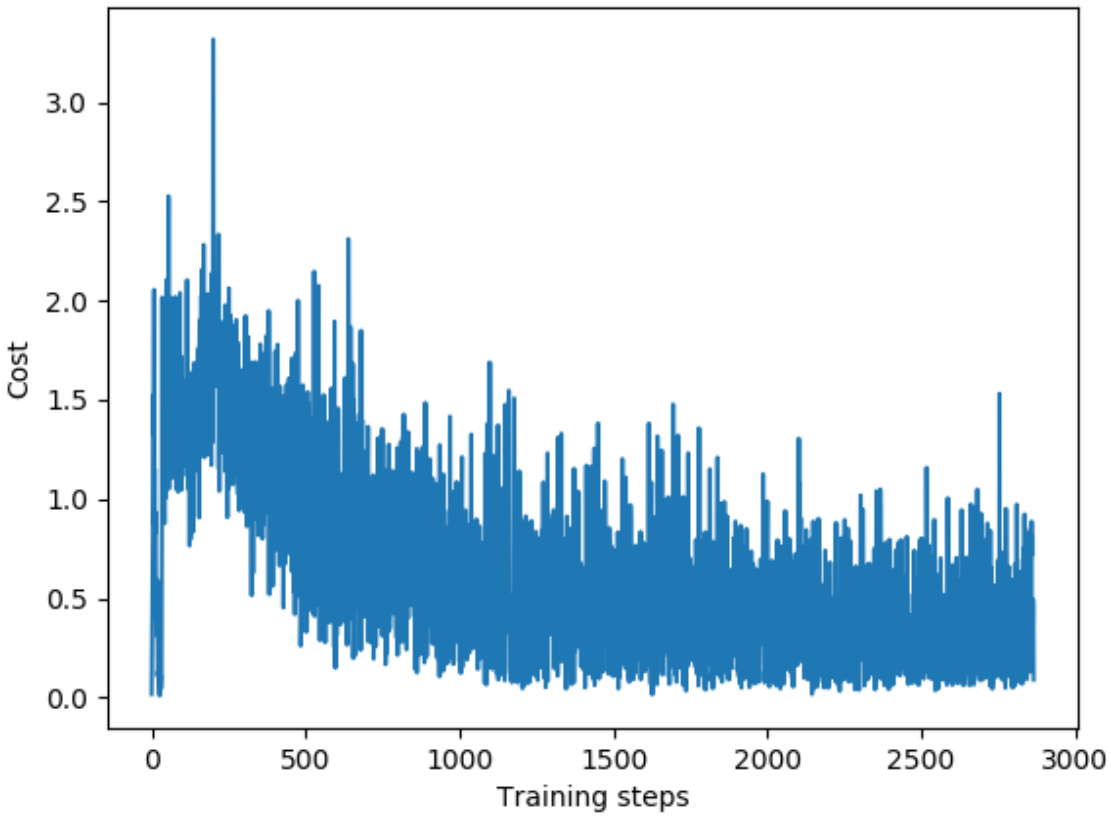


Figure 4-2: Cost vs. iterations for the first 30-days of the 13 month run.

4.2 Conclusions

The development was able to save 33.5% in a simulated home environment over a fixed setpoint baseline, while maintaining comfort throughout. Due to the consistent money-saving performance even during cold months, and since setpoint governance acts as a failsafe for preserving comfort, this model could be implemented into homes as it is today with confidence that customers could save money and maintain comfort. We consider our results satisfactory and have begun live, in-home testing.

There are several directions future researchers might take to expand this work. Heating was only briefly examined, with the expectation of future incorporation. The ORNL work will expand to include other loads, like an RL-controlled water heater. When more loads are added, we may test parallel operation of AI decision making. Negative dynamic prices were never tested, which allow a learner to sell power back to the grid and potentially make profits through clever scheduling. The existing code should support this phenomenon without significant alteration. Overall, we consider this project a success, and the AI development is ready to move into a real-time home environment.

REFERENCES

[1] S. Gyamfi, S. Krumdieck and T. Urmee, "Residential peak electricity demand response—Highlights of some behavioural issues.," *Renewable and Sustainable Energy Reviews,* vol. 25, pp. 71-77, 2013.

[2] T. Wei, Y. Wang and Q. Zhu, "Deep reinforcement learning for building hvac control," *Proceedings of the 54th Annual Design Automation Conference,* p. 22, 2017.

[3] A. Aribali, M. Ghofrani, M. Etezadi-Amoli, M. S. Fadali and Y. Baghzouz, "Genetic-algorithm-based optimization approach for energy management," *IEEE Transactions on Power Delivery,* vol. 28, no. 1, pp. 162-170, 2012.

[4] B. Asare-Bedaiko, W. Kling and P. Ribeiro, "Home energy management systems: Evolution, trends, and frameworks," *2012 47th International Universities Power Engineering Conference (UPEC),* pp. 1-5, 2012.

[5] M. Fiorentini, P. Cooper and Z. Ma, "Development and optimization of an innovative HVAC system with integrated PVT and PCM thermal storage for a net-zero energy retrofitted house.," *Energy and Buildings,* vol. 94, pp. 21-32, 2015.

[6] L. Klein, J. Y. Kwak, G. Kavulya, F. Jazizadeh, B. Becerik-Gerber, P. Varakantham and M. Tambe, "Coordinating occupant behavior for building energy and comfort management using multi-agent systems," *Automation in Construction,* vol. 22, pp. 525-536, 2012.

[7] A. Gonzalez-Briones, J. Prieto, F. De La Prieta, E. Herrera-Viedma and J. Corchado, "Energy optimization using a case-based reasoning strategy," *Sensors,* vol. 18, no. 3, p. 865, 2018.

[8] B. Cui, J. Munk, R. Jackson, D. Fugate and M. Starke, "Building thermal model development of typical house in u.s. for virtual storage control of aggregated building loads based on limited available information," *2017 30th International Conference on Efficiency, Cost, Optimization, Stabilization, and Environmental Impact of Energy Systems,* 2017.

[9] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, MIT press, 2018.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare and S. Petersen, "Human-level control through deep reinforcement learning," *Nature,* vol. 518, no. 7540, p. 529, 2015.

[11] G. Clark and P. Mehta, "Artificial intelligence and networking in integrated building management systems.," *Automation in Construction,* Vols. 6(5-6), pp. 481-498, 1997.

[12] M. Hadjiski, V. Sgurev and V. Bioshina, "Multi agent intelligent control of centralized HVAC systems," *IFAC Proceedings,* vol. 39.19, pp. 195-200, 2006.

[13] D. Monfet, M. Corsi, D. Choiniere and E. Arkhipova, "Development of an energy prediction tool for commercial buildings using case-based reasoning," *Energy and Buildings,* vol. 81, pp. 152-160, 2014.

[14] S. Salakij, N. Yu, S. Paolucci and P. Antsaklis, "Model-based predictive control for building energy management. I: Energy modeling and optimal control," *Energy and Buildings,* vol. 133, pp. 345-358, 2016.

[15] Z. A. Vale, H. Morais and H. Khodr, "Intelligent multi-player smart grid management considering distributed energy resources and demand response," *IEEE PES General Meeting,* 2010.

[16] G. Serale, M. Fiorentini, A. Capozzoli, D. Bernardini and A. Bemporad, "Model predictive control (MPC) for enhancing building and HVAC system energy efficiency: Problem formulation, applications and opportunities.," *Energies,* vol. 11, no. 3, p. 631, 2018.

[17] K. Dalamagkidis, D. Kolokotsa and K. Kalaitzakis, "Reinforcement learning for energy conservation and comfort in buildings," *Building and Environment,* vol. 42, no. 7, pp. 2686-2698, 2007.

[18] Y. Wang, K. Velswamy and B. Huang, "A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems," *Processes,* vol. 5, no. 3, p. 46, 2017.

[19] C. Cheng and D. Lee, "Artificial intelligence-assisted heating ventilation and air conditioning and the unmet demand for sensors," Department of Energy and Refrigerating Air-Conditioning Engineering, National Taipei University of Technology, Taipei, 2019.

[20] F. Lara-Rosano and N. K. Valverde, "Knowledge-based systems for energy conservation programs.," *Expert Systems with Applications,* Vols. 1-2, no. 14, pp. 25-35, 1998.

[21] S. Wang and X. Jin, "Model-based optimal control of VAV air-conditioning system using genetic algorithm.," *Buillding and Environment,* vol. 35.6, pp. 471-487, 2000.

[22] S. S. Intille, "Designing a home of the future," *IEEE pervasive computing,* vol. 2, pp. 76-82, 2002.

[23] D. Kolokotska, "Comparison of the performance of fuzzy controllers for the management of the indoor environment.," *Building and Environment,* vol. 38, no. 12, pp. 1439-1450, 2003.

[24] I. H. Yang, M. S. Yeo and K. W. Kim, "Application of artificial neural network to predict the optimal start time for heating system in building.," *Energy Conversion and Management,* vol. 44, no. 17, pp. 2791-2809, 2003.

[25] J. K. Wong, H. Li and S. W. Wang, "Intelligent building research: a review," *Automation in Construction,* vol. 14, no. 1, pp. 143-159, 2005.

[26] M. Mozer, "The adaptive house," *Proceedings of the IEEE Seminar on Intelligent Building Environment,* pp. 39-79, 2005.

[27] M. Teriyska, Y. Todorov and M. Petrov, "Fuzzy-neural model predictive control of a building heating system," *IFAC Proceedings,* vol. 39.19, pp. 69-74, 2006.

[28] G. Huang, S. Wang and X. Xu, "A robust model predictive control strategy for improving the control performance of air-conditioning systems," *Energy Conversion and Management,* vol. 50, no. 10, pp. 2650-2658, 2009.

[29] G. Jahedi and M. M. Ardehali, "Genetic algorithm-based fuzzy-PID control methodologies for enhancement of energy efficiency of a dynamic energy system," *Energy Conversion and Management,* vol. 52, no. 1, pp. 725-732, 2011.

[30] I. Petri, H. Li, Y. Rezgui, Y. Chunfeng, B. Yuce and B. Jayan, "A modular optimisation model for reducing energy consumption in large scale building facilities," *Renewable and Sustainable Energy Reviews,* vol. 38, pp. 990-1002, 2014.

[31] T. G. Stavropoulos, E. Kontopoulos, N. Bassiliades, J. Argyriou, A. Bikakis, D. Vrakas and I. Vlahavas, "Rule-based approaches for energy savings in an ambient intelligence environment," *Pervasive and Mobile Computing,* vol. 19, pp. 1-23, 2015.

[32] P. H. Shaikh, N. B. M. Nor, P. Nallagownden, I. Elamvazuthi and T. Ibrahim, "Intelligent multi-objective control and management for smart energy efficient buildings," *International Journal of Electrical Power & Energy Systems,* vol. 74, pp. 403-409, 2016.

[33] R. Godina, E. Rodrigues, E. Pouresmaeil, J. Matias and J. Catalao, "Model predictive

control home energy management and optimization strategy with demand response," *Applied Sciences,* vol. 8, no. 3, p. 408, 2018.

[34] G. Gao, J. Li and Y. Wen, "Energy-efficient thermal comfort control in smart buildings via deep reinforcement learning," *CoRR,* vol. 1901.04963, 2019.

[35] V. Mnih, K. Kavukcouglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv:1312.5602,* vol. 1, 2013.

[36] R. L. Hu, R. Skorupski, R. Entriken and Y. Ye, "A mathematical programming formulation for optimal load shifting of electricity demand for the smart grid," *IEEE Transactions on Big Data,* vol. 1, pp. 1-1.

[37] U.S.-Energy-Information-Adminiistration, "Residential energy consumption survey.," 2015.

APPENDICES

Appendix A: Controller Code

Controller_2zone.py

```
"""
For 2 zone model
This script is an outer script and does following
1. Sets up the environment
2. Interact with the RL brain
3. Interact with the Building model
Author: Kuldeep Kurte, Olivera
Created: 04/09/2019
Email: kurtekr@ornl.gov
"""

from Building_env_2Zone import Building_env
import Building_Model_Yarnell_Lennox_v2 as build
import Building_Model_Yarnell_wTstat_simplifiedinput as build_old
#from RL_brain_DQN_1_gpu import DeepQNetwork
from RL_brain_DQN_1_2Zone import DeepQNetwork
import argparse, sys
import numpy as np
import datetime as dt
from matplotlib import pyplot as plt


class ControllerK():

    def __init__(self, n_episodes, initialSteps, start_ts, end_ts, k):



        self.n_episodes = n_episodes # number of episodes =100(Source: Wei et al., 2017).
        self.initialSteps = initialSteps # initial steps without learning to acquire enough experience. default 200
        self.k = k # control step = k* simulation step (Source: Wei et al., 2017). default 15.
        self.maxSteps = end_ts - start_ts # 1440*n_days # Depends on whether we want to optimize for a week
        self.CumReward = 0
        self.all_rewards = []
        self.all_costs = []
        self.all_comfort = []
        self.com_comfort = []
        self.action_changes = 0
        self.start_ts = start_ts
        self.end_ts = end_ts
        self.Temp_inputs = []
        self.AC_status_by_sim = []
        self.ActionsByRL_1 = []
        self.ActionsByRL_2 = []
        self.setPt_1 = []
        self.setPt_2= []
```

```
47.
48.         # Evan Edit
49.         self.setpoint_governance = True
50.         self.use_fixed_setpoint = False  # Baseline
51.         self.use_months_queue = True
52.         # self.months_queue = [0, 1, 3, 4, 2]  # 0,1 = May, June; StartDay mus
    t equal 0
53.         # self.months_queue = [1, 1, 1, 1, 2]
54.         # self.months_queue = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
55.         self.months_queue = [4, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
56.         # self.months_queue = [0, 1, 3, 4, 0, 1, 3, 4, 0, 1, 3, 4, 2]
57.         self.fixed_setpoint_choice = 24.0  # 24
58.         self.offset_last_episode = False
59.         self.vary_span = False
60.         self.num_chunks = 4
61.         self.unvaried_ut = 24
62.         self.unvaried_lt = 21
63.         self.use_two_setpoints = True
64.         self.use_jeffs_hero = False
65.         self.single_zone = False
66.         self.use_const_price = False
67.         if self.single_zone is True:
68.             self.sz_T_inputs = [21, 21, 21, 21]
69.             self.sz_Tin = 21.
70.             self.sz_Twall = 21.
71.             self.sz_Tattic = 21.
72.             self.sz_Tmass = 21.
73.             self.sz_SetPt = 23.
74.             self.sz_ACStatus = 1.0
75.             self.sz_T_outputs = [21, 21, 21, 21]
76.             self.sz_power = 0.
77.
78.     def plotReward(self):
79.         print('self.rewards ', len(self.rewards))
80.
81.         plt.plot(list(range(1,len(self.rewards) + 1)), self.rewards)
82.         plt.xlabel('Episodes')
83.         plt.ylabel('Total reward')
84.         plt.show()
85.
86.     def plotCost(self):
87.         print('self.costs ', len(self.costs))
88.
89.         plt.plot(list(range(1,len(self.costs) + 1)), self.costs)
90.         plt.xlabel('Episodes')
91.         plt.ylabel('Total cost')
92.         plt.show()
93.
94.
95.
96.     def run_building(self, build_env, RL, start_ts, expId):
97.         '''''
98.         Write main controller logic here
99.         [observation] --(a)--> [observation_]
100.             '''
```

```
101.
102.             T_inputs = []
103.
104.             '''''
105.             These following variables are used to keep track of the fact tha
    t whateven action RL is
106.             suggesting and the action Jeff's model is taking inside are same
    .
107.             '''
108.
109.             ###Evan Edit
110.             t = 0
111.             n = 0
112.             global_MOC = 0
113.
114.             for episode in range(self.n_episodes):
115.                 #print("Episode %d" % (episode))
116.
117.                 ### Evan edit
118.                 if self.offset_last_episode is True and episode == self.n_ep
    isodes - 1:
119.                     start_ts = start_ts + 1440 * 30
120.                 if self.use_months_queue is True:
121.                     this_month = self.months_queue[episode]
122.                     start_ts = this_month * 1440 * 30
123.
124.                 #1. Fetch the initial observation from environment here
125.                 build_env.reset_env(start_ts)
126.                 S_pre = build_env.getInitialObservation(start_ts) #initializ
    ation of previous state
127.                 #S_curr = S_pre
128.                 #action = build_env.getInitialAction() # initial action "AC:
     off"
129.                 action = 0 #at the beginning of each episode the both zone's
     AC are off
130.                 action_1 = action_2 =0
131.                 self.CumReward = 0 #for episode
132.
133.                 ###Evan Edit
134.                 t_length = self.maxSteps * self.n_episodes
135.                 self.ActionsByRL_1 = np.zeros(t_length)
136.                 self.ActionsByRL_2 = np.zeros(t_length)
137.                 self.setPt_1 = [] # Unused
138.                 self.setPt_2 = []
139.                 episodic_MOC = 0
140.                 num_chunks = self.num_chunks
141.                 chunks = int((1440 * 30) / num_chunks)
142.
143.                 SavingModel=0
144.
145.                 for TS in range(self.maxSteps):
146.
147.                     ### Evan Edit
148.                     if TS % 1000 == 0 or TS == self.maxSteps - 1:
```

```
149.                         sys.stdout.write('\r'+'Episode:'+str(episode)+' TS:'
    +str(TS))
150.                         sys.stdout.flush()
151.
152.                     if self.vary_span is True:
153.                         if TS >= 0 and TS < chunks:
154.                             build_env.LT = 20
155.                             build_env.UT = 23
156.                         elif TS >= chunks and TS < 2*chunks:
157.                             build_env.LT = 23
158.                             build_env.UT = 26
159.                         elif TS >= 2*chunks and TS < 3*chunks:
160.                             build_env.LT = 20
161.                             build_env.UT = 23
162.                         else:
163.                             build_env.LT = 23
164.                             build_env.UT = 26
165.                     else:
166.                         build_env.LT = self.unvaried_lt
167.                         build_env.UT = self.unvaried_ut
168.
169.                     ### Evan Edit
170.
171.                     if(TS > 0 and TS % self.k == 0):    #start from first kt
    h observation
172.                         #1. Get current observations
173.                         S_curr = build_env.getCurrentObservationState(TS, st
    art_ts)
174.
175.                         #2. Calculate reward for [S_pre, action, observation
    _]
176.                         reward = build_env.calculateReward(TS, self.k)
177.                         self.CumReward += reward
178.
179.                         #3. Store tuple (S_pre, action, reward, observation_
    ) in memory
180.                         RL.store_transition(S_pre, action, reward, S_curr)
181.
182.                         #4. RL training logic
183.                         if (TS > self.initialSteps):
184.                             #print("Here the training should start")
185.                             RL.learn(episode)
186.                             action = RL.choose_action(S_curr) #RL's decision
     for AC status
187.                             #print("Action:%d"%(action))
188.
189.                         leadoff = 0.5
190.
191.                         if self.use_two_setpoints is True:
192.                             hi = build_env.UT
193.                             lo = build_env.LT
194.                             if (action == 0):
195.                                 build_env.ecobee[0].Cool_SetPt = hi * 1.8 +
    32  # C to F conversion
```

```
                                    build_env.ecobee[1].Cool_SetPt = hi * 1.8 +
    32
                                    action_1 = 0
                                    action_2 = 0
                                elif (action == 1):
                                    build_env.ecobee[0].Cool_SetPt = hi * 1.8 +
    32
                                    build_env.ecobee[1].Cool_SetPt = lo * 1.8 +
    32
                                    action_1 = 0
                                    action_2 = 1
                                elif (action == 2):
                                    build_env.ecobee[0].Cool_SetPt = lo * 1.8 +
    32
                                    build_env.ecobee[1].Cool_SetPt = hi * 1.8 +
    32
                                    action_1 = 1
                                    action_2 = 0
                                else:
                                    build_env.ecobee[0].Cool_SetPt = lo * 1.8 +
    32
                                    build_env.ecobee[1].Cool_SetPt = lo * 1.8 +
    32
                                    action_1 = 1
                                    action_2 = 1
                                if self.single_zone is True:
                                    if action < 2:
                                        self.sz_SetPt = lo
                                    else:
                                        self.sz_SetPt = hi
                            else:
                                if(action == 0):
                                    build_env.ecobee[0].Cool_SetPt= (build_env.T
    in1[TS] + leadoff)*1.8+32 #C to F conversion
                                    build_env.ecobee[1].Cool_SetPt= (build_env.T
    in2[TS] + leadoff)*1.8+32
                                    action_1=0
                                    action_2=0
                                elif(action == 1):
                                    build_env.ecobee[0].Cool_SetPt= (build_env.T
    in1[TS] + leadoff)*1.8+32
                                    build_env.ecobee[1].Cool_SetPt= (build_env.T
    in2[TS] - leadoff)*1.8+32
                                    action_1=0
                                    action_2=1
                                elif(action == 2):
                                    build_env.ecobee[0].Cool_SetPt= (build_env.T
    in1[TS] - leadoff)*1.8+32
                                    build_env.ecobee[1].Cool_SetPt= (build_env.T
    in2[TS] + leadoff)*1.8+32
                                    action_1=1
                                    action_2=0
                                else:
                                    build_env.ecobee[0].Cool_SetPt= (build_env.T
    in1[TS] - leadoff)*1.8+32
```

```
                                    build_env.ecobee[1].Cool_SetPt= (build_env.T
in2[TS] - leadoff)*1.8+32
                                    action_1=1
                                    action_2=1

                                ### Evan Edit
                                add_noise = False
                                if self.setpoint_governance is True:
                                    for k in [0,1]:
                                        if add_noise is True:
                                            noise = np.random.normal(loc=0.0, scale=
0.4)
                                        else:
                                            noise = 0.0
                                        if build_env.ecobee[k].Cool_SetPt > (build_e
nv.UT + 0.5)*1.8+32:
                                            build_env.ecobee[k].Cool_SetPt = (build_
env.UT)*1.8+32 + noise
                                        if build_env.ecobee[k].Cool_SetPt < (build_e
nv.LT - 0.5)*1.8+32:
                                            build_env.ecobee[k].Cool_SetPt = (build_
env.LT)*1.8+32 + noise
                                if self.use_fixed_setpoint is True:
                                    fixed_setpoint = self.fixed_setpoint_choice
                                    for k in [0,1]:
                                        build_env.ecobee[k].Cool_SetPt = (fixed_setp
oint) * 1.8 + 32
                                ###

                                #5. swap states S_pre = S_curr
                                S_pre = S_curr
                            if self.use_jeffs_hero is True:
                                if build_env.dfinput['Price1'].iloc[TS + start_ts] <
 0.25:
                                    build_env.ecobee[0].Cool_SetPt = (build_env.LT)
* 1.8 + 32
                                    build_env.ecobee[1].Cool_SetPt = (build_env.LT)
* 1.8 + 32
                                else:
                                    build_env.ecobee[0].Cool_SetPt = (build_env.UT)
* 1.8 + 32
                                    build_env.ecobee[1].Cool_SetPt = (build_env.UT)
* 1.8 + 32


                            build_env.setPt_1[TS] = (build_env.ecobee[0].Cool_SetPt)
 #store the setpoint
                            build_env.setPt_2[TS] = (build_env.ecobee[1].Cool_SetPt)


                            #6. Execute 'action' in building environment
                            if self.single_zone is True:
                                self.sz_T_outputs, self.sz_power, self.sz_ACStatus =
 build_old.simulate(self.sz_T_inputs, build_env.dfinput.iloc[TS+start_ts], sel
f.sz_SetPt, self.sz_ACStatus)
```

```
276.                             self.sz_T_inputs = self.sz_T_outputs
277.                             self.sz_Tin = self.sz_T_inputs[0]
278.                             build_env.Total_energy += self.sz_power/60000.0
279.                             if self.use_const_price is True:
280.                                 build_env.Total_Cost += ((self.sz_power/60000.0)
    *0.05)
281.                             else:
282.                                 build_env.Total_Cost += (self.sz_power / 60000.0
    ) * build_env.dfinput['Price1'].iloc[
283.                                     TS + start_ts]  # converting W (power) to kW
    h (energy) and multiplied by cost in $/kWh
284.                         else:
285.                             build_env.yarnell,build_env.lennox,build_env.zoning,
    build_env.ecobee=build.simulate(build_env.yarnell,build_env.lennox,build_env.z
    oning,build_env.ecobee,build_env.dfinput.iloc[TS+start_ts])
286.
287.                             #calculate energy consumption and cost
288.                             build_env.Total_energy += build_env.zoning.Power/600
    00.0 #conversion from watts to kW
289.                             if self.use_const_price is True:
290.                                 build_env.Total_Cost += ((build_env.zoning.Power
    /60000.0)*0.05)
291.                             else:
292.                                 build_env.Total_Cost += (build_env.zoning.Power
    / 60000.0) * build_env.dfinput['Price1'].iloc[
293.                                     TS + start_ts]  # converting W (power) to kW
    h (energy) and multiplied by cost in $/kWh
294.
295.                         if self.single_zone is True:
296.                             build_env.cost_t_1[TS] = (self.sz_power/60000.0)*bui
    ld_env.dfinput['Price1'].iloc[TS+start_ts]
297.                             if self.use_const_price is True:
298.                                 build_env.cost_t_1[TS] = ((build_env.zoning.Powe
    r/60000.0)*0.05)
299.                             build_env.Tout[TS] = (build_env.dfinput['Tout'].iloc
    [TS+start_ts])
300.                             build_env.Tin1[TS] = self.sz_Tin
301.                             build_env.Tin2[TS] = 0
302.                             build_env.Power[TS] = self.sz_power
303.
304.                         else:
305.                             build_env.cost_t_1[TS] = ((build_env.zoning.Power/60
    000.0)*build_env.dfinput['Price1'].iloc[TS+start_ts])
306.                             if self.use_const_price is True:
307.                                 build_env.cost_t_1[TS] = ((build_env.zoning.Powe
    r/60000.0)*0.05)
308.                             build_env.Tout[TS] = (build_env.dfinput['Tout'].iloc
    [TS+start_ts])
309.                             build_env.Tin1[TS] = (build_env.yarnell.Tin[0])
310.                             build_env.Tin2[TS] = (build_env.yarnell.Tin[1])
311.                             build_env.Power[TS] = (build_env.zoning.Power)
312.
313.                         build_env.Capacity1[TS] = (build_env.zoning.Qzone[0])
314.                         build_env.Capacity2[TS] = (build_env.zoning.Qzone[1])
315.                         build_env.Stage[TS] = (build_env.zoning.Stage)
```

```
                        build_env.CFM1[TS] = (build_env.zoning.CFM[0])
                        build_env.CFM2[TS] = (build_env.zoning.CFM[1])
                        build_env.CFM[TS] = (build_env.zoning.TotalCFM)
                        build_env.CoolOn1[TS] = (build_env.ecobee[0].CoolOn)
                        build_env.CoolOn2[TS] = (build_env.ecobee[1].CoolOn)
                        build_env.price_t[TS] = build_env.dfinput['Price1'].iloc
    [TS+start_ts] * 10
                        if self.use_const_price is True:
                            build_env.price_t[TS] = 0.05 * 10

                        self.ActionsByRL_1[TS] = (action_1+10) #store the action

                        self.ActionsByRL_2[TS] = (action_2+5)

                        ### Evan Edit

                        t += 1
                        # MOC = minutes outside comfort
                        MOCstrike = False
                        if self.single_zone is True:
                            if build_env.Tin1[TS] > build_env.UT + 0.5:
                                MOCstrike = True
                        else:
                            if build_env.Tin1[TS] > build_env.UT + 0.5:
                                MOCstrike = True
                            if build_env.Tin2[TS] > build_env.UT + 0.5:
                                MOCstrike = True
                        if MOCstrike is True:
                            episodic_MOC += 1
                            global_MOC += 1

                    #save model at the end of every episode
                    RL.saveModel(expId, episode)

                    ### Evan Edit
                    print("  CumReward: %f Cost: %f MOC: %i"%(self.CumReward, bu
    ild_env.Total_Cost, episodic_MOC))


                #All episodes are finished
                print("Finish RL training")

                ### Evan Edit
                print('Global MOC: %i'%(global_MOC))

        #Lower: 70F -> 21C
        #Upper: 75F -> 24C
        if __name__ == "__main__":
            '''
            1. create environment object
            2. create RL object
            3. execute run method
            '''
            #Usage:
```

```
367.          #python Controller.py --
   filePath "C:\Users\kk0\Documents\ORNL\Programming\rl-
   sandbox\data\inputs.csv" --episodes 50 --initialSteps 200 --k 15
368.          # python Controller_2zone.py --
   filePath "C:\Users\Evan\OneDrive\Sp19\ORNL\Kuldeep\rl-sandbox-RL-setPoint-
   Building_Model_2_Zone\rl-sandbox-RL-setPoint-
   Building_Model_2_Zone\Building_Model_2_Zone\inputs.csv" --episodes 50 --
   initialSteps 200 --k 15 --expId 10
369.          # python Controller_2zone.py --
   filePath "C:\Users\emckee5\OneDrive\Sp19\ORNL\Kuldeep\rl-sandbox-RL-setPoint-
   Building_Model_2_Zone\rl-sandbox-RL-setPoint-
   Building_Model_2_Zone\Building_Model_2_Zone\inputs.csv" --episodes 50 --
   initialSteps 200 --k 15 --expId 10
370.
371.          '''''
372.          beginTime = time.time()
373.          fileNameStats = "%s/stats/%a"%(os.getcwd(),time.ctime(beginTime))
374.          fileNameFigs = "%s/figs/%a"%(os.getcwd(),time.ctime(beginTime))
375.          os.mkdir(fileNameStats)
376.          os.mkdir(fileNameFigs)
377.          file = open("%s/log.txt"%(fileNameStats), "w")
378.          file.write("Start Execution %s\r\n" % time.ctime(beginTime))
379.          '''
380.
381.          parser = argparse.ArgumentParser()
382.          parser.add_argument( "--
   filePath", help="Path to .csv file containing weather data" )
383.          parser.add_argument( "--episodes", help="Training episodes" )
384.          parser.add_argument( "--
   initialSteps", help="Initial steps before the first learning" )
385.          parser.add_argument( "--
   k", help="Time interval between trainings" )
386.          parser.add_argument( "--expId", help="ExperiementId")
387.          args = parser.parse_args()
388.
389.          StartDay=0 #31 june 61 july
390.          Duration=30  #days
391.
392.          start_ts = int(StartDay*24*60)
393.          end_ts = int((StartDay+Duration)*24*60) #convert to the timestamp
394.          days = end_ts - start_ts + 1
395.
396.
397.          #1. read data
398.          build_env = Building_env(filePath = args.filePath) # [t, T_in, T_out
   , ..]
399.          build_env.readData2()
400.          print("Initialized building environment and Read input file")
401.
402.          #2. Initialize the neural network
403.          RL = DeepQNetwork(build_env.n_actions, build_env.n_features,
404.                            learning_rate=0.01,
405.                            reward_decay=0.9,
406.                            e_greedy=0.9,
407.                            replace_target_iter=200,
```

```
                                batch_size= 32,
                                memory_size=20000)

            print("Initialized RL Neural network structure")

            #3.
            con_obj = ControllerK(n_episodes = int(args.episodes), initialSteps
    = int(args.initialSteps), start_ts = start_ts, end_ts = end_ts, k = int(args.k
    ))

            print("starting training...")
            #<-
            con_obj.run_building(build_env, RL, start_ts, int(args.expId))

            #4. plotting
            RL.plot_cost()

            build_env.printTotalPrice()
            build_env.printIndoorTemp(con_obj.ActionsByRL_1, con_obj.ActionsByRL
    _2,start_ts = start_ts, end_ts = end_ts)

            '''''
            endTime = time.time()
            executionTime = str(endTime-beginTime)
            file.write("Stop Execution %s\r\n" % time.ctime(endTime))
            file.write("Execution time %s" % executionTime)
            file.close()
            '''

```

Appendix B: Building Environment Class Code

Building_env_2Zone.py

```
"""
This script defines and controls the building environment

Author: Kuldeep Kurte
Created: 04/09/2019
Email: kurtekr@ornl.gov
"""

import random
import pandas as pd
import datetime as dt
import Building_Model_Yarnell_Lennox_v2 as build
import Building_Model_Yarnell_wTstat_simplifiedinput as build_old
import numpy as np
from matplotlib import pyplot as plt
# import pickle as pkl

class Building_env():

    #Constructor definition

### Evan Edit num features
    def __init__(self, filePath, n_zones=2, n_features=8 , action_space=[0, 1, 2, 3], LT=21, UT=24):

        self.filePath=filePath
        self.yarnell,self.lennox,self.zoning,self.ecobee=build.initialize() #initialization


        self.action_space = action_space #action space for RL with respect to the HVAC "0: OFF 1:ON"
        self.n_actions = len(self.action_space) #number of actions = action values*number of zones
        self.n_features = n_features #features (state variables) = input to the neural n/w, [TOD, In_Temp1, In_Temp2, Out_Temp, Price, Price+, Price++ ]


        #User's comfort range
        self.LT = LT
        self.UT = UT

        self.dfinput = pd.DataFrame()

        #list to catch various values
        self.Tout=[] #outside temperature
        self.Tin1=[] #zone 1 temperature
        self.Tin2=[] #zone 2 temperature
        self.Power=[]
        self.Capacity1=[]
```

```
46.         self.Capacity2=[]
47.         self.Stage=[]
48.         self.CFM1=[]
49.         self.CFM2=[]
50.         self.CFM=[]
51.         self.CoolOn1=[] #HVAC status at zone1
52.         self.CoolOn2=[] #HVAC status at zone2
53.         self.price_t = []
54.
55.         self.cost_t_1 = [] #store the cost values
56.         self.Total_energy=0.0 #total energy used
57.         self.Total_Cost=0.0 #Total cost
58.
59.         self.setPt_1 = []
60.         self.setPt_2 = []
61.         self.setPt_1.append(72)
62.         self.setPt_2.append(72)
63.
64.         self.priceHigh = [180, 540, 900, 1260]
65.         self.priceLow = [0, 360, 720, 1080]
66.
67.         self.Lambda = 100 #(Source: Wei et al., 2017)
68.
69.         # Evan Edit
70.         self.use_absolute_temps = False
71.         self.use_comfort_penalty = True
72.         self.use_point_slope_features = False
73.         self.use_acstatus = False
74.         self.use_only_price = False
75.         self.use_const_price = False
76.         self.kite_tail_cycles = 4
77.         self.ot_slope_cycles = 4
78.         self.price_slope_cycles = 12
79.         self.vary_span = False
80.         self.fixed_ut = 24  # 24
81.         self.fixed_lt = 21  # 21
82.         self.select_prices = []
83.         self.select_slopes = []
84.
85.     def timeToPeak(self, ts):
86.         if(ts>=0 and ts < 180):
87.             tp = 180 - ts
88.         elif(ts < 540):
89.             tp = 540 - ts
90.         elif(ts < 900):
91.             tp = 900 - ts
92.         else:
93.             tp = 1260 - ts
94.
95.         return tp
96.
97.     def readData2(self):
98.         self.dfinput = pd.read_csv(self.filePath) #read that input data inside
    panda's dataframe
99.
```

```
100.
101.         def build_env(self):
102.             '''''
103.             implement any functions related to building the environment
104.             '''
105.
106.         def reset_env(self, ts):
107.             '''''
108.             implement any functions related to the resetting the environment

109.             '''
110.
111.             ### Evan Edit
112.             n_length = 1440 * 30 + 1
113.
114.             self.Tout=np.zeros(n_length)
115.             self.Tin1=np.zeros(n_length)
116.             self.Tin2=np.zeros(n_length)
117.             self.Power=np.zeros(n_length)
118.             self.Capacity1=np.zeros(n_length)
119.             self.Capacity2=np.zeros(n_length)
120.             self.Stage=np.zeros(n_length)
121.             self.CFM1=np.zeros(n_length)
122.             self.CFM2=np.zeros(n_length)
123.             self.CFM=np.zeros(n_length)
124.             self.CoolOn1=np.zeros(n_length)
125.             self.CoolOn2=np.zeros(n_length)
126.             self.price_t=np.zeros(n_length)
127.
128.             self.cost_t_1 = np.zeros(n_length)
129.             self.cost_t_1[0] = 0.0
130.             self.Total_energy=0.0
131.             self.Total_Cost=0.0
132.
133.             self.CoolOn1[0] = 0
134.             self.CoolOn2[0] = 0
135.             self.Tin1[0] = (self.dfinput['Tin'].iloc[ts]) #initial indoor te
    mperature
136.             self.Tin2[0] = (self.dfinput['Tin'].iloc[ts])
137.
138.             #reset Yarnell building's parameters
139.             self.yarnell.Tin[0]=self.dfinput['Tin'].iloc[ts] #Zone-
    0 indoor temp
140.             self.yarnell.Tin[1]=self.dfinput['Tin'].iloc[ts] #Zone-
    1 indoor temp
141.             self.yarnell.Tattic=self.dfinput['Tattic'].iloc[ts]
142.             self.yarnell.Tfloor=self.dfinput['Tin'].iloc[ts]
143.             self.yarnell.Twall[0]=self.dfinput['Twall'].iloc[ts] #Zone-
    0 wall temp
144.             self.yarnell.Twall[1]=self.dfinput['Twall'].iloc[ts] #Zone-
    1 wall temp
145.             self.yarnell.Tmass[0]=self.dfinput['Tmass'].iloc[ts] #Zone-
    0 mass temp
146.             self.yarnell.Tmass[1]=self.dfinput['Tmass'].iloc[ts] #Zone-
    1 mass temp
```

```
147.                 self.ecobee[0].Cool_SetPt=72.0 #Zone-0 setpoint
148.                 self.ecobee[1].Cool_SetPt=72.0 #Zone-1 setpoint
149.
150.                 self.setPt_1 = np.zeros(n_length)
151.                 self.setPt_2 = np.zeros(n_length)
152.                 self.setPt_1[0] = (self.ecobee[0].Cool_SetPt)
153.                 self.setPt_2[0] = (self.ecobee[1].Cool_SetPt)
154.                 ### End Evan Edit
155.
156.
157.
158.             def simulateBuildingModel(self, TS, ACStatus, SetPt, start_idx):
159.                 #here, action is a set point value
160.                 #1. fetch weather input for "TS"
161.                 weather_inputs = (self.T_outside[TS], self.DNI[TS], self.HDI[TS]
    , self.WS[TS]) #get weather attribute values for timestamp "TS"
162.                 #self.tstamp_current = self.Timestamps[TS]
163.                 self.Temp_inputs,power,self.ACStatus = build.simulate(self.Temp_
    inputs, self.dfinput.iloc[TS+start_idx], SetPt, ACStatus, TS)
164.
165.                 #No need to have this price index calculation since price data i
    s available for every minute.
166.                 #2. Fetch the price information for time "TS"
167.                 #mTOD = (TS)%1440 #this is essentially minute of the day
168.                 #priceIdx = mTOD//5 # since, price data is available for 5mins;
    find the price for that time of the day
169.
170.                 #3. append all the values in the variables
171.                 self.Indoor_Air_Temp.append(self.Temp_inputs[0]) #this indoor te
    mperatures for next minute
172.                 self.AC_state.append(self.ACStatus*5) #action taken at "TS" 1/0

173.                 self.SetPoints.append(SetPt)#here, action is a set point value
174.                 self.priceInfo.append(self.price[TS]*10) #price at "TS"
175.                 self.cost_t_1.append((power/60000)*self.price[TS]) #[power consu
    med due to previous action at "TS"] * [price per unit]
176.
177.                 self.Total_power += power/60000
178.                 self.Total_cost += self.cost_t_1[TS]
179.                 self.cost_over_time.append(300*self.cost_t_1[TS])
180.                 return (self.Total_cost, self.ACStatus, self.Temp_inputs) #this
    is a cumulative cost and will be used to display total cost for each episode
181.
182.
183.             def getInitialAction(self):
184.                 return self.action_space[random.getrandbits(1)] #adding some ran
    domness in choosing initial action
185.                 #return self.SetPt
186.
187.             def getInitialObservation(self, ts):
188.                 '''''
189.                 [MOD, In_Temp, Out_Temp ]
190.                 '''
191.                 mTOD = 0 #0th minute of the day
192.                 ### Evan Edit
```

```
193.             # For relative temp test
194.             if self.use_absolute_temps is True:
195.                 In_Temp_1=self.dfinput['Tin'].iloc[ts] /(30) #Zone-
    0 indoor temp #21
196.                 In_Temp_2=self.dfinput['Tin'].iloc[ts] /(30) #Zone-
    1 indoor temp
197.                 Out_Temp = self.dfinput['Tout'].iloc[ts] / (30)  # Outside t
    emp
198.             else:
199.                 In_Temp_1=(self.dfinput['Tin'].iloc[ts]-
    self.UT) /(30) #Zone-0 indoor temp #21
200.                 In_Temp_2=(self.dfinput['Tin'].iloc[ts]-
    self.UT) /(30) #Zone-1 indoor temp
201.                 Out_Temp = (self.dfinput['Tout'].iloc[ts]-
    In_Temp_1) / (30)  # Outside temp
202.             ###
203.
204.             if self.use_point_slope_features is True:
205.                 it_m1 = 0
206.                 it_m2 = 0
207.                 ot_m = 0
208.                 p_m = 0
209.             price1=self.dfinput['Price1'].iloc[ts] /(0.25-
    0.05)#current price
210.             price2=self.dfinput['Price1'].iloc[ts+5]/(0.25-
    0.05) #current price
211.             price3=self.dfinput['Price1'].iloc[ts+15]/(0.25-
    0.05) #current price
212.             price4=self.dfinput['Price1'].iloc[ts+30]/(0.25-0.05)
213.             if self.use_const_price is True:
214.                 price1=0
215.                 price2=0
216.                 price3=0
217.                 price4=0
218.             if self.use_point_slope_features is True:
219.                 S_curr = np.array([In_Temp_1, it_m1, In_Temp_2, it_m2, Out_T
    emp, ot_m, price1, p_m])
220.                 if self.use_only_price is True:
221.                     S_curr = np.array([price1, p_m,0,0,0,0,0,0])
222.             else:
223.                 S_curr = np.array([In_Temp_1, In_Temp_2, Out_Temp, price1, p
    rice2, price3, price4, 0])
224.             if self.use_acstatus is True:
225.                 S_curr = np.array([In_Temp_1, In_Temp_2, Out_Temp, 0, 0, pri
    ce3, price4, 0])
226.             else:
227.                 S_curr = np.array([In_Temp_1, In_Temp_2, Out_Temp, price1, p
    rice2, price3, price4, 0])
228.             # self.select_prices.append(price1)
229.             # self.select_slopes.append(m2)
230.             return S_curr
231.
232.
233.         def getCurrentObservationState(self, ts, start_ts):
234.             #this is not needed
```

```
            mTOD = ((ts)%(1440))/1440 #this is essentially hour of the day

            if self.use_absolute_temps is True:
                In_Temp_1 = (self.Tin1[ts % (1440 * 30)]) / 30
                In_Temp_2 = (self.Tin2[ts % (1440 * 30)]) / 30
                Out_Temp = self.dfinput['Tout'].iloc[ts+start_ts] / (30)  # Outside temp
            else:
                In_Temp_1 = (self.Tin1[ts % (1440 * 30)]-self.UT) / 30
                In_Temp_2 = (self.Tin2[ts % (1440 * 30)]-self.UT) / 30
                Out_Temp = (self.dfinput['Tout'].iloc[ts+start_ts] - In_Temp_1) / (30)  # Outside temp

            ### Evan Edit Kite tail
            cycle_time = 15  # fix; import
            ktl = self.kite_tail_cycles  # Kite tail length
            psl = self.price_slope_cycles
            otl = self.ot_slope_cycles
            if self.use_point_slope_features is True:
                if ts < (ktl*cycle_time) - 1:
                    it_m1 = 0
                    it_m2 = 0
                else:
                    y = np.array(self.Tin1[ts+1-(cycle_time*ktl):ts+1])
                    x = np.linspace(0, (ktl-1)*cycle_time, ktl*cycle_time)
                    # Linear regression
                    it_m1 = (len(x) * np.sum(x * y) - np.sum(x) * np.sum(y)) / (len(x) * np.sum(x * x) - np.sum(x) * np.sum(x))
                    y = np.array(self.Tin2[ts + 1 - (cycle_time * ktl):ts + 1])
                    it_m2 = (len(x) * np.sum(x * y) - np.sum(x) * np.sum(y)) / (
                                        len(x) * np.sum(x * x) - np.sum(x) * np.sum(x))
                if ts >= 43200 - psl * cycle_time:  ### Fix; generalize t_length for other than 30 days
                    p_m = 0
                else:
                    y = np.array(self.dfinput['Price1'].iloc[ts+start_ts:ts+start_ts+psl*cycle_time])
                    x = np.linspace(0, (psl-1)*cycle_time, psl*cycle_time)
                    # Linear regression
                    p_m = (len(x) * np.sum(x * y) - np.sum(x) * np.sum(y)) / (len(x) * np.sum(x * x) - np.sum(x) * np.sum(x))
                if ts >= 43200 - otl * cycle_time:  ### Fix; generalize t_length for other than 30 days
                    ot_m = 0
                else:
                    y = np.array(self.dfinput['Tout'].iloc[ts+start_ts:ts+start_ts+otl*cycle_time])
                    x = np.linspace(0, (otl-1)*cycle_time, otl*cycle_time)
                    # Linear regression
                    ot_m = (len(x) * np.sum(x * y) - np.sum(x) * np.sum(y)) / (len(x) * np.sum(x * x) - np.sum(x) * np.sum(x))
```

```
            price1=self.dfinput['Price1'].iloc[ts+start_ts] /(0.25-0.05) #current price
            price2=self.dfinput['Price1'].iloc[ts+start_ts+5] /(0.25-0.05) #current price
            price3=self.dfinput['Price1'].iloc[ts+start_ts+15] /(0.25-0.05) #current price
            price4=self.dfinput['Price1'].iloc[ts+start_ts+30]/(0.25-0.05)
            if self.use_const_price is True:
                price1=0
                price2=0
                price3=0
                price4=0
            if self.use_point_slope_features is True:
                S_curr = np.array([In_Temp_1, it_m1, In_Temp_2, 0, Out_Temp, ot_m, price1, p_m])
                if self.use_only_price is True:
                    S_curr = np.array([price1, p_m,0,0,0,0,0,0])
            else:
                S_curr = np.array([In_Temp_1, In_Temp_2, Out_Temp, price1, price2, price3, price4, 0])
            if self.use_acstatus is True:
                S_curr = np.array([In_Temp_1, In_Temp_2, Out_Temp, self.CoolOn1[ts-15], self.CoolOn2[ts-15], price3, price4, 0])
            else:
                S_curr = np.array([In_Temp_1, In_Temp_2, Out_Temp, price1, price2, price3, price4, 0])
            return(S_curr)


        def calculateReward(self, ts, K):

            pu1 = pl1 = pu2 = pl2 = CumCost = 0
            preCoolAdv_1 = preCoolAdv_2 = 0

            for i in range(0,K):
                pu1 += (self.Tin1[ts-i] - self.UT) if self.Tin1[ts-i] > self.UT else 0
                pl1 += (self.LT - self.Tin1[ts-i]) if self.Tin1[ts-i] < self.LT and self.CoolOn1[ts-i] == 1 else 0
                pu2 += (self.Tin2[ts-i] - self.UT) if self.Tin2[ts-i] > self.UT else 0
                pl2 += (self.LT - self.Tin2[ts-i]) if self.Tin2[ts-i] < self.LT and self.CoolOn2[ts-i] == 1 else 0

                #preCoolAdv_1 += np.max([(self.Tin1[ts-i] - self.LT), 0]) * np.max([(30-self.timeToPeak(ts)),0])
                #preCoolAdv_2 += np.max([(self.Tin2[ts-i] - self.LT), 0]) * np.max([(30-self.timeToPeak(ts)),0])
            '''''
            for i in range(0,K):
                pu1 += (self.setPt_1[ts-i] - self.UT) if self.setPt_1[ts-i] > self.UT else 0
                pl1 += (self.LT - self.setPt_1[ts-i]) if self.setPt_1[ts-i] < self.LT else 0
```

```
                    pu2 += (self.setPt_2[ts-i] - self.UT) if self.setPt_2[ts-i] > self.UT else 0
                    pl2 += (self.LT - self.setPt_2[ts-i]) if self.setPt_2[ts-i] < self.LT else 0
                '''

                for j in range(0, K):
                    CumCost += self.cost_t_1[ts-j]

                # reward_t = -100*CumCost - (pu1 + pl1 + pu2 + pl2)

                if self.use_comfort_penalty is True:
                    reward_t = -100*CumCost - (pu1 + pl1 + pu2 + pl2)
                else:
                    reward_t = -100*CumCost
                return(reward_t)


            def printActionSpace(self):
                print(self.action_space)

            def printIndoorTemp(self, ActionByRL_1, ActionByRL_2, start_ts, end_ts ):
                #timesteps=1440*n_days # read one extra data point
                timesteps = end_ts - start_ts

                # plt.plot(self.select_prices[0:48])
                # plt.show()
                # plt.plot(self.select_slopes[0:48])
                # plt.show()

                fig = plt.figure( figsize=[12, 6], dpi=100 )
                ax = fig.add_subplot(111)
                ax.plot( [x for x in range(len(self.Tout))], self.Tout, lw=1, color='pink', label='Outside Temperature')
                ax.plot( [x for x in range(len(self.Tin1))], self.Tin1, lw=1, color='blue', label='Indoor_Air_Temperature Zone 1')
                ax.plot( [x for x in range(len(self.Tin2))], self.Tin2, lw=1, color='orange', label='Indoor_Air_Temperature Zone 2')
                ax.plot( [x for x in range(len(ActionByRL_1))], ActionByRL_1, lw=1, color='red', label='AC state by RL Zone 1')
                ax.plot( [x for x in range(len(ActionByRL_2))], ActionByRL_2, lw=1, color='red', label='AC state by RL Zone 2', alpha=0.3)
                ax.plot( [x for x in range(len(self.price_t))], self.price_t, lw=1, color='green', label='Normalized Price')


                #ax.plot( [x for x in range(len(self.CoolOn2))], self.CoolOn2, lw=1, color='black', label='AC state1')

                #ax.plot( [x for x in range(len(priceInfo))], priceInfo, lw=3, color='red', label='Price')
                #ax.plot( [x for x in range(len(SetPoints))], SetPoints, lw=1, color='yellow', label='SetPt')
```

```
359.                 ax.set_xlim( (0, timesteps) )
360.                 ax.set_ylim( (0, 40) )
361.                 if self.vary_span is True:
362.                     upper_line = list([23] * 10800) + list([26] * 10800) + list(
    [23] * 10800) + list([26] * 10800)
363.                     lower_line = list([20] * 10800) + list([23] * 10800) + list(
    [20] * 10800) + list([23] * 10800)
364.
365.                     ax.plot(upper_line, color='k', label='Comfort Zone')
366.                     ax.plot(lower_line, color='k')
367.                 else:
368.                     plt.axhspan(self.fixed_lt, self.fixed_ut, color='green', alp
    ha=0.5)
369.                 plt.xlabel('Minutes', fontsize=16)
370.                 ### Evan Edit F to C
371.                 plt.ylabel('Temperature ($^\circ$C) ', fontsize=16)
372.                 plt.legend(loc="upper left")
373.                 plt.tight_layout()
374.                 plt.show()
375.
376.
377.
378.             def printTotalPrice(self):
379.                 print("Total Energy used during last episode: ", self.Total_ener
    gy)
380.                 print("Total cost during last episode: ", self.Total_Cost)
```

Appendix C: DQN Algorithm Code

RL_brain_DQN_1_2Zone.py

```
"""
This part of code is the DQN brain, which is a brain of the agent.
All decisions are made in here.
Using Tensorflow to build the neural network.
View more on tutorial page: https://morvanzhou.github.io/tutorials/
Using:
Tensorflow: 1.0
gym: 0.7.3

Author: Kuldeep Kurte
Created: 04/09/2019
Email: kurtekr@ornl.gov
Adpated from: https://morvanzhou.github.io/tutorials/
"""

import numpy as np
import pandas as pd
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import tensorflow as tf
import time
import datetime

np.random.seed(1)
tf.set_random_seed(1)
tf.reset_default_graph()


# Deep Q Network off-policy
class DeepQNetwork:
    def __init__(
            self,
            n_actions,
            n_features,
            learning_rate=0.01,
            reward_decay=0.9,
            e_greedy=0.9,
            replace_target_iter=300,
            memory_size=500,
            batch_size=32,
            e_greedy_increment=None,
            output_graph=False,

    ):
        self.n_actions = n_actions
        self.n_features = n_features
        self.lr = learning_rate
        self.gamma = reward_decay
        self.epsilon_max = e_greedy
        self.replace_target_iter = replace_target_iter
```

```
51.         self.memory_size = memory_size
52.         self.batch_size = batch_size
53.         self.epsilon_increment = e_greedy_increment
54.         self.epsilon = 0 if e_greedy_increment is not None else self.epsilon_m
    ax
55.
56.         # total learning step
57.         self.learn_step_counter = 0
58.
59.         # initialize zero memory [s, a, r, s_]
60.         self.memory = np.zeros((self.memory_size, n_features * 2 + 1 + 1)) # a
    =1, r = 1 and s, s_ * 2
61.
62.         # consist of [target_net, evaluate_net]
63.         self._build_net()
64.         t_params = tf.get_collection('target_net_params')
65.         e_params = tf.get_collection('eval_net_params')
66.         self.replace_target_op = [tf.assign(t, e) for t, e in zip(t_params, e_
    params)]
67.
68.         #self.config = tf.ConfigProto()
69.         #self.config.gpu_options.allow_growth = True
70.         self.sess = tf.Session()
71.         #self.sess = tf.Session(config=self.config)
72.
73.         if output_graph:
74.             # $ tensorboard --logdir=logs
75.             # tf.train.SummaryWriter soon be deprecated, use following
76.             tf.summary.FileWriter("logs/", self.sess.graph)
77.
78.         self.sess.run(tf.global_variables_initializer())
79.         self.saver = tf.train.Saver(max_to_keep=50) # define a saver for savin
    g model and restoring
80.
81.         self.cost_his = []
82.         self.QVal = []
83.
84.
85.
86.     def _build_net(self):
87.         # ------------------ build evaluate_net ------------------
88.         self.s = tf.placeholder(tf.float32, [None, self.n_features], name='s')
      # input
89.         self.q_target = tf.placeholder(tf.float32, [None, self.n_actions], nam
    e='Q_target')  # for calculating loss
90.         with tf.variable_scope('eval_net'):
91.             # c_names(collections_names) are the collections to store variable
    s
92.             c_names, n_l1, w_initializer, b_initializer = \
93.                 ['eval_net_params', tf.GraphKeys.GLOBAL_VARIABLES], 10, \
94.                 tf.random_normal_initializer(0., 0.3), tf.constant_initializer
    (0.1)  # config of layers
95.
96.             # first layer. collections is used later when assign to target net
```

```
            with tf.variable_scope('l1'):
                w1 = tf.get_variable('w1', [self.n_features, n_l1], initializer=w_initializer, collections=c_names)
                b1 = tf.get_variable('b1', [1, n_l1], initializer=b_initializer, collections=c_names)
                    l1 = tf.nn.relu(tf.matmul(self.s, w1) + b1)

                # first hidden layer. collections is used later when assign to target net
                with tf.variable_scope('l2'):
                    w2 = tf.get_variable('w2', [n_l1, 10], initializer=w_initializer, collections=c_names)
                    b2 = tf.get_variable('b2', [1, 10], initializer=b_initializer, collections=c_names)
                    l2 = tf.nn.relu(tf.matmul(l1, w2) + b2)

                # last layer. collections is used later when assign to target net
                with tf.variable_scope('l3'):
                    w3 = tf.get_variable('w3', [10, self.n_actions], initializer=w_initializer, collections=c_names)
                    b3 = tf.get_variable('b3', [1, self.n_actions], initializer=b_initializer, collections=c_names)
                    self.q_eval = tf.matmul(l2, w3) + b3

            with tf.variable_scope('loss'):
                self.loss = tf.reduce_mean(tf.squared_difference(self.q_target, self.q_eval)) #MSE
            with tf.variable_scope('train'):
                self._train_op = tf.train.AdamOptimizer(self.lr).minimize(self.loss)
                #self._train_op = tf.train.RMSPropOptimizer(self.lr).minimize(self.loss) # default RMSPropOptimizer

            # ------------------ build target_net ------------------
            self.s_ = tf.placeholder(tf.float32, [None, self.n_features], name='s_')    # input
            with tf.variable_scope('target_net'):
                # c_names(collections_names) are the collections to store variables
                c_names = ['target_net_params', tf.GraphKeys.GLOBAL_VARIABLES]

                # first layer. collections is used later when assign to target net
                with tf.variable_scope('l1'):
                    w1 = tf.get_variable('w1', [self.n_features, n_l1], initializer=w_initializer, collections=c_names)
                    b1 = tf.get_variable('b1', [1, n_l1], initializer=b_initializer, collections=c_names)
                    l1 = tf.nn.relu(tf.matmul(self.s_, w1) + b1)

                # first hidden layer. collections is used later when assign to target net
                with tf.variable_scope('l2'):
```

```
                    w2 = tf.get_variable('w2', [n_l1, 10], initializer=w_initializer, collections=c_names)
                    b2 = tf.get_variable('b2', [1, 10], initializer=b_initializer, collections=c_names)
                    l2 = tf.nn.relu(tf.matmul(l1, w2) + b2)

                # last layer. collections is used later when assign to target net
                with tf.variable_scope('l3'):
                    w3 = tf.get_variable('w3', [10, self.n_actions], initializer=w_initializer, collections=c_names)
                    b3 = tf.get_variable('b3', [1, self.n_actions], initializer=b_initializer, collections=c_names)
                    self.q_next = tf.matmul(l2, w3) + b3


        def store_transition(self, s, a, r, s_):
            if not hasattr(self, 'memory_counter'):
                self.memory_counter = 0

            transition = np.hstack((s, [a, r], s_)) # why we need '[]'

            # replace the old memory with new memory
            index = self.memory_counter % self.memory_size
            self.memory[index, :] = transition

            self.memory_counter += 1

        def choose_action(self, observation):
            # to have batch dimension when feed into tf placeholder
            #print(observation)
            observation = observation[np.newaxis, :]

            if np.random.uniform() < self.epsilon:
                # forward feed the observation and get q value for every actions
                actions_value = self.sess.run(self.q_eval, feed_dict={self.s: observation})
                action = np.argmax(actions_value)
            else:
                action = np.random.randint(0, 4)
            return action


        def learn(self, e_idx): # 'e_idx' is not used
            # check to replace target parameters
            if self.learn_step_counter % self.replace_target_iter == 0:
                self.sess.run(self.replace_target_op)
    #            print('target_params_replaced')

            # sample batch memory from all memory
            if self.memory_counter > self.memory_size:
```

```
181.                 sample_index = np.random.choice(self.memory_size, size=self.
    batch_size)
182.             else:
183.                 sample_index = np.random.choice(self.memory_counter, size=se
    lf.batch_size)
184.             batch_memory = self.memory[sample_index, :]
185.
186.             q_next, q_eval = self.sess.run(
187.                 [self.q_next, self.q_eval],
188.                 feed_dict={
189.                     self.s_: batch_memory[:, -
    self.n_features:],  # fixed params
190.                     self.s: batch_memory[:, :self.n_features],  # newest par
    ams
191.                 })
192.
193.             # change q_target w.r.t q_eval's action
194.             q_target = q_eval.copy()
195.
196.             batch_index = np.arange(self.batch_size, dtype=np.int32)
197.             eval_act_index = batch_memory[:, self.n_features].astype(int)
198.             reward = batch_memory[:, self.n_features + 1]
199.
200.             q_target[batch_index, eval_act_index] = reward + self.gamma * np
    .max(q_next, axis=1)
201.
202.             #q_target[batch_index, eval_act_index] = np.max(reward/1000 + se
    lf.gamma * np.max(q_next, axis=1), -1)
203.
204.
205.             """
206.             For example in this batch I have 2 samples and 3 actions:
207.             q_eval =
208.             [[1, 2, 3],
209.              [4, 5, 6]]
210.             q_target = q_eval =
211.             [[1, 2, 3],
212.              [4, 5, 6]]
213.             Then change q_target with the real q_target value w.r.t the q_ev
    al's action.
214.             For example in:
215.                 sample 0, I took action 0, and the max q_target value is -
    1;
216.                 sample 1, I took action 2, and the max q_target value is -
    2:
217.             q_target =
218.             [[-1, 2, 3],
219.              [4, 5, -2]]
220.             So the (q_target - q_eval) becomes:
221.             [[(-1)-(1), 0, 0],
222.              [0, 0, (-2)-(6)]]
223.             We then backpropagate this error w.r.t the corresponding action
    to network,
224.             leave other action as error=0 cause we didn't choose it.
225.             """
```

```
226.
227.             # train eval network
228.             _, self.cost = self.sess.run([self._train_op, self.loss],
229.                                          feed_dict={self.s: batch_memory[:,
    :self.n_features],
230.                                                     self.q_target: q_target}
    )
231.             self.cost_his.append(self.cost)
232.             self.QVal.append(q_target[batch_index, eval_act_index])
233.
234.             #print("[step:{}] loss:{}".format(e_idx, self.cost))
235.
236.
237.             #saver_path = self.saver.save(self.sess, "model/model"+self.Expe
    riement_id+".ckpt")
238.
239.             # increasing epsilon
240.             self.epsilon = self.epsilon + self.epsilon_increment if self.eps
    ilon < self.epsilon_max else self.epsilon_max
241.             self.learn_step_counter += 1
242.
243.         def plot_cost(self):
244.             import matplotlib.pyplot as plt
245.             plt.plot(np.arange(len(self.cost_his)), self.cost_his)
246.             plt.ylabel('Cost')
247.             plt.xlabel('Training steps')
248.             plt.show()
249.
250.             '''''
251.             plt.plot(np.arange(len(self.QVal)), self.QVal)
252.             plt.ylabel('Qval')
253.             plt.xlabel('training steps')
254.             plt.show()
255.             '''
256.         def saveModel(self, expId, ep, path=None):
257.             ''''' Save the current model to a file.
258.
259.             @param path The location, including file name, for the model to
    be saved. If None, model will be saved to
260.             model/(Experiment_id)/model.ckpt
261.             '''
262.             ts = time.time()
263.             st=datetime.datetime.fromtimestamp(ts).strftime('%Y_%m_%d_%H_%M_
    %S')
264.             os.makedirs(os.getcwd()+'\\model\\Exp'+str(expId)+'\\'+str(ep),
    exist_ok=True)
265.             modelPath='model/Exp'+str(expId)+'/'+str(ep)+'/model.ckpt'
266.             self.saver.save(self.sess, modelPath)  # meta_graph is not recom
    mended
267.
268.
269.         def restoreModel(self, modelPath):
270.             # Restore variables from disk.
271.             self.saver.restore(self.sess, modelPath)
272.             print("Model restored.")
```

```
273.
274.         def getAction_Inference(self, observation):
275.             #this function is used to choose action during inference phase
276.             observation = observation[np.newaxis, :]
277.             actions_value = self.sess.run(self.q_eval, feed_dict={self.s: ob
    servation})
278.             action = np.argmax(actions_value)
279.
280.             return action
281.
```

Appendix D: Yarnell Station House Simulation Code

Building_Model_Yarnell_Lennox_v2.py

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

"""
import numpy as np
import math
#from pyswarm import pso
#import pandas as pd
#import matplotlib.pyplot as plt
import datetime as dt
from scipy.integrate import odeint
#import solar_calcv2 as solarcalc
#import time
#from calendar import monthrange
#import os
Twb_assumed=(63-32)/1.8
TS= 1 #minute
P_ATM=14.4 #psi for Knoxville


class Tstat:
    Heat_SetPt=70  #F
    Cool_SetPt=75  #F
    # Deadband=0.5   #F  Updated 8-2-2019
    Deadband=0.5
    CoolOn=0.0
    HeatOn=0.0
    Mode='Cool'
    def Update(self,Tin):
        Tin=Tin*1.8+32.0
        if self.Mode=='Cool':
            self.HeatOn=0.0
            self.CoolCheck(Tin)
        if self.Mode=='Auto':
            self.CoolCheck(Tin)
            self.HeatCheck(Tin)
        if self.Mode=='Heat':
            self.CoolOn=0.0
            self.HeatCheck(Tin)
        if self.Mode=='Off':
            self.CoolOn=0.0
            self.HeatOn=0.0
    def CoolCheck(self,Tin):
        if Tin>=self.Cool_SetPt+self.Deadband:
            self.CoolOn=1.0
```

```
47.         elif Tin<self.Cool_SetPt:
48.             self.CoolOn=0.0
49.     def HeatCheck(self,Tin):
50.         if Tin<=self.Heat_SetPt-self.Deadband:
51.             self.HeatOn=1.0
52.         if Tin>self.Heat_SetPt:
53.             self.HeatOn=0.0
54.
55. def Psat(T):
56.     T=T*1.8+32
57.     if T<=32:
58.         Ps=math.exp(-10214.165/(T+459.67)+-4.8932428+-
    0.0053765794*(T+459.67)+0.00000019202377*(T+459.67)**2+0.00000000035575832*(T+
    459.67)**3+-9.0344688E-14*(T+459.67)**3+4.1635019*math.log(T+459.67))
59.     else:
60.         Ps=math.exp(-10440.397/(T+459.67)+-11.29465+-
    0.027022355*(T+459.67)+0.00001289036*(T+459.67)**2+-
    0.0000000024780681*(T+459.67)**3+6.5459673*math.log(T+459.67))
61.     return Ps
62.
63. def omega(Tdb,Twb,Ws):
64.     Tdb=Tdb*1.8+32.0
65.     Twb=Twb*1.8+32.0
66.     if Tdb<=32:
67.         w=((1220-0.04*Twb)*Ws-0.24*(Tdb-Twb))/(1220+0.444*Tdb-0.48*Twb)
68.     else:
69.         w=((1093-0.556*Twb)*Ws-0.24*(Tdb-Twb))/(1093+0.444*Tdb-Twb)
70.     return w
71.
72. def enthalpy(Tdb,w):
73.     Tdb=Tdb*1.8+32.0
74.     h=Tdb*0.24+w*(1061+0.444*Tdb)
75.     return h
76.
77.
78. class AC_Low:
79.     C1=0.812449495
80.     C2=0.027954545
81.     C3=0
82.     C4=-0.010272727
83.     C5=0
84.     C6=0
85.     C7=0.388838488
86.     C8=0.002969775
87.     C9=0
88.     C10=0.023786565
89.     C11=0.000768399
90.     C12=-0.001405613
91.     CapfFF_C1=0.785582082
92.     CapfFF_C2=.213657028
93.     EIRfFF_C1=1.293093166
94.     EIRfFF_C2=-0.290723318
95.     def __init__(self):
96.         self.Qrated=26400/3.4121
97.         self.EIRrated=1/(15.26/3.4121)
```

```
        self.CFMrated=815
        self.BFrated=0.043577
              self.A0=11491.14
              self.SHRrated=0.78
          #Total cooling capacity
          def Capacity(self,Tout,CFM,Tin,Twb=Twb_assumed):
              TempMod=self.C1+self.C2*Twb+self.C3*Twb*Twb+self.C4*Tout+self.C5*Tout*Tout+self.C6*Tout*Twb
              FFMod = self.CapfFF_C1+self.CapfFF_C2*CFM/self.CFMrated
              TotalCool=self.Qrated*TempMod*FFMod
              Pws=Psat(Twb)
              Ws_star=0.62198*Pws/(P_ATM-Pws)
              omega_in=omega(Tin,Twb,Ws_star)
              h_in=enthalpy(Tin,omega_in)
              BF=math.exp(-self.A0/(CFM*.075*60))
              hADP=h_in-TotalCool*3.4121/CFM/.075/60/(1-BF)
              omega_ADP=0.000000001056600991*hADP**4 - 0.00000019734430649*hADP**3 + 0.000016070636536*hADP**2 + 0.000074718341939*hADP + 0.00098051700574
              h_Tin_wADP=enthalpy(Tin,omega_ADP)
              SHR=min((h_Tin_wADP-hADP)/(h_in-hADP),1)
              SensibleCool=TotalCool*SHR
              return -SensibleCool
      #EIR for cooling
          def EIR(self,Tout,CFM,Twb=Twb_assumed):
              TempMod=self.C7+self.C8*Twb+self.C9*Twb*Twb+self.C10*Tout+self.C11*Tout*Tout+self.C12*Tout*Twb
              FFMod = self.EIRfFF_C1+self.EIRfFF_C2*CFM/self.CFMrated
              EIR=self.EIRrated*TempMod*FFMod
              return EIR

      class AC_High:
          C1=0.835916
          C2=0.027472
          C3=0
          C4=-0.01079
          C5=-0
          C6=0
          C7=0.43789434
          C8=0.006855168
          C9=0
          C10=0.007961
          C11=0.000671194
          C12=-0.000977162
          CapfFF_C1=0.795663
          CapfFF_C2=.20406
          EIRfFF_C1=1.194069
          EIRfFF_C2=-0.19353
          def __init__(self):
              self.Qrated=36200/3.4121
              self.EIRrated=1/(15.02/3.4121)
              self.CFMrated=1225
              self.BFrated=0.127574
              self.A0=11350.58
              self.SHRrated=0.78
```

```
149.          #Total cooling capacity
150.          def Capacity(self,Tout,CFM,Tin,Twb=Twb_assumed):
151.              TempMod=self.C1+self.C2*Twb+self.C3*Twb*Twb+self.C4*Tout+self.C5
   *Tout*Tout+self.C6*Tout*Twb
152.              FFMod = self.CapfFF_C1+self.CapfFF_C2*CFM/self.CFMrated
153.              TotalCool=self.Qrated*TempMod*FFMod
154.              Pws=Psat(Twb)
155.              Ws_star=0.62198*Pws/(P_ATM-Pws)
156.              omega_in=omega(Tin,Twb,Ws_star)
157.              h_in=enthalpy(Tin,omega_in)
158.              BF=math.exp(-self.A0/(CFM*.075*60))
159.              hADP=h_in-TotalCool*3.4121/CFM/.075/60/(1-BF)
160.              omega_ADP=0.000000001056600991*hADP**4 -
    0.00000019734430649*hADP**3 + 0.000016070636536*hADP**2 + 0.000074718341939*h
   ADP + 0.00098051700574
161.              h_Tin_wADP=enthalpy(Tin,omega_ADP)
162.              SHR=min((h_Tin_wADP-hADP)/(h_in-hADP),1)
163.              SensibleCool=TotalCool*SHR
164.              return -SensibleCool
165.      #EIR for cooling
166.          def EIR(self,Tout,CFM,Twb=Twb_assumed):
167.              TempMod=self.C7+self.C8*Twb+self.C9*Twb*Twb+self.C10*Tout+self.C
   11*Tout*Tout+self.C12*Tout*Twb
168.              FFMod = self.EIRfFF_C1+self.EIRfFF_C2*CFM/self.CFMrated
169.              EIR=self.EIRrated*TempMod*FFMod
170.              return EIR
171.
172.      class HP_High:
173.          C1=0.769502019 #Constant
174.          C2=0 # ID DB
175.          C3=0 # ID DB^2
176.          C4=0.020735431 # OD DB
177.          C5=0.00027153227
178.       # OD DB^2
179.          C6=9.0078224121E-06 # OD DB^3
180.          C7=7.6950202E-01 # Constant
181.          C8=0 # ID DB
182.          C9=0 # ID DB^2
183.          C10 = 2.0735431E-02 # OD DB
184.          C11=2.7153227E-04 # OD DB^2
185.          C12 = 9.0078224E-06 # OD DB^3
186.          CapfFF_C1=0.738234792
187.          CapfFF_C2=0.262218318
188.          EIRfFF_C1=1.576830735
189.          EIRfFF_C2=-0.571402315
190.          def __init__(self):
191.              self.Qrated=33200/3.4121
192.              self.EIRrated=0.230214
193.              self.CFMrated=1225
194.          def Capacity(self,Tout,CFM,Tin):
195.              TempMod=self.C1+self.C2*Tin+self.C3*Tin^2+self.C4*Tout+self.C5*T
   out^2+self.C6*Tout^3
196.              FFMod=self.CapfFF_C1+self.CapfFF_C2*CFM/self.CFMrated
197.              TotalHeat=self.Qrated*TempMod*FFMod
198.              return TotalHeat
```

```
199.         def EIR(self,Tout,CFM,Tin):
200.             TempMod=self.C7+self.C8*Tin+self.C9*Tin^2+self.C10*Tout+self.C11
    *Tout^2+self.C12*Tout^3
201.             FFMod=self.EIRfFF_C1+self.EIRfFF_C2*CFM/self.CFMrated
202.             EIR=self.EIRrated*TempMod*FFMod
203.             return EIR
204.     class HP_Low:
205.         C1=0.784532897
206.         C2=0
207.         C3=0
208.         C4=0.025736096
209.         C5=0
210.         C6=0
211.         C7=1.251998595
212.         C8=0
213.         C9=0
214.         C10=-0.035785145
215.         C11=0.000679714
216.         C12=0
217.         CapfFF_C1=0.846380642
218.         CapfFF_C2=0.154494066
219.         EIRfFF_C1=1.533707442
220.         EIRfFF_C2=-0.526303988
221.         def __init__(self):
222.             self.Qrated=33200/3.4121
223.             self.EIRrated=0.230214
224.             self.CFMrated=1225
225.         def Capacity(self,Tout,CFM,Tin):
226.             TempMod=self.C1+self.C2*Tin+self.C3*Tin^2+self.C4*Tout+self.C5*T
    out^2+self.C6*Tout^3
227.             FFMod=self.CapfFF_C1+self.CapfFF_C2*CFM/self.CFMrated
228.             TotalHeat=self.Qrated*TempMod*FFMod
229.             return TotalHeat
230.         def EIR(self,Tout,CFM,Tin):
231.             TempMod=self.C7+self.C8*Tin+self.C9*Tin^2+self.C10*Tout+self.C11
    *Tout^2+self.C12*Tout^3
232.             FFMod=self.EIRfFF_C1+self.EIRfFF_C2*CFM/self.CFMrated
233.             EIR=self.EIRrated*TempMod*FFMod
234.             return EIR
235.
236.     class TwoStageHP:
237.         CoolHigh=AC_High()
238.         CoolLow=AC_Low()
239.         HeatHigh=HP_High()
240.         HeatLow=HP_Low()
241.         tau=1.5
242.         SuppHeat=10000 #W
243.         SSCapacity=0
244.         Capacity=0
245.         Power=0
246.
247.
248.     class ZoneControl:
249.         def __init__ (self,zones):
250.             self.Cool=[]
```

```
251.                self.Heat=[]
252.                self.Fan=[]
253.                self.PIAB=[]
254.                self.CFM=[]
255.                self.Qzone=[]
256.                for i in range(0,zones):
257.                    self.Cool.append(0)
258.                    self.Heat.append(0)
259.                    self.Fan.append(0)
260.                    self.PIAB.append(0.4)
261.                    self.CFM.append(0)
262.                    self.Qzone.append(0)
263.                self.num_zones=zones
264.                self.Theat_target=90.0
265.                self.Theat_max=self.Theat_target+10.0
266.                self.Tcool_target=55
267.                self.Tcool_max=self.Tcool_target+7
268.                self.Tsupply=0
269.                self.Stage=0
270.                self.OffTimer=6
271.                self.OpposingDemandtimer=0
272.                self.StageTimer=0
273.                self.CurrentCapacity=0
274.                self.OldCapacity=0
275.                self.SSCapacity=0
276.                self.MinCFM=450
277.                self.MaxCFM=1200
278.                self.HeatingAirReduction=0.0
279.                self.ContinuousFanReduction=0.5
280.                self.Mode='Off'
281.                self.OldMode='Off'
282.                self.TotalCFM=0
283.                self.Power=0
284.
285.            def PIABtoCFM(self):
286.                if self.Mode=='Cool':
287.                    ZonesCalling=sum(self.Cool)
288.                    ZonePIABsum=0
289.                    for i in range(0,len(self.Cool)):
290.                        ZonePIABsum=ZonePIABsum+self.Cool[i]*self.PIAB[i]
291.                    TotalPIAB=min(ZonePIABsum,1)+(ZonesCalling-1)/3
292.                    if TotalPIAB>0:
293.                        self.TotalCFM=TotalPIAB*(self.MaxCFM-
    self.MinCFM)+self.MinCFM
294.                        for i in range(0,len(self.Cool)):
295.                            self.CFM[i]=self.Cool[i]*self.PIAB[i]/ZonePIABsum*se
    lf.TotalCFM
296.                elif self.Mode=='Heat':
297.                    ZonesCalling=sum(self.Heat)
298.                    ZonePIABsum=0
299.                    for i in range(0,len(self.Heat)):
300.                        ZonePIABsum=ZonePIABsum+self.Heat[i]*self.PIAB[i]
301.                    TotalPIAB=min(ZonePIABsum+(ZonesCalling-1)/3,1.0)
302.                    TotalPIAB=TotalPIAB*(1.0-self.HeatingAirRecution)
303.                    if TotalPIAB>0:
```

```
                    self.TotalCFM=TotalPIAB*(self.MaxCFM-
self.MinCFM)+self.MinCFM
                    for i in range(0,len(self.Heat)):
                        self.CFM[i]=self.Heat[i]*self.PIAB[i]/ZonePIABsum*se
lf.TotalCFM
            elif self.Mode=='Fan':
                ZonesCalling=sum(self.Fan)
                ZonePIABsum=0
                for i in range(0,len(self.Fan)):
                    ZonePIABsum=ZonePIABsum+self.Fan[i]*self.PIAB[i]
                TotalPIAB=min(ZonePIABsum+(ZonesCalling-1)/3,1.0)
                TotalPIAB=TotalPIAB*(1.0-self.ContinuousFanReduction)
                self.TotalCFM=TotalPIAB*(self.MaxCFM-
self.MinCFM)+self.MinCFM
                for i in range(0,len(self.Fan)):
                    self.CFM[i]=self.Fanl[i]*self.PIAB[i]/ZonePIABsum*self.T
otalCFM
            else:
                self.TotalCFM=0
                for i in range(0,self.num_zones):
                    self.CFM[i]=0

        def Update(self,HP,Tstat,Tin,Tout,Twb=Twb_assumed):
            self.OldCapacity=self.CurrentCapacity
            self.OldMode=self.Mode
            self.Power=0
            for i in range(0,len(Tstat)):
                if Tstat[i].CoolOn==1:
                    self.Cool[i]=1
                    self.Heat[i]=0
                    self.Fan[i]=0
                elif Tstat[i].HeatOn==1:
                    self.Cool[i]=0
                    self.Heat[i]=1
                    self.Fan[i]=0
                elif Tstat[i].Mode=='Fan':
                    self.Cool[i]=0
                    self.Heat[i]=0
                    self.Fan[i]=0
                else:
                    self.Cool[i]=0
                    self.Heat[i]=0
                    self.Fan[i]=0

            if self.OffTimer>=5:
                if (sum(self.Cool)>0 and sum(self.Heat)==0) or (self.OldMode
=='Cool' and self.StageTimer<=4):
                    self.Mode='Cool'
                    self.PIABtoCFM()
                    if self.Stage==0:
                        self.Stage=1
                        self.StageTimer+=1
                        self.SSCapacity=HP.CoolLow.Capacity(Tout,self.TotalC
FM,Tin)
```

```
                            self.Power=HP.CoolLow.EIR(Tout,self.TotalCFM,Tin)*-
    self.SSCapacity
                        else:
                            self.Tsupply=(Tin*1.8+32.0)+self.OldCapacity*3.4121/
    self.TotalCFM/60.0/0.075/0.24
                            if self.Tsupply>self.Tcool_max and self.StageTimer>4
    :
                                if self.Stage==1:
                                    self.StageTimer=0
                                self.Stage=2
                            elif self.Tsupply<self.Tcool_target and self.StageTi
    mer>4:
                                if self.Stage==2:
                                    self.StageTimer=0
                                self.Stage=1
                            self.StageTimer+=1
                            if self.Stage==1:
                                self.SSCapacity=HP.CoolLow.Capacity(Tout,self.To
    talCFM,Tin)
                                self.Power=HP.CoolLow.EIR(Tout,self.TotalCFM,Tin
    )*-self.SSCapacity
                            if self.Stage==2:
                                self.SSCapacity=HP.CoolHigh.Capacity(Tout,self.T
    otalCFM,Tin)
                                self.Power=HP.CoolHigh.EIR(Tout,self.TotalCFM,Ti
    n)*-self.SSCapacity
                        self.CurrentCapacity=self.SSCapacity-(self.SSCapacity-
    self.OldCapacity)*math.exp(-TS/HP.tau)
                    elif sum(self.Heat)>0 or (self.OldMode=='Heat' and self.Stag
    eTimer<=4):
                        self.Mode='Heat'
                        self.PIABtoCFM()
                        if self.Stage==0:
                            self.Stage=1
                            self.StageTimer+=1
                            self.SSCapacity=HP.HeatLow.Capacity(Tout,self.TotalC
    FM,Tin)
                            self.Power=HP.HeatLow.EIR(Tout,self.TotalCFM,Tin)*se
    lf.SSCapacity
                        else:
                            self.Tsupply=(Tin*1.8+32.0)+self.OldCapacity*3.4121/
    self.TotalCFM/60.0/0.075/0.24
                            if self.Tsupply<self.Theat_target and self.StageTime
    r>4:
                                if self.Stage==1:
                                    self.StageTimer=0
                                self.Stage=2
                                if self.Stage==2:
                                    self.ElecHeatTimer=0
                                    self.Stage=3
                            elif self.Tsupply>self.Theat_max and self.StageTimer
    >4:
                                if self.Stage==2:
                                    self.StageTimer=0
                                self.Stage=1
```

```
                                if self.Stage==3 and self.ElecHeatTimer>2:
                                    self.Stage=2
                            self.StageTimer+=1
                            if self.Stage==1:
                                self.SSCapacity=HP.HeatLow.Capacity(Tout,self.TotalCFM,Tin)
                                self.Power=HP.HeatLow.EIR(Tout,self.TotalCFM,Tin)*self.SSCapacity
                            if self.Stage>=2:
                                self.SSCapacity=HP.HeatHigh.Capacity(Tout,self.TotalCFM,Tin)
                                self.Power=HP.HeatHigh.EIR(Tout,self.TotalCFM,Tin)*self.SSCapacity
                            if self.Stage==3:
                                self.ElecHeatTimer+=1
                                self.SSCapacity=self.SSCapacity+HP.SuppHeat
                                self.Power=self.Power+HP.SuppHeat
                        self.CurrentCapacity=self.SSCapacity-(self.SSCapacity-self.OldCapacity)*math.exp(-TS/HP.tau)
                    elif sum(self.Heat)==0 and sum(self.Cool)==0 and sum(self.Fan)==0 and self.StageTimer>4:
                        if self.OldMode == 'Cool' or self.OldMode=='Heat':
                            self.OffTimer=0
                        self.Mode='Off'
                        self.PIABtoCFM()
                        self.Stage=0
                        self.OffTimer+=1
                        self.SSCapacity=0
                        self.CurrentCapacity=0
                    elif sum(self.Fan)!=0 and sum(self.Cool)==0 and sum(self.Heat)==0:
                        if self.OldMode == 'Cool' or self.OldMode=='Heat':
                            self.OffTimer=0
                        self.Mode='Fan'
                        self.PIABtoCFM()
                        self.Stage=0
                        self.OffTimer+=1
                        self.SSCapacity=0
                        self.CurrentCapacity=self.SSCapacity-(self.SSCapacity-self.OldCapacity)*math.exp(-TS/HP.tau)
                    else:
                        print('ERROR IN DETERMINING MODE')
                else:
                    self.Mode='Off'
                    self.Stage=0
                    self.PIABtoCFM()
                    self.OffTimer+=1
                    self.SSCapacity=0
                    self.CurrentCapacity=0
                if self.TotalCFM==0:
                    for i in range(0,len(Tstat)):
                        self.Qzone[i]=0
                else:
                    for i in range(0,len(Tstat)):
```

```
                    self.Qzone[i]=self.CurrentCapacity*self.CFM[i]/self.Tota
lCFM


    class Area:
        def __init__(self, front, back, left, right):
            self.front=front
            self.back=back
            self.left=left
            self.right=right
        def sum(self):
            return(self.front+self.back+self.right+self.left)

    class Building():
        def __init__(self):
            self.Tin=[22.0,22.0]
            self.Twall=[22.0,22.0]
            self.Tmass=[22.0,22.0]
            self.Tattic=22.0
            self.time=0

            self.Rwall=[0.03093243,0.03093243]
            self.Cwall=[3525806.234,3525806.234]

            self.Rmass=[0.002100298,0.00019]
            self.Cmass=[11313219.26,4629470.281]

            self.Rattic=0.199797116
            self.Cattic=2181004.52
            self.Rroof=0.003

            self.Cin=[2830497.042,2830497.042]

            self.Rf=[0.007423197,0.00223278]
            self.Cfl=10640487
            self.C1=0.992013217
            self.C2=0.608357208
            self.C3=0.15

            self.Rwin=[0.011956817,0.011956817]

            self.Sp1=0.8
            self.Sp2=0.9
            self.Sp3=0.2

            self.orientation= 88.0
            self.SHGC=0.56
            self.WA=19.0
            self.WallAbsorptivity=0.4
            self.RoofAbsorptivity=0.8
            self.latitude = 35.926905
            self.longitude=-84.159739
            self.timezone=-4
            self.walls=Area(698.0*0.092903,822.7*0.092903,650.0*0.092903,482
.0*0.092903)
```

```
491.             self.windows=Area((37.0+59.5)*0.092903,(100.8+61.0)*0.092903,13.
   0*0.092903,6.0*0.092903)
492.             self.roofs=Area(1.0,1.0,0.0,0.0)
493.             self.Qac=0
494.             self.eir=0
495.             self.Tsolr=0
496.             self.Tsolw=0
497.             self.Qsolar=0
498.             self.Tout=0
499.             self.IHL=[0,0]
500.             self.Qzone=[0,0]
501.
502.         def model3_1(self,z,t): # 16 parameters; when XU>=XD: Rf1; Cfl & Rwi
   n
503.
504.             XD = z[0] #XD=Tin,downstairs; YD=Twall,downstairs; MD=Tmass,down
   stair; W=Tfloor (between downstair and upstair);
505.             YD = z[1] #XU=Tin,upstairs; YU=Twall,upstairs; MU=Tmass,upstair;
    Z=Tattic
506.             MD = z[2]
507.             W  = z[3]
508.             XU = z[4]
509.             YU = z[5]
510.             MU = z[6]
511.             Z  = z[7]
512.
513.             #dXDdt = ((YD-XD)/(Rwall/2)*0.54+(MD-XD)/Rmass1+(Tambient-
   XD)/0.0269*0.67+(W-XD)/(0.0071/2)+IHL1*C1*0.84-CQ1*((XD-
   24.4444)*0.0352+1.0055)*C2*0.9+Qsolar1*C3*0.17+min(Cmin*(XU-XD),0))/Cin
514.             dXDdt = ((YD-XD)/(self.Rwall[0]/2)+(MD-
   XD)/self.Rmass[0]+(self.Tout-XD)/self.Rwin[0]*0.6+(XU-XD)/self.Rf[0]+(W-
   XD)/(0.0071/2)+self.IHL[0]*self.C1*self.Sp1+self.Qzone[0]*self.C2*self.Sp2+sel
   f.Qsolar*self.C3*self.Sp3)/self.Cin[0]
515.             dYDdt = ((self.Tsolw-YD)/(self.Rwall[0]/2) - (YD-
   XD)/(self.Rwall[0]/2))/self.Cwall[0]
516.             dMDdt = (-(MD-XD)/self.Rmass[0]+self.IHL[0]*self.C1*(1-
   self.Sp1)+self.Qzone[0]*self.C2*(1-self.Sp2)+self.Qsolar*self.C3*(1-
   self.Sp3))/self.Cmass[0]
517.             #dWdt = ((XU-W)/(Rf1/2)-(W-XD)/(Rf1/2))/3546829
518.             dWdt = ((XU-W)/(0.0071/2)-(W-XD)/(0.0071/2))/self.Cfl
519.
520.             #((max(XU-XD,0)*Rf1+max(XD-XU,0)*Rf2)/2)=(min(max(XU-
   XD,0)*Rf1+max(XD-XU,0)*Rf2,Rf1)/2)
521.             dXUdt = ((YU-XU)/(self.Rwall[1]/2)*1+(MU-
   XU)/self.Rmass[1]+(self.Tout-XU)/self.Rwin[1]*0.4-(XU-XD)/self.Rf[0]-(XU-
   W)/(0.0071/2)+(Z-
   XU)/self.Rattic+self.IHL[1]*self.C1*self.Sp1+self.Qzone[1]*self.C2*self.Sp2+se
   lf.Qsolar*self.C3*self.Sp3)/self.Cin[1]
522.             dYUdt = ((self.Tsolw-YU)/(self.Rwall[1]/2) - (YU-
   XU)/(self.Rwall[1]/2))/self.Cwall[1]
523.             dMUdt = (-(MU-XU)/self.Rmass[1]+self.IHL[1]*self.C1*(1-
   self.Sp1)+self.Qzone[1]*self.C2*(1-self.Sp2)+self.Qsolar*self.C3*(1-
   self.Sp3))/self.Cmass[1]
524.             dZdt = ((self.Tsolr-Z)/0.003-(Z-XU)/self.Rattic)/self.Cattic
525.
```

```
526.                 dZZdt=[dXDdt,dYDdt,dMDdt,dWdt,dXUdt,dYUdt,dMUdt,dZdt]
527.                 return dZZdt
528.
529.             def model3_2(self,z,t): # 16 parameters: when XU<XD: Rf2; Cfl & Rwin

530.
531.
532.                 XD = z[0] #XD=Tin,downstairs; YD=Twall,downstairs; MD=Tmass,down
    stair; W=Tfloor (between downstair and upstair);
533.                 YD = z[1] #XU=Tin,upstairs; YU=Twall,upstairs; MU=Tmass,upstair;
     Z=Tattic
534.                 MD = z[2]
535.                 W  = z[3]
536.                 XU = z[4]
537.                 YU = z[5]
538.                 MU = z[6]
539.                 Z  = z[7]
540.
541.                 #dXDdt = ((YD-XD)/(Rwall/2)*0.54+(MD-XD)/Rmass1+(Tambient-
    XD)/0.0269*0.67+(W-XD)/(0.0071/2)+IHL1*C1*0.84-CQ1*((XD-
    24.4444)*0.0352+1.0055)*C2*0.9+Qsolar1*C3*0.17+min(Cmin*(XU-XD),0))/Cin
542.                 dXDdt = ((YD-XD)/(self.Rwall[0]/2)+(MD-
    XD)/self.Rmass[0]+(self.Tout-XD)/self.Rwin[0]*0.6+(XU-XD)/self.Rf[1]+(W-
    XD)/(0.0071/2)+self.IHL[0]*self.C1*self.Sp1+self.Qzone[0]*self.C2*self.Sp2+sel
    f.Qsolar*self.C3*self.Sp3)/self.Cin[0]
543.                 dYDdt = ((self.Tsolw-YD)/(self.Rwall[0]/2) - (YD-
    XD)/(self.Rwall[0]/2))/self.Cwall[0]
544.                 dMDdt = (-(MD-XD)/self.Rmass[0]+self.IHL[0]*self.C1*(1-
    self.Sp1)+self.Qzone[0]*self.C2*(1-self.Sp2)+self.Qsolar*self.C3*(1-
    self.Sp3))/self.Cmass[0]
545.                 #dWdt = ((XU-W)/(Rf1/2)-(W-XD)/(Rf1/2))/3546829
546.                 dWdt = ((XU-W)/(0.0071/2)-(W-XD)/(0.0071/2))/self.Cfl
547.
548.                 #((max(XU-XD,0)*Rf1+max(XD-XU,0)*Rf2)/2)=(min(max(XU-
    XD,0)*Rf1+max(XD-XU,0)*Rf2,Rf1)/2)
549.                 dXUdt = ((YU-XU)/(self.Rwall[1]/2)*1+(MU-
    XU)/self.Rmass[1]+(self.Tout-XU)/self.Rwin[1]*0.4-(XU-XD)/self.Rf[1]-(XU-
    W)/(0.0071/2)+(Z-
    XU)/self.Rattic+self.IHL[1]*self.C1*self.Sp1+self.Qzone[1]*self.C2*self.Sp2+se
    lf.Qsolar*self.C3*self.Sp3)/self.Cin[1]
550.                 dYUdt = ((self.Tsolw-YU)/(self.Rwall[1]/2) - (YU-
    XU)/(self.Rwall[1]/2))/self.Cwall[1]
551.                 dMUdt = (-(MU-XU)/self.Rmass[1]+self.IHL[1]*self.C1*(1-
    self.Sp1)+self.Qzone[1]*self.C2*(1-self.Sp2)+self.Qsolar*self.C3*(1-
    self.Sp3))/self.Cmass[1]
552.                 dZdt = ((self.Tsolr-Z)/0.003-(Z-XU)/self.Rattic)/self.Cattic
553.
554.                 dZZdt=[dXDdt,dYDdt,dMDdt,dWdt,dXUdt,dYUdt,dMUdt,dZdt]
555.                 return dZZdt
556.
557.
558.             def Energy_Balance(self,zoning,df):
559.                 tspan=[0,60]
560.                 z0=[0]*8
561.                 z0[0] = self.Tin[0]
```

```
562.                z0[1] = self.Twall[0]
563.                z0[2] = self.Tmass[0]
564.                z0[3] = self.Tfloor
565.                z0[4] = self.Tin[1]
566.                z0[5] = self.Twall[1]
567.                z0[6] = self.Tmass[1]
568.                z0[7] = self.Tattic
569.
570.                self.Tout = df.Tout
571.                self.Tsolw = df.Tsolw
572.                self.Tsolr = df.Tsolr
573.                self.Qsolar = df.Qsolar
574.
575.
576.
577.
578.                self.IHL[0] = df.IHL*0.65
579.                self.IHL[1] = df.IHL*0.35
580.
581.
582.
583.                ###
584.                # self.IHL[0] = 0
585.                # self.IHL[1] = 0
586.                ###
587.
588.
589.
590.                self.Qzone[0]=zoning.Qzone[0]
591.                self.Qzone[1]=zoning.Qzone[1]
592.
593.                if z0[4]>=z0[0]:
594.                    z=odeint(self.model3_1,z0,tspan)
595.                if z0[4]<z0[0]:
596.                    z=odeint(self.model3_2,z0,tspan)
597.
598.                self.Tin[0]=z[1][0]
599.                self.Twall[0]=z[1][1]
600.                self.Tmass[0]=z[1][2]
601.                self.Tfloor=z[1][3]
602.                self.Tin[1]=z[1][4]
603.                self.Twall[1]=z[1][5]
604.                self.Tmass[1]=z[1][6]
605.                self.Tattic=z[1][7]
606.
607.    def changetype(x):
608.        if type(x) is not 'float':
609.            try:
610.                x=x[0]
611.            except:
612.                'Error Changing Data Type'
613.        return x
614.
615.    def initialize(num_zones=2):
616.        tstat=[]
```

```
617.          for i in range(0,num_zones):
618.              tstat.append(Tstat())
619.          b=Building()
620.          z=ZoneControl(num_zones)
621.          hp=TwoStageHP()
622.          #b.time=time
623.          return b,hp,z,tstat
624.
625.      def simulate(building,hp,zoning,tstat,df):
626.          Tavg=0
627.          for i in range(0,len(tstat)):
628.              tstat[i].Update(building.Tin[i])
629.              Tavg+=building.Tin[i]
630.          Tavg=Tavg/len(tstat)
631.          zoning.Update(hp,tstat,Tavg,df.Tout)
632.          building.Energy_Balance(zoning,df)
633.          return building,hp,zoning,tstat
634.
635.
636.
637.
638.      #
639.      #
640.      #T_inputs=[21,22,21,22]
641.      #weather_inputs=[35,900,200,5]
642.      #SetPt=23.0
643.      #ACStatus=1.0
644.      #IHL=200
645.      #t_current=dt.datetime(2019,2,10,12)
646.      #
647.      #T_outputs,t_new,power,ACStatus=simulate(T_inputs,weather_inputs,t_curre
    nt,SetPt,IHL,ACStatus)
648.      #print(T_outputs,t_new,power,ACStatus)
```

VITA

Evan McKee was born in Knoxville, TN to Mike and Kim McKee and is the youngest of three siblings alongside Molly and Jesse. He graduated from Lipscomb University with a B.S. in Marketing in 2010, and spent five years playing music professionally in Nashville. In 2015, he entered the University of Tennessee to study electrical engineering and received the Pete Barile, Sr. design scholarship his sophomore year. After graduating in 2018, Evan worked as a graduate research assistant for CURENT and later for Oak Ridge National Laboratory on the project described in this work. Evan graduated with an M.S. degree in Electrical Engineering in 2019 and went on to work in industry.