UNIVERSITY OF
LINCOLN

Deep Reinforcement Learning and Model Predictive Control Approaches for the Scheduled Operation of Domestic Refrigerators

Mohammad Reza Zavvar Sabegh

Doctor of Philosophy

School of Engineering

College of Science

June 2022

Deep Reinforcement Learning and Model Predictive Control Approaches for the Scheduled Operation of Domestic Refrigerators

Mohammad Reza Zavvar Sabegh

A thesis submitted in partial fulfilment of the requirements of the University of Lincoln for the degree of Doctor of Philosophy

School of Engineering

College of Science

June 2022

DECLARATION

This dissertation is the result of my own work and includes nothing, which is the outcome of work done in collaboration except where specifically indicated in the text. It has not been previously submitted, in part or whole, to any university of institution for any degree, diploma, or other qualification.

Signed: Mohammad Reza Zavvar Sabegh

Date:29/04/2022

ABSTRACT

Excess capacity of the UK's national grid is widely quoted to be reducing to around 4% over the coming years [1] as a consequence of increased economic growth (and hence power usage) and reductions in power generation plants. There is concern that short term variations in power demand could lead to serious wide-scale disruption on a national scale. This is therefore spawning greater attention on augmenting traditional generation plants with renewable and localized energy storage technologies, and consideration of improved demand side responses (DSR), where power consumers are incentivized to switch off assets when the grid is under pressure. It is estimated, for instance, that refrigeration/HVAC systems alone could account for ~14% of the total UK energy usage, with refrigeration and water heating/cooling systems, in particular, being able to act as real-time 'buffer' technologies that can be demand-managed to accommodate transient demands by being switched-off for short periods without damaging their outputs. Large populations of thermostatically controlled loads (TCLs) hold significant potential for performing ancillary services in power systems since they are well-established and widely distributed around the power network. In the domestic sector, refrigerators and freezers collectively constitute a very large electrical load since they are continuously connected and are present in almost most households. The rapid proliferation of the 'Internet of Things' (IoT) now affords the opportunity to monitor and visualise smart buildings appliances performance and specifically, schedule the operation of the widely distributed domestic refrigerator and freezers to collectively improve energy efficiency and reduce peak power consumption on the electrical grid. To accomplish this, this research proposes the real-time estimation of the thermal mass of individual refrigerators in a network using on-line parameter identification, and the co-ordinated (ON-OFF) scheduling of the refrigerator compressors to maintain their respective temperatures within specified hysteresis bands—commensurate with accommodating food safety standards. Custom Model Predictive Control (MPC) schemes and a Machine Learning algorithm (Reinforcement Learning) are researched to realize an appropriate scheduling methodology which is implemented through COTS IoT hardware. Benefits afforded by the proposed schemes are investigated through experimental trials which show that the co-ordinated operation of domestic refrigerators can 1) reduce the peak power consumption as seen from the perspective of the electrical power grid (i.e. peak power shaving), 2) can adaptively control the temperature hysteresis band of individual refrigerators to increase operational efficiency, and 3)

contribute to a widely distributed aggregated load shed for Demand Side Response purposes in order to aid grid stability. Comparative studies of measurements from experimental trials show that the co-ordinated scheduling of refrigerators allows energy savings of between 19% and 29% compared to their traditional isolated (non-co-operative) operation. Moreover, by adaptively changing the hysteresis bands of individual fridges in response to changes in thermal behaviour, a further 20% of savings in energy are possible at local refrigerator level, thereby providing benefits to both network supplier and individual consumer.

ACKNOWLEDGEMENTS

I want to thank my supervisor Prof. Chris Bingham, for his exhaustive guidance throughout my PhD study. His enthusiasm towards developing original ideas and applying to practical problems has influenced me deeply. Chris always provided me with valuable suggestions and explained things in such an intuitive way, which helped me tremendously.

Finally, thank the University of Lincoln, School of Engineering and Dr Argyrios Zolotas for giving me the opportunity to study for a PhD.

Publications stemming from research in this thesis

- Zavvar Sabegh MR, Bingham C, (2019) Model Predictive Control with Binary Quadratic Programming for the Scheduled Operation of Domestic Refrigerators. Energies 12:4649.
- Zavvar Sabegh MR, Bingham C, (2021) Synchronised Power Scheduling of Widely Distributed Refrigerators using IoT. In: Proceedings of the 10th International Conference on Smart Grids and Green IT Systems.
- Zavvar Sabegh MR, Bingham CM, (2019) Impact of Hysteresis Control and Internal Thermal Mass on the Energy Efficiency of IoT-Controlled Domestic Refrigerators. In: IEEE 7th International Conference on Smart Energy Grid Engineering (SEGE). pp 103–107.
- Zavvar Sabegh MR, Bingham CM, Power Scheduling of Widely Distributed Refrigerators using Deep Reinforcement Learning, in writing.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Demand Side Response in the Residential Sector

Demand Side Response (DSR) focuses on adjusting demand to follow power production and infrastructure availability for effective and efficient power system operation. With fossil fuels becoming ever more costly and the use of renewable energy power rapidly increasing in the UK [1], it has become increasingly problematic to match supply with demand. By increasing the penetration of renewables in power systems, demand side participation becomes more important. It is well known that demand response increases system reliability and flexibility to manage the variability and uncertainty of some renewable energy resources, decreases the cost of operation, and enhances system efficiency. Participation of demand response can be achieved by active consumer participation in real-time to maintain the balance between generation and demand using two-way communication [2], [3]. DSR can vary from a temporary short period demand adjustment to a permanent change in the load. Power system demands can be broadly classified as industrial, commercial, and domestic [4]. The residential sector, in particular, has suitable appliances present in most households eg. refrigerators and freezers, which provides an opportunity to devise and test candidate technologies that can be widely deployed. The total load that can be released by controlling small

domestic loads cannot individually match those of large industries. However, the unavailability of large industrial loads for control has a substantial impact on the service commitment to the power system. Studies have therefore considered exploiting the potential availability of the widely distributed residential sector [5], [6]. The authors of [5] found that more frequent and short-term switching (On/Off) of suitable residential loads are more acceptable than infrequent and long curtailment. The author of [6] analysed different household appliances and their potential to delay (remain Off) their load consumption and concluded that 5% to 20% of these devices could use a delay option in the future.

1.2 Thermostatically Controlled Loads

Thermostatically Controlled Loads (TCLs) have been identified as key contributors to facilitate the implementation of improved power control and demand side response schemes. Refrigerators in particular form a fundamental part of daily power consumption with around 1.4 billion fridges being used across the world which consume around 650 TWh per year [7].

In 2018, the total demand for electricity in the UK was 334 TWh over the year, with domestic energy consumption accounting for 31.7% of the total [8], [9]. Buildings are therefore natural candidates for providing demand-side flexibility. A study in [10] attempted to quantify the DSR potential of TCLs by calculating their average electricity consumption in households, as shown in Figure 1.1. The number of appliances in the calculation is estimated through a questionnaire survey. The average annual electricity consumption of a specific appliance is calculated from the average consumption of both new and decade old equipment. Among those loads, thermostatically controlled appliances, including space and water heaters, refrigerators and freezers, have presented great value, due to their considerable volume and stable profile during the day and throughout the year. For example, refrigerators have the highest electricity

consumption, as shown in Figure 1.1, followed by the freezers and heaters, and are all thermostatically controlled.

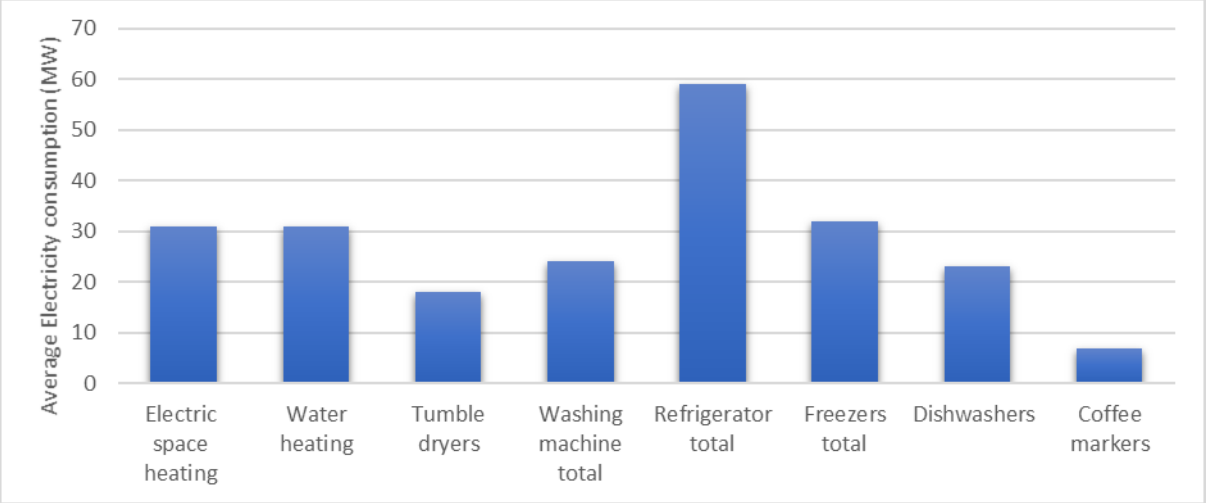


Figure 1.1: The average electricity consumption of different household appliance [11]

In 2019, there were 27.8 million households in the UK (Office for National Statistics, 2015), and it is estimated that there are around 50 million refrigerators in the UK along with an annual energy consumption approaching 18 TWh/year [12]. Notably, the UK consumes ~1/69th of the total electrical power generated globally, and ~1/36th of the electricity globally consumed for refrigeration and freezing [13], [14]. Based on Statista reports, Figure 1.2 presents the brands of refrigerators and combined fridge/freezers ranked by their number of users in the UK in 2018. As can be seen, the highest value is owned by Beko. Beko fridges and fridge/freezers use a dial to control both the fridge and freezer temperatures. The dial displays 5 numbers, where 0 indicates no cooling, 1 and 5 indicate the warmest and coldest temperatures, respectively. Beko's American Style Fridge Freezers offer digital control panel that sits flush within the main doors. Moreover, the American Style are powered by two separate cooling systems which use automatic fans. The fridge focuses on providing 2x faster cooling and keeping humidity levels high to maintain food freshness while the freezer focuses on maintaining a dry environment to prevent ice build-up. In a fridge freezer with a single cooling system,

both the fridge and freezer compartments work collectively. Whereas in a dual cooling system, the fridge and freezer compartments work separately which allows each compartment to focus on specific tasks, such as keeping food fresh and the freezer free of ice build-up [124].

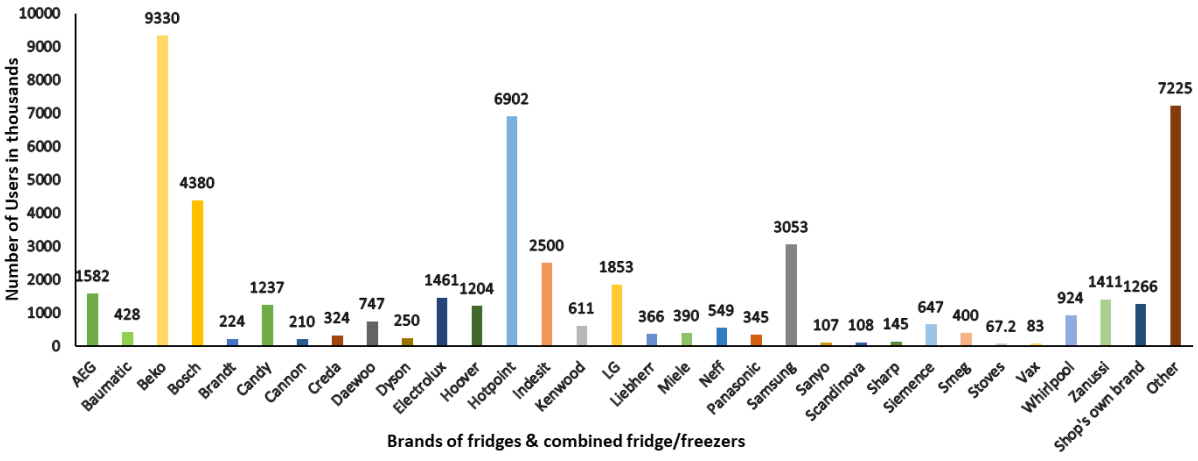


Figure 1.2: The number of UK users of brands of refrigerators and fridge/freezers in 2018

The main feature which makes refrigerators a strong candidate for direct load control demand response programs is that they have a stable aggregate load profile throughout the day regardless of the small variations resulting from changes in the ambient temperature in summer and winter and the frequency of door openings which increases in the evening around dinner time. Due to the thermal storage of the chilled contents, short interruptions in the power supply of refrigerators should not affect the service provided so, fridges with thermal are best suited for DSR.

1.3 Research Objectives

The research presented in this thesis is divided into four main parts. The first provides a real-time recursive based system identification strategy to monitor and estimate the

internal temperature of individual domestic refrigerators based on their internal thermal mass (product). A first-order ARX (Autoregressive with Exogenous Variables) model is implemented then, real-time recursive based system identification strategies with and without considering ambient temperature effects are implemented and compared. The results of the experimental trials demonstrate that a real-time model update can identify the refrigerator dynamics and respond to common disturbances such as door openings. The second proposes the deployment of an IoT scheme for local and remote accesses to monitor and control the appliances and schedule the operation of aggregated domestic refrigerators. An IP-based synchronized wireless mesh network is implemented through IoT hardware (based on a NodeMCU) and Google Sheets to monitor and schedule the operation of aggregated domestic refrigerators. Benefits afforded by the proposed technique are investigated through experimental trials sited in three different domestic locations in the city of Lincoln, UK. Results show that the proposed network is able to monitor the widely distributed refrigerators and schedule the appliances. In parts three and four of the thesis, the use of a new, custom MPC control scheme and a Machine Learning algorithm (Reinforcement Learning) are developed, respectively, for jointly scheduling the operation of multiple distributed refrigerators. The effectiveness of the scheduling approaches is analysed through experimental trials on a suite of common domestic refrigerators distributed across Lincoln (UK), which is shown to facilitate peak power shaving which, if expanded to a larger scale, could aid grid stability and contribute to effective DSR. Moreover, the impact of temperature hysteresis bands is investigated, and it is shown that by adaptively changing the hysteresis bands of individual fridges in response to changes in thermal behaviour, 20% of savings in energy are possible at the local refrigerator level.

2 LITERATURE SURVEY

2.1 Smart Grid and Demand Side Response

The smart grid was introduced in 1980 to provide cooperation between suppliers and consumers in response to price and system reliability concerns. This approach was initially termed “homeostasis utility control”. An approach that uses suppliers' and consumers' economic reactions to price together with developments in communication and computation hardware were used to devise internally correcting control schemes that are efficient, accurate, and consistent [15]. To build a national smart grid, a large number of renewable sources are required (both large scale and distributed) along with the electrification of transportation and heating. An integrated approach to control and communication will enable: 1) consumer participation in the energy market, 2) energy storage and 3) the integration of local and national electricity markets [16]. As well as changing the supply side, the demand side can also be transformed. The term demand side response (DSR) refers to the modification of consumer loads as a result of price or grid stability [17]. In order to improve the efficiency of electricity markets and maintain system stability, DSR mechanisms have been widely investigated eg. [18]. In a power system, aggregated generation and consumption must be balanced almost

instantaneously and continuously, and the network operator has various controllable reserves to achieve this balance [19]. As a result, power systems are traditionally designed and operated according to the 'supply follows demand' principle, which has been recently criticized [15]:

1. Fast activation of reserves leads to inefficient fuel use.
2. In order to supply the peak demand, additional generation and transportation capacity must be available.
3. The use of fixed electricity prices discourages the consumers to follow the supply.
4. The absence of real-time supply-side information for the consumers, leads to vulnerable situation to both short-term and long-term emergencies.

Accordingly, the concept of demand response in Europe entails that end-user consumers adjust their electric usage away from their normal behavioural patterns in response to changes in electricity prices, or if their bid is accepted, they can adjust their usage if they respond to an incentive payment [20]. The two general types of demand response are: 1) controllable demand response that can be dispatched similarly to generation and 2) dynamic tariff scheme based on price-based demand response [21].

Electricity demand is affected by numerous factors: 1) weather [22], [23], 2) building characteristics [24]-[26], 3) design of appliances [27], [28] and 4) Control of the domestic appliances [29], [30]. These factors contribute to the intensity of demand. Homes that are poorly insulated, for example, use more energy to reach the same temperature as those that are well insulated. Older appliances tend to be less efficient and use more electricity, gas, and water than newer ones. Heating and cooling loads are directly affected by the weather. As a result, it helps to clarify how energy demand is considered.

2.2 Peak Energy Demand and the Residential Sector

Providing Demand Side Response in the residential sector and balancing supply and demand will require a combination of using aggregate load profiles and patterns of activity to inform interventions on load shifting. Using social practice approaches, it can be identified what is routine and what activities in the household are flexible in terms of energy consumption. In other words, the use of electricity is typically more predictable and repetitive during a particular time of day [31]. Flexible consumption and consumption timing are strongly influenced by household economics. Individuals' behaviour can be described as flexible decision makers that require little or no investment in the short or long term [32]. As a result of consumers' unwillingness to swap out one appliance use for another, the supposedly flexible component of consumption is in fact limited in its utility as an indicator of energy elasticity [33]. The term inflexibility is commonly used when describing inelasticity in energy economics literature [34]. Residential electricity demand is inelastic due to two main factors. Due to the flat tariffs that have been imposed on the vast majority of residential consumers over the past decade, information about true electricity costs are unavailable. It is typical to expect that a 100% price increase will result in a 20% change in demand [35]. As a result of the prevalence of flat tariffs in the retail market, customers are particularly unresponsive. As a result of the low supply of energy during a time of simultaneous demand, network operators are therefore compelled to produce additional energy that is expensive and often highly carbon-intensive. The second problem is that electricity prices are usually too low in developed countries for any significant change in behaviour to take place [36].

According to UK usage data, people's activities vary dramatically depending on their gender and whether or not they have children with them [31]. There are differences between men and women in housework, paid work, and media use, for example. For instance, early evening consumption is predominantly initiated by women who, broadly

over the population, are more likely often finish paid work sooner, pick up children from school, take them home, do activities such as entertainment, housework and cooking. Over the population men generally return home later and then contribute heavily to the evening peak. This type of household behaviour is unlikely change as a result of requests to ‘load shift’ during the evening peaks. Meanwhile, a significant percentage of people now work from home. Approximately 14 percent of UK workers work from home regularly. The number of people working from home has reached its highest level since records were kept by the Office for National Statistics: 4.2 million in 2014 compared to 1.3 million in 1998 [37]. In addition, these statistics indicate that home workers are generally employed in high-skilled occupations and statistically earn higher salaries. People who work from home are generally less restricted from carrying out power consuming activities at particular times of the day. In these scenario, it is clear which households will avoid load shifting, but which households could benefit more from DSR: electric storage, TCLs, smart appliances, and electric vehicles are all potential opportunities. The challenges presented by morning peaks are similar to those presented by evening peaks, though the latter includes a greater variety of activities. It is therefore more likely to see people doing the same thing in the morning than in the evening. There are some examples of DSR trialled in the UK such as "Sunshine Tariff" and "TIDE". The “Sunshine Tariff” in Wadebridge, Cornwall looked to understand how different customers engage with a cheaper daytime tariff and how active they become in changing their consumption patterns in response to this price signal. The project aimed to resolve network capacity issues in the local area by incentivising customers to use electricity between 10am and 4pm in the summer months, which is often when solar generation is at its greatest. Participants on the “Sunshine Tariff” on average shifted 10 percent of their demand into the 10:00-16:00 period. Moreover, the average household shifted a total of just under 150 kWh over the 10:00-16:00 period from April to September. In order to offset the generation from a 250 kW solar farm, approximately 650 Sunshine Tariff customers would be required. This would be approximately 20% of the homes in Wadebridge [125]. TIDE uses 100% renewable electricity and it helps consumers access low overnight prices and take back control of energy bill [126].

2.3 History of DSR Using Cooling Appliances

The concept of using domestic cooling appliances to stabilize the grid frequency was introduced in the 1980s [38], although more recent investigations have set out to model refrigerator populations with authors developing models for largescale aggregated networks of TCLs and the impact of cooling appliances on the grid frequency. Studies on load shifting by adaptive control of air conditioning of buildings and refrigerated warehouses have been considered for some time [39], [40]. DSR with refrigerators as primary assets have been considered for several smart grid applications [41]-[45]. Specifically, [43] recommend the use of refrigerators for flexible power balancing, whilst the authors of [44] propose the inclusion of domestic appliances in congestion management and recommend refrigerators and freezers as suitable for load shifting.

The most widely used control structures used in industry for numerous applications is based on traditional Proportional+Integral_Derivative (PID) implementations. The main reason for its wide usage is that it is easy to understand and realise eg. it is used from simple temperature control applications to complex systems [46], including for refrigerators. For instance, [46] used a classical PID control scheme to adjust the fan speed of a refrigerator in order to respond to sudden plant outages and maintaining the power system's frequency for a longer period compared to that of a relay controller. There are also other, more complex control techniques that have been amalgamated PID to provide improved performance, such as fuzzy techniques and neural networks, and others that have been developed for adaptive and self-tuning of the PID parameters [47] in response to system dynamic changes eg. [48]. The authors of [49] present a study of a PI-fuzzy controller for temperature and humidity. The outputs of their controller are changed as a result of the power consumption of the refrigerator and evaporator. In [50], a controller for temperature and humidity of frost-free refrigerators was designed based on fuzzy logic and controls the speed of the compressor to control temperature and relative humidity, and can change the speed of the evaporator fan for flow control.

The authors of [51]-[54] proposed a decentralized stochastic controller for the aggregated control of refrigerators to respond to mains frequency fluctuations. The relative merits of both centralized and decentralized control approaches are summarized in Table 2.1 [55]. Moreover, the use of food retailing refrigeration systems for a large supermarket chain to contribute to Firm Frequency Response (FFR) Demand Side Response (DSR) is presented in [56]. The authors show the beneficial impact of responding to a DSR event on the temperature profiles of the refrigerators and the active power consumed by the compressors. In particular, it is shown that using refrigerators to respond to DSR events can actually provide greater overall efficiency since the refrigerators operate in more efficient regions of their operating envelope. In [57], large-scale control of domestic refrigerators is used to reduce peak power demand and reduce losses in a power distribution system. The proposed strategy considered the refrigerator's thermal characteristics and incorporates models of door opening and food insertion. In [58], three experimental fuzzy logic control systems for a single domestic refrigerator are used to investigate its thermal and energy characteristics, whilst taking into consideration the frequency and duration of door opening. The fuzzy system controls the speed of the compressor in order to reduce energy consumption while keeping the temperature as close as possible to desired temperature boundaries.

Table 2.1 Comparison of centralized and decentralized control techniques

Centralized Control Approach	Decentralized Control Approach
Pros 1. High degree of controllability practiced over lower tiers. 2. High level of system reliability and security.	Pros 1. Do not require safe and reliable communication network. 2. Reduced amount of computation for every controller.
Cons 1. The High cost associated with maintaining secure communication network. 2. Huge processing burden of a centralized controller. 3. Feasible only with a limited number of control loads.	Cons 1. Local measurement of frequency signal with accuracy is difficult. 2. Frequency measurement units are expensive to be installed in every house.

Recent trends in the scheduling and control of TCLs show that MPC is implemented for use in various domestic system. It provides a control approach that determines optimal actuation inputs based on a model of known system dynamics, with 'forward looking' predictions of behaviour and the ability to inherently incorporate constraints and

accommodate exogenous disturbances. A further advantage of MPC is given by its ease of reconfiguration and adaptability to changes in the controlled system. With the development of IoT and Cloud computing MPC is increasingly applied to building management and energy systems [59]. In [60] the use of MPC has showed improved heating and energy savings in an old residential building, whilst [59] reports on the efficacy of various existing MPC algorithms for heating ventilation and air conditioning systems. In [61] a model predictive controller is developed for a domestic freezer, whilst the work in [62] develops a grey-box model for a domestic freezer and applies MPC to control its power consumption as a Demand Side Management (DSM) application. Finally, a novel non-parametric adaptive MPC scheme for domestic refrigerators is proposed in [63] which reacts to hourly pricing DSM programs and facilitates a decrease in energy consumption during the peak periods.

2.4 Refrigerator Modelling

Domestic refrigerators are widely modelled in literature, with work usually concentrating on the three main elements of i) the cabinet e.g. [64], ii) evaporator e.g. [65] and iii) compressor e.g. [66]. A common property of these models is that they tend to be very detailed, albeit very accurate, but are not computationally effective for use with low-cost microcontrollers. The objective of developing a model for control purposes is to create a linear, discrete, parametric model of the refrigerator which is sufficiently accurate for intelligent control.

Many modeling techniques already exist to estimate the available thermal capacity of the refrigerator [67], [68]. For instance, the black box method presented in [41] requires only two measurements, namely the refrigerator power consumption and the refrigerator cool chamber temperature, to accurately predict cool chamber temperature at steady-state. This also provides a generalized modelling methodology suitable for other thermostatically-controlled loads. Such a model is suitable for experiments with a large number of refrigerators, where the number of parameters measured is limited. However,

the model provides little physical insight into the structure of the dynamic behaviour and interaction of states.

Three dynamic models of a domestic refrigerator were investigated in [69]. The simplest describes the system with a single linear inhomogeneous ordinary differential equation with constant coefficients. The model captures the inside air temperature of the refrigerator correctly, but the back of the cabinet is not included in the model. The second model improves on the first, but still has significant modeling error in the cool-down phase. Figure 2.1 shows the most exact model is the third presented contribution, containing three capacitances: 1) cabinet load heat storage capacity (C_c) with cabinet temperature (T_c). This temperature is increased by the ambient temperature (T_a). Heat transfers are bounded by the thermal resistance of the insulation (R_i) and the thermal resistance of the wall between the cabinet and the evaporator (R_{ec} 2) the back of the cabinet, with the evaporator capacitance (C_e) with temperature (T_e). 3) condenser capacity C_{cond} with voltage source (T_{cond}) and the compressor supplies refrigerant T_{comp} to the condenser through a narrow tube R_{cond}. It applies two differential equations to describe the warm-up phase and three equations when the compressor is on. It is based on the second model, but the cool-down phase model is extended with an equation for the condenser of the refrigerator.

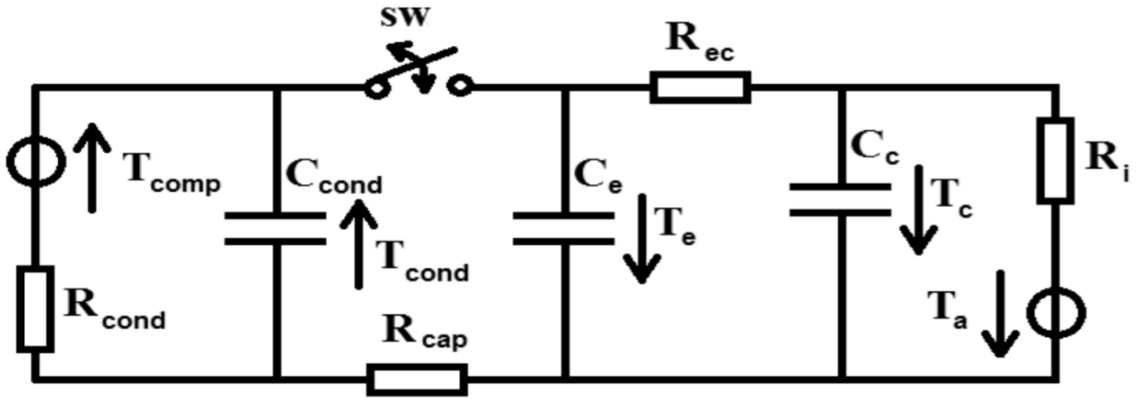


Figure 2.1: Refrigerator model with three capacitances [69]

A ‘grey box’ model is used to capture the dominant dynamics rather than all the physical dynamics of the system in order to cover different cooling technologies such as compressor and thermoelectric and different insulations and sizes that affect the thermal mass. Because this thesis wants to cover the different technologies of cooling and didn’t want to be specific about underpinning technology but be adaptive about the dominant characteristics. This model will be used in other chapters in controllers since it is the fundamental part of model predictive control and reinforcement learning.

2.5 Effect of Ambient Temperature on Refrigerators' Energy Consumption

The energy consumption of refrigerators is greatly affected by room temperature, door openings and thermal regulation, hence energy consumption is highly variable and sensitive to consumer behaviour and conditions in private homes. Daily energy consumption of a domestic appliance can be up to 2000 Wh, depending on refrigerator capacity, type (fridge, fridge/freezer, and cooler), consumer behaviour and ambient temperature [70].

Early research into refrigerator energy consumption identified that external ambient temperature was a key parameter that influences energy consumption [71], [72]. This has been confirmed in many recent studies, which conclude that the room temperature in which the appliance operates is the most important factor that impacts on energy consumption in conditions of normal use [73]-[76]. The literature has many examples of simulations of the energy consumption of households and many of these models can estimate the energy consumption of a specific refrigerator in various ambient temperatures [77]-[79].

Authors of [80] examine laboratory data for 111 appliances where energy consumption is measured at four ambient temperatures from 10 $°C$ to 40 $°C$. Field data for 235 appliances in homes is also examined. They found that room temperature has two main

effects on the energy consumption of refrigerating appliances. Firstly, the temperature difference between compartment temperatures and the room temperature dictates the heat gain into the appliance through the cabinet wall insulation and door seals. The second main effect is that a change in room temperature affects the condensing temperature. An increase in room temperature reduces overall refrigerating system coefficient of performance by increasing the difference between the evaporating and condensing temperatures.

2.6 Monitor and Control of Appliances Using IoT

A key factor for improving the management of energy consumption in the residential sector is by the remote real-time supervision and control of domestic appliances. Recently published research shows the potential of accessing home appliances remotely and implementing intelligent smart home systems based on web-based and smartphone applications [81]-[88]. The authors of [82] propose the implementation of IoT connections for home appliances through Google Assistant (voice commands) as well as World Wide Web services to switch on/off devices such as fans and lighting etc. without requiring any physical interaction, for the elderly or those with disabilities for instance, whilst [83] proposes an IoT based home automation system using an Arduino microcontroller for indoor and outdoor remote control of appliances. It develops an Android-based application to provide on/off control of six home electrical appliances namely lamp, AC, fan, refrigerator, TV and washing machine. Furthermore [84] uses a STM32F407 embedded development board for remote environmental monitoring. It shows how the integration of the STM32F407, Reduced Media Independent Interface (RMII), Flexible Static Memory Controller (FSMC) interface and web development can provide a real-time remote monitor function. The authors of [85] develop an Android-based application for a smart home automation system using ThingSpeak for data acquisition and visualisation, whilst [87] proposes the inclusion of Big Data in IoT with a unique IP address in a mesh wireless network of devices and recommend users to

remotely monitor and control devices and generate on-line bills via a mobile app. This platform can accommodate up to 4000 users. In [88] an ESP8266 and a MCU STM32F103 microcontroller are used to realise a home appliance control system with a mobile terminal for remote control, whilst in [89] a low-cost smart switch system based on an Arduino UNO interface to facilitate remote controllability and cloud analytics. This system also helps in not overloading the power grid at the peak usage time and take advantage of lower tariffs if provided by the electricity board. Finally, home automation using Message Queuing Telemetry Transport (MQTT) and a Raspberry Pi is proposed in [90] to enable measurements of temperature and humidity. Such systems enable users to benefit from the remote monitoring and control of devices at isolated sites. However, more substantial benefits can be obtained from the coordinated control of widely distributed home appliances for aggregated load management and load shed for DSR purposes. In Chapter 4, a IP-based synchronized wireless network is proposed and implemented for monitoring and jointly scheduling the widely distributed domestic refrigerators in DSR events.

2.7 Reinforcement Learning Approaches for the Scheduled Operation of the Heating and Cooling Systems

In the field of computer science, Machine Learning (ML) is a relatively new emerging branch of artificial intelligence (at the time of writing), and amalgamates concepts of physiology, data mining, statistics and probability, and control theory. Recently, several new fields of research have developed and emerged that are related to ML due to advancements in computational hardware. The following provide some merits of machine learning and why it might be considered for load scheduling [91]:

- New and unknown environments: during the process of designing or controlling a system, some of the system's features remain unknown or may change. ML can

learn and adapt to these overtime. Moreover, much greater amounts of data can be accommodated that can be handled by humans.

- In some cases, problems cannot be accurately described due to high complexity; in such cases, they could be specified as generic input/output problems and ML used to design and control complex systems through learning.
- The environment may change over time; The ML agent can adapt to these changes and modifies agents of decision-making processes to improve performance over the time.

Types of ML can be classified in many ways, but in general, there are three major categories [92]:

1. Supervised Learning: There is a set of training data that specifies the value of each input, output, or function. This learning agent aims to obtain a hypothesis that describes the function or relationship between input and output. Classification and regression are examples of supervised learning.
2. Unsupervised Learning: An example of training data exists in which only the inputs are known, and the correct outputs remain unknown. For instance, Clustering can be described as unsupervised learning.
3. Reinforcement Learning (RL): The agent tries to interact with the environment through trial and error and learns to choose the optimal action to achieve the goal. In this thesis, the RL is proposed for the scheduled operation of the domestic refrigerators.

The Table 2.2 provides examples for each of the proposed ML categories used in different fields [93].

Table 2.2 Examples of the three different ML categories

Supervised Learning	Unsupervised Learning	Reinforcement Learning
<u>Regression</u> 1. Ad popularity prediction. 2. Weather forecasting. 3. Market forecasting. 4. Population growth prediction.	<u>Clustering</u> 1. Recommender systems. 2. Targeted marketing. 3. Customer segmentation	1.Real-time decisions. 2. Robot navigation. 3. Learning tasks. 4. Skill acquisition. 5. Games.
<u>Classification</u> 1. Image classification. 2. Customer retention. 3. Identify fraud detection. 4. Diagnostics.	<u>Dimensionality Reduction</u> 1. Big data visualisation. 2. Meaningful compression. 3. Structure discovery. 4. Feature elicitation.	

Artificial intelligence has been used in many fields recently, including for the control of household appliances, especially TCLs. In [94], a Reinforcement Learning (RL) agent is proposed to control a HVAC system by optimising both occupant comfort and energy costs. Results show that the 'learning thermostat' can save up to 10% in energy costs when compared to a traditional thermostat. A model free RL based on Q-functions is presented in [95] to construct a day-ahead schedule for a heat-pump thermostat, whilst in [96] model-free Q-learning is used to control HVAC and window systems. Experimental results in hot-and-humid Miami and warm-and-mild Los Angeles shows

that the proposed strategy led to 13% and 23% lower HVAC system energy consumption, respectively, compared to heuristic control. In [97], two different RL algorithms, including Deep Q Network (DQN) and Deep Deterministic Policy Gradient (DDPG), are used for load shifting in a simulated cooling network which leads to weekly cost savings of 14 % compared to direct load coverage. A DQN based centralised and decentralized controller is studied in [98] to improve setpoint tracking and reduce energy costs in smart buildings. In [99], the Proximal Policy Optimisation (PPO) RL algorithm is employed for the efficient scheduling and control of the HVAC system in a commercial building, whilst taking into consideration the achievement of demand response objectives. Simulation results show that a maximum weekly energy reduction of up to 22% can be achieved compared to use of a traditional controller. The authors in [100] show up to 30% and 21% cost reduction for the simulation and experimental results, respectively, when using DQN based HVAC control on different house models with varying user comforts. HVAC and refrigerators have different models due to different thermal mass and insulation but in terms of control objectives, they have similarities such as thermal comfort tracking. DQN can also be tested on refrigerators, given that there are many similarities between HVAC control at home and domestic refrigerator control, such as user thermal comfort. Therefore, the DQN reward function (thermal comfort tracking or reference tracking) structure provided for HVAC can be used and tested in a similar manner for refrigerators. [101], [102] introduces a multi-advisor DQN approach with user-defined importance weighting objectives for tracking temperature set-points and reducing power usage in a domestic heating system. Finally, a novel DDPG method for a multi-zone residential HVAC system is proposed in [103] which generates an optimal heating profile with a 15% and 79% reduction in energy consumption comfort violation, respectively, compared to the DQN controller. It is due to these successes that this promising suite of techniques is researched for investigation of scheduling TCLs in this study.

2.8 Conclusion

Literature review indicates that continuous programming is used for scheduling HVAC systems, however, binary programming is required for scheduling TCLs. As a result, a binary quadratic programming algorithm is developed utilizing Model Predictive Control (MPC) and Reinforcement Learning to accomplish the scheduling methodology through the Internet of Things. Of notable importance is the formulation of life-time indices as part of the MPC and RL strategies to accommodate a maximum number of compressor starts per hour as part of the scheduling to avoid too frequent on/off switching events while the literature does not address life-time indices. Moreover, when power consumption is beyond what can be reasonably supplied to support the cooling of all the refrigerators in the network, the proposed MPC formulation allows for the prioritization of power distribution to 'preferred' units. The thesis also proposes and implements an IP-based synchronized wireless mesh network for monitoring and jointly scheduling widely distributed domestic refrigerators, whereas in the literature only wireless networks for single appliances are considered.

As can be seen from figure 2.2, a fundamental part of RL and MPC control schemes is the identification of the underlying system dynamics. The application to refrigerators is complicated because of significant dynamic changes due to ambient conditions, opening and closing the door and the changes in product mass and constitution within the refrigerator (the product). Real-time model updates are therefore essential for high performance control i.e. adaptive mechanisms are necessary. Finally, IoT platforms with local and remote access are designed to undertake experimental trials and implement the proposed MPC and RL on domestic refrigerators.

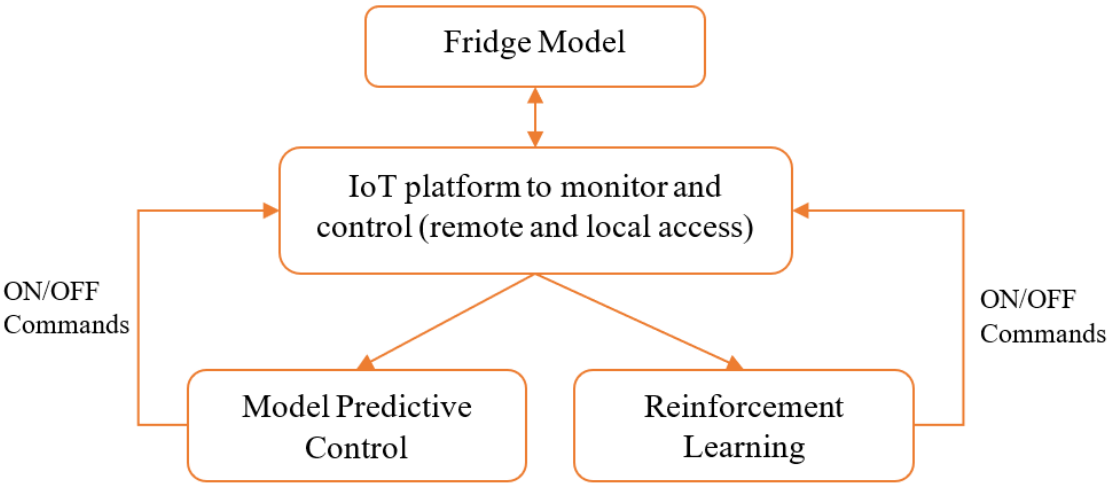


Figure 2.2: The thesis research overview

3 IDENTIFICATION OF REFRIGERATOR DYNAMICS AND MODEL PREDICTIVE CONTROL WITH BINARY QUADRATIC PROGRAMMING

3.1 Introduction

Here, real-time system identification techniques are considered to identify refrigerator dynamics. The first is based on a first-order ARX (Autoregressive with Exogenous Variables) model, and then a real-time recursive based system identification strategy to monitor and estimate the internal temperature of individual domestic refrigerators based on their internal thermal mass (product), is proposed. The latter is ultimately used to adaptively modify the hysteresis temperature bounds of individual refrigerators, and in so doing, show that significant overall energy savings can be obtained. An important

feature is that the proposed model has an ability to accommodate uncertain events e.g., ambient conditions, opening and closing the door and changes in product mass. Initially, the experiments are conducted using the simple ARX model, followed by ARX with RLS without a temperature consideration, and then ARX with RLS considering a temperature consideration. The objective of these experiments is to examine how ambient temperature and updating *a*, *b*, and *c* parameters affect refrigerator temperature prediction. Several different thermal masses and events are also used in each experiment to test model accuracy, such as an empty refrigerator, six litres of water, and opening and closing the door.

Therefore, the research presented in this chapter proposes a time-varying priority-based on-off scheduling algorithm that can effectively schedule networks of widely distributed refrigerators using an IoT controlled platform. Specifically, Binary Quadratic Programming is used to formulate a Model Predictive Control problem. Importantly, the number of compressor 'starts per hour' for each refrigerator is also bounded as an inherent design feature of the algorithm so as not to operationally overstress the compressors and reduce their lifetime—typically, refrigerator compressor lifetime is based on 6 ON-OFF events per hour of operation. Experimental trials will show that such co-ordinated operation of refrigerators can reduce energy consumption by ~30% whilst also providing peak power shaving, thereby affording benefits to both individual consumers as well as electrical network suppliers. Moreover, the chapter considers the impact of various temperature hysteresis bands on the projected annual energy consumption of a typical domestic refrigerator when operating empty of product and when including internal product. Benefits afforded by the proposed technique are investigated through experimental trials on VonShef 13/291 (50W, compressor based technology), iGENIX IG 3920 (55W, compressor based technology) and Russell Hobbs RHCLRF17B (50W, thermoelectric technology) domestic refrigerators. The NodeMCU is used as a microcontroller because it has an integrated Wi-Fi module to connect smart plug and send/receive data from the cloud data centre (ThingSpeak). The TP-LINK

HS110 smart plug is used to measure power usage and receive ON/OFF commands using the NodeMCU Wi-Fi module, and it is easy to communicate with this plug using Arduino and Python codes. In order to measure fridge internal temperatures, the DS18B20 waterproof version sensor is used. This sensor is used within the fridge so it should be waterproof. Data acquisition and monitoring are accomplished using ThingSpeak, since this platform is connected to MATLAB and displays the data graphically in real-time.

3.2 System Identification

System identification is a methodology for building mathematical models of dynamic systems by utilizing measurements of the system's input and output measurements. Two common classifications of methods are available [104]:

- Non-parametric Methods: The results are curves, tables, etc. These methods are simple to apply; they give basic information about time delays and time constants of the system, for instance.
- Parametric Methods: The results are values of the parameters in the model such as ARX, AR-MAX (Autoregressive–moving-average), OE (Output Error) and BJ (Box-Jenkins). These methods may provide better accuracy but are often computationally more demanding.

3.3 ARX structure

This model uses a generalized notion of transfer function to express the relationship between the input, *u(t)*, and the output, *y(t)* using the equation [104]:

$$y(t) + a_1 y(t-1) + \cdots + a_n y(t-n) = b_1 u(t-1) + \cdots + b_m u(t-m) \qquad (3.1)$$

$y(t-1)$ ... $y(t-n)$— Previous outputs on which the current output depends ($y(t)$)

$u(t-1)$ ... $u(t-m)$— Previous inputs

So, in ARX model the estimated value ($y(t)$) depends on previous measured outputs ($y(t-1)$ ... $y(t-n)$) and inputs $u(t-1)$ ... $u(t-m)$.

ARX (equation 3.1) time series models are a linear representation of a dynamic system in discrete time (it is not accurate at the first step because it is linear and have constant *a* and *b* parameters as can be seen in the measurement trials Figure 3.3). Putting a model into ARX form is the basis for many methods in process dynamics and control analysis — See figure 3.3

The system is represented in discrete time, primarily since observed data are always collected by sampling. It is thus more straightforward to relate observed data to discrete time models. (3.1) can be reduced to a more compact notation:

$$y(t) = \varphi^T(t)\theta \tag{3.2}$$

where:

$$\theta(t) = [a_1, \dots, a_n, b_1, \dots, b_n]^T \tag{3.3}$$

$$\varphi(t) = [-y(t-1), \dots, -y(t-n), u(t-1), \dots u(t-m)]^T \tag{3.4}$$

To emphasize that the estimated value of *y(t)* from past data depends on the parameters in θ, the estimated value is denoted $\hat{y}(t|\theta)$ where:

$$\hat{y}(t|\theta) = \varphi^T(t)\theta \tag{3.5}$$

Based on the equations, θ in ARX model always has a constant value based on a single set of data, therefore, in presence of disturbance and uncertainties, it is not possible to always provide an accurate model.

3.4 Real-time Identification of Refrigerator Dynamics

Since the dynamics of refrigerators change with ambient conditions, opening and closing of the door and changes in product thermal mass, the provision of an adaptive dynamic model is essential. Here, the author proposes the use of an online recursive identification algorithm that captures the dominant dynamics and disturbance patterns of the refrigerator. It is based on work originally proposed in [105]. The model (3.6) is widely used in refrigerator control literature, e.g. [51-53, 61, 106-108]:

$$T(t) = e^{\frac{-\tau \times A}{m_c}} \times T(t-1) + (1 - e^{\frac{-ST \times A}{m_c}}) \times (T_{amb}(t-1) - \frac{\eta \times P(t-1)s(t-1)}{A}) \qquad (3.6)$$

where *T(t)* is the estimated internal temperature of the refrigerator at time *t.* Parameter *P(t)* denotes the electrical power required during the last time interval, and is dependent on whether the compressor is turned ON or OFF*;* $s(t) \in [0,1]$ is the state of device at time *t* (a binary ON (1) /OFF (0)); *A* is the overall thermal insulation *(W/ºC)*; η is the coefficient of performance; m_c is the thermal mass (*J/ºC*), and *ST* is the sample time between *t-1* and *t*. T_{amb} *(t)* describes the ambient temperature at time *t*. To simplify notation, the following model (3.7) is used, where *a* represents the thermal characteristics of refrigerator inner temperature, *b* the impact of the energy transfer from the compressor due to the operation of the system, and *c* the impact of ambient temperature. The thermal insulation (A) can be calculated, from the physical characteristics of the fridge. Knowing the thermal conductivity coefficient (k), the area of the fridge S and approximating the thickness of the fridge wall x, the thermal insulation parameter A can be computed. It is assumed to be a constant parameter since it strictly depends on the physical construction properties of the fridge: $A = \frac{k \times S}{x}$

The coefficient of performance represents the ratio between the cabinet heat loss Q_{tot} and the energy consumption measured $W_{consumed}$. Assuming that 60% of the heat loss is

due to convection [61], the total heat loss Q_{tot} (cabinet heat loss) is computed: $Q_{tot}(t) = 0.6 \times Q_{conv}(t)$

The convective heat loss Q_{conv} is calculated as a function of the thermal insulation A, the difference between the ambient temperature (T_{amb}), average temperature of the fridge compartment (T_{avg}) and the time duration of the off cycle Δt_{off}: $Q_{conv}(t) = A \times (T_{amb}(t) - T_{avg}) \times \Delta t_{off}$

The overall expected shape of this model is provided in figure 3.1 [61]:

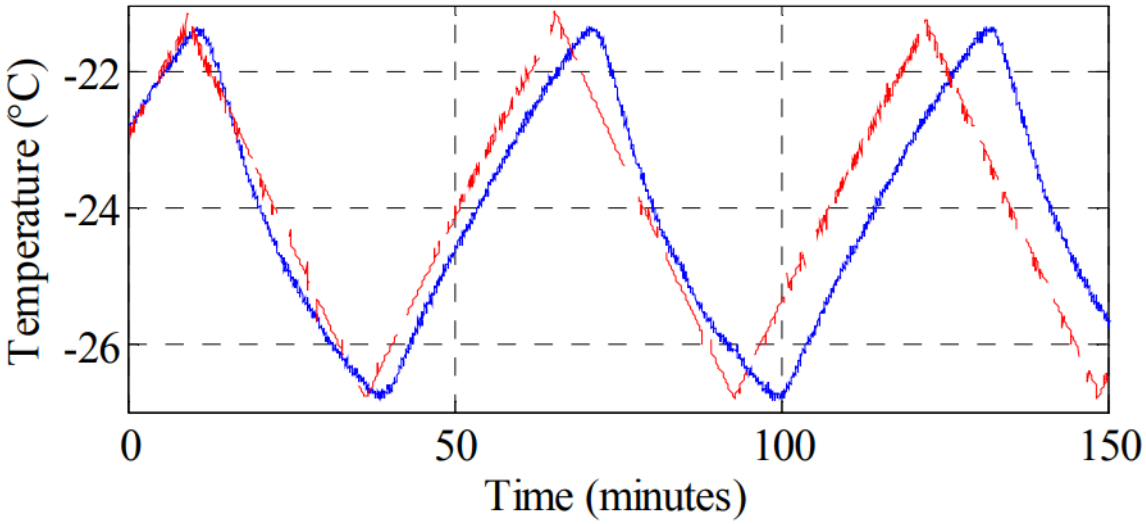


Figure 3.1: The measured (continuous line) and estimated (interrupted line) temperature of the freezer [61].

Figure 3.1 [61] shows a model in which at the beginning the predicted and measured distances are very close, but as time passes, because the parameters stay constant, the predicted and measured distances get further apart. The constant parameters are used here, and section 3.5.2 shows the results if the parameters are allowed to change.

Based on equation 3.6, parameters are in the form of coefficients for refrigerator temperature ($T(t-1)$), power consumption ($P(t-1)$), and ambient temperature

($T_{amb}(t-1)$) from the previous measured time step. Therefore, instead of using equations and the physical characteristics of the fridge to calculate these coefficients, they are identified by the proposed ARX method and the ARX with RLS method. Compared to a model with fixed coefficients, the ARX with the RLS method offers a more accurate estimate since its coefficients are updated every time a sample is taken.

Refrigerator internal temperature ($T(t-1)$), power consumption ($P(t-1)$), and ambient temperature ($T_{amb}(t-1)$) from the previous measured time step. *a* represents the thermal characteristics of refrigerator inner temperature, *b* the impact of the energy transfer from the compressor due to the operation of the system, and *c* the impact of ambient temperature. *T(t)* is the estimated internal temperature of the refrigerator at time *t* but $T_{amb}(t)$ describes the ambient temperature at time *t*: $a = e^{\frac{-\tau \times A}{m_c}}$, $b = -(\frac{\eta}{A}) \times (1 - e^{\frac{-ST \times A}{m_c}})$, $c = 1 - e^{\frac{-ST \times A}{m_c}}$

$$T(t) = a \times T(t-1) + b \times P(t-1)s(t-1) + c \times T_{amb}(t-1) \tag{3.7}$$

This can be reduced to the more compact notation:

$$T(t) = \varphi^T(t)\theta \tag{3.8}$$

where:

$$\theta(t) = [a, b, c]^T \tag{3.9}$$

$$\varphi^T(t) = [T(t-1), P(t-1)s(t-1), T_{amb}(t-1)] \tag{3.10}$$

Subsequently, $\theta(t)$ is obtained using Recursive Least Squares (RLS) to update the $\theta(t)$ based on the dominant dynamics and disturbance patterns of the refrigerator in each sample time [104]:

$$\theta(t) = \theta(t-1) + K(t)[T(t) - \varphi^T(t)\theta(t-1)] \tag{3.11}$$

$$K(t) = \frac{q(t-1)\varphi(t)}{1+ \varphi^T(t)q(t-1)\varphi(t)} \tag{3.12}$$

$$q(t) = [q(t-1) - \frac{q(t-1)\varphi(t)\varphi^T(t)q(t-1)}{1+ \varphi^T(t)q(t-1)\varphi(t)}] \tag{3.13}$$

The expected response for equation 3.6 is provided in figure 3.1 [61]. The experimental results for ARX and ARX with RLS are illustrated in sections (3.5.1) -(3.5.3).

3.5 Experimental System Identification

To show the efficacy of the parameter estimation using ARX, first-order ARX with RLS and first-order ARX with RLS considering ambient temperature, trials are initially undertaken on a single VonShef 13/291 ($50W$) refrigerator. The experimental setup employs a NodeMCU microcontroller and an IoT smart-plug based platform to monitor the power usage of the refrigerator. ThingSpeak is used for a data acquisition and monitoring—see Figure 3.2. The refrigerator is instrumented with a DS18B20 waterproof sensor and a TMP102 module to monitor, respectively, the internal refrigeration temperature T and ambient temperature T_{amb}. A fixed sampling period of 20 seconds is used.

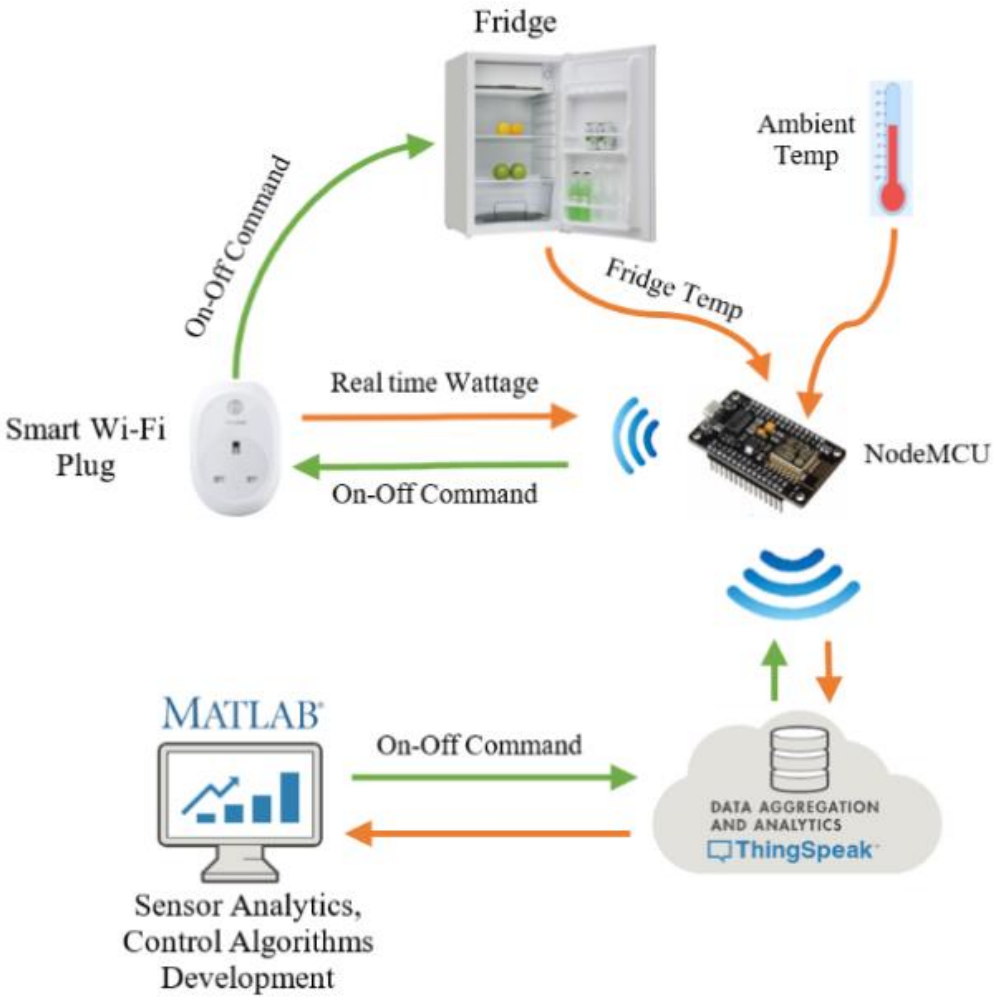


Figure 3.2: Experimental setup

3.5.1 First-order ARX

The first-order ARX model is used to identify the fridge model using experimental trials with 6-litres of water in the fridge:

$$T(t) = a \times T(t-1) + b \times P(t-1)s(t-1) \tag{3.14}$$

Parameters a and b are calculated using equations (3.2) -(3.5) and the results are as follows:

$a = 1.0022, b = -0.002$ (3.15)

The resultant dynamics are presented in Figure 3.3, with constant ***a*** and ***b*** parameters (thereby providing a linear model). There are notable discrepancies between the measured and estimated temperatures—this first-order model therefore cannot accurately estimate the internal temperature during uncertain events such as door opening.

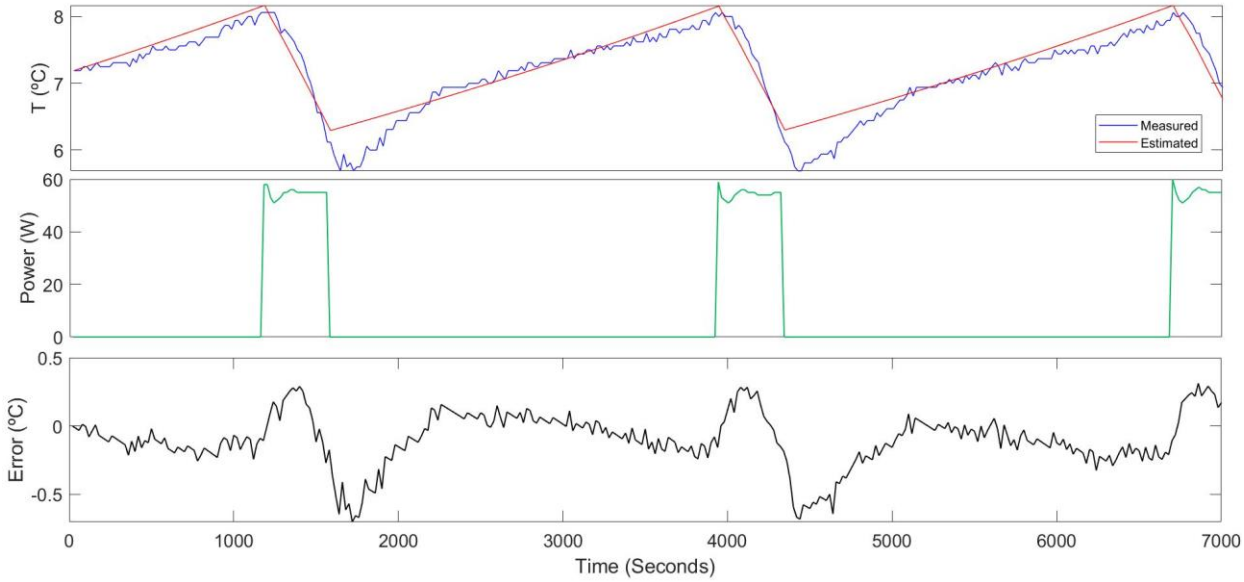


Figure 3.3: Internal temperature and parameter estimation using first-order ARX model

Since the parameters are constant, and according to Equation 3.14, the estimated internal temperature also depends on the power consumption. Figure 3.3 illustrates when the refrigerator is turned on, the temperature drops immediately without delay, while in the measured temperature there is a delay between power usage (input) and internal temperature (output) due to thermal mass (the same trend in figure 3.1 [61]).

3.5.2 ARX model with RLS

The first-order ARX model proposed in (3.14) is now used with RLS to estimate the $\boldsymbol{a}$ and $\boldsymbol{b}$ parameters at each sample time step. Experimental trials are carried out under three different conditions: 1) empty fridge 2) with additional product (6-litres of water) 3) door opening and closing events. Figures 3.4 and 3.5 show the results when the refrigerator is empty and when it contains 6-litres of water respectively. As can been seen from the results, the model is able estimate the internal temperature correctly with and without additional products in the refrigerator due to the recursive parameter optimisation in each time step.

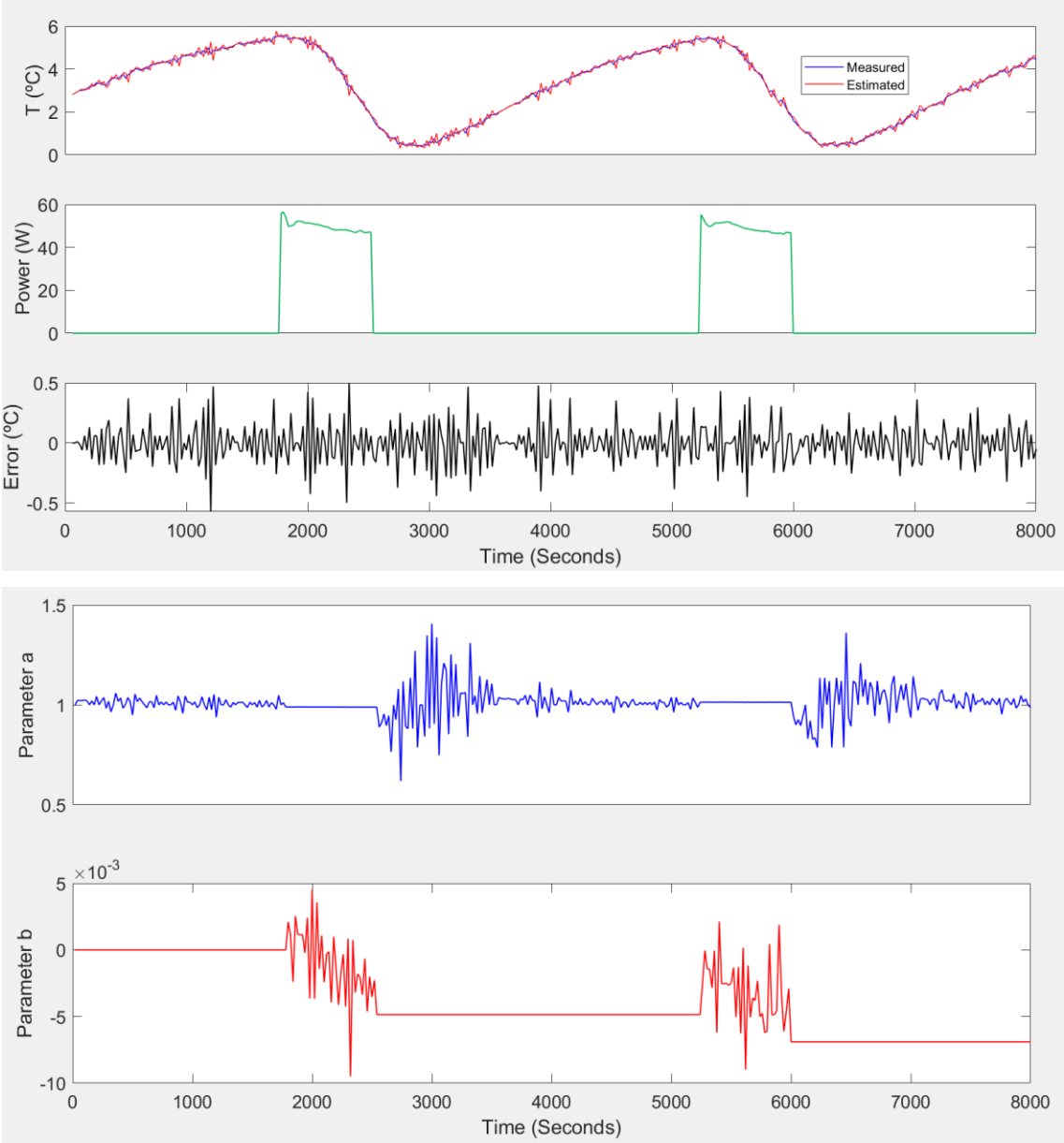


Figure 3.4: Internal temperature and parameter estimation using ARX model with RLS without product

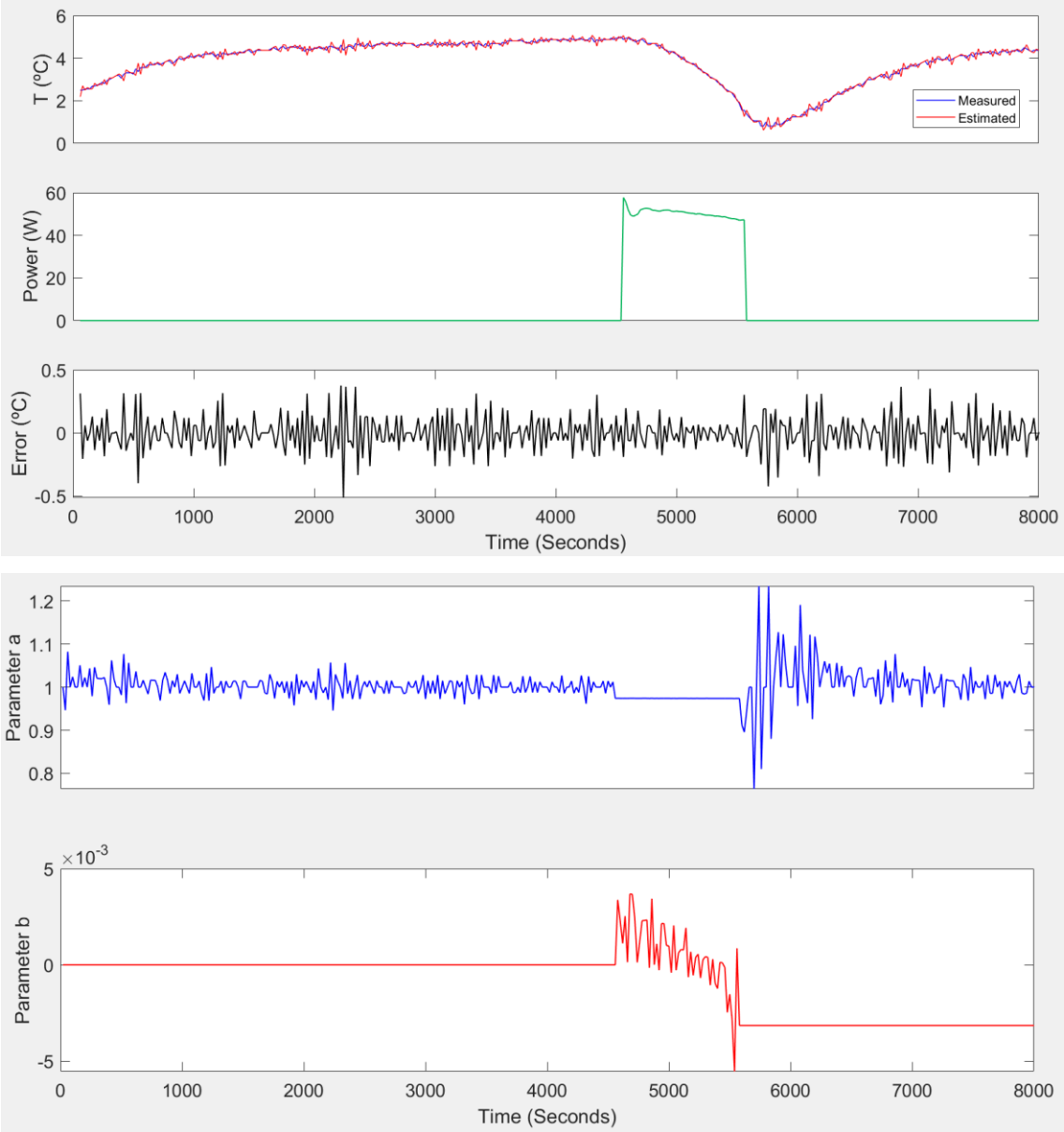


Figure 3.5: Internal temperature and parameter estimation using ARX and RLS model with additional product (6L water)

Figure 3.6 presents the results when there are 3 door opening and closing events. It can be seen that the model cannot estimate the internal temperature correctly when the door opening event occurred, because the impact of the ambient temperature is not considered in this model.

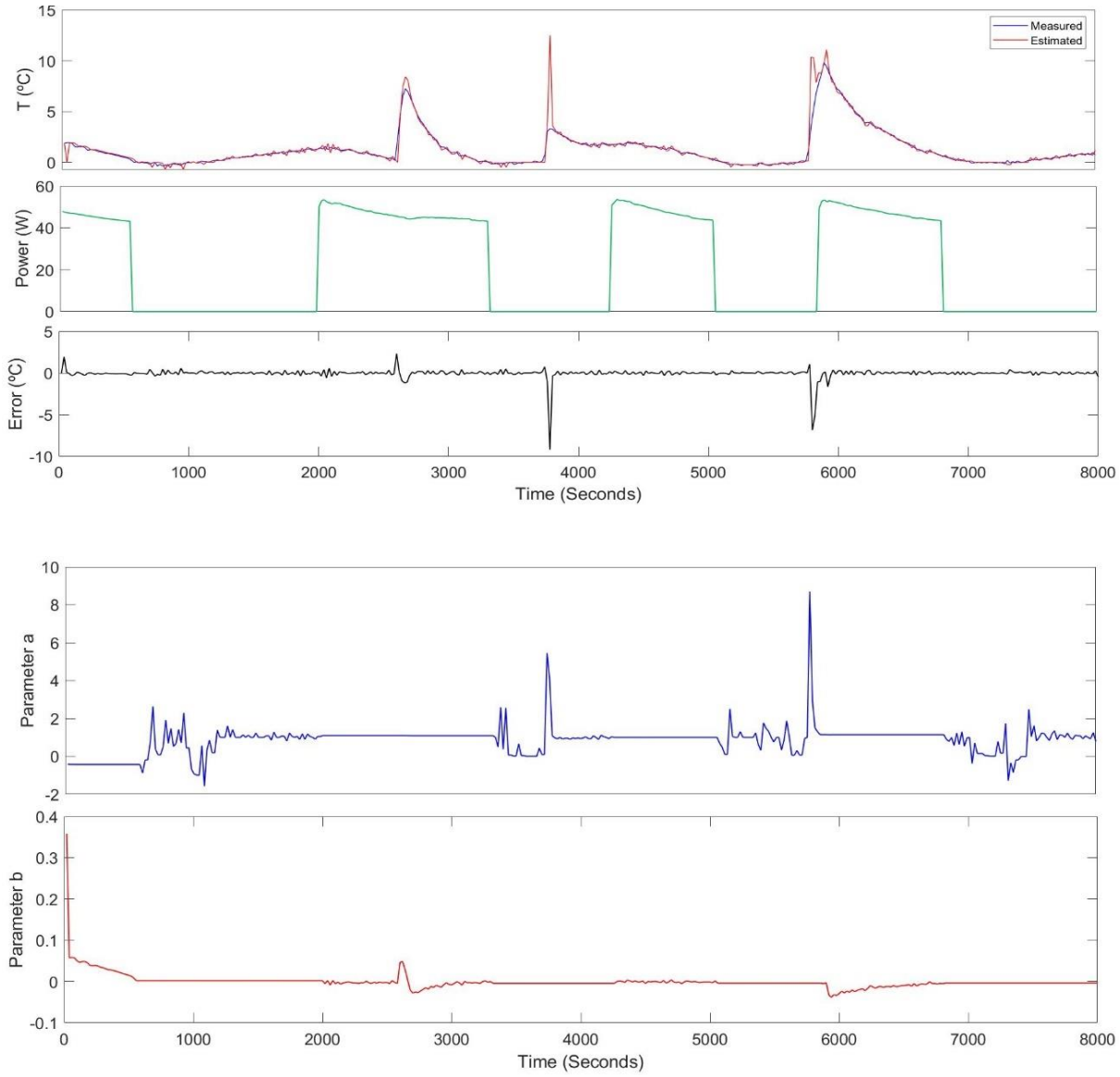


Figure 3.6: Internal temperature and parameter estimation using ARX and RLS model with 3 different door opening events

3.5.3 ARX model with RLS considering Ambient Temperature

Experimental measurements are taken to identify parameters when the refrigerator is both empty and when it contains product (6-litres of water in this instance). The results

are presented in Figures 3.7 and 3.8 for each condition, respectively. It can be seen that the estimation of temperature rapidly converges to the correct value (due to the recursive optimization of model parameters) and errors become negligible (<0.5 ^{o}C) in steady state. Notably when employing RLS in this way, $\boldsymbol{b}$ responds to observations from the power usage of the compressor. When the compressor is in the OFF state, $\boldsymbol{b}$ remains constant, and when the compressor is in the ON state, $\boldsymbol{b}$ is allowed to dynamically adapt.

Figure 3.9 presents parameter identification results when the VonShef refrigerator is subject to 3 door opening and closing events that induce transient disturbances. The first event lasts for 20 seconds (one sample time), the second for 60 seconds and the third for 120 seconds. It can be seen that parameter tracking remains robust to the induced disturbances and very good temperature tracking performance is maintained.

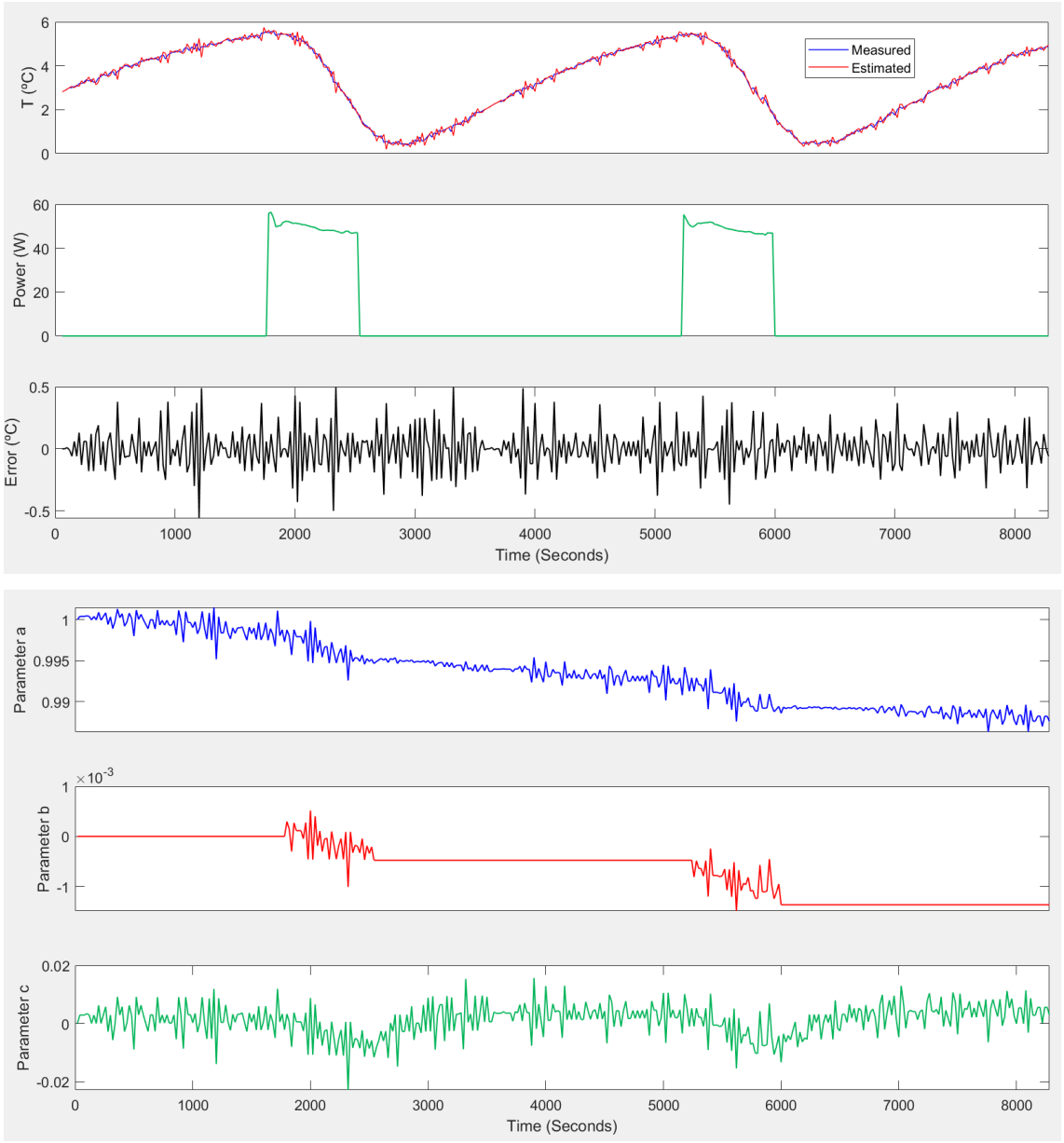


Figure 3.7: Internal temperature and parameter estimation without product (empty)

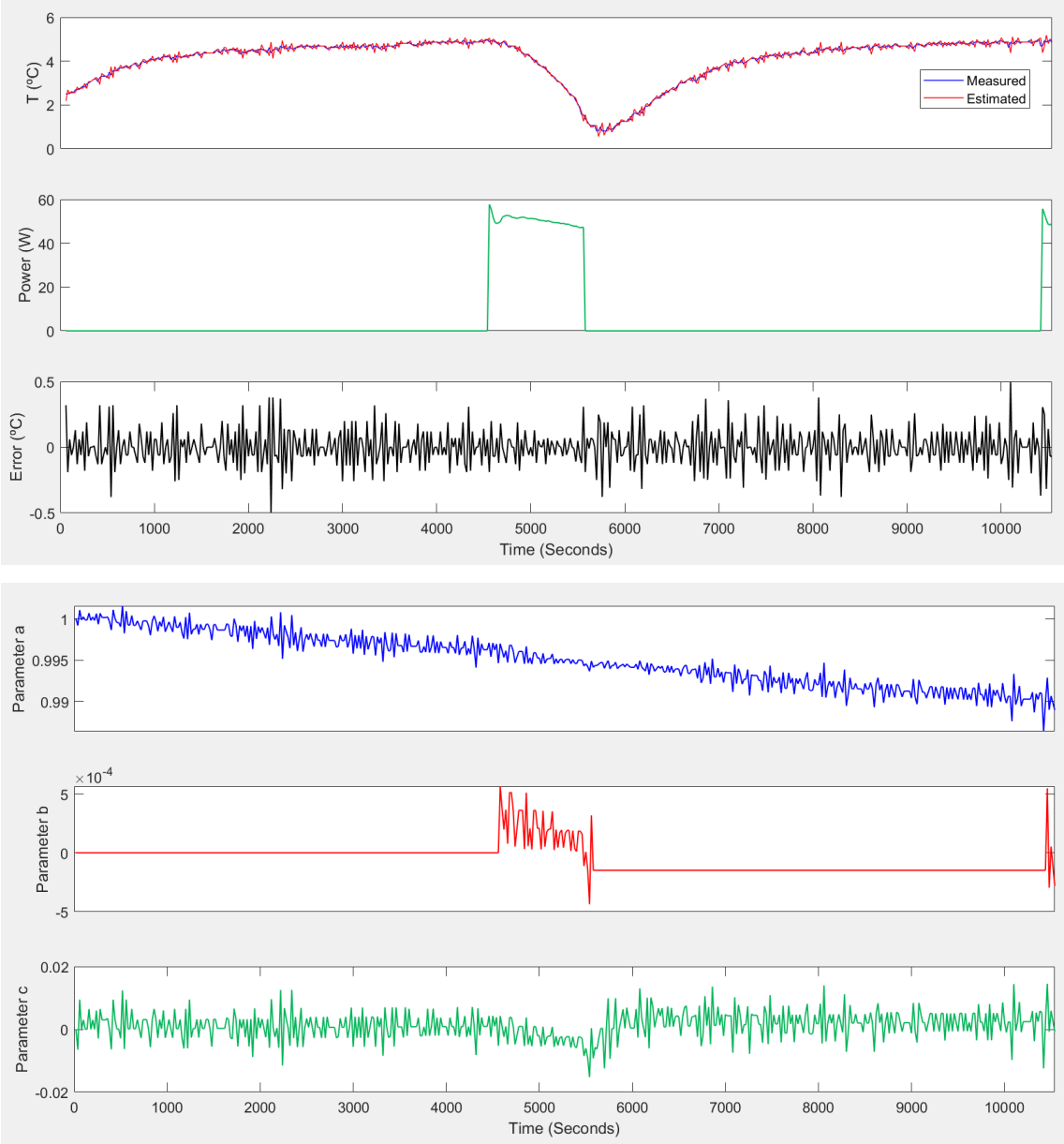


Figure 3.8: Internal refrigerator temperature and parameter estimation with additional product (6L water)

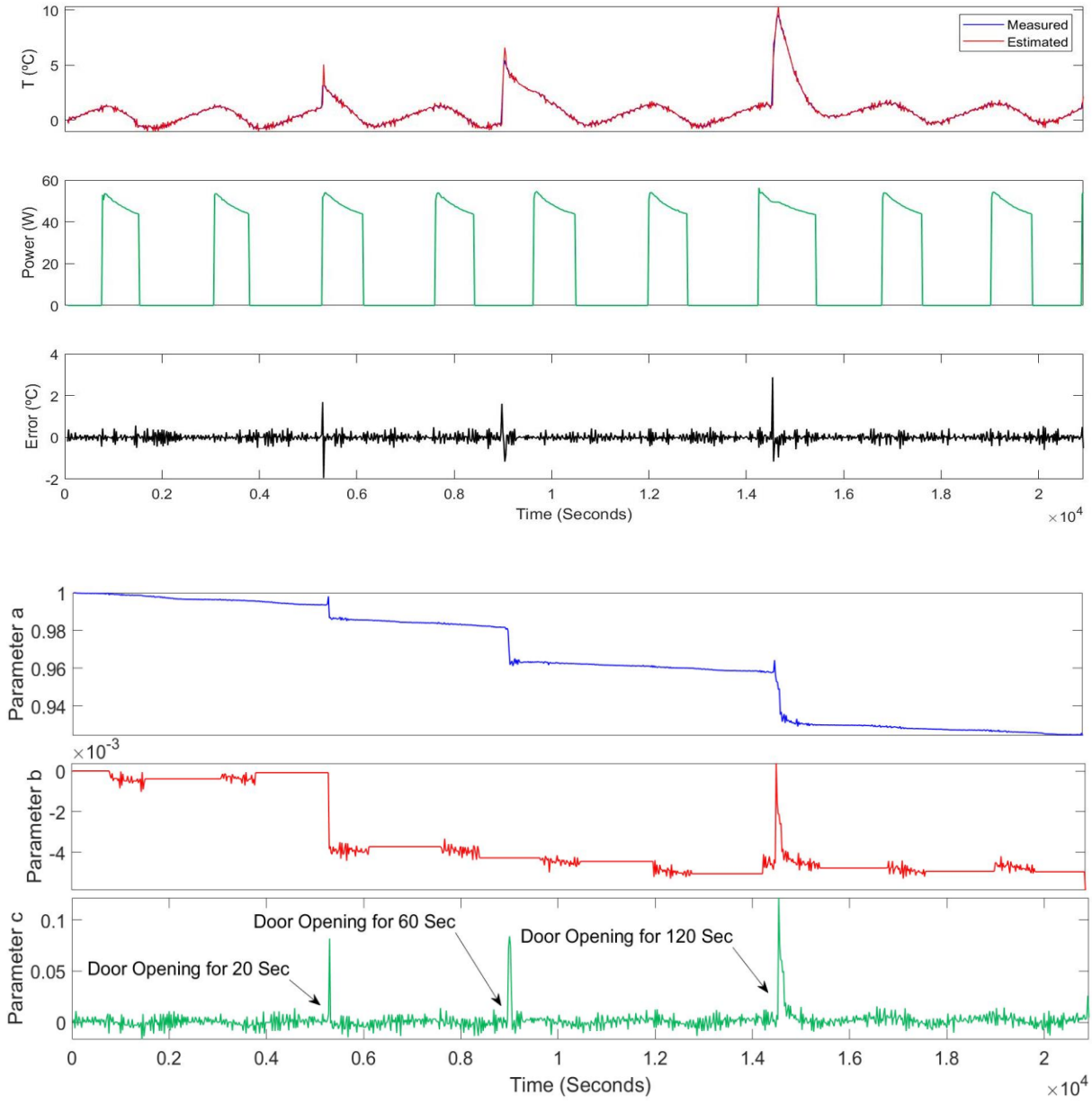


Figure 3.9: Internal temperature and parameter estimation with 3 different door opening events

Figure 3.9 also shows how parameter ***c*** responds to the door opening events by recognizing the impact of exposure to the ambient temperature i.e, the internal temperature rises, and hence so does parameter ***c***, and time span of the change is

reflected by how long the door was open—for instance, the rise in $\boldsymbol{c}$ for the 120 second door opening case is greater than of the 60 seconds door opening scenario etc.

"*a*" is linked to fridge internal temperature. In addition to previously measured data ($T(t-1)$), ($P(t-1)$) and ($T_{amb}(t-1)$)), the parameters *a*, *b*, and *c* are also updated based on previous *a*, *b*, and *c* parameters (equations 3.11 to 3.13). This results in different parameters over time.

3.6 Comparison of Refrigerator Identification Techniques

The Root Mean Square Error (RMSE) is the standard deviation of prediction errors which is always non-negative, and a value of 0 (almost never achieved in practice) would indicate a perfect fit to the data. Therefore, the RMSE is used to compare the proposed models for the refrigerator. RMSE is calculated using:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}} \tag{3.16}$$

The RMSE is obtained for each model, and the results given in Table 3.1. It can be seen that the RMSE of First-order ARX model is 0.2175 which shows a large difference between predicted and measured values (as expected), whilst the RLS has a good performance with and without additional product, but with door opening and closing events the RMSE is 0.67 which shows the lack of ambient temperature information. After considering ambient temperature, the RMSE in each case is less than 0.24 which shows a negligible difference between values predicted by the model and the values observed.

Table 3.1 RMS errors in each case

Algorithm	Case name	RMSE
First-order ARX	6-litres of water	0.2175
First-order ARX with RLS	Empty	0.1722
	6-litres of water	0.1467
	Door opening and closing events	0.67
RLS with ambient temperature	Empty	0.1704
	6-litres of water	0.1450
	Door opening and closing events	0.2383

3.7 Load Levelling by the Scheduled Operation of Multi-Refrigerator Systems

A Model Predictive Scheduling Control scheme is used to control a set of domestic refrigerators [109]. A state space model of the refrigerator network with *r* inputs and *n* outputs is given in (3.17) where the parameters are obtained from the online identification process given previously.

$$\begin{cases} x(t+1) = Ax(t) + Bu(t) \\ T(t) = Cx(t) \end{cases} \tag{3.17}$$

where $x \in \mathbb{R}^{n\times 1}$ as the state vector, $u(t) \in \mathbb{R}^{r\times 1}$ input vector ($s_i(t) \in [0,1]$ and $T_{amb}(t)$ are considered as input variables), $A \in \mathbb{R}^{n\times n}$ system matrix, $B \in \mathbb{R}^{n\times r}$ input matrix, $T(t) \in \mathbb{R}^{n\times 1}$ the estimated fridge temperature, $C \in \mathbb{R}^{n\times n}$ output matrix and where the *t* denotes discrete time quantities:

$$x(t) = \begin{bmatrix} T_1(t) \\ T_2(t) \\ \vdots \\ T_n(t) \end{bmatrix},\ u(t) = \begin{bmatrix} s_{1(t)} \\ s_{2(t)} \\ \vdots \\ s_{n(t)} \\ T_{amb1(t)} \\ T_{amb2(t)} \\ \vdots \\ T_{ambn(t)} \end{bmatrix},\ A = \begin{bmatrix} a_1 & 0 & 0 & \dots & 0 \\ 0 & a_2 & 0 & \dots & 0 \\ 0 & 0 & a_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_n \end{bmatrix} \tag{3.18}$$

$$B = \begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 & \dots & 0 \\ 0 & b_2 & c_2 & 0 & 0 & \dots & 0 \\ 0 & 0 & b_3 & c_3 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & b_n & c_n \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \tag{3.19}$$

The proposed general form of optimization model for controlling of aggregated refrigerators uses a Binary Quadratic cost function (*J*):

$$J = \sum_{j=N_1}^{j=N_2} R(j) \times [T(t+j) - T_{ref}(t+j)]^2 + \sum_{j=1}^{j=N_u} Q(j) \times [u(t+j) - u(t-1+j)]^2 \tag{3.20}$$

$$s.t. \sum_i P_i \times s_i \leq P_{max}, \quad \forall t \in \tau \tag{3.21}$$

Refrigerator *i* operational constraint, $\forall i$

$$\text{when } s_i(t-1) = 0 \; and \; s_i(t-2) = 1, \text{ then } \sum_{j=1}^{j=minoff} s_i(t+j) = 0 \tag{3.22}$$

$$\text{when } s_i(t-1) = 1 \; and \; s_i(t-2) = 0, \text{ then } \sum_{j=1}^{j=minon} s_i(t+j) = 1 \tag{3.23}$$

where N_1 and N_2 are the minimum and maximum prediction horizons and N_u is the control horizon. Weighting factors for predicted error and control increments are *R(j)* and *Q(j)*, respectively. The parameter $T_{ref}(t)$ specifies the internal temperature references for each refrigerator which should be kept within upper and lower bounds; *i* is the refrigerator identifier and τ is a set of indices in the scheduling horizon. Constraint (3.20) ensures that maximum power consumption at a given time (P_{max}) does not exceed a specified boundary value, and constraints (3.22) and (3.23) ensure the minimum off-time (*minoff*) and minimum on-time (*minon*) per cycle for each refrigerator, respectively. This is an important consideration as it allows the number of compressor starts per hour to be bounded so that the scheduling algorithm does not detrimentally overstress any of the compressors and reduce operational lifetime. The solution to the optimization problem to minimize *J* and calculate $s_i(t)$ follows the

procedure presented in [110]. In what follows, the parameters used in the MPC are $N_1=1$, $N_2=5$ and $N_u=5$. As the prediction and control horizons are set at five, the MPC provides prediction and control for the next 5 time steps in the output, however, information is received from the controller in each step using the IoT platform, so only the first output is used for the next time step. The parameters N1, N2 and Nu have not changed during all the tests.

The procedure to obtain the scheduled operation of domestic Refrigerators using MPC can be described as follows:

Algorithm_ MPC with BQP for the scheduled operation of domestic refrigerators

Input:
N_1, N_2, N_u
For each appliance *i:*
$T_{ref}(i)$
Minoff(i)
Minon(i)
Q(i)
R(i)
for t = 0:T_{finish}
    Receive the measured $T_i(t)$, $P_i(t)$ and $T_{amb}(t)$ from ThingSpeak
    Calculate the *A, B* and *C* matrices using (3.8)-(3.13) and (3.17)-(3.19)
    Receive $P_{max}(t+1)$ from ThingSpeak
    Minimize *J* considering the constraints (3.21)-(3.23) and calculate $s_i(t+1)$, using opti in MATLAB
    Send $s_i(t+1)$ to ThingSpeak
end

3.8 Experimental Setup

The laboratory-based test facility is shown in Figure 3.10 and includes a NodeMCU microcontroller to implement the MPC and an IoT smart-plug based platform to provide ON-OFF control of the iGENIX IG 3920 (55W), VonShef 13/291(50W) and the Russell Hobbs RHCLRF17B (50W) domestic refrigerators (each refrigerator is ON-OFF controlled via its own smart-plug). It is important to note that the RHCLRF17B uses thermoelectric cooling technology, so no refrigerant is used. Consequently, no compressor is required, and the unit normally operates at 100% duty (i.e., always ON)

[111]. Internal and ambient temperatures are measured using a DS18B20 waterproof sensors and a TMP102 module, respectively.

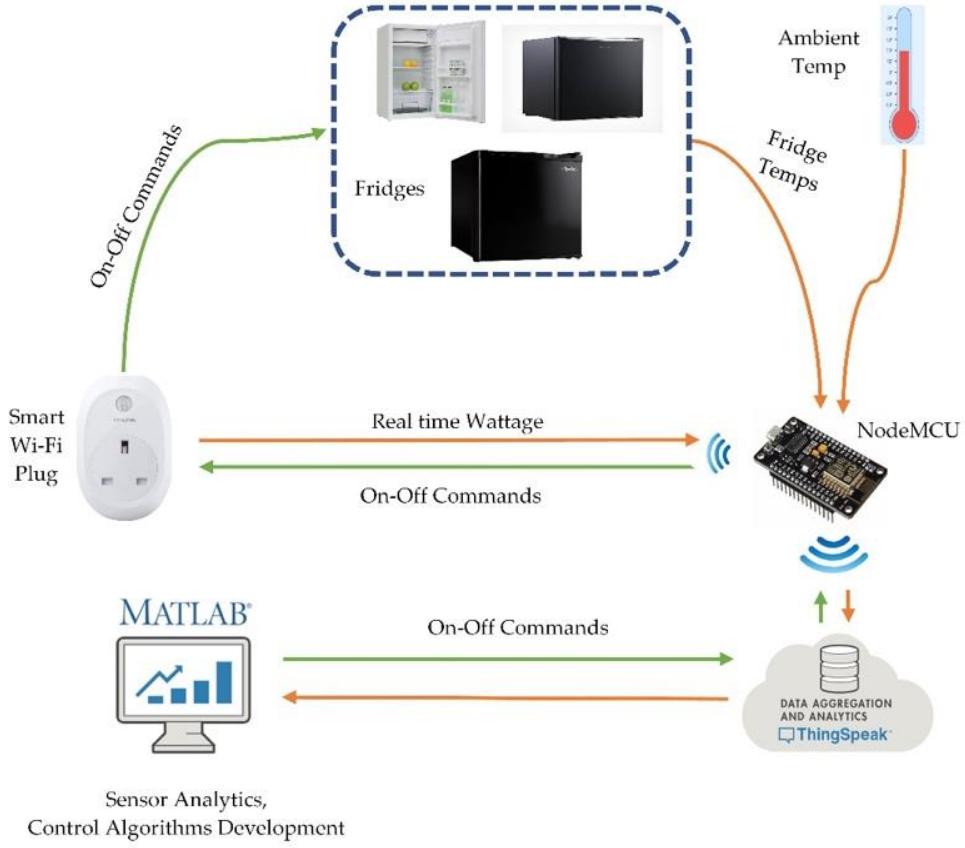


Figure 3.10: Experimental setup for local access

The hardware test facility comprises of three domestic refrigerators, see Figure 3.11. A detailed summary of the appliances is given in Table 3.2. Each refrigerator is instrumented with a DS18B20 waterproof sensor to measure the internal temperature. The sensors have a $-55\ ^{\circ}C$ to $+125\ ^{\circ}C$ temperature range and a $\pm 0.5\ ^{\circ}C$ accuracy. During the tests, the ambient temperature is measured with an accuracy of $\pm 0.5\ ^{\circ}C$ from $-40\ ^{\circ}C$ to $+125\ ^{\circ}C$ by a TMP102 module. Real-time power consumption of each refrigerator is measured using TP-Link Smart Wi-Fi Plug (HS110) with an accuracy of

±0.2 *W*. The Smart plug is also used to provide ON-OFF control of the refrigerator. The network connection is established using a NodeMCU which is an open-source IoT platform that includes integrated support for Wi-Fi. The experimental setup uses ThingSpeak for data acquisition and monitoring in the cloud.

Table 3.2 Specification of appliances

	iGENIX	VonShef	Russell Hobbs
Model	IG 3920	13/291	RHCLRF17B
Type	Compressor	Compressor	Thermoelectric
Energy rating	A+	A+	A+
Total storage capacity (Litres)	90	47	17
Power (W)	55	50	50
Voltage (V)	220-240	220-240	220-240
Frequency (Hz)	50	50	50

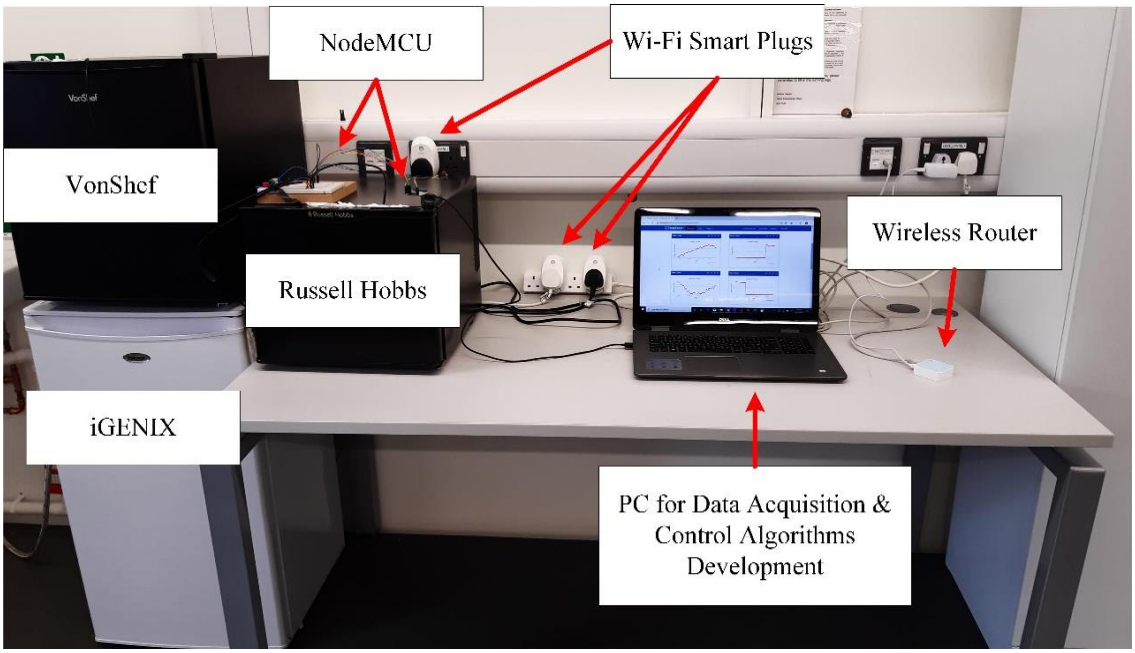


Figure 3.11: The hardware setup used in the measurement tests for local access

3.9 Experimental Results

The developed MPC algorithm is implemented with a sampling rate of 20 seconds and experimental trials are undertaken over a period of 210 minutes in each case. A key feature of the algorithm is a constraint on the total maximum power consumption (accumulative sum of the 3 refrigerators), and thereby facilitate peak power shaving. Also, the number of ON-OFF events is constrained to be 6 in each case. The test schedule is as follows:

A) Refrigerators operate in isolation without any scheduling controller. This aligns with the normal operating conditions of domestic refrigerators and provides a comparative benchmark.
B) P_{max} is limited to 110W: the maximum aggregated power for all refrigerators is constrained with equal power supply priority given to all refrigerators.
C) P_{max} is limited to 60W and all refrigerators are given equal power supply priority weightings.
D) P_{max} is limited to 60W and the refrigerators are given unequal power supply priority weightings.

For consistency, the ambient temperature is controlled to be within the range 23ºC ± $1^{o}C$ since it is known that it can have a significant influence on energy consumption [112], [113]. The temperature bands are taken from reference [127]. According to the food legislation [127], different upper and lower band temperatures are considered for each refrigerator. For example, the temperature of the cooler, which is used to keep drinks cool, should not exceed 8 °C, and that of ready-to-eat foods should not exceed 6 °C (preferably 2 to 4 °C). Three different types of refrigerators with different technologies (compressor and thermoelectric) that were cheap, and best-selling were selected.

The iGENIX, VonShef and Russell Hobbs refrigerators are unevenly loaded with 10L, 6L and 2L of water, respectively, and the doors remained closed for the duration of the

trials. Desired upper and lower temperature setpoints and minimum OFF and ON times per cycle for each refrigerator, are chosen as in Table 3.3.

Table 3.3 Data for iGENIX, VonShef and Russell Hobbs

Name	Upper Band (*ºC*)	Lower Band (*ºC*)	Minimum on time (*sec*)	Minimum off time (*sec*)
iGENIX	3.5	2	220	240
VonShef	2.5	1.5	260	200
Russell Hobbs	7.5	4.5	380	100

3.9.1 Trial A: Refrigerators operate in isolation without a scheduling MPC controller

This initial trial investigates how the refrigerators operate with no co-ordinated MPC scheduling applied. This effectively mimics how each would operate in a normal isolated domestic setting, and the aggregated power characteristic that would be obtained. Figure 3.12 shows each refrigerator's internal temperature, the ambient temperature, individual power consumption and the total aggregated power consumption. The hysteresis controller described in [114] is used to adjust the upper and lower temperature setpoints for the VonShef and iGENIX units, whilst the Russell Hobbs unit employs thermoelectric cooling technology and, as such, it nominally has a 100% operational duty (not ON-OFF) with variable power usage, as can be seen from Figure 3.12. Of particular note from Figure 3.12 is that without any constraints or co-ordinated (scheduling) control, there are significant periods when all units are ON, and periods of relatively high peak power consumption are therefore evident.

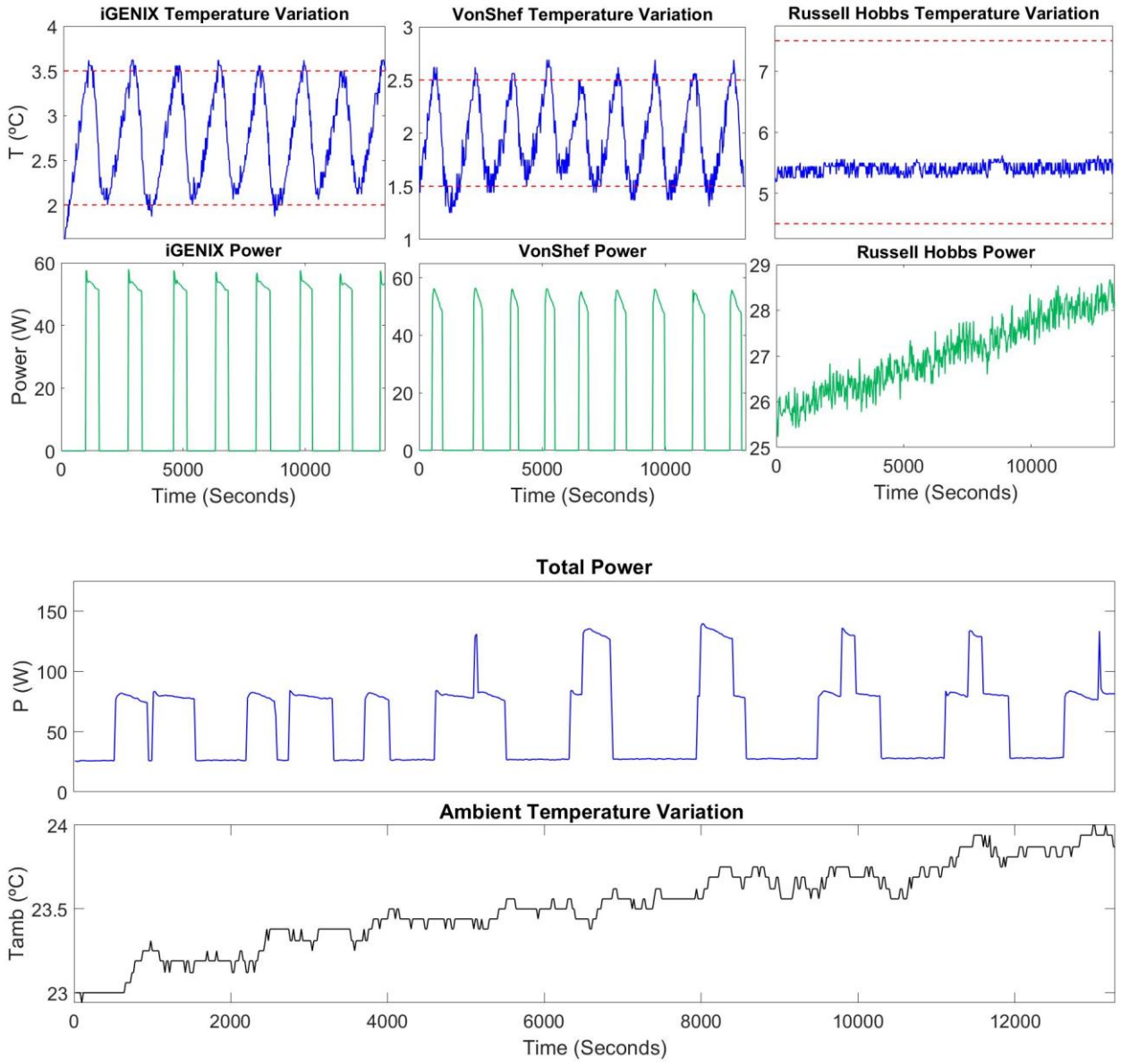


Figure 3.12: Refrigerator's results and total power consumption for trial A

Figure 3.13 shows total power and power of the individual refrigerators on the same timescale for clarity.

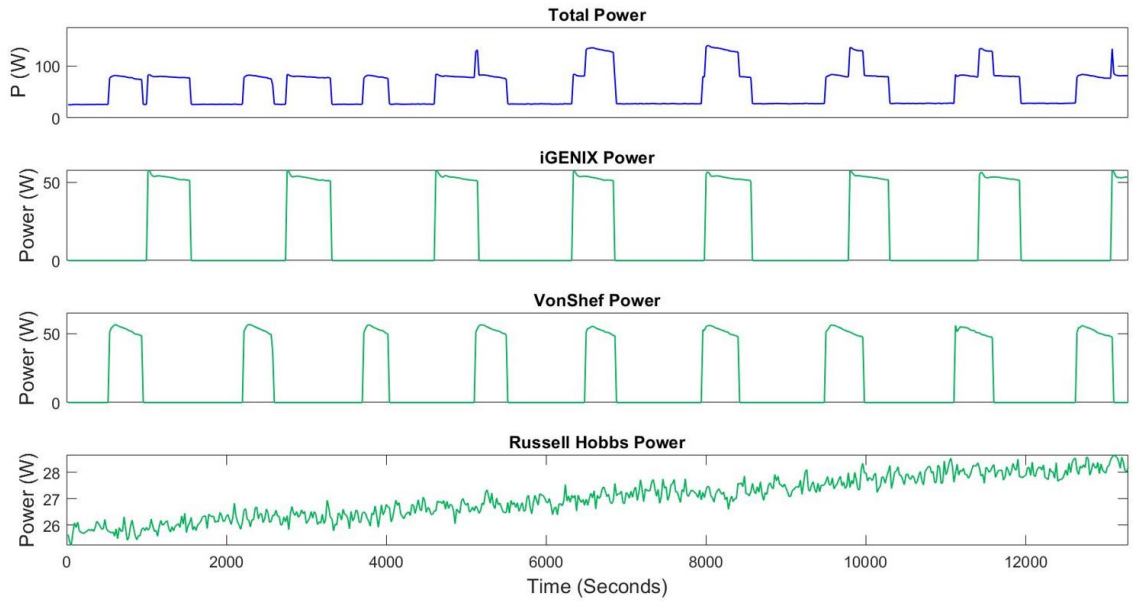


Figure 3.13: Refrigerator's power and total power for trial A

Russell Hubbs uses thermoelectric technology instead of a compressor, and has very small dimensions, so it is sensitive to changes in ambient temperature. As the ambient temperature increases, its power consumption rises in order to maintain a constant internal temperature.

3.9.2 Trial B: MPC scheduling with Pmax=110W and equal supply priority is given to all refrigerators

Figure 3.14 shows results of a trial under the condition of P_{max}=110W, effectively limiting the MPC algorithm to supplying power to a maximum of 2 refrigerators at any instant. In this case, the supply priority weightings are chosen to be equal with Q = *[1,1,1]* and R = *[1e-5,1e-5,1e-5]*. From the results of Figure 3.14, it is clear that all of the refrigerators can maintain their temperatures within required bounds and demonstrates that although the peak power has been constrained, there remains

sufficient power overhead to supply the cooling requirements of each unit so long as they are appropriately scheduled.

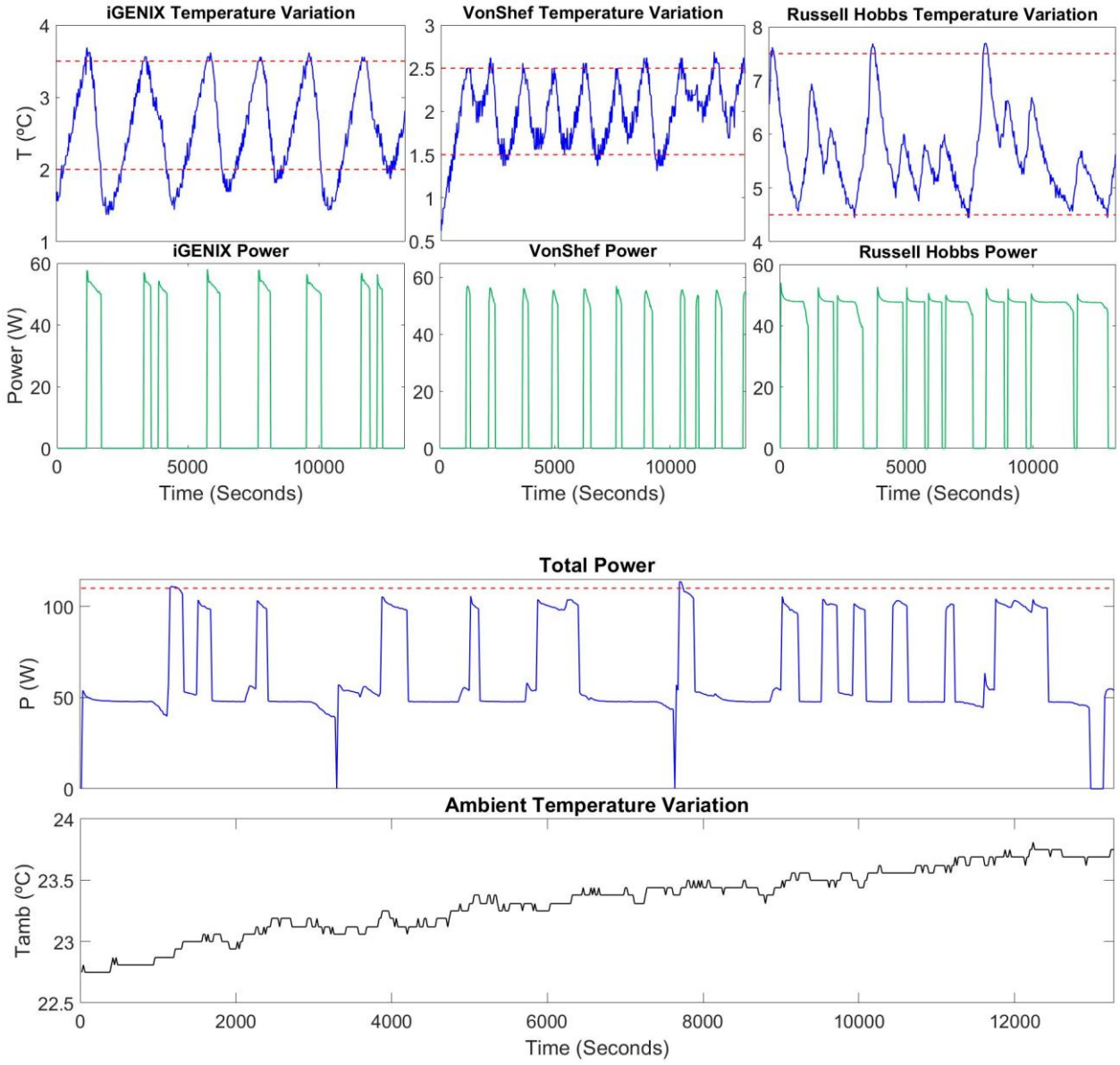


Figure 3.14: Refrigerator's results and total power consumption for trial B

Figure 3.15 shows total power and power of the individual refrigerators on the same timescale for clarity.

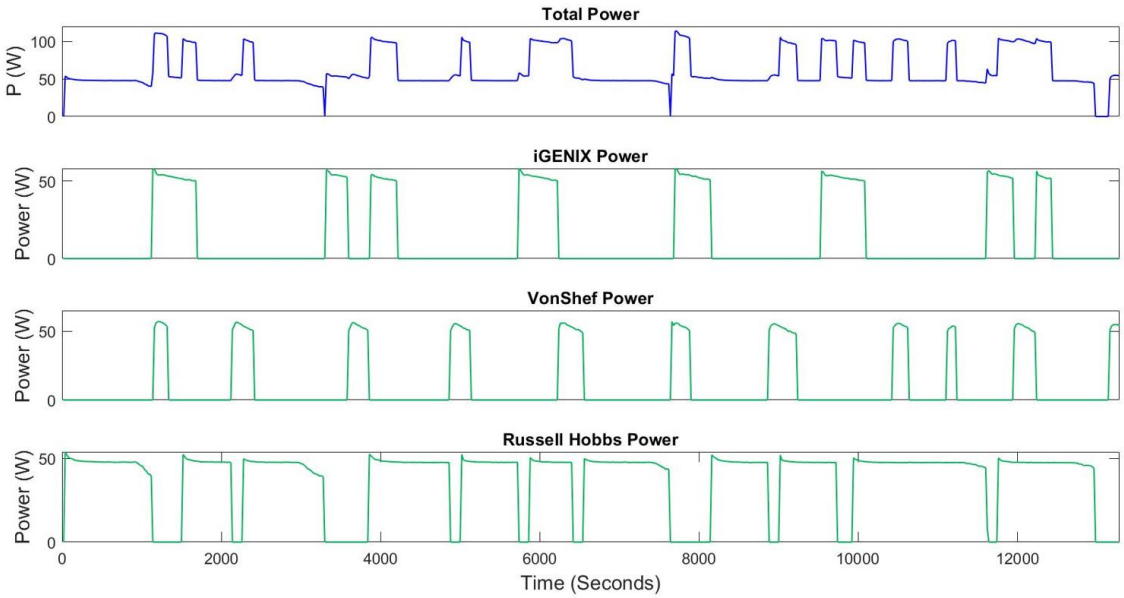


Figure 3.15: Refrigerator's power and total power for trial B

3.9.3 Trial C: MPC scheduling with Pmax = 60W and equal supply priority given to all refrigerators

Since there will always exist a minimum average power necessary to control the internal temperatures of all the refrigerators, it is instructive to investigate what happens when the demanded constraint on peak power (P_{max}) is reduced to a value that is below what is required for all the refrigerators to adequately cool their product. For this scenario, P_{max} is now reduced to 60W, effectively constraining the MPC to allow power delivery to allow only a single refrigerator at any instant—this constitutes a very severe power constraint for this refrigerator network. Again, the supply priority weighting matrices are chosen to be *Q = [1,1,1]* and *R = [1e-5,1e-5,1e-5]*. From the results of Figure 3.16, it can be seen that the temperatures now exceed the required bounds due to the severe power constraint, although the temperature of the iGENIX unit is less affected as a result of its higher thermal product mass (10L of water), and hence it takes longer for the temperature to rise and exceed the bounds. Nevertheless, it is clear from the results

that the MPC controller still constrains the power to $< 60W$ i.e. forces peak power shaving.

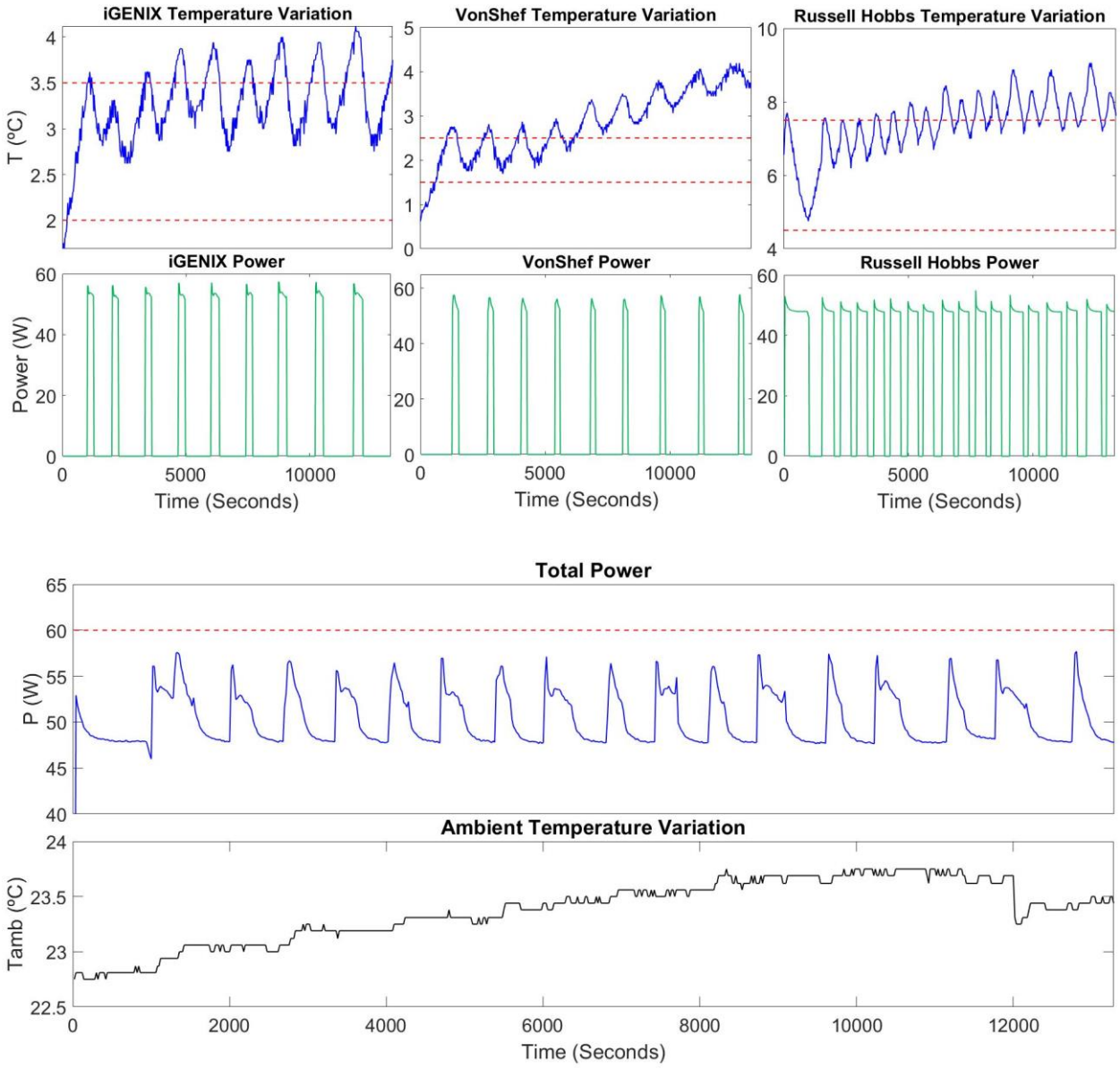


Figure 3.16: Refrigerator's internal temperature, ambient temperature, power consumption and total power consumption for trial C

Food safety and the quality of refrigerated food depend on the good performance of the refrigerator and are closely linked to temperature distribution and airflow inside the refrigerator. Depending on the exceeding time and the type of food or medicine in the refrigerator, it can cause spoilage and increase the risk of food-borne outbreaks in domestic households [127].

Figure 3.17 shows total power and power of the individual refrigerators on the same timescale for clarity.

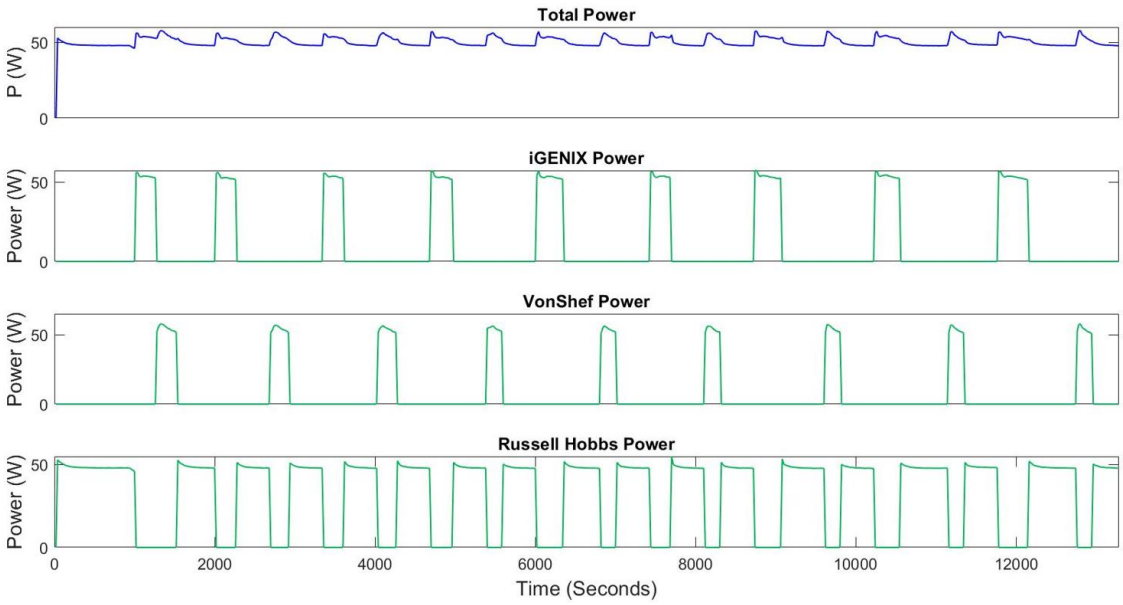


Figure 3.17: Refrigerator's power and total power for trial C

3.9.4 Trial D: MPC scheduling with Pmax = 60W and power preferentially delivered to the VonShef unit

Finally, under conditions where P_{max} is again limited so that it does not allow all refrigerators to maintain their temperature with required bounds, (as in test trial C), the proposed MPC algorithm can readily accommodate preferred priority scheduling, where the refrigerators can be allowed power preferentially. To show this, a trial similar to C)

is conducted with $P_{max} = 60W$ but with the priority biased towards delivering power to the VonShef unit i.e. in this case the weighting matrices are chosen to be $Q = [1,2,1]$ and $R = [1e\text{-}5,1e\text{-}5,1e\text{-}5]$. Figure 3.18 shows the results of the experimental trial. A comparison with those from Trial C shows that the Russell Hobbs refrigerator exceeds its bounds more rapidly, whilst the VonShef unit is given preferential power to better maintain its temperature, albeit it still exceeds its bounds periodically due to the extremely severe power limitation.

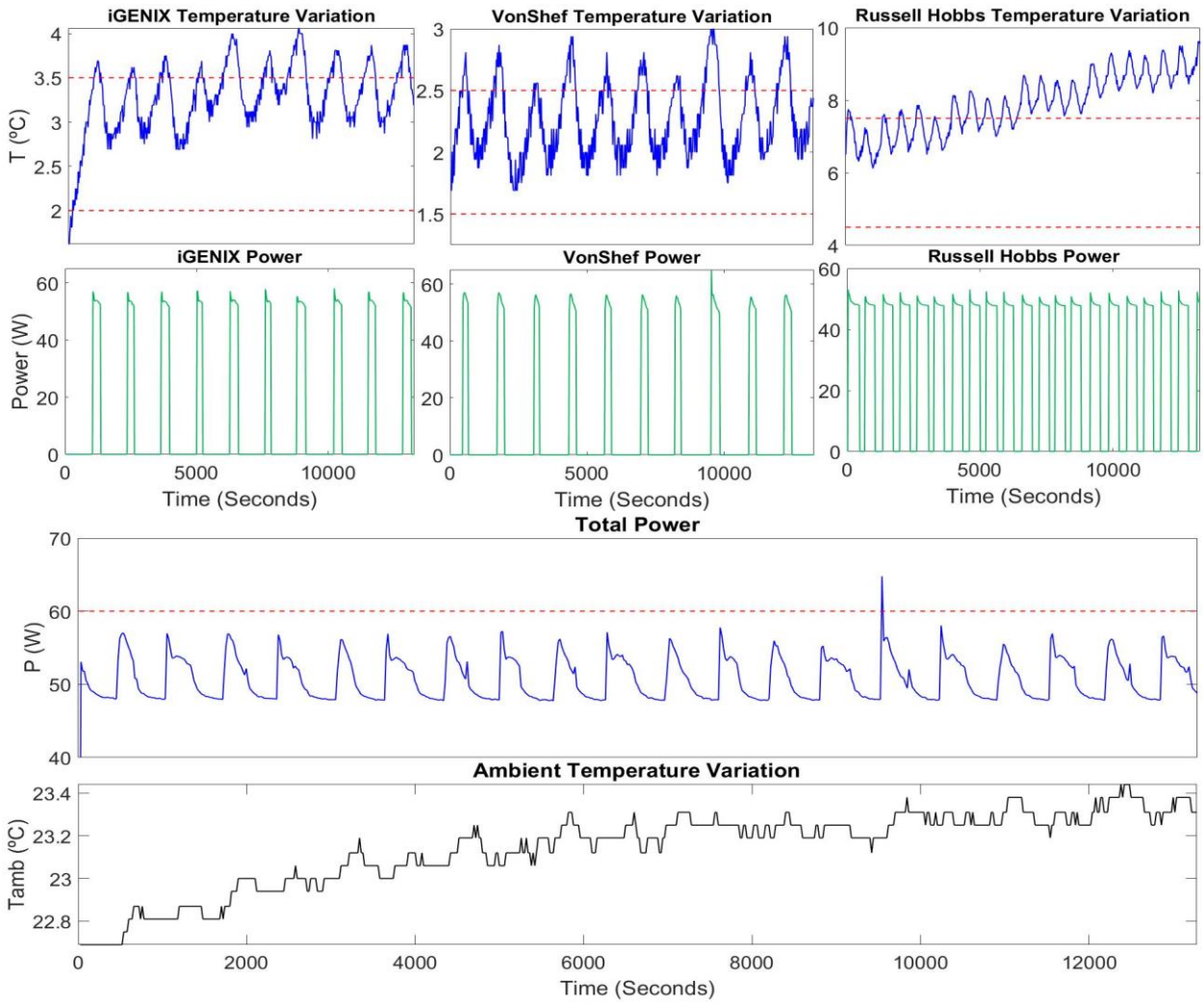


Figure 3.18: Refrigerator's internal temperature, ambient temperature, power consumption and total power consumption for trial D

Figure 3.19 shows total power and power of the individual refrigerators on the same timescale for clarity.

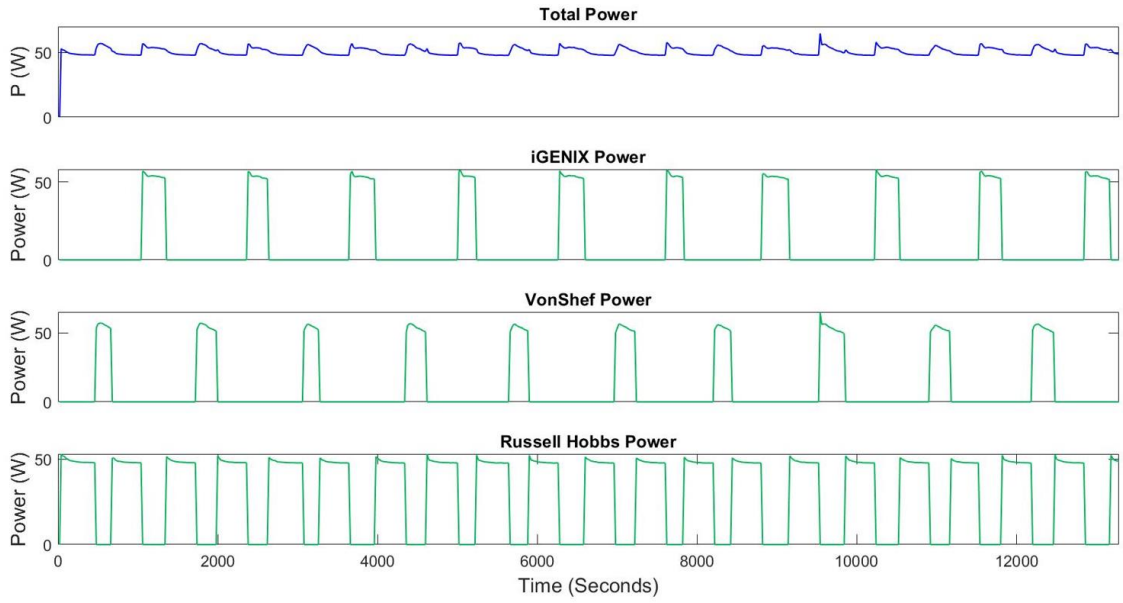


Figure 3.19: Refrigerator's power and total power for trial D

3.9.5 Comparison of Energy Consumption

The energy consumption during each of the trials has been measured, Figure 3.20. It can be seen that in trial D, the VonShef unit consumed 29.52*Watt-seconds* more energy and Russel Hobbs 34.53 *Watt-seconds* less energy compared to trial C. This is due to the higher priority weightings for the VonShef unit in D. In trial B, iGENIX and VonShef have energy savings of up to 19% and 29%, respectively, compared to the trial A, though in both A and B all of refrigerators remain within required temperature bounds. In contrast, Russell Hobbs consumed more energy in B compared to the A because it uses thermoelectric technology instead of a compressor and therefore loses stored thermal energy more rapidly, and hence is turned ON more. Moreover, the Russell Hobbs unit operates at around 27*W* in isolated mode, but this power usage reaches 50*W* when the MPC controller schedules its operation. These results indicate that in addition

to facilitating peak power shaving, operational energy savings can be accrued in compressor-based refrigerators. This is because the compressor-based refrigerators have compressors to move the refrigerant through the system, and cools through repeated refrigeration cycles. On the other hand, the cooler (Russell Hobbs) uses thermoelectric cooling technology, so no refrigerant is used and it transfers heat from one side to the other directly using electrical energy which is less efficient and the internal temperature rises faster when it is switched OFF. Since Russell Hobbs does not have a compressor, it loses cold air quickly, so when the total power is limited to 110 W in trial B, Russell Hobbs consumes more energy to keep the temperature of the refrigerator within the defined upper and lower temperatures. However, the other two refrigerators have compressors and lose cold air later. A well-designed compressor-based system will use 30% to 35% less power than the equivalent thermoelectric system [128].

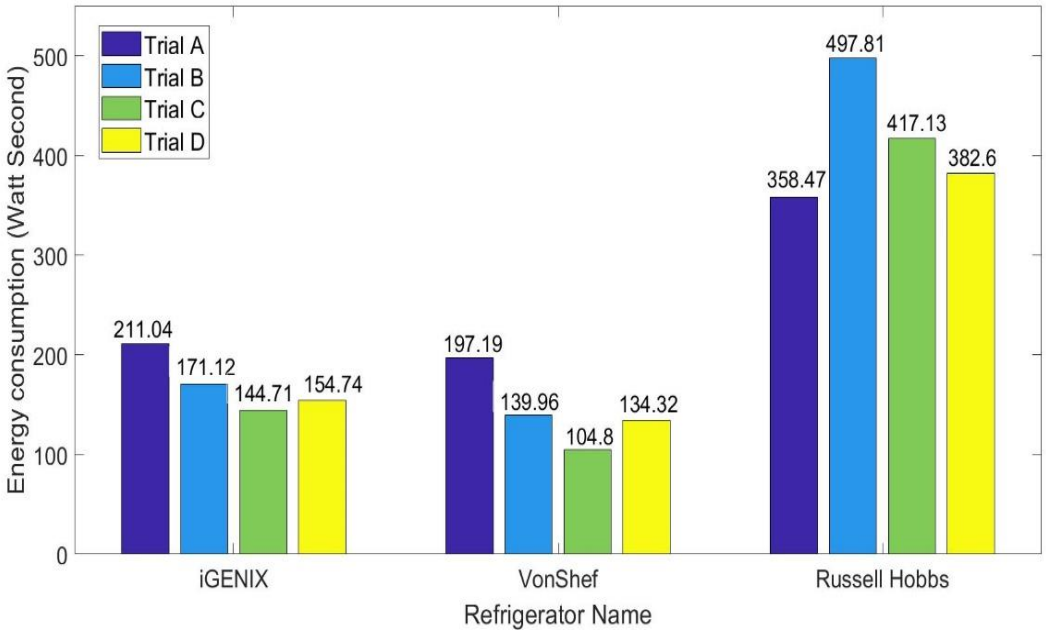


Figure 3.20: Refrigerator's energy consumption for Scenario A, B, C and D

The authors in [56] present results to show how the temperature profiles of the refrigeration cases, the active power and the current drawn by the compressors are

affected by responding to a DSR event. Performed DSR tests revealed that, depending on the amount of contents of refrigeration case, this can result in an increase of product temperature to up to 2 °C above its nominal value for the duration of the pull-down. During the July tests, the pack operation post-DSR demonstrated less stable cycling of active compressors, varying from 1 to 4 active compressors compared to 1 to 2 active compressors during March tests, with all 4 compressors being active immediately after the end of DSR. Measurements have shown an increase in supply current (by approximately 30%) after responding to the DSR event compared to the normal operation. It has been found that a 101 A transient inrush current over a period of 10 ms creates a drop in line voltages of up to 1.5 kV (~10% of rated) at the primary side and 100 V (~30% of rated) at the secondary side of the transformer, with a transition time of 25 μs. Inrush currents of the load have direct impact on the terminal voltage and current of the HV and LV sides of the step-down transformer and can cause voltage fluctuations on the power supply and potentially cause instability to the local power supply system. Moreover, inrush current can be much greater if the subsequent start-up sequence of multiple compressors is synchronized.

3.10 Domestic Refrigerators and Demand Side Response (DSR)

The advent and proliferation of IoT will ultimately allow the aggregation of widely distributed networks of domestic appliances, such as refrigerators and freezers, to take part in Demand Side Response (DSR) load-shedding events to help maintain grid frequency stability. Indeed, the advantages of using widely distributed networks of retail refrigerators to contribute to DSR events has already been recognized and reported in [56] and [115]. Here then, an experimental study investigates how domestic refrigerators can respond to DSR events using the presented MPC methodology. Specifically, results stemming from the initiation of two DSR events for the small network of refrigerators used in this study are given in Figure 3.21, where each refrigerator unit is given equal supply priority weighting on power, and load shedding is

initiated by instantaneously reducing P_{max} to $60W$. The first event occurs at t = 7140s and lasts for 30 minutes and the second occurs at t = 16060s and lasts for one hour. Moreover, Figure 3.23 shows a similar condition with a DSR demanded at t = 7200s and ending at t = 10780s but where the Russell Hobbs unit is given preferential access to power through the weighting matrices $Q = [1,1,3]$ and $R = [1e\text{-}5,1e\text{-}5,1e\text{-}5]$. As can be seen from the measurements in Figures 3.21 and 3.23, the refrigerators are able to respond instantly to power shedding events and the total power usage has been reduced to $60W$ when required. In addition, from Figure 3.23 it can be seen that the Russell Hobbs unit largely remains within temperature limits due to the additional priority biasing given to it by the MPC. Although only on a very small scale, this demonstrates the potential for the co-ordinated scheduling of widely distributed domestic refrigerators for contributing to aggregated load shedding events. In this section, only the controller performance is evaluated in terms of DSR event response, so there is no control for the ambient temperature and residents can change the ambient temperature as desired.

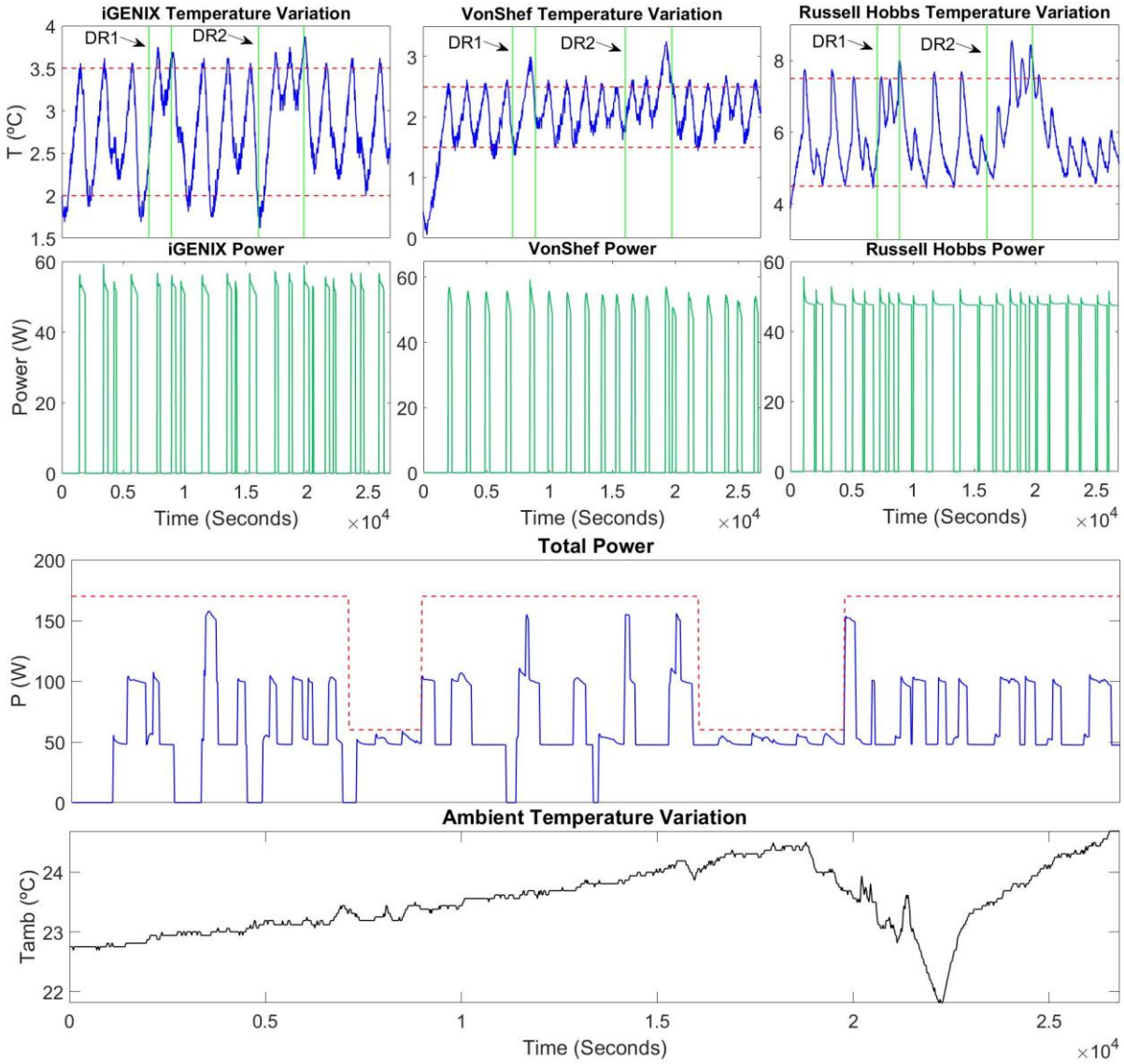


Figure 3.21: Results for DSR event with equal priority weighting given to all units

Figure 3.22 shows total power and power of the individual refrigerators on the same timescale for clarity.

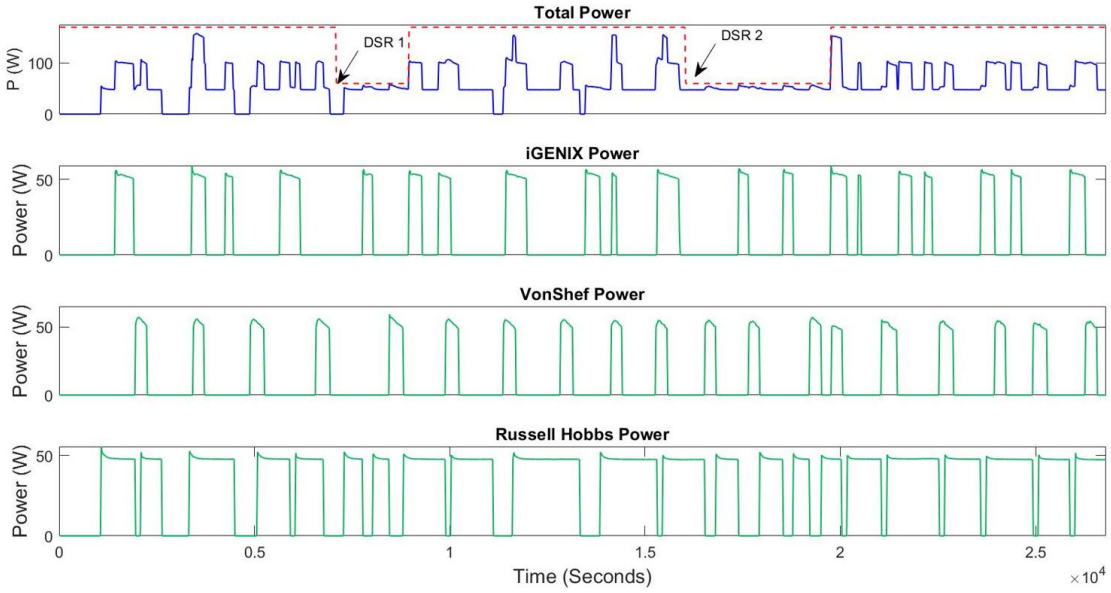


Figure 3.22: Refrigerator's power and total power for DSR event with equal priority weighting given to all units

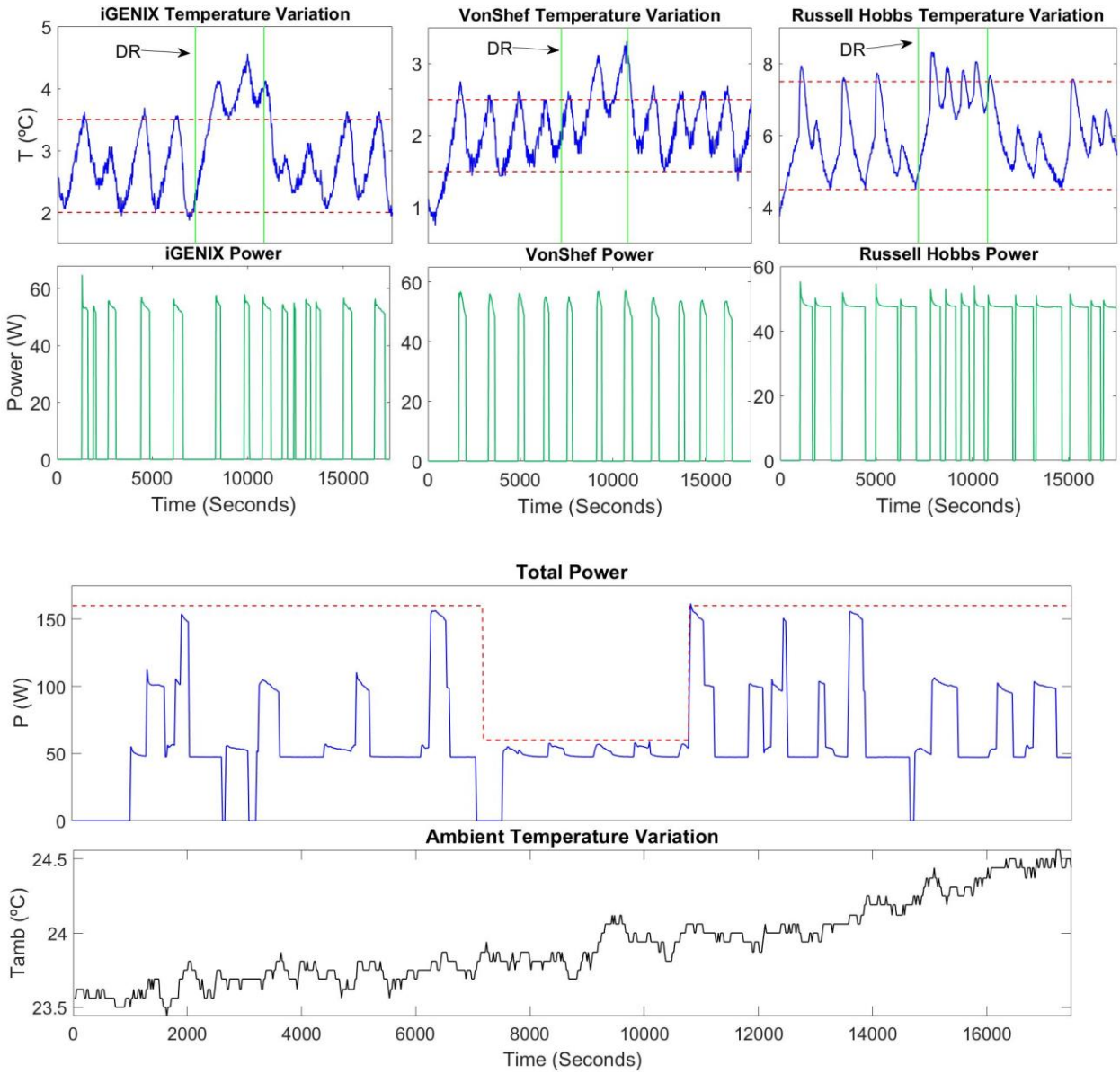


Figure 3.23: Results for DSR event (Russell Hobbs unit is given greater priority weighting)

Figure 3.24 shows total power and power of the individual refrigerators on the same timescale for clarity.

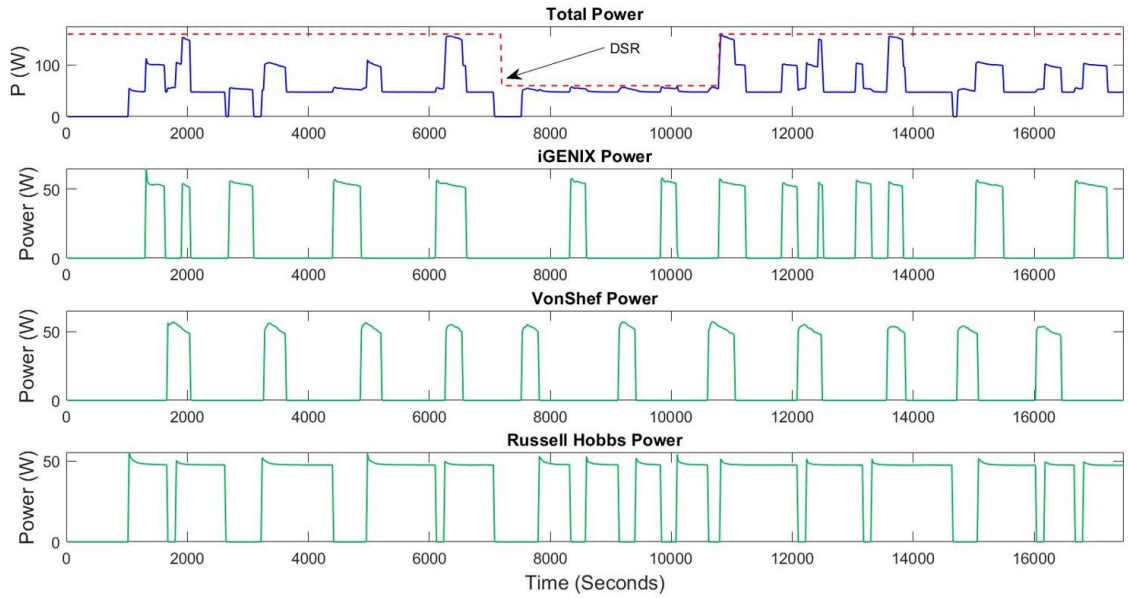


Figure 3.24: Refrigerator's power and total power for DSR event (Russell Hobbs unit is given greater priority weighting)

3.11 Impact of Hysteresis Band and Internal Thermal Mass on Refrigerator Operational Efficiency

As well as contributing to the co-ordinated operation of multiple refrigerators, the proposed MPC algorithm also allows for reductions in energy consumption of individual refrigerator units by virtue of being able to adaptively change their temperature hysteresis boundaries in real-time. It is shown below that knowledge of the underlying dynamics of each refrigerator from its identified parameters (an integral part of the proposed Binary Quadratic MPC) allows the hysteresis bands to be tailored to how much product is contained within the refrigerator. The benefits afforded by this are shown through experimental trials on the candidate iGENIX unit. Firstly, an initial experimental trial is undertaken with the temperature-controlled hysteresis band set to $\pm 0\ ^{o}C$ i.e., effectively mimicking a non-hysteresis type control scheme. The results are

given in Figure 3.25, which shows a high degree of compressor switching activity to try and maintain ideal temperature tracking. As can be seen from figure 3.25, due to the thermal mass in compressor-based refrigerator, there is a delay between power usage and fridge temperature changes.

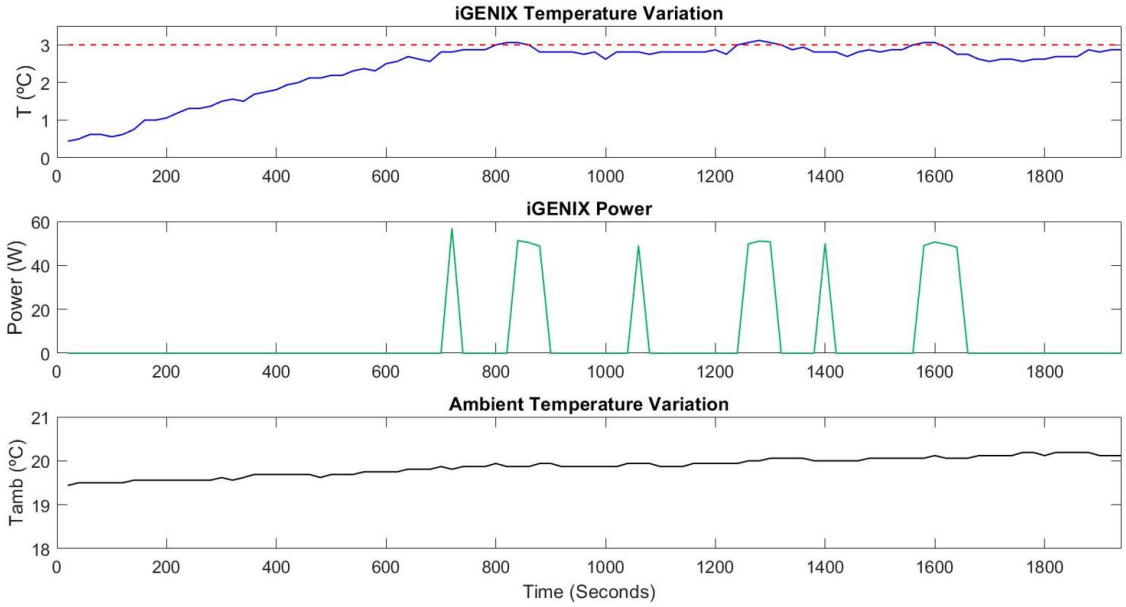


Figure 3.25: Internal temperature and power variation with ±0 ºC hysteresis band

Two additional trials are subsequently undertaken with i) no product in the refrigerator, and ii) with 10L of water in the refrigerator. Operation of the refrigerator using different hysteresis bands is investigated under the two scenarios. Specifically, hysteresis bands of ±0.5 *ºC*, ±1 *ºC*, ±1.5 *ºC* and ±2 *ºC* are used in each case. The results are shown in Figures 3.26 and 3.27. Notably, it can be seen that smaller hysteresis bands create more compressor ON-OFF events in both cases. Moreover, the empty condition requires a greater number of compressor starts than when there is product in the refrigerator, due to the availability of increased thermal mass in the latter case. For commercial compressors the number of starts per hour are typically assumed to be ≤6 [116], [117], and this is the value used in the proposed Binary Quadratic MPC algorithm. Based on

reference [127], 3 °C was selected as the reference temperature and different hysteresis bands were tested to investigate the effect of hysteresis band on energy consumption.

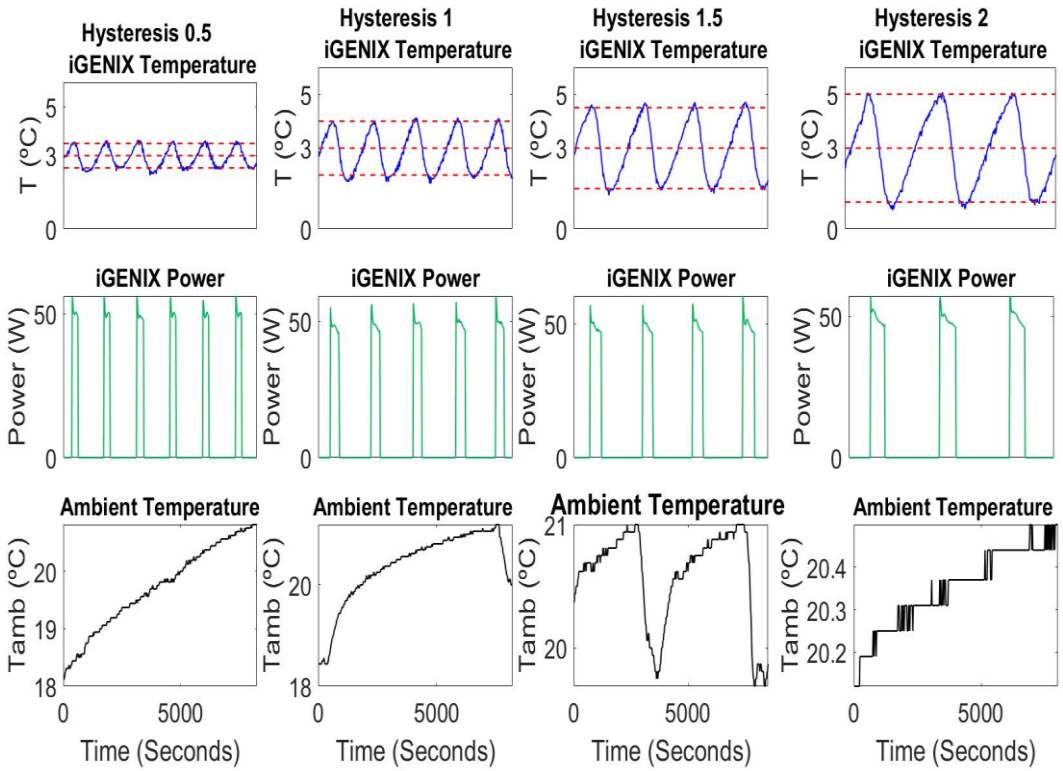


Figure 3.26: Internal temperature, ambient temperature, and consumed power for hysteresis bands of ±0.5 ºC, ±1 ºC, ±1.5 ºC and ±2 ºC without internal product (refrigerator empty)

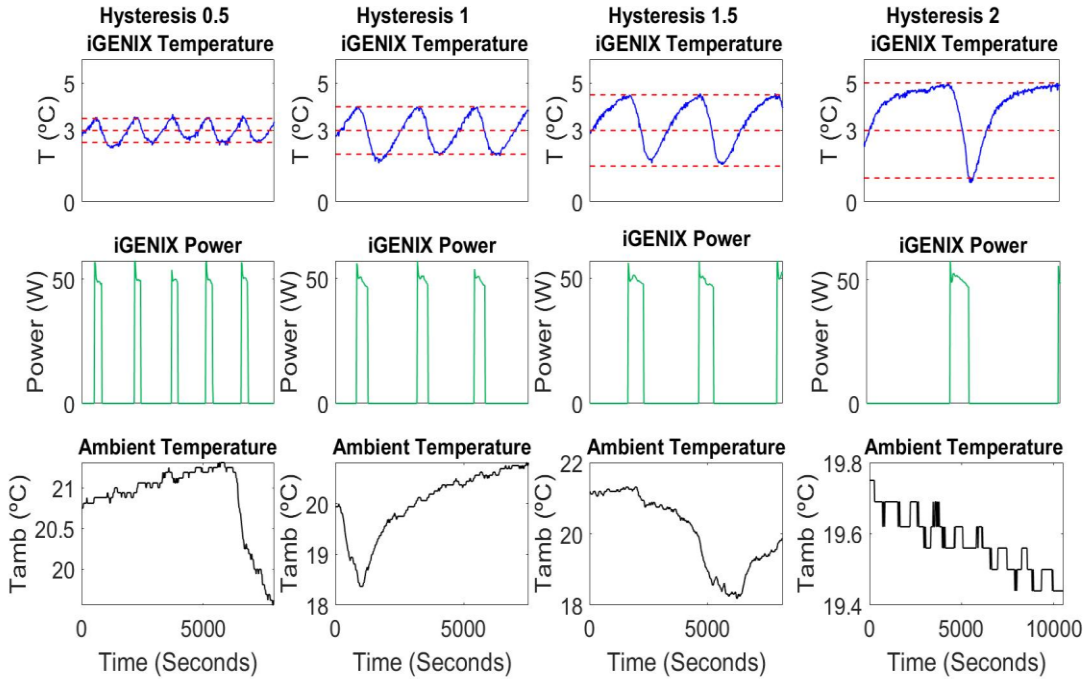


Figure 3.27: Internal temperature, ambient temperature and consumed power for hysteresis bands of ±0.5 ºC, ±1 ºC, ±1.5 ºC and ±2 ºC with internal product consisting of 10L of water

Of particular interest is the amount of energy consumed by the refrigerators when different hysteresis band are employed. For the cases considered, Figure 3.28 shows the energy consumption of each case projected over 1 year of usage. The energy consumption in each hysteresis band can be calculated using the area under the power usage curve. Figures 3.29 and 3.30 shows the energy savings for different hysteresis bands under empty and with 10L of water conditions respectively. The results indicate that with appropriate real-time adaptive identification and control the hysteresis band can be changed to accommodate varying product, and hence improve the long-term energy consumption. For instance, in the two scenarios identified, energy savings of up to 20% and 10%, respectively, can be expected between best- and worst-case conditions. Notably, increasing the product thermal mass has the impact of making the refrigerator less sensitive to the imposed hysteresis band. The authors in [56] states that

the absence of (product) thermal inertia and undermined cooling capacity contributed to the extended pull-down state during DSR events. The authors in [56] shows the effect of the thermal inertia, which assists the air-off temperature to slowly reach the set point value, with only a short-period region.

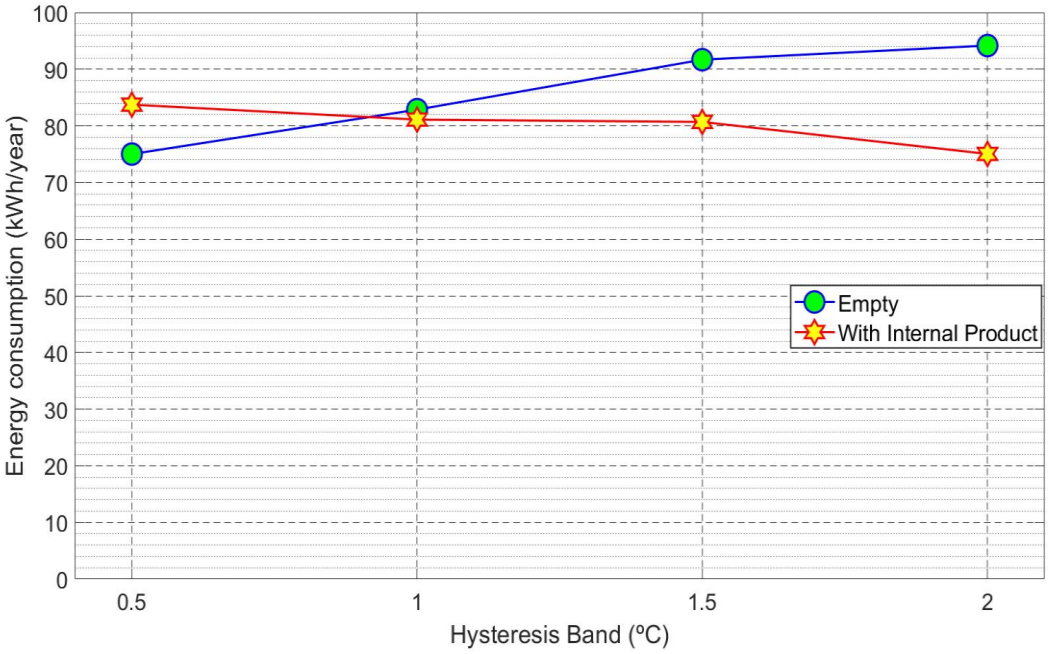


Figure 3.28: Projected annual energy consumption for different Hysteresis bands (Empty and including internal product (10L of water))

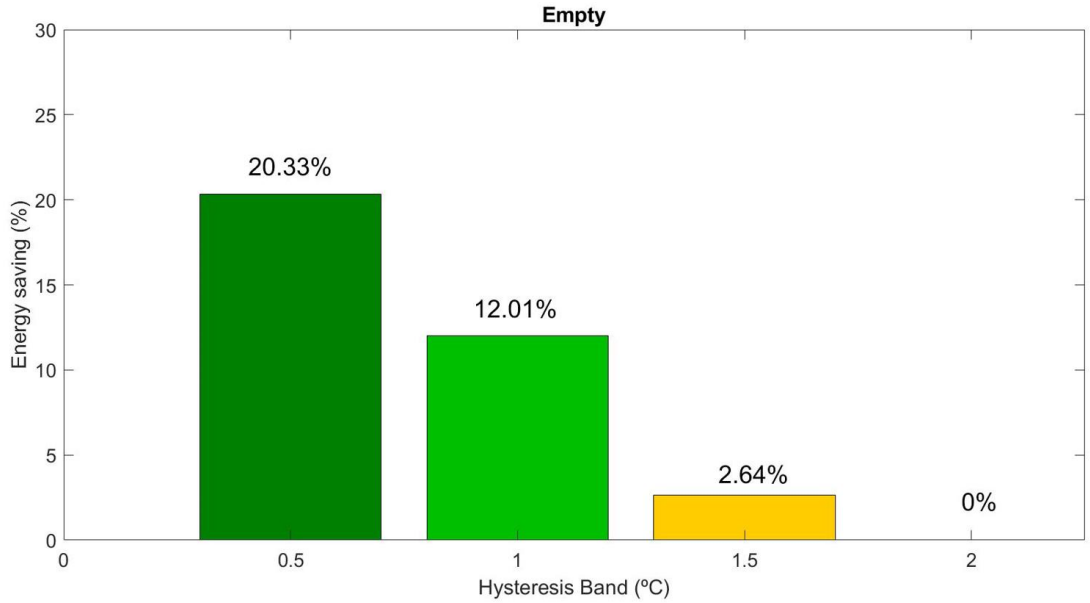


Figure 3.29: Projected annual energy savings for different Hysteresis bands (Empty)

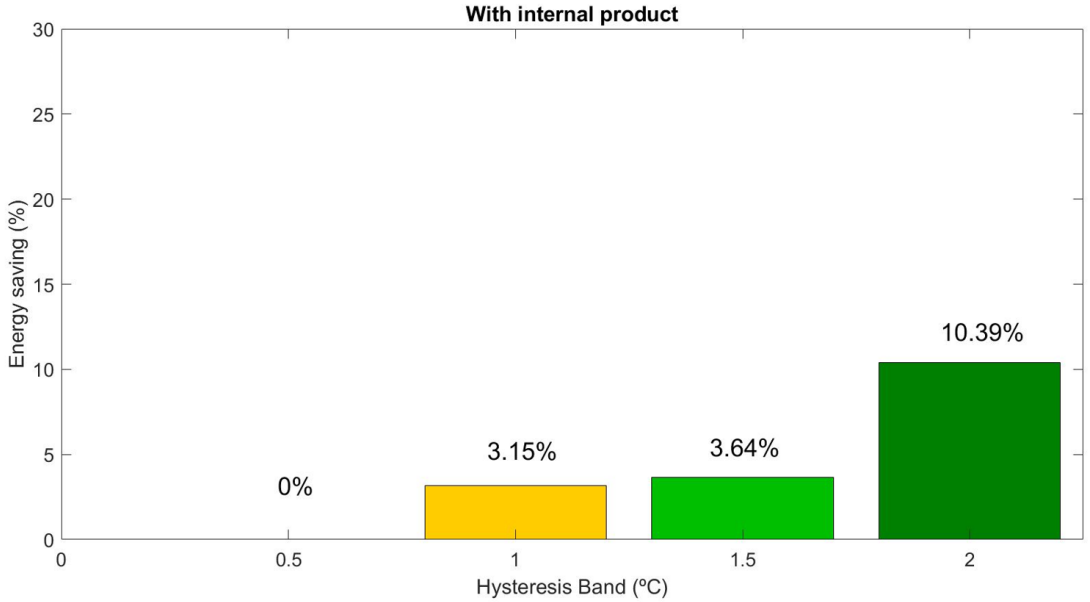


Figure 3.30: Projected annual energy savings for different Hysteresis bands (including internal product (10L of water))

3.12 Conclusions

The research presented in this chapter has proposed a real-time identification of refrigerator dynamics. Experimental trials show that such a real-time model update can identify the refrigerator dynamics, and respond to common disturbances such as door openings. Moreover, a fundamental part of subsequently investigated MPC control schemes is the identification of the underlying system dynamics. Real-time model updates are therefore essential for high-performance control i.e., adaptive mechanisms are necessary. In addition, the research presented in this chapter has proposed a time-varying priority-based on-off scheduling algorithm that can effectively schedule networks of widely distributed refrigerators. Comparative studies of measurements from experimental trials show that the co-ordinated scheduling of refrigerators allows energy savings of between 19% and 29% compared to their traditional isolated (non-co-operative) operation. Moreover, by adaptively changing the hysteresis bands of individual fridges in response to changes in thermal behaviour, a further 20% of savings in energy are possible at local refrigerator level, thereby providing benefits to both network supplier and individual consumer. Importantly, manufacturers do not need to make any significant hardware changes to reap these benefits, as the control methodology uses only sensor and actuation mechanisms already present in modern domestic refrigerators.

It should be noted that whilst the proposed methodology has been specifically directed towards the co-ordinated operation of refrigerators, the underlying techniques are more widely applicable, for instance, for the preferential charging of a multiple electric vehicles with constrained total aggregate power availability, or HVAC systems in large buildings.

The MPC controller described in Chapter 3 will be used in Chapter 4. In chapter 4, an IoT platform is proposed to remotely monitor and schedule multiple fridges using MPC.

Customers can monitor the fridge remotely using a phone or laptop, and adaptive hysteresis bands could be beneficial for customers. Also, customers can define the upper and lower band temperatures for each fridge. Based on food safety standards, setpoint temperatures are selected for this experiment [127]. The stability of the grid can be improved using scheduling of multiple fridges and instant response to DSR events.

4 SYNCHRONISED POWER SCHEDULING OF WIDELY DISTRIBUTED REFRIGERATORS USING IOT

4.1 Introduction

This chapter proposes an IoT controlled platform to remotely monitor and control appliances in the residential sector. An IP-based synchronized wireless mesh network is implemented through IoT hardware (based on a NodeMCU) and Google Sheets to monitor and schedule the operation of aggregated domestic refrigerators under a Model Predictive Control (MPC) scheme. Benefits afforded by the proposed technique are investigated through experimental trials from VonShef 13/291 (50*W*), iGENIX IG 3920 (55*W*) and Russell Hobbs RHCLRF17B (50*W*) domestic refrigerators sited in three different domestic locations in the city of Lincoln, UK. Results show that the proposed network is able to monitor the widely distributed refrigerators and adaptively schedule the appliances to reduce peak operational loads and facilitate Demand Side Response.

Further widespread expansion of the proposed technique would allow a rapidly deployed regional Demand Side Response (DSR) strategy to aid grid stability. This platform can be used with Wi-Fi, 4G and 5G networks, as long as the network has an assigned public IP address. In the event of an internet connection failure, based on the prediction and control horizons in MPC (chapter 3), the scheduling decision will be available for 5 sample times, if internet connectivity does not return for the next five steps the refrigerators will return to normal operation.

4.2 IP-based synchronized wireless network (remote control and monitoring)

This section proposes an IoT controlled platform to remotely monitor and control appliances in the residential sector. An IP-based synchronized wireless network is implemented through IoT hardware. A suitably widespread mesh network affords the opportunity to schedule and monitor the operation of distributed domestic appliances simultaneously to collectively improve energy efficiency and reduce peak power consumption on the electrical grid. Here then, this chapter proposes the implementation of a Model Predictive Control (MPC) scheme, initially proposed in chapter 3, to provide the co-ordinated scheduling of power to domestic refrigerators in a small number of households around the City of Lincoln, UK. Measurements are taken from the refrigerators under the IP-based wireless mesh network and Google Sheets is used for cloud data acquisition and monitoring. The proposed network shown in Figure 4.1 consists of two main parts: the server and client units. Client units are deployed at the different refrigerator locations. Each client is assigned a unique public IP address to provide remote access, and is used to collect measurements (e.g., internal fridge temperature, ambient temperature and the power consumption of the fridge), send the measurements to the server via received HTTP requests, receive control instructions from the server and apply them to the house appliances (e.g., fridge on/off commands).

The server can be located anywhere with Wi-Fi access and is used to send control commands and HTTP requests to the clients using the ‘port forwarding’ feature of the routers to collect sensor data simultaneously and send them to the Google Sheets for data acquisition and monitoring—see Figure 4.1.

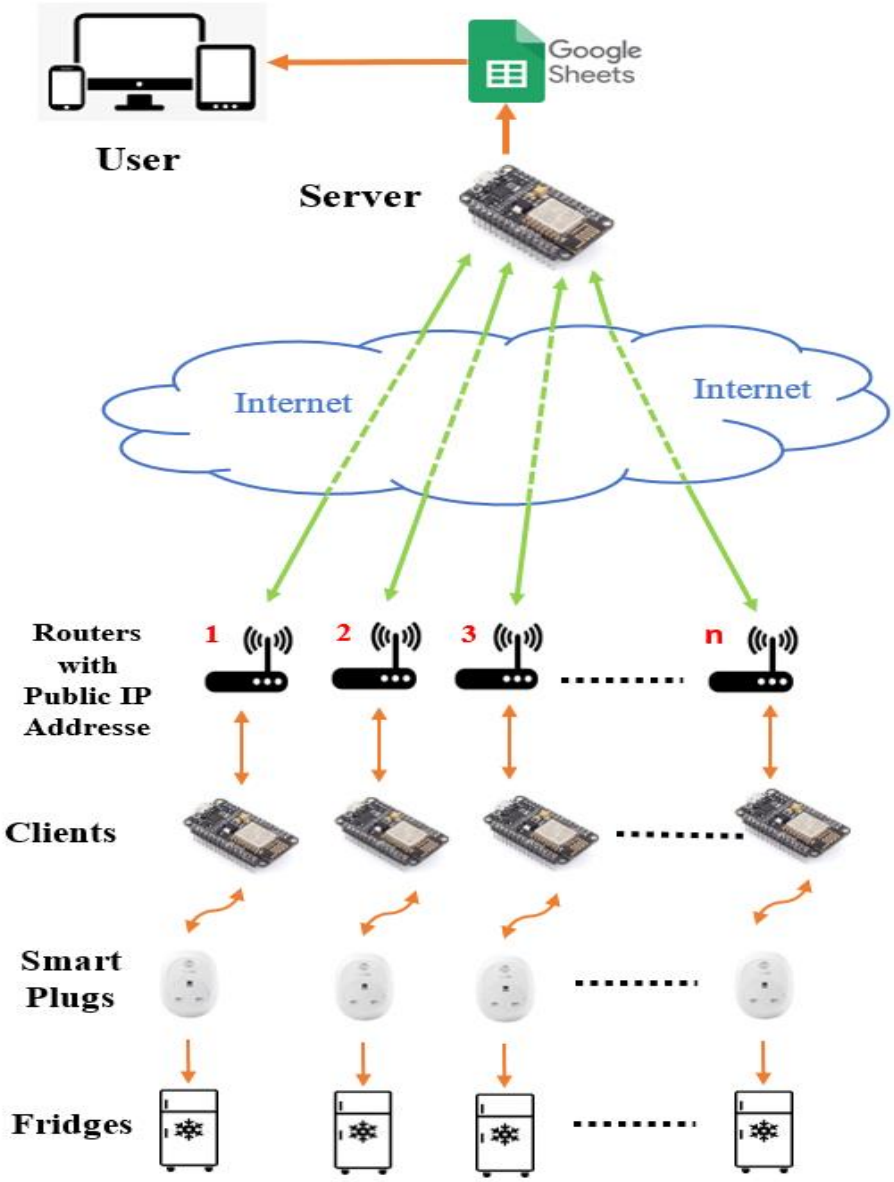


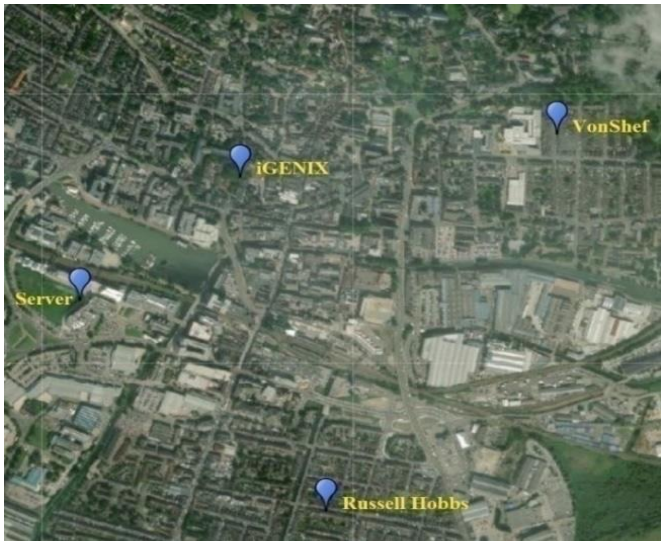
Figure 4.1: IP-based synchronized wireless network structure

The controller is in the server side. Server receives all the data from clients then make the decisions based on MPC and finally, send the scheduling decisions to clients (fridges).

The customer sets the upper and lower temperature bands for each refrigerator. The measured ambient and internal temperatures and power consumption are sent to the server using the mentioned sensors. The server makes decisions using the received data and MPC and sends them to clients to apply decisions for each refrigerator using the smart plug.

Customers will gain benefit if they participate in DSR events also they can monitor their appliances remotely from everywhere. Also, it helps to improve the grid stability.

As previously indicated, the study uses a wireless IoT scheme based on a synchronized wireless mesh network to remotely monitor and schedule three domestic refrigerators under Binary Quadratic MPC control to simulate DSR load-shedding events. The locations of the refrigerators are distributed around Lincoln (UK) as shown in Figure 4.2. These locations were selected because they were easily accessible because of COVID-19 restrictions. Any place can be chosen as long as the available Wi-Fi, 4G or 5G network has an assigned public IP address. The test facility components and hardware setup are also shown in Figure 4.2. Each refrigerator is instrumented with two DS18B20 waterproof sensors to measure the internal and ambient temperatures, a NodeMCU microcontroller to implement the Binary Quadratic MPC algorithm and a TP-Link Smart Wi-Fi Plug (HS110) to provide on-off control and measure the real-time power consumption of the refrigerators. Moreover, another NodeMCU is located at the University of Lincoln as a server to request the data from clients simultaneously and send them to Google Sheets for data logging.



(a)

(b)

Figure 4.2: System setup (a) The location of the refrigerators in the City of Lincoln, UK (b) hardware and fridges for measurement and control for remote access

An overview of router port forwarding features is given in the Appendices. The instructions for sending data to Google Sheets, related MicroPython code for client and the Arduino code for server can also be found in Appendices 1, 2 and 3.

4.3 Experimental Results

Measurements are taken with a fixed sampling period of 60 seconds to show the performance of the IP-based synchronized wireless mesh network under conditions of i) the refrigerators operating in isolation without any scheduling controller to show the monitoring feature of the proposed network, and ii) the custom Binary Quadratic MPC algorithm is implemented for jointly scheduling the operation of refrigerators in DSR events. The Server receives the DSR events from utility and based on the events and measured data the decisions using MPC will send these to clients (smart plugs). An initial experiment is conducted under normal operating conditions with no co-ordinated MPC scheduling applied. This effectively shows how the proposed network can monitor the domestic fridges remotely. The second trial shows the performance of the proposed network to investigate how the widely distributed domestic refrigerators can respond to DSR events using the MPC controller proposed in chapter 3.

4.3.1 Trial 1: Refrigerators operate under normal conditions without a scheduling controller (Monitoring mode)

An initial experiment is conducted under normal operating conditions with no co-ordinated MPC scheduling applied. This effectively shows how the proposed network can monitor the domestic fridges remotely. Figure 4.3 shows examples of 12-hour experiment trial intervals for each refrigerator’s internal temperature, the ambient temperature, individual power consumption and the total aggregated power consumption.

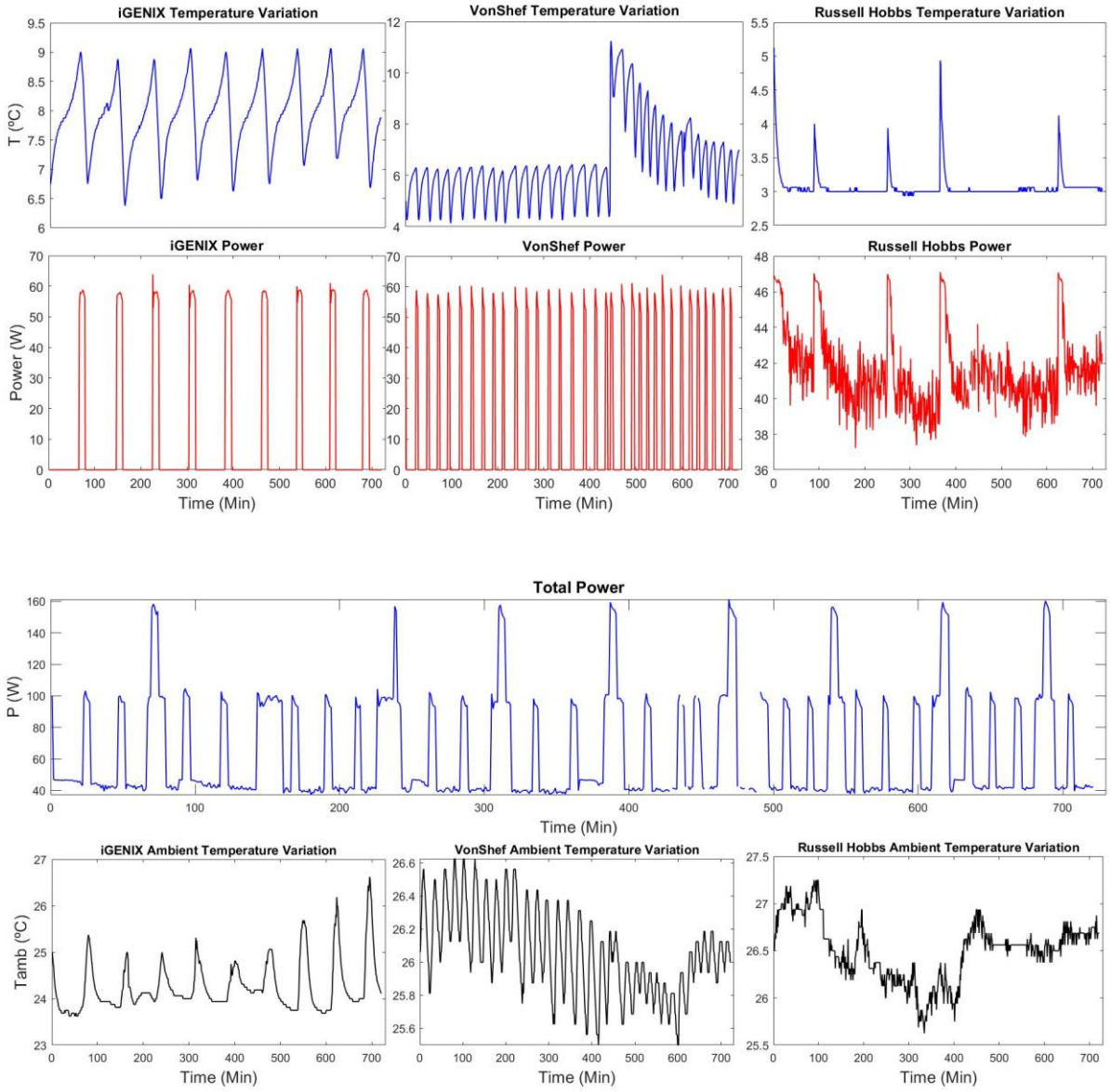


Figure 4.3: Refrigerators' internal temperatures, ambient temperatures, power consumption and total power for trial 1

4.3.2 Trial 2: Responding to DSR events using power scheduling

Measurements now show the performance of the synchronized wireless mesh network to investigate how the widely distributed domestic refrigerators can respond to DSR events using the MPC controller proposed in chapter 3. The iGENIX, VonShef and Russell Hobbs refrigerators are loaded with 12L, 6L and 3.5L of water, respectively, and the doors remained closed for the duration of the trials. The prediction and control horizon parameters used in the MPC are set to 5 samples. Desired upper and lower temperature setpoints and minimum non-working (minimum OFF) and working (minimum ON) times per cycle for each refrigerator, are shown in Table 4.1. These are required in order to limit the ON-OFF demand frequency so as to not unduly stress the compressor.

Table 4.1 Data for iGENIX, VonShef and Russell Hobbs

Name	Upper Band (ºC)	Lower Band (ºC)	Minimum on time (sec)	Minimum off time (sec)
iGENIX	3	0.5	220	240
VonShef	2.5	0.5	260	200
Russell	8	4	380	100

Results are given in Figure 4.4. In this scenario the Russell Hobbs unit is given preferential access to power during the DSR event via the weighting matrices in the MPC controller. DSR demanded with the total power usage of 60W occurs at t = 90 mins and lasts for 45 minutes. As can be seen from the measurements (Figure 4.4), the refrigerators are able to respond (virtually) instantaneously to power shedding events and the peak power consumption is limited to 60W. Moreover, all the refrigerators maintain their temperatures within required bounds before the DSR event whilst the Russell Hobbs unit remains within temperature limits due to the additional power supply priority attributed to it by the MPC.

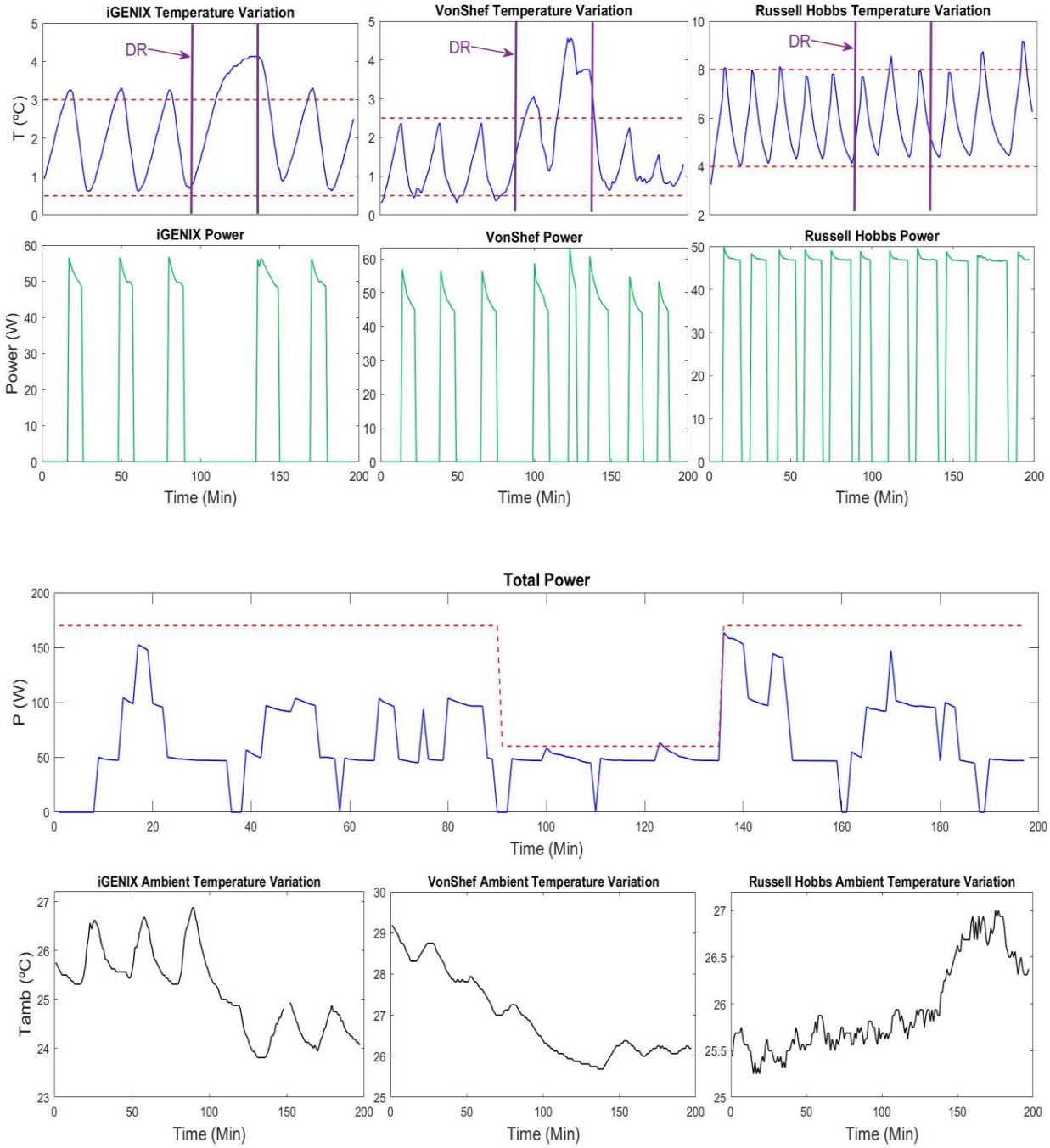


Figure 4.4: Refrigerators' internal temperatures, ambient temperatures, power consumption and total power for trial 2

Figure 4.5 shows total power and power of the individual refrigerators on the same timescale for clarity.

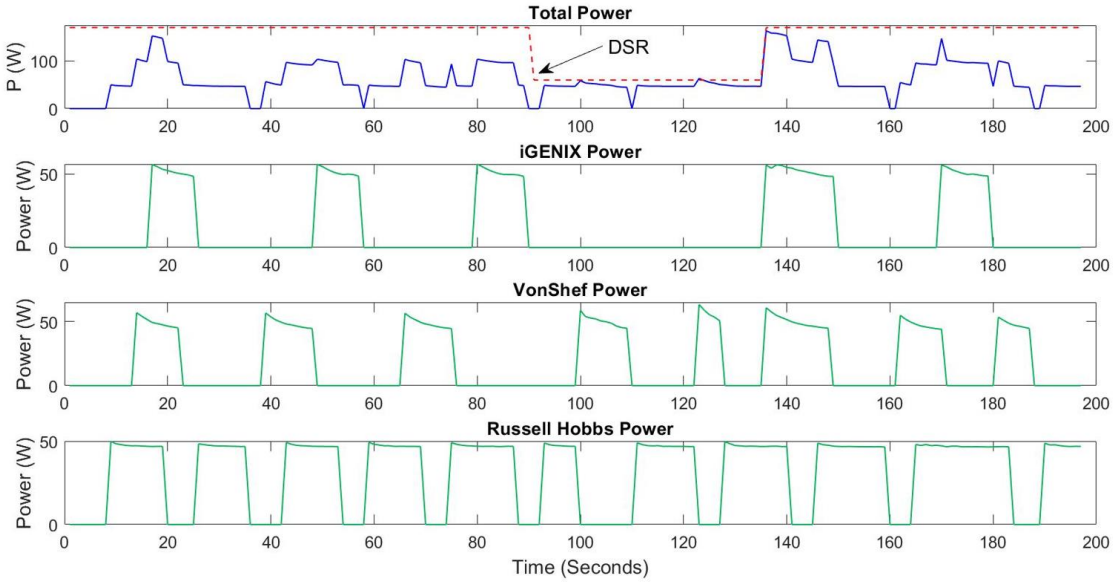


Figure 4.5: Refrigerators' power consumption and total power usage for trial 2

In the event of an internet connection failure, based on the prediction and control horizons in MPC (chapter 3), the scheduling decision will be available for 5 sample times, if internet connectivity does not return for the next five steps the refrigerators will return to normal operation.

4.4 Conclusions

In this chapter, the IP-based synchronized wireless network is proposed and implemented for monitoring and jointly scheduling the widely distributed domestic refrigerators in DSR events using the MPC controller. It is shown that the refrigerators are remotely monitorable from everywhere that internet access is available and they are able to respond to power shedding events almost instantaneously. It should be noted that the proposed methodology can be implemented more widely for other house appliances,

for instance, heating ventilation and air conditioning (HVAC) systems, or other TCLs. The proposed network performance was evaluated through two tests. During the first experiment, the monitoring feature was examined, and it was found that each refrigerator was monitored remotely. Second, the refrigerator's response to DSR events was examined, and the refrigerators responded instantly after receiving the DSR signal.

5 REINFORCEMENT LEARNING APPROACHES FOR THE SCHEDULED OPERATION OF TCLS

5.1 Introduction

Emerging technologies like Artificial Intelligence (AI) and home appliances are increasingly shaping the future of the world. By employing machine learning techniques (such as deep Reinforcement Learning), homes can become more energy-efficient while improving occupant comfort and reducing energy consumption. To accomplish this, this chapter proposes Reinforcement Learning (RL) approaches for the scheduled operation of the aggregated domestic refrigerators. Initially, a hysteresis band controller is designed for a single refrigerator using Q and Deep Q Networks (DQNs), then a new DQN model is developed for jointly scheduling the operation of multiple refrigerators.

The effectiveness of the scheduling approach is analysed through the designed environment for three different domestic refrigerators using OpenAI Gym.

5.2 Introduction to Reinforcement Learning

Scheduling the cooperation of refrigerators requires a controller that can make decisions. The RL method in ML is used to control and schedule refrigerators based on the interaction between agent and environment, so the RL method is used to control and schedule refrigerators. Table 2.2 shows the examples of the three different ML categories (chapter 2). The RL method can learn over time and has an interaction with the environment. So, RL can adapt itself to changes in the environment, but MPC has a fixed objective function and constraints and does not change with time. RL, on the other hand, decides based on the trained network, which has a faster response than MPC, since optimization calculations are performed in each sample time. The RL is a paradigm within Machine Learning (ML) whereby a self-learning agent learns some type of interaction between it and the environment [118]. In a RL task (see Figure 5.1):

1. The agent wants to achieve a **goal** within the environment.
2. In each time step, **action** is performed by the agent.
3. This action changes the **state** of the environment.
4. Based on the success, the agent will receive a certain **reward**.

Therefore, to start with a RL problem, these four components should be clearly defined:

1. Goal
2. State space
3. Action space
4. Reward function

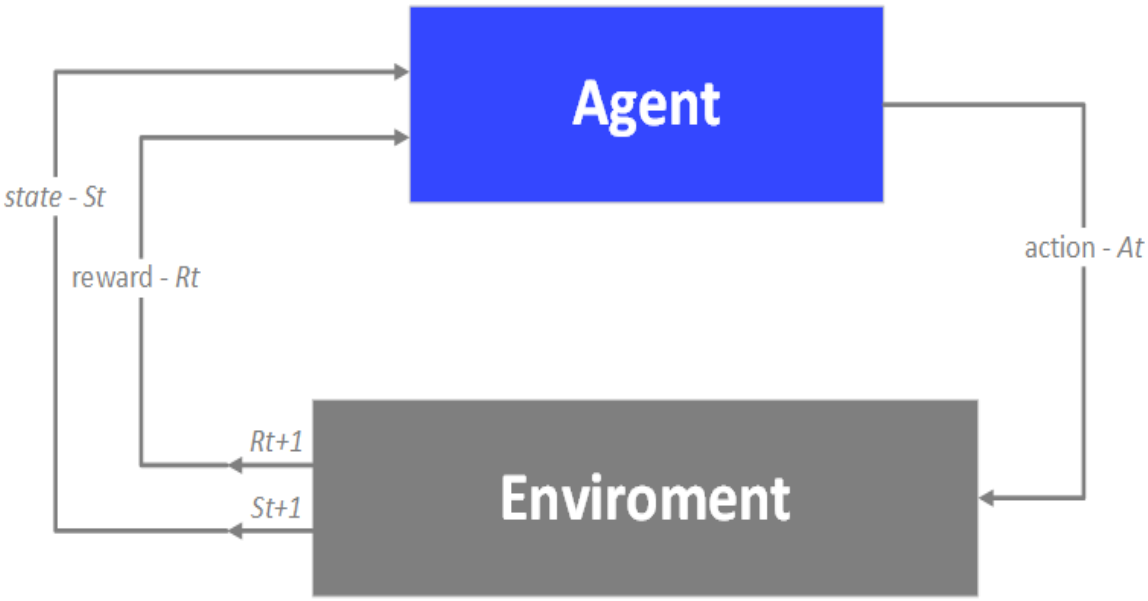


Figure 5.1: Basic RL structure

5.3 Markov Decision Process and Q-Learning

In the Markov Decision Process (MDP) each state within an environment is a consequence of its previous state which in turn is a result of its previous state. Therefore, for sequential decision problems, the MDP provides a formalized description. The MDP is characterized by its S: state-space, A: action-space, transition probabilities P: $S_t \times A_t \times S_{t+1}$, and a reward function R: $S_t \times A_t$. This framework provides a formal description of the problem in which an agent seeks an optimal policy. Policy maps the current states of the environment to the best action that the agent can take in a particular environment state [119]. The policy is learned implicitly in the form of state-action-values. Using Bellman's equation, Q-values are updated in Q-Learning [120]:

$$Q^{new}(s_t, a_t) = (1 - \alpha) \times Q^{old}(s_t, a_t) + \alpha \times (r_t(s_t, a_t) + \gamma \times \max(Q(s_{t+1}, a))) \qquad (5.1)$$

where the learning-rate α represents the sensitivity against new experiences, the discount factor γ determines the weighting between immediate and future rewards, $r_t(s_t, a_t)$ represents the reward and $\max\big(Q(s_{t+1}, a)\big)$ estimates the optimal future value. All these Q-values are stored in a Q-table with the rows for states and the columns for actions —Table 5.1 provides an example.

Table 5.1 The example of Q-table structure

Q-Table		Actions				
		Action 1	Action 2	...	Action n-1	Action n
States	State 1	0.789112	0.745642	...	0.212485	0.256545
	State 2	5.123455	5.11565	...	5.156545	4.155612
	...	...	...	...	...	...
	State n-1	2.156454	2.15567	...	2.144423	2.454658
	State n	6.156212	6.154556	...	6.145441	6.444444

For clarity, here is a breakdown of the Q-Learning steps —see Figure 5.2:

1. Initialize the table with random Q-values.
2. Pick the action a_t, from the set of actions defined for that state s_t using $\max\big(Q(s_t, a_t)\big)$.
3. Perform action a_t.
4. Observe reward $r_t(s_t, a_t)$ and the next state s_{t+1}.
5. Select the a_{t+1}using $\max\big(Q(s_{t+1}, a_{t+1})\big)$ for the state s_{t+1}.
6. Update the value for the state using the equation (5.1).
7. Repeat steps 2-6 for each episode.

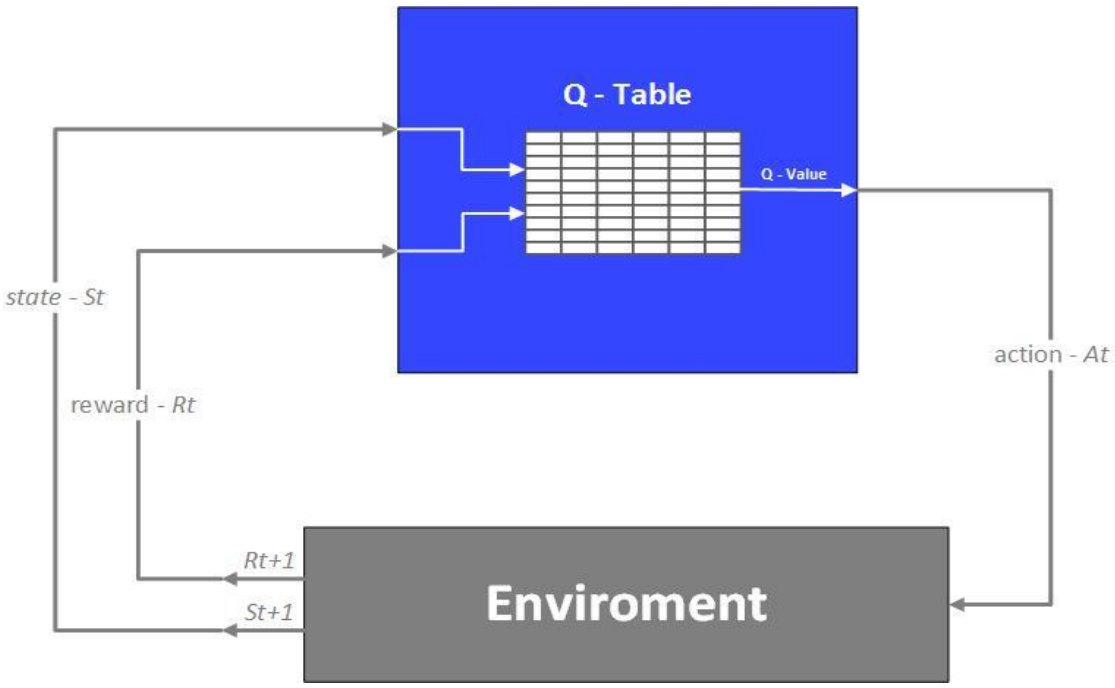


Figure 5.2: The Q-learning structure

A drawback of this type of learning is that it can converge towards scenarios which may not offer the best solution. Therefore, the parameter epsilon (ε) is used to explore new actions and perhaps provide improved 'exploration' or adopt a previously learned agent, termed exploitation.

5.4 Deep Q-Networks

In complex environments, the number of states and actions can significantly increase, and in such circumstances it becomes unrealistic to infer the Q-value of new states from already explored states. Moreover, the amount of memory required to save and update the table also increases. Thus, Q-values can be approximated using machine learning models such as artificial neural networks. The schematic structure of a Deep Q-Network for state-action-value approximation is shown in Figure 5.3. The input and output layers

contain state vector and related Q-value for each possible action, respectively, while there are a certain number of hidden layers between the input and output layers [121].

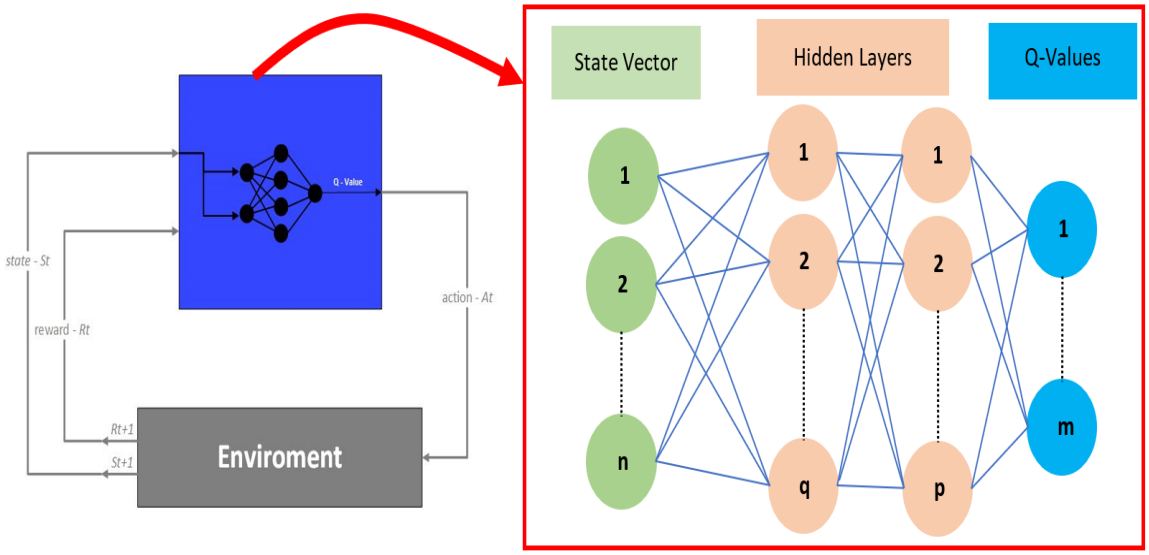


Figure 5.3: The Deep Q-Network structure

5.5 Hysteresis Band Control Using the RL

In this section, the Q-Learning and DQN are tested on a single refrigerator to schedule on/off control for various temperature hysteresis bands.

5.5.1 Implementation of Q-Learning

Q-Learning is realised with a fixed sampling period of *1 min* on the VonShef refrigerator model proposed in equation (3.7). The door remains closed for the duration of the training. The detailed parameters of the Q-Learning are shown in Table 5.2. In simulations, it is assumed that the ambient temperature is controlled between 21 °C and 22 °C and can fluctuate within this range and the temperature bands are taken from reference [127].

Table 5.2 Q-Learning structure for the VonShef

Parameter	Value
Upper band	2.6 (^oC)
Lower band	1.4 (^oC)
a fridge	0.98
b fridge	0.0032
c fridge	0.004
Operation power	51 *W*
Min-on time	8 *min*
Min-off time	14 *min*
Number of training episodes	100000
Each episode length	200 *min*
epsilon (ε)	0.5
Learning rate	0.1
Discount factor	0.95

The controller turns on/off the refrigerator based on a lower and upper bound set points therefore, the action space is as follows:

$$Actions = \{ON = 1,\ OFF = 0\} \tag{5.2}$$

States are used to describe the environment at different points in time. In this model, a state is defined as a set of five values: setpoint temperature, the internal temperature, ambient temperature, and minimum on and off times—see equation (5.3).

$$States = \left\{T_{set}, T_{in}, T_{amb}, min_{on}, min_{off}\right\} \tag{5.3}$$

The main goal of this RL algorithm is to control the internal temperature within the defined upper and lower bands without turning the refrigerator on/off too frequently. The agent will receive a high penalty when it chooses the wrong action based on min_{on} *or* min_{off} values. Conversely, if the internal temperature hits the upper or lower bands, the agent will receive a large reward, otherwise, the difference between internal temperature and setpoint temperature will be considered as a reward. The reward function is given by equation (5.4). The important aspect of choosing the numbers is the difference between the penalty and reward should be large, since a large difference helps the agent to learn the right and wrong actions during the training. For this study, four different penalty and reward pairs are chosen to show the agent can learn using any

arbitrary number with a large difference between reward and penalty eg. (-300, 200), (-100, 35), (-100, 500) and (-4000, 4000) are chosen for the penalty and reward pairs, respectively. Any other arbitrary numbers can be chosen.

$$Reward = \begin{cases} -300, -100, -100, -4000 & min_{on} > 0 \,\&\, action = 0 \\ -300, -100, -100, -4000 & min_{off} > 0 \,\&\, action = 1 \\ 200, 35, 500, 4000 & T_{set} = upband \quad T_{in} \geq upband \\ 200, 35, 500, 4000 & T_{set} = lowband \quad T_{in} \leq lowband \\ -(T_{in} - T_{set})^2 & Otherwise \end{cases} \tag{5.4}$$

The episode will be finished when the episode-length (*200 min*) is 0 or the current reward is $-300, -100, -100, -4000$ —see equation (5.5).

$$Episode\ is\ done = \begin{cases} Episode\ length = 0 \\ reward = -300, -100, -100, -4000 \end{cases} \tag{5.5}$$

Details of the proposed Q-learning algorithm are given in Algorithm 1. Initially, the refrigerator parameters are randomly selected based on the data given in table 5.2, and the Q-table is randomly initialized, as shown in lines 1–2. Starting from line 3, for each iteration, the system state is first initialized, then the on/off action is chosen based on the epsilon, as shown in lines 7-11. Next, in lines 12–14, the selected action is executed in the environment for the entire control interval, and the received reward and the next state are observed. The maximum future Q and current Q values are calculated, and the Q-table is updated based on the rewards and (5.1), as shown in lines 15-24. When the current reward is -300, the current episode is finished, and the agent will start the next episode. Finally, after finishing all the episodes, the trained Q-table is stored to be further used for tests. Full python code is provided in Appendix 4.

Algorithm 1: Q-Learning method for hysteresis bands controller

```
Initialize the parameters of the fridge and Q-table using table 5.2
Initialize the Q-table with random numbers
  for episode = 1 to Number of training episodes, do
    Initialize system states (T_set, T_in, T_amb, min_on, min_off) with random numbers
    for step = 1 to max steps, do
      Get the observation space (T_set, T_in, T_amb, min_on, min_off)
      if random number > epsilon, do
        Get the action using argmax (Q-table [observation space])
      else, do
        Get the random action [0,1]
      end if
      Apply the action and calculate the new internal temperature using equation (3.7),
      calculate the new min_on, min_off, ambient temperature and setpoint temperature
      Calculate the reward using equation (5.4)
      Get the new observation space
      Calculate the max Q value for this new observation using
      Max-future-Q = argmax (Q-table [new observation])
      Calculate the current Q for the chosen action using
      Current-Q = Q-table[observation][action]
      Update the Q-table using:
      if reward == setpoint-reward (200), do
        new-Q = setpoint-reward (200)
      elseif reward == large-penalty (-300), do
        new-Q = large-penalty (-300)
      else, do
        Calculate the new-Q using equation (5.1)
      end if
      if reward == large-penalty (-300), do
        break
      end if
    end for
  end for
Save the trained Q-table
```

The Q-table is created after 100,000 training episodes in this case. Figures 5.4, 5.5, 5.6 and 5.7 show the average reward values during the training for the various penalty-reward pairs. It can be seen that the reward starts from negative numbers during the first episodes. In between, there is a consistent trend of greater rewards as the episodes increase until 70,000, then tends to be relatively constant thereafter, indicating that the agent (Q-table) is well trained.

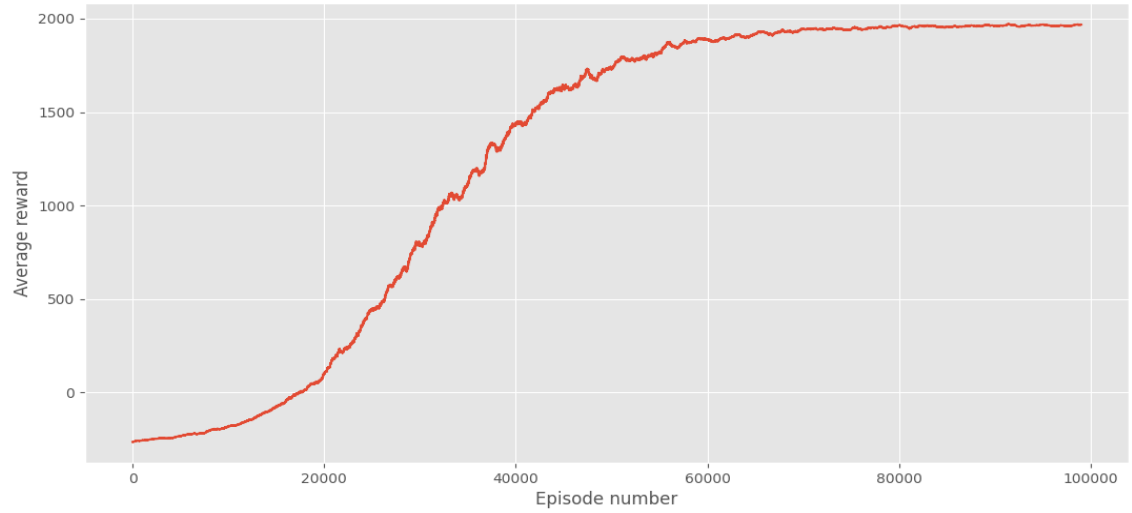


Figure 5.4: The average reward values during the Q-table training for penalty-reward pair of (-300, 200)

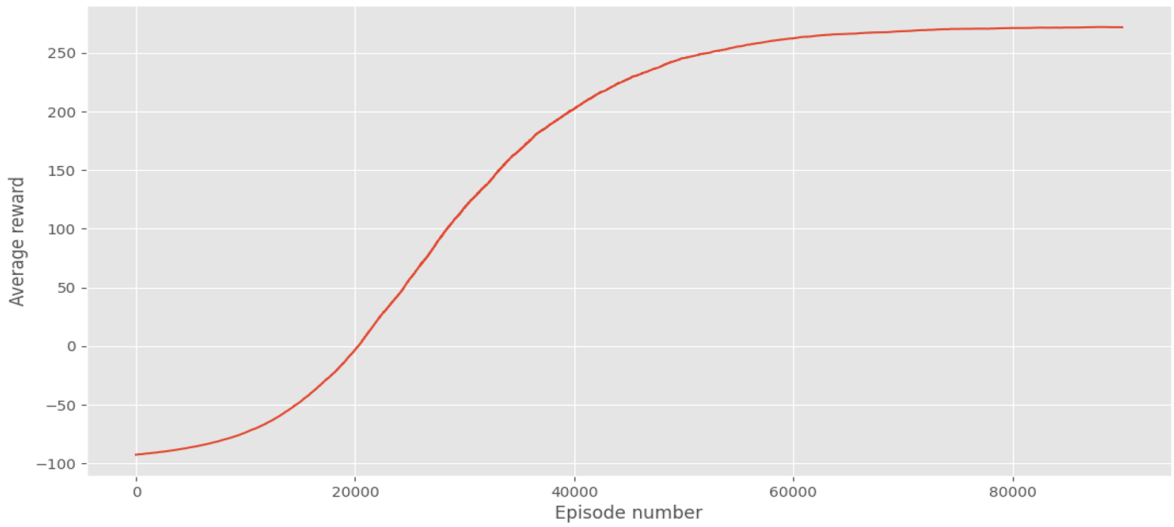


Figure 5.5: The average reward values during the Q-table training for penalty-reward pair of (-100, 35)

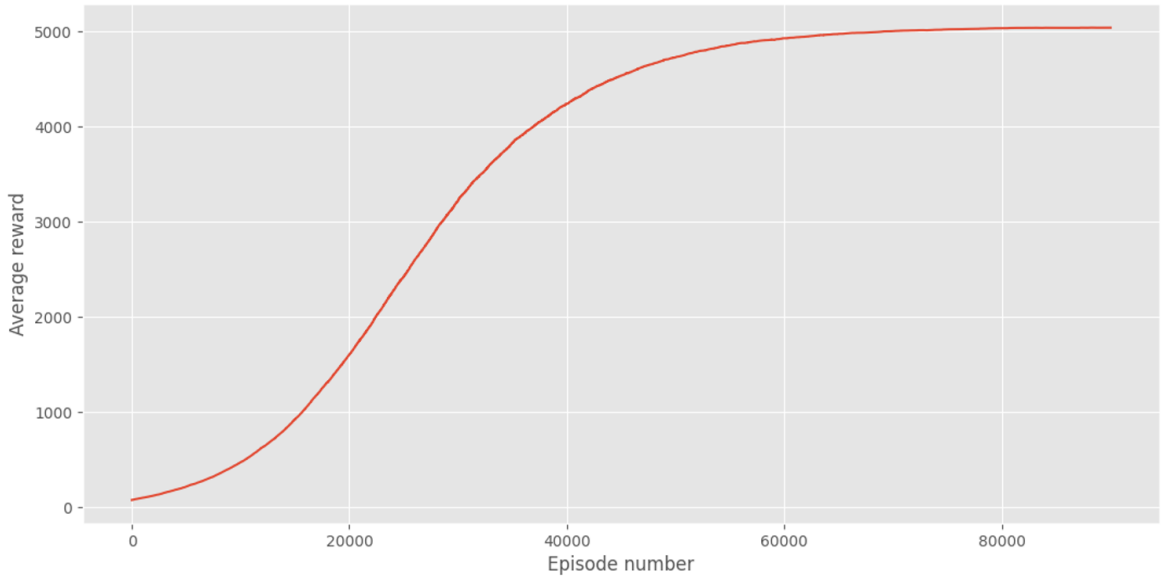


Figure 5.6: The average reward values during the Q-table training for penalty-reward pair of (-100, 500)

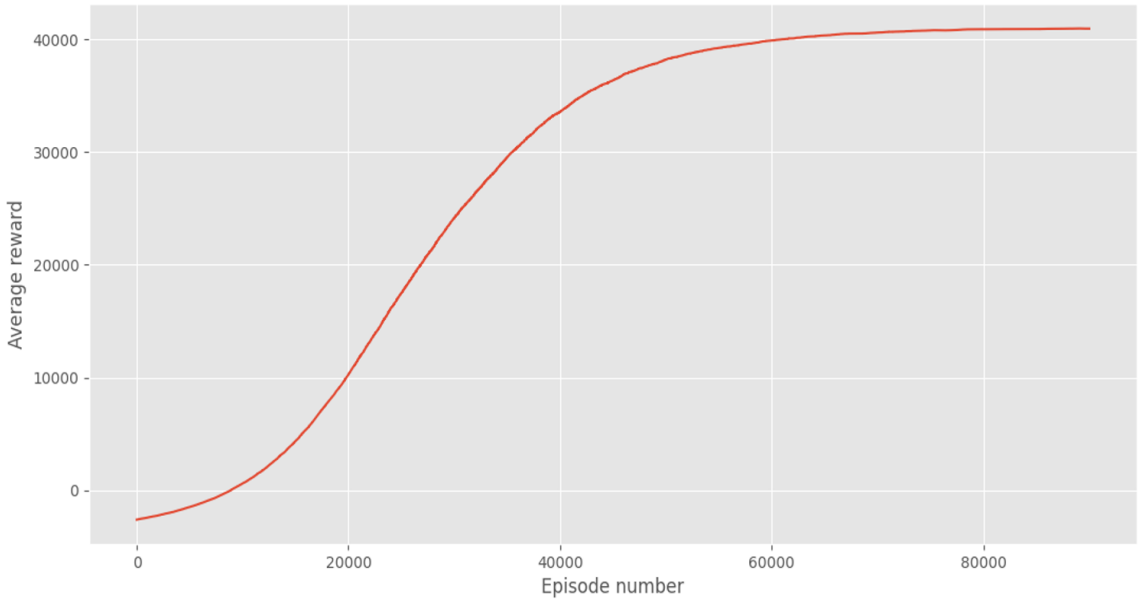


Figure 5.7: The average reward values during the Q-table training for penalty-reward pair of (-4000, 4000)

The trained Q-table is tested to schedule the on/off states to control the internal temperature between the defined setpoints (1.4, 2.6). Figures 5.8, 5.9, 5.10 and 5.11 show the refrigerator's internal temperature, the ambient temperature and the on/off states using the trained table for different penalty-reward pairs. As can been seen the agent is able to schedule the refrigerator compressor to maintain its respective temperature within the specified hysteresis bands whilst considering the number of compressor starts per hour.

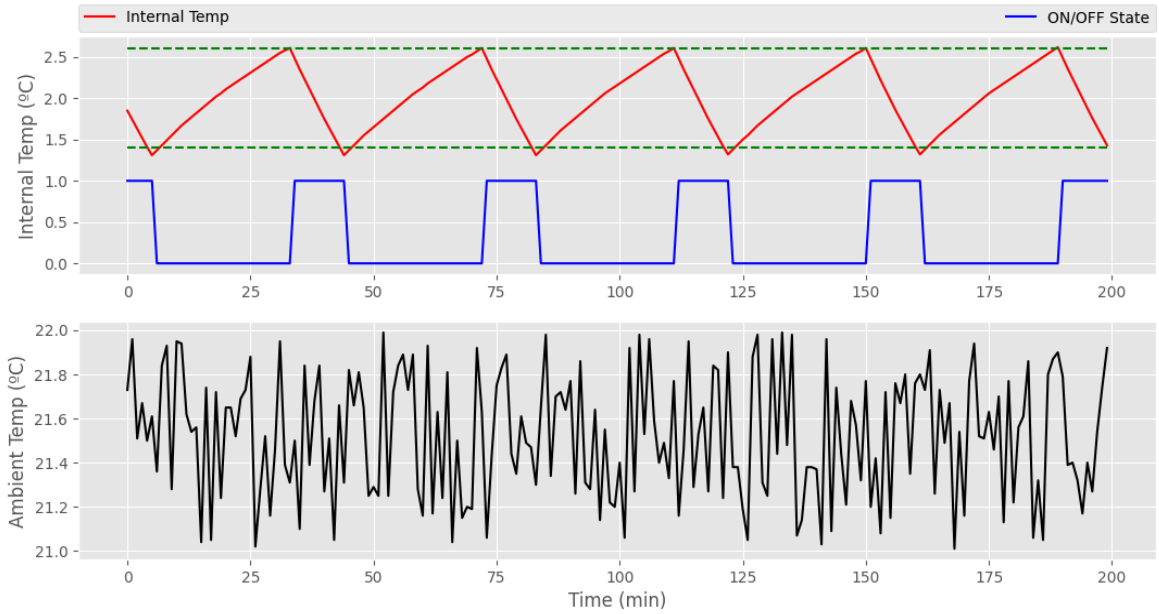


Figure 5.8: Refrigerator's internal temperature, ambient temperature, and on/off state for Q-Learning for penalty-reward pair of (-300, 200)

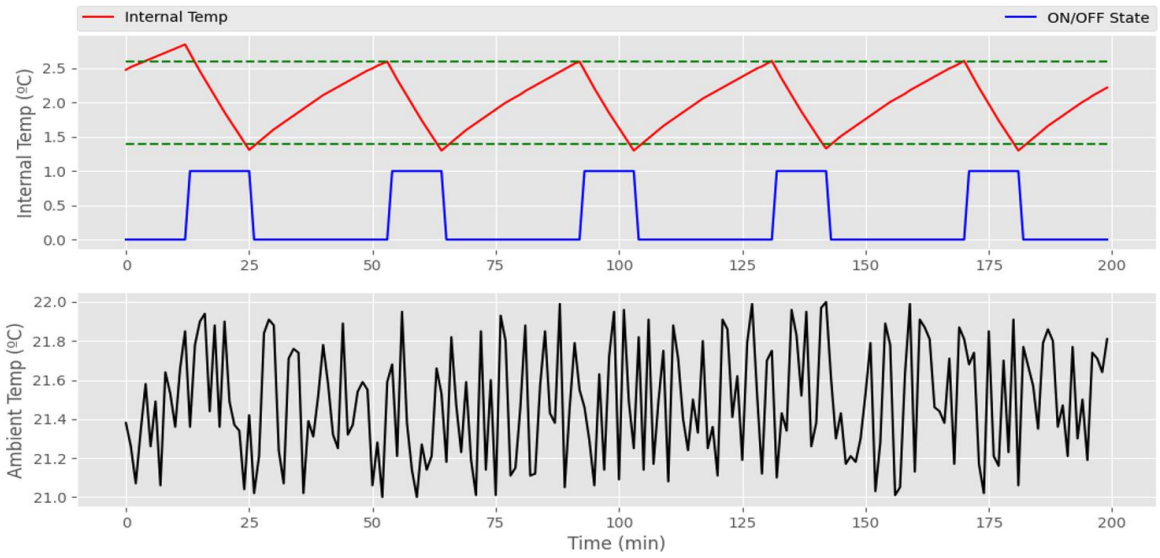


Figure 5.9: Refrigerator's internal temperature, ambient temperature, and on/off state for Q-Learning for penalty-reward pair of (-100, 35)

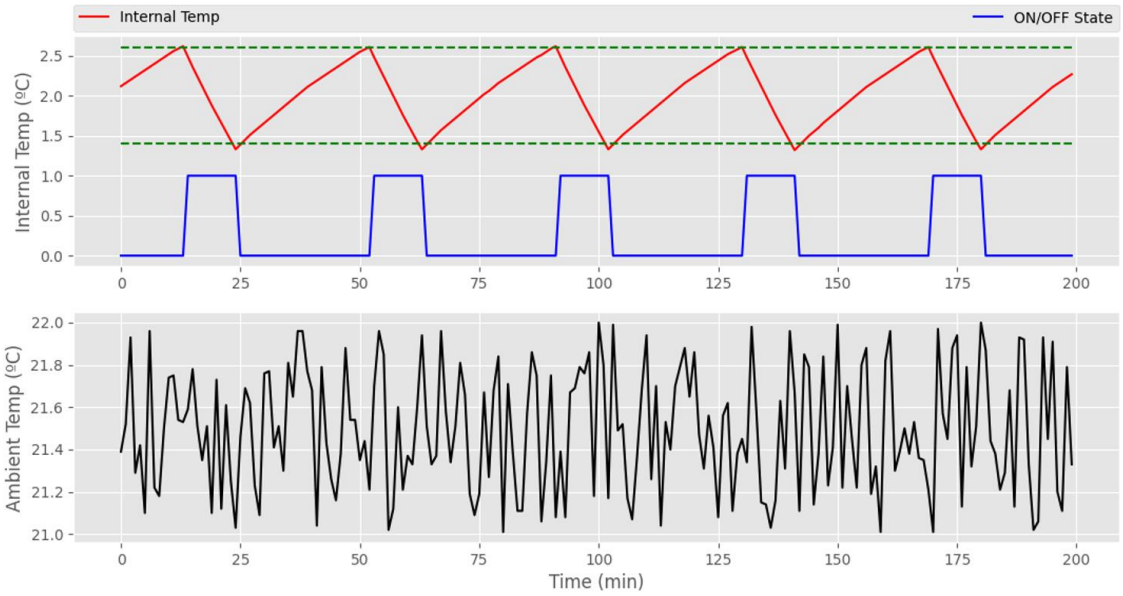


Figure 5.10: Refrigerator's internal temperature, ambient temperature, and on/off state for Q-Learning for penalty-reward pair of (-100, 500)

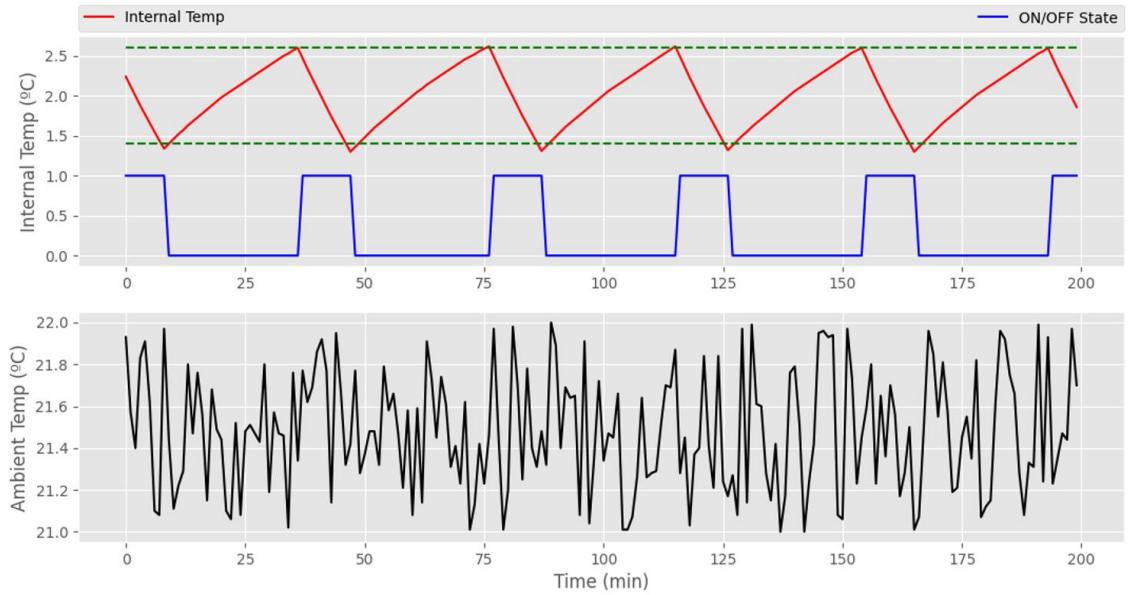


Figure 5.11: Refrigerator's internal temperature, ambient temperature, and on/off state for Q-Learning for penalty-reward pair of (-4000, 4000)

5.5.2 Implementation of DQN

An agent is trained using DQN to control the VonShef internal temperature. The agent environment is designed using the OpenAI Gym [122]. The implementation of the DQN algorithm is carried out using the Python library Tensorflow and Keras-RL [123]. Table 5.3 summarizes the parameters used in DQN. All other parameters, action space, state space and reward function are the same as they were for previous Q-Learning case. To build a good DQN, the settings are tried from the simplest settings, layers, networks until the network is properly trained and the suggestions provided in [94]-[103] the references are also used.

Table 5.3 DQN structure for the VonShef

Parameter	Value
Input layer	States (5)
Number of hidden layers	2
Number of neurons in each hidden layer	24
Output layer	Actions (2)
Activation function	Rectified Linear Unit (ReLU)
Number of training episodes	20000
Each episode length	500 *min*
Learning rate	0.001
Optimiser	Adam
Loss	Mean Square Error (MSE)

Algorithm 2 and figure 5.12 outline the pseudocode for the proposed DQN. After initializing the parameters of the refrigerator (line 1), the custom fridge environment is created using the OpenAI Gym environment. To create an environment with OpenAI Gym, the following functions should be defined:

1. init: Initialize the system states and get the first observation space, as shown in lines 4-7.
2. step: Apply the selected action to the environment and calculate the new observation space, reward, and episode status, as shown in lines 8-12.
3. reset: Reset the environment, episode rewards and the observation space, as shown in lines 13-15.
4. render: add graphical options to the environment such as plots.

Next, in lines 18-30, the deep neural network model and the DQN agent are defined based on the data given in table 5.3. Finally, the DQN agent is trained and stored in lines 31-32. The full python code for the proposed DQN can be found in Appendix 5.

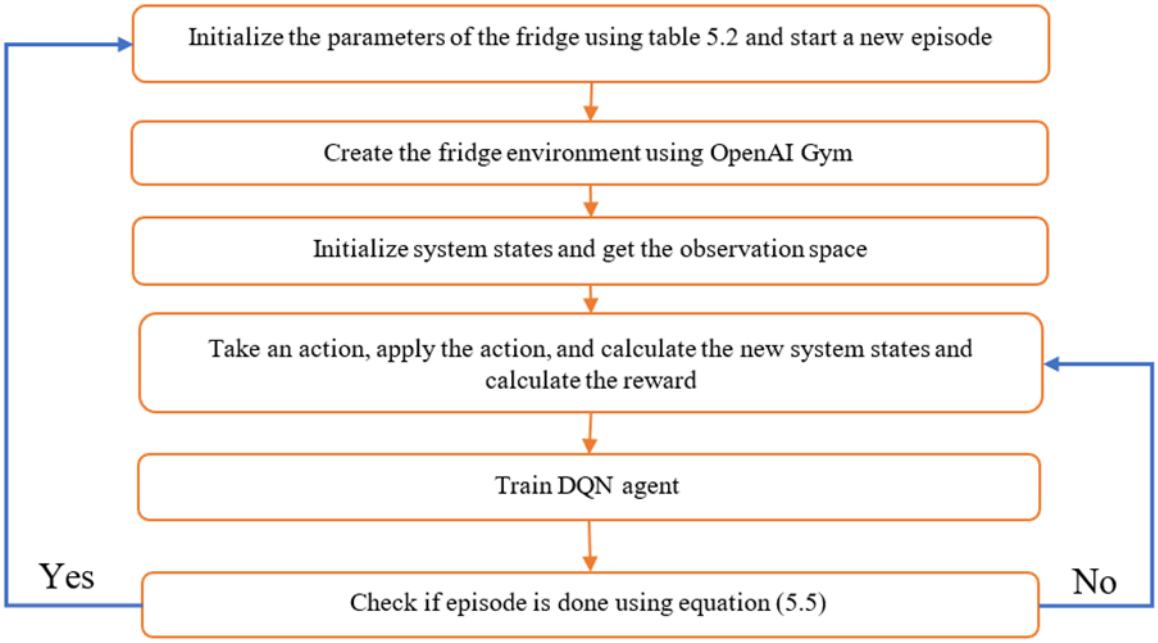


Figure 5.12: DQN flowchart for hysteresis bands controller

Algorithm 2: DQN method for hysteresis bands controller

1: Initialize the parameters of the fridge using table 5.2
2: Create the fridge environment using OpenAI Gym (init, step, render and reset functions)
3: **Class** fridge-environment
4: **def** init:
5: Initialize system states ($T_{set}, T_{in}, T_{amb}, min_{on}, min_{off}$) with random numbers
6: Get the observation space ($T_{set}, T_{in}, T_{amb}, min_{on}, min_{off}$)
7: **end def**
8: def step:
9: Apply the action and calculate the new internal temperature using equation (3.7), calculate the new min_{on}, min_{off}, ambient temperature and setpoint temperature
10: Calculate the reward using equation (5.4)
11: Check if episode is done using equation (5.5)
12: **end def**
13: **def** rest:
14: Reset the states ($T_{set}, T_{in}, T_{amb}, min_{on}, min_{off}$)
15: **end def**
16: end Class
17: Create the deep neural network model
18: **def** deep model:
19: model.add (input layer = states)
20: model.add (hidden layer 1, activation function)
21: model.add (hidden layer 2, activation function)
22: model.add (output layer = actions)
23: end def
24: Build the DQN agent
25: **def** DQN:
26: Initialize the parameters of the DQN using table 5.3
27: policy = BoltzmannQPolicy
28: model = deep model
29: environment = fridge-environment
30: end def
31: DQN.train
32: Save the trained DQN

Tests are carried out on the trained agent for three different setpoints: (1.4–2.6 ^{o}C), (1.5–4 ^{o}C) and (2–3.5 ^{o}C). Figures 5.13, 5.14 and 5.15 show the refrigerator's internal temperature, the ambient temperature, the on/off state, and step reward value under three different upper and lower bands. By comparison with the results from Q-table, here it can be seen that the relative number of training episodes has reduced from 100,000 to 20,000. Moreover, the DQN model is able to control the internal temperature for different setpoint other than the trained setpoint (1.4–2.6 ^{o}C), whereas the Q-table counterpart only works for the trained setpoint.

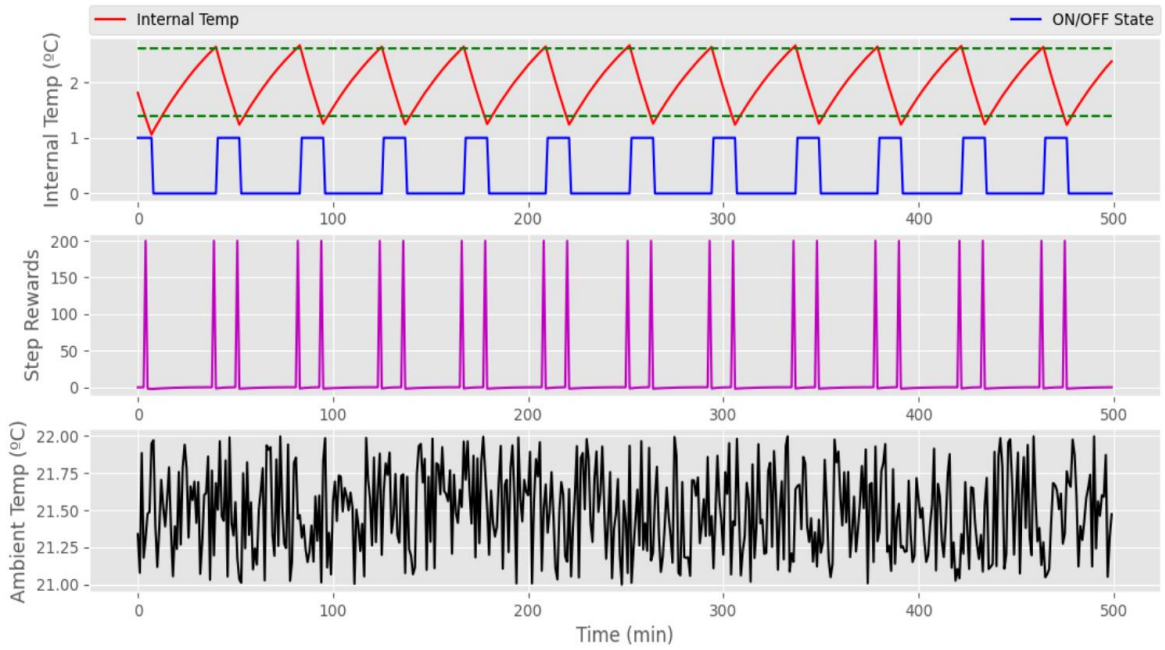


Figure 5.13: Refrigerator's internal temperature, ambient temperature, on/off state and step reward for setpoint (1.4–2.6 ºC) under DQN scheme

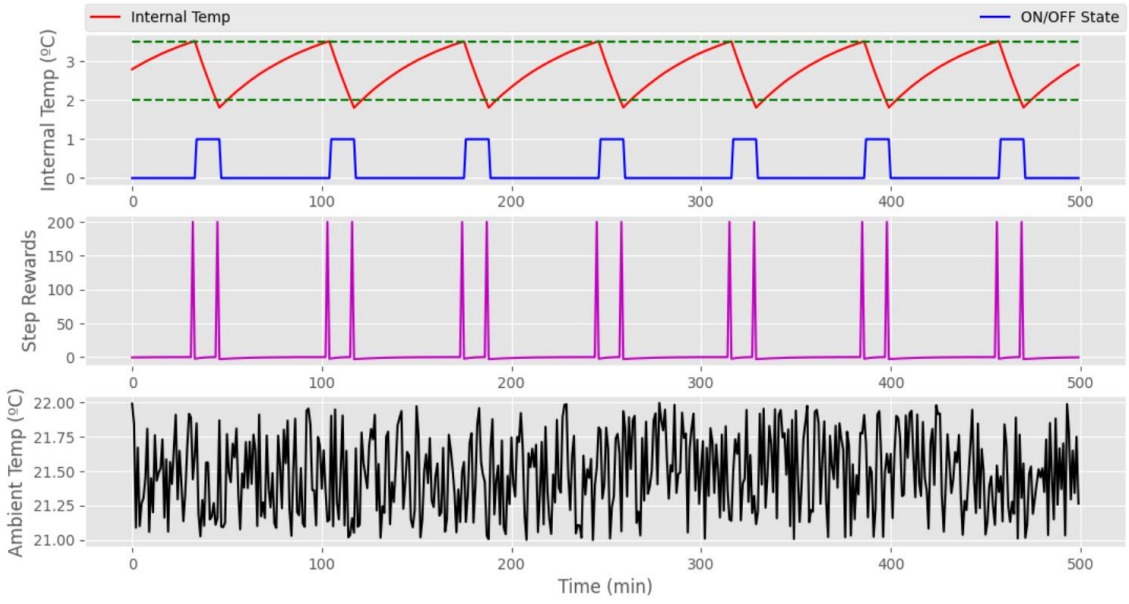


Figure 5.14: Refrigerator's internal temperature, ambient temperature, on/off state and step reward for setpoint (2–3.5 ºC) under DQN scheme

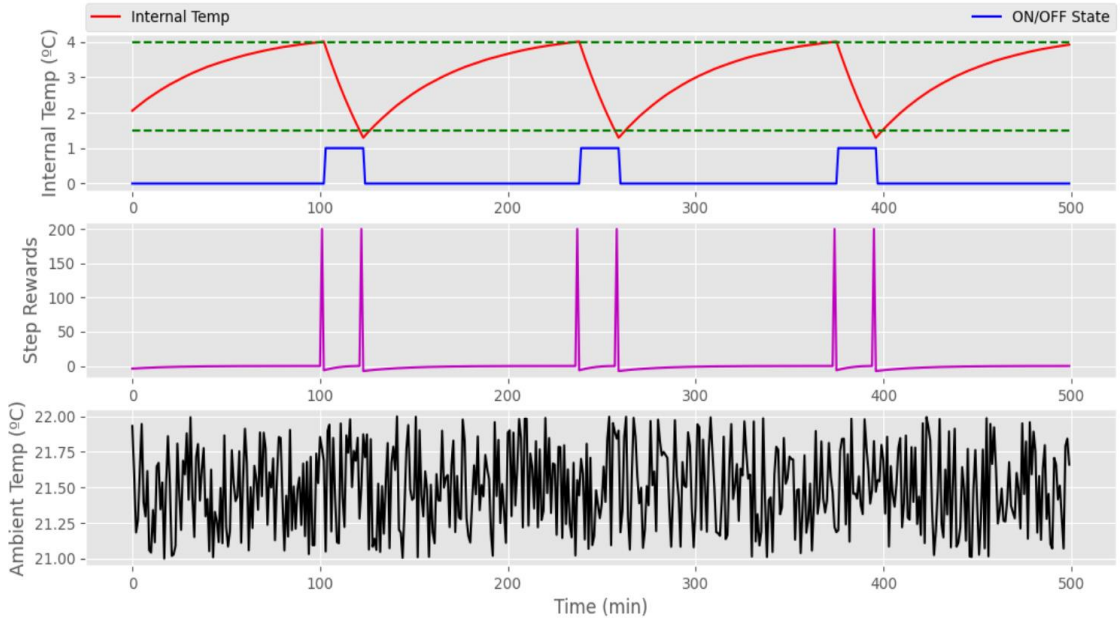


Figure 5.15: Refrigerator's internal temperature, ambient temperature, on/off state and step reward for setpoint (1.5–4 ºC) under DQN scheme

5.6 Multi-Refrigerator Systems and DSR Using DQN

This study investigates the use of the previously presented DQN to respond to DSR events in domestic refrigerators. Table 5.4 summarise the data for three different domestic refrigerators models that are designed using the OpenAI Gym environment.

Table 5.4 Data for the designed three domestic refrigerators

Fridge Number	Upper band (*ºC*)	Lower band (*ºC*)	Min-on time (*min*)	Min-off time (*min*)	Parameter *a*	Parameter *b*	Parameter *c*	Operation power (*W*)
VonShef	2.6	1.4	6	12	0.98	0.0032	0.004	51
iGenix	4.5	2.5	5	15	0.99	0.0047	0.003	55
Russell Hobbs	5	3	9	10	0.95	0.005	0.006	50

The DQN is used to train the agent using Keras-RL library in Python. In this DSR event, the agent can turn on only one fridge at each time step. Therefore, the action space is a set of four values:

$$Actions = \{OFF\_OFF\_OFF\ (000), ON\_OFF\_OFF\ (100),\ OFF\_ON\_OFF\ (010),\ OFF\_OFF\_ON\ (001)\} \tag{5.5}$$

The state space includes setpoint temperature, internal temperature, ambient temperature, minimum on and off times for each refrigerator:

$$State\ space = \begin{Bmatrix} T_{set1}, T_{in1}, T_{amb1}, min_{on1}, min_{off1}, \\ T_{set2}, T_{in2}, T_{amb2}, min_{on2}, min_{off2}, \\ T_{set3}, T_{in3}, T_{amb3}, min_{on3}, min_{off3} \end{Bmatrix} \tag{5.6}$$

The reward functions for each refrigerator are given by equations (5.7), (5.8) and (5.9).

$$Reward_1 = \begin{cases} -1000 & min_{on1} > 0\ \&\ action_1 = 0 \\ -1000 & min_{off1} > 0\ \&\ action_1 = 1 \\ 200 & T_{set1} = upband_1 \quad T_{in1} \geq upband_1 \\ 200 & T_{set1} = lowband_1 \quad T_{in1} \leq lowband_1 \\ -(T_{in} - T_{set})^2 & Otherwise \end{cases} \tag{5.7}$$

$$Reward_2 = \begin{cases} -1000 & min_{on2} > 0\ \&\ action_2 = 0 \\ -1000 & min_{off2} > 0\ \&\ action_2 = 1 \\ 200 & T_{set2} = upband_2 \quad T_{in2} \geq upband_2 \\ 200 & T_{set2} = lowband_2 \quad T_{in2} \leq lowband_2 \\ -(T_{in} - T_{set})^2 & Otherwise \end{cases} \tag{5.8}$$

$$Reward_3 = \begin{cases} -1000 & min_{on3} > 0\ \&\ action_3 = 0 \\ -1000 & min_{off3} > 0\ \&\ action_3 = 1 \\ 200 & T_{set3} = upband_3 \quad T_{in3} \geq upband_3 \\ 200 & T_{set3} = lowband_3 \quad T_{in3} \leq lowband_3 \\ -(T_{in} - T_{set})^2 & Otherwise \end{cases} \tag{5.9}$$

Finally, the integrated total reward function is given by equation (5.10).

$$Reward = Reward_1 + Reward_2 + Reward_3 \tag{5.10}$$

The structure of the DQN is proposed in Table 5.5.

Table 5.5 DQN structure for the VonShef

Parameter	Value
Input layer	States (15)
Number of hidden layers	2
Number of neurons for hidden layer 1	120
Number of neurons for hidden layer 2	32
Output layer	Actions (4)
Activation function	Gaussian Error Linear Units (GELUs)
Number of training episodes	33000
Each episode length	500 *min*
Learning rate	0.001
Optimiser	Adam
Loss	Mean Square Error (MSE)

The full python code for the proposed DQN structure can be found in Appendix 6.

Tests on refrigerator models now show the performance of the DQN to investigate how the widely distributed domestic refrigerators can respond to DSR events. Tests on refrigerators 1, 2, and 3 include two different sets of setpoint data and the trained setpoint, and the doors remained closed for the duration of the trials. Desired upper and lower temperature setpoints for different tests for each refrigerator are shown in Table 5.6.

Table 5.6 Desired upper and lower temperature setpoints for the tests

	Test 1		Test 2		Test 3	
Fridge Number	Upper band (Trained) (^oC)	Lower band (Trained) (^oC)	Upper band (^oC)	Lower band (^oC)	Upper band (^oC)	Lower band (^oC)
VonShef	2.6	1.4	2.5	1.5	2.5	1.5
iGenix	4.5	2.5	3	2	3	1.5
Russell Hobbs	5	3	4	1.5	6	4

Results for test 1 are given in Figures 5.16, 5.17 and 5.18, for test 2 are given in Figures 5.19, 5.20 and 5.21 and for test 3 are given in Figures 5.22, 5.23 and 5.24. DSR

demanded with the total power usage of 60W lasts for 500 minutes. As can be seen from the measurements, the refrigerators are able to respond (virtually) instantaneously to power shedding events and the peak power consumption is limited to 60W. Moreover, all the refrigerators maintain their temperatures within required bounds during the DSR event.

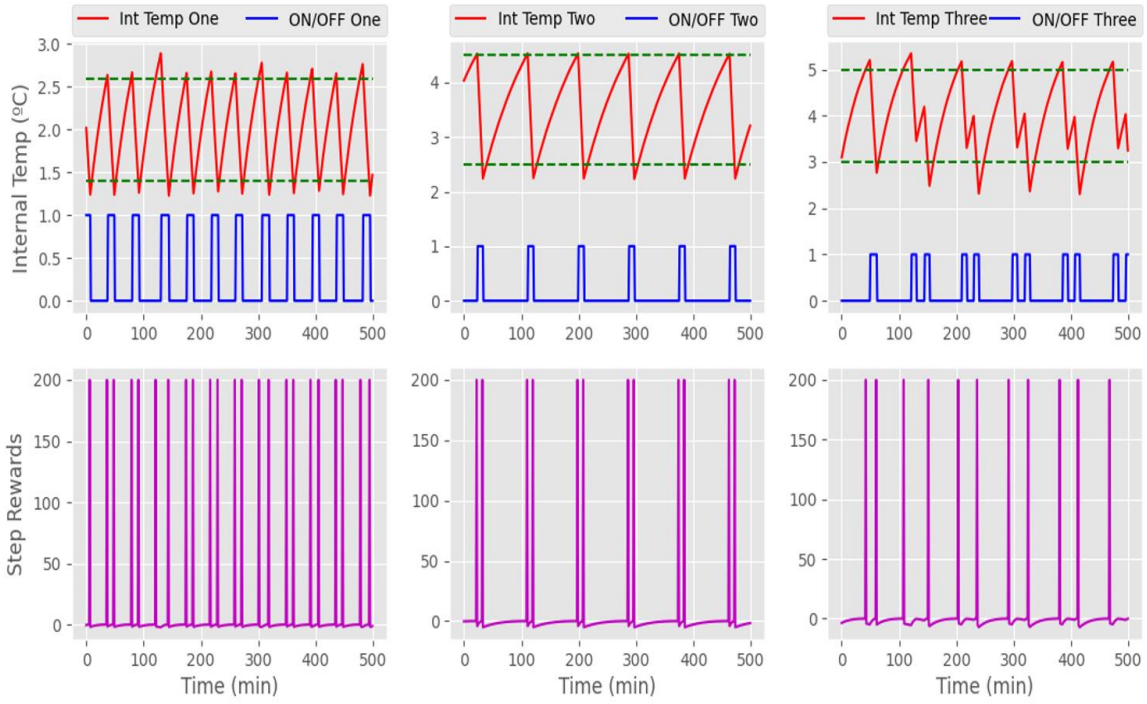


Figure 5.16: Refrigerator's internal temperatures, on/off state and step reward for Test 1

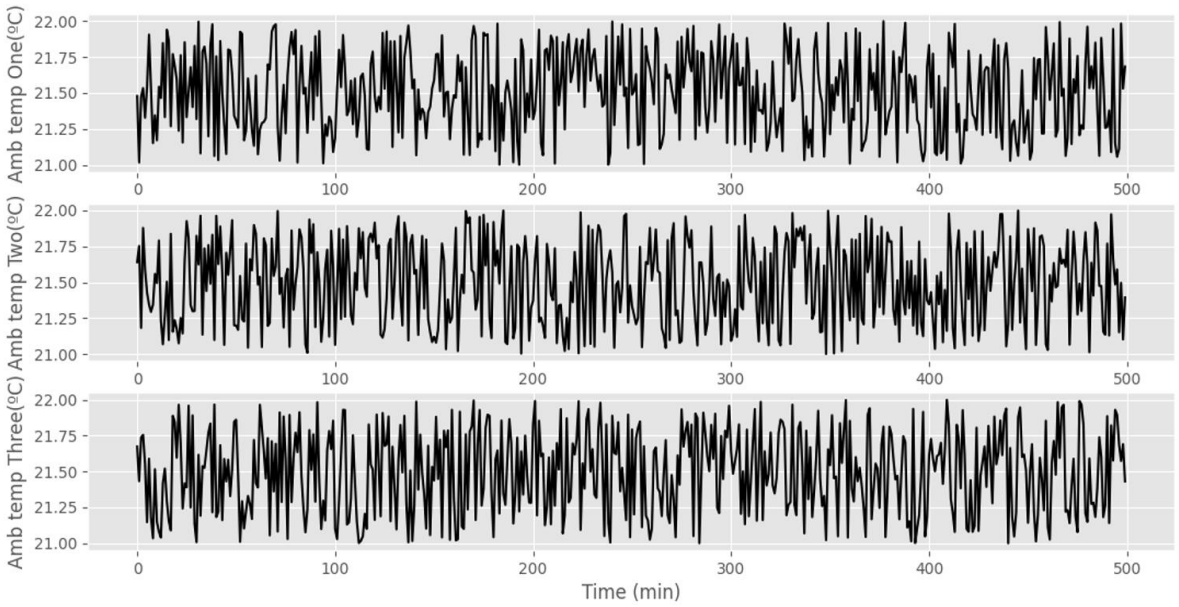


Figure 5.17: Refrigerator's ambient temperatures for Test 1

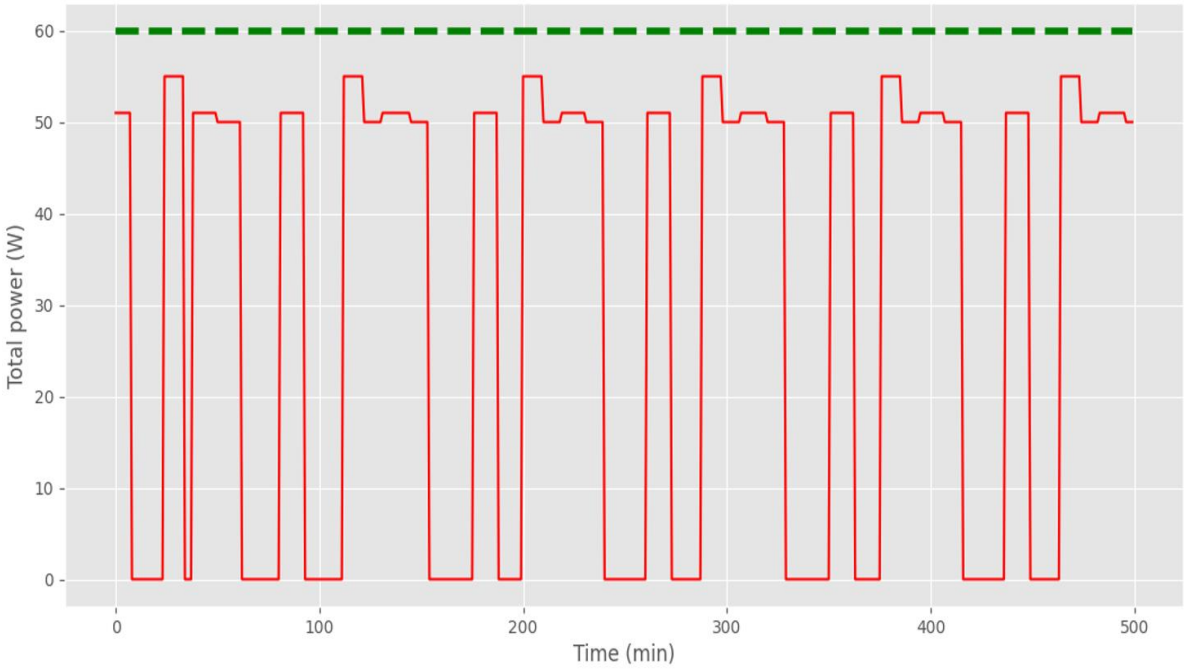


Figure 5.18: Refrigerator's total power usage for Test 1

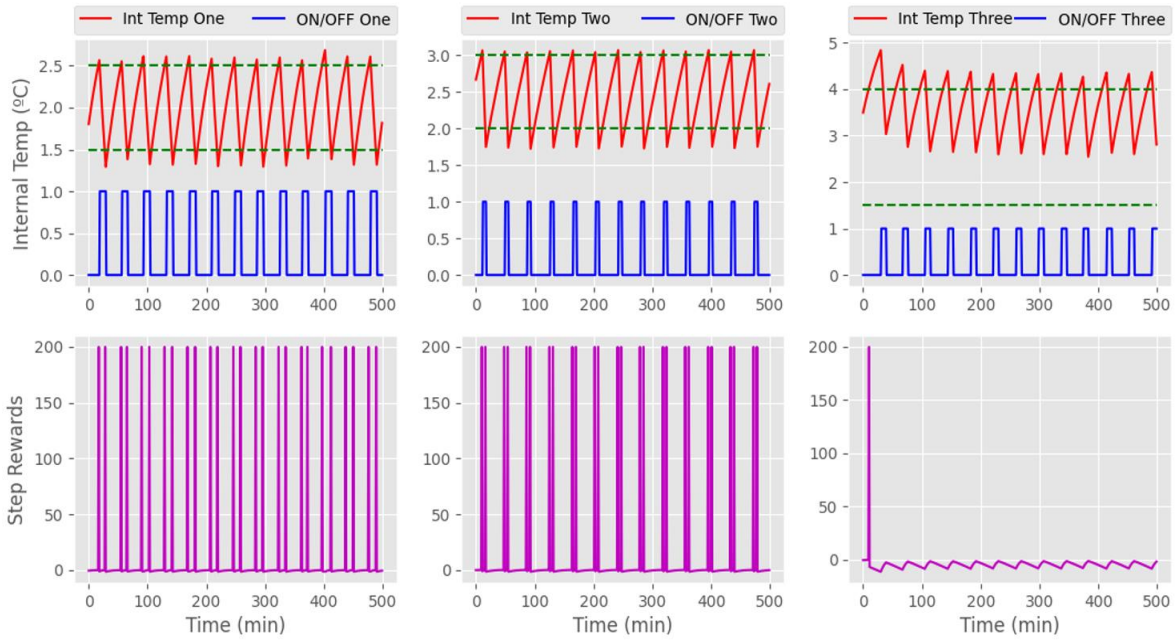


Figure 5.19: Refrigerator's internal temperatures, on/off state and step reward for Test 2

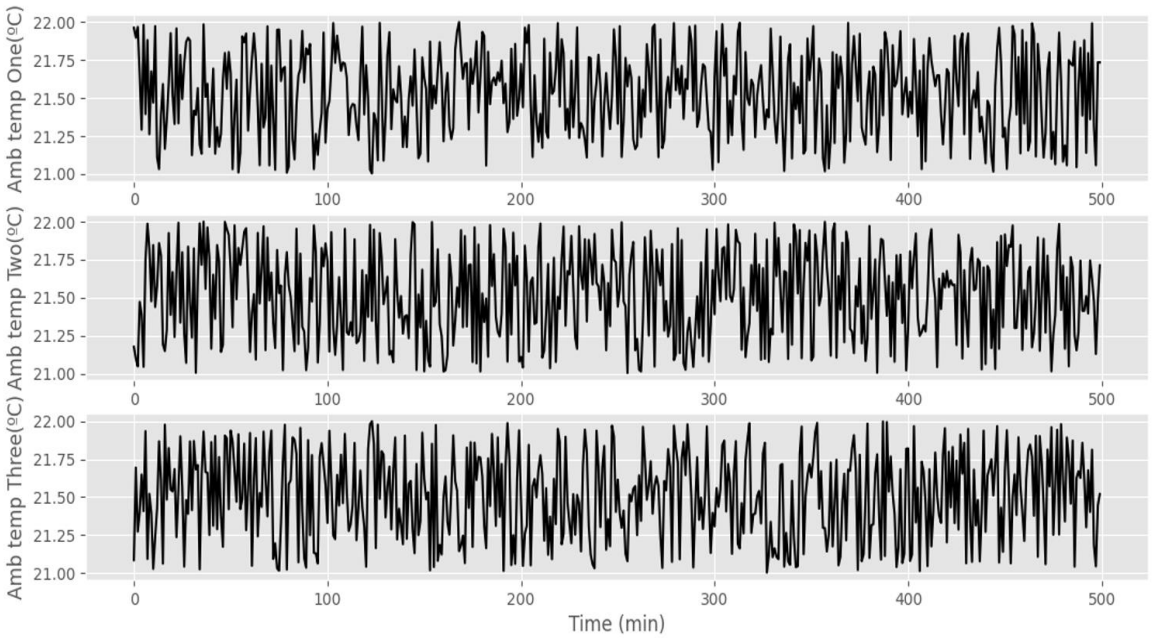


Figure 5.20: Refrigerator's ambient temperatures for Test 2

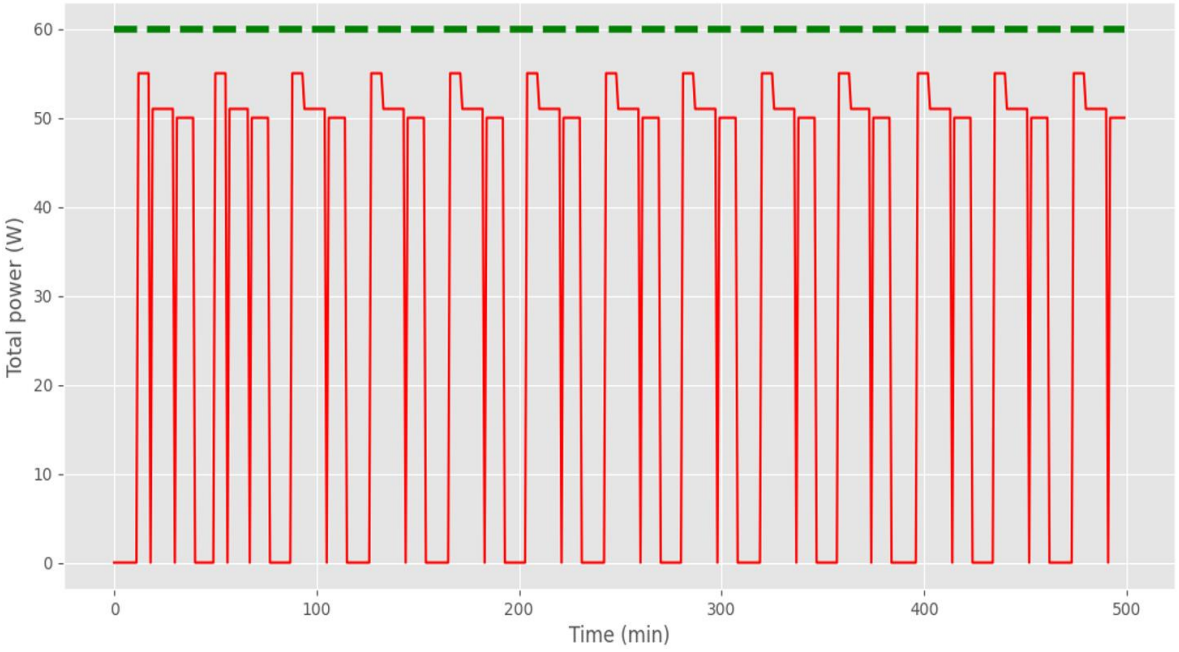


Figure 5.21: Refrigerator's total power usage for Test 2

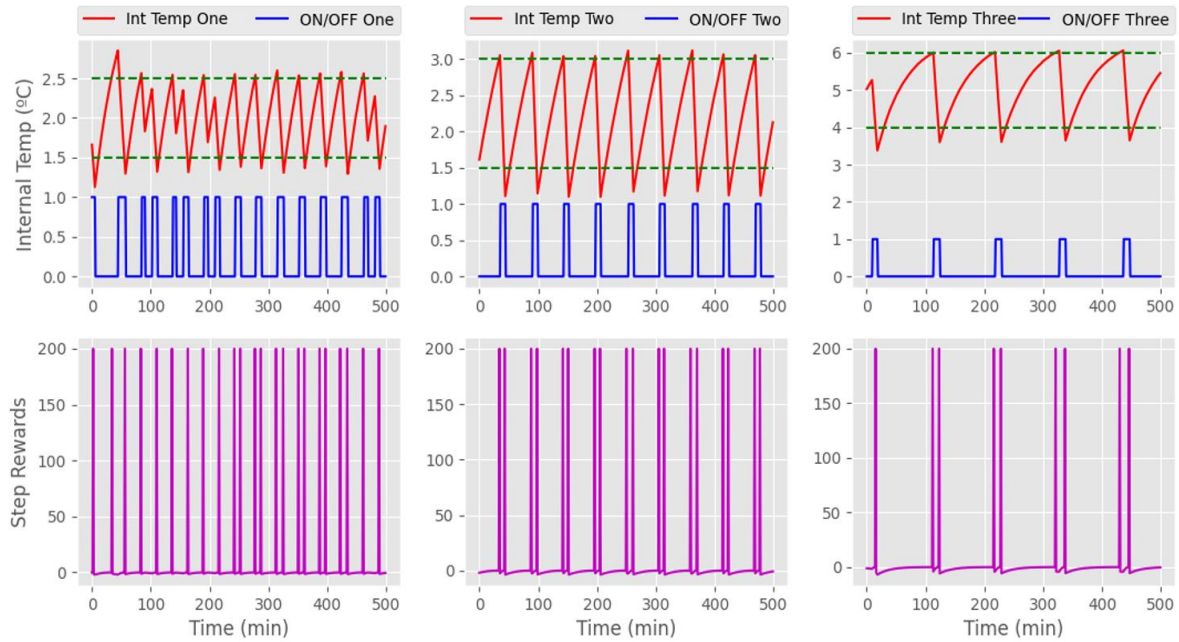


Figure 5.22: Refrigerator's internal temperatures, on/off state and step reward for Test 3

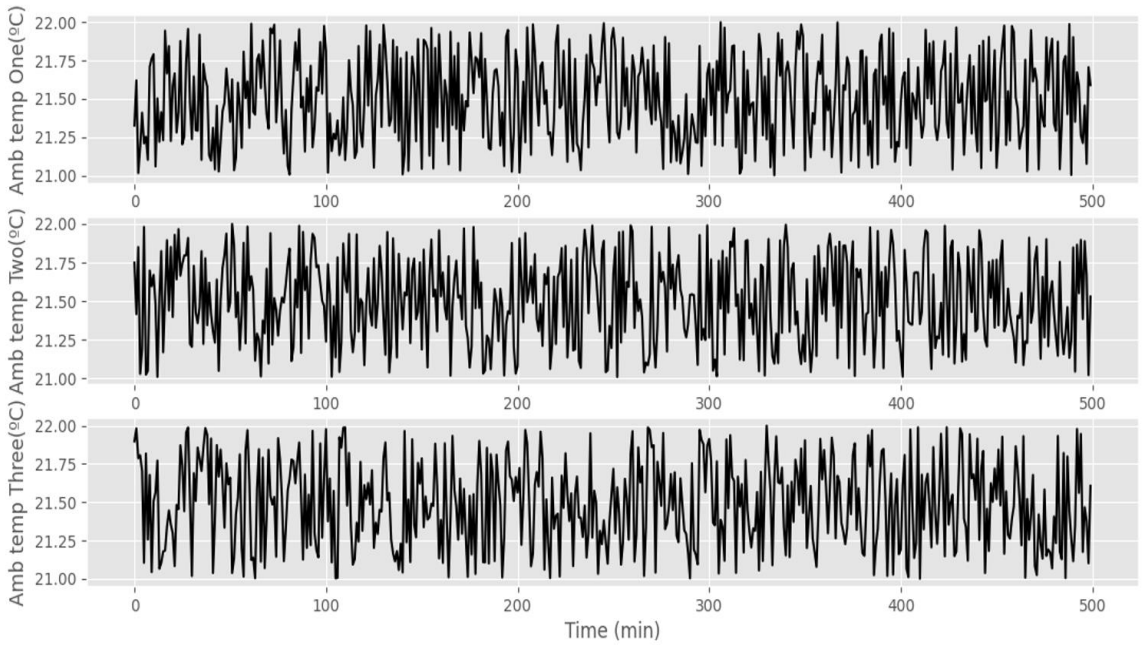


Figure 5.23: Refrigerator's ambient temperatures for Test 3

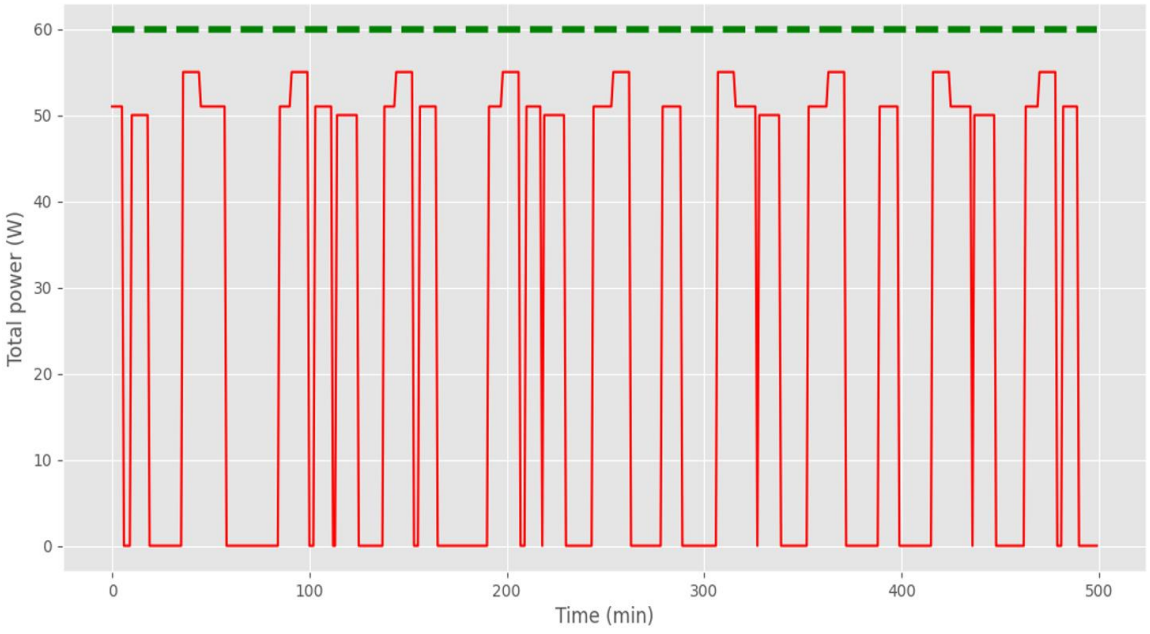


Figure 5.24: Refrigerator's total power usage for Test 3

5.7 Conclusions

In this chapter, the Q-Learning and Deep Q-Network are proposed and implemented for hysteresis band controlling and jointly scheduling the widely distributed domestic refrigerators in DSR events. The Q-table is trained with 2.6 and 1.4 (^{o}C) setpoint temperatures to implement hysteresis control but Q-table only works for the trained setpoint. In order to resolve this problem, the deep neural network is used instead of a tabular model. The custom fridge environment is created using the OpenAI Gym environment and DQN is implemented and tested using Keras-RL and it is shown that the DQN model is able to control the internal temperature for different setpoint other than the trained setpoint. Moreover, the DQN is implemented to respond to DSR events in three domestic refrigerators. Tests are carried out with two different sets of setpoint data and the trained setpoint and the peak power consumption is limited to $60W$ (the agent can only provide power one fridge at each time step due to the strict power constraint). It is shown that the refrigerators are able to respond to power shedding events almost instantaneously. It should be noted that the proposed methodology can be implemented more widely for other house appliances, for instance, boilers, or other TCLs. Since the RL method uses a trained network and optimization calculations are not performed every time step, it can provide an instant response. In contrast to MPC, there are no control and prediction horizon parameters in the RL method, so the refrigerators will return to normal operation immediately after a failure of the Internet connection.

There are some difficulties in experimental work in chapter 5. It was an odder of magnitude more difficult than experimental trails for the other chapters and would investigate in future works. The practical difficulties in the implementation for a practical system compared with previous methods: The previous controller was developed in the MATLAB environment with Arduino, but RL is created in the Python environment, so implementing RL on refrigerators requires a new communication setup

using new hardware (such as Raspberry Pi) or a new programming language such as MicroPython. Additionally, it needs to be figured out how to communicate between sensors and how to receive and send data via smart plugs and the Python environment.

6 CONCLUSIONS AND RECOMMENDATIONS

This thesis has presented new methods for monitoring and jointly scheduling of widely distributed domestic refrigerators using MPC and Reinforcement Learning approaches. Here, we draws together the various elements undertaken in the course of the research. The main conclusions are discussed, with recommendations for future work presented.

6.1 Conclusions

There are four objectives in section 1.3 of this thesis which are summarised:

First, the real-time identification of refrigerator dynamics is proposed, and experimental trials are carried out when the refrigerator is empty, when it contains product (6-litres of water) and with door opening and closing events to show the recursive based system identification strategy is able to accommodate uncertainty e.g., ambient conditions, opening and closing the door and changes in product mass. The results show that the difference between the predicted and actual data become negligible (<0.5 ºC) in steady state and RMSE in each trial case is lower than 0.24.

A suitably widespread network affords the opportunity to schedule and monitor the operation of distributed domestic appliances simultaneously to collectively improve energy efficiency and reduce peak power consumption on the electrical grid. In this thesis, the IP-based synchronized wireless network is proposed and implemented for monitoring and jointly scheduling the widely distributed domestic refrigerators in DSR events using the proposed Model Predictive Control with Binary Quadratic Programming. Benefits afforded by the proposed technique are investigated through experimental trials from VonShef 13/291 (50W), iGENIX IG 3920 (55W) and Russell Hobbs RHCLRF17B (50W) domestic refrigerators sited in three different domestic locations in the city of Lincoln, UK. The algorithm is able to put a constraint on the total maximum power consumption of the fridges and the number of ON-OFF events is constrained to be 6-8 per hour for each fridge. Also, supply priority is considered. The experimental results are carried out under four different conditions including: 1) refrigerators operate in isolation 2) total maximum power consumption is limited to 110W with equal supply priority 3) total maximum power consumption is limited to 60W with equal supply priority 4) total maximum power consumption is limited to 60W with unequal supply priority. The results show that in trial 2, iGENIX and VonShef have energy savings of up to 19% and 29%, respectively, compared to the trial 1, though in trials A and B all of refrigerators remain within required temperature bounds based on food safety standards proposed in [127]. Moreover, an experimental study investigates how domestic refrigerators can respond to DSR events. Results stemming from the initiation of two DSR events for the small network of refrigerators with equal and unequal supply priorities, show that the proposed network is able to adaptively schedule the appliances to reduce peak operational loads and facilitate Demand Side Response. Further widespread expansion of the proposed technique would allow a rapidly deployed regional DSR strategy to aid grid stability.

The effect of varying hysteresis bands and internal product on the energy consumption of a domestic refrigerator is investigated as part of an IoT controlled platform. It is shown that the energy consumption of the empty refrigerator increases as the hysteresis

band increases. However, as the amount of product is increased (greater thermal mass) the opposite trend is shown to occur. It is demonstrated that potential energy savings of up to 20% are possible by judicious choice of hysteresis band with respect to the amount of product within the refrigerator. This requires a real-time adaption scheme based on identifying an underlying model of the refrigerator dynamics, and its control, as presented in this thesis for example. For largescale distributed refrigeration systems, as found in superstores for example, the use of IoT in the manner presented in this thesis will also allow candidacy algorithms to be implemented (to facilitate DSM) as well as local hysteresis band control to accommodate product mass changes in real time.

Finally, the custom fridge environment is created using OpenAI Gym environment and the RL based methods using Q-table and DQN are developed to control a single refrigerator within a defined upper and lower temperature bands. The results reveal that the Q-table works only with the trained setpoint temperatures, but that DQN can control the internal temperature other than at the trained setpoint point. Moreover, the DQN is used and tested to schedule the aggregated domestic refrigerators for peak shaving purposes considering refrigerators' compressor life-time indices to accommodate a maximum number of compressor starts per hour as part of the scheduling, to avoid too frequent on/off switching events. The results show that the proposed DQN can respond to power shedding events (total maximum power consumption is limited to 60W) and all of refrigerators remain within required temperature bounds not only for the trained setpoint but also for the setpoints other than at the trained setpoint.

In the experiments, smart plugs were used to connect the refrigerators to the Internet and control their ON/OFF states. In a wider dimension, smart plugs and sensors can also be integrated inside refrigerators as a way to transmit and receive data, as well as ON/OFF commands. Moreover, some smart refrigerators have this feature and are currently connected to the Internet such as LG GSXV91BSAE Wifi Connected American Fridge Freezer, Bosch Serie 8 KFF96PIEP Wifi Connected American Fridge Freezer and LG GBB92STAXP Wifi Connected 70/30 Frost Free Fridge Freezer.

The novelty in this thesis include:

1. A grey box model is proposed and tested for refrigerators to be used in control methods.
2. Model predictive control with binary quadratic programming for the scheduled operation of domestic refrigerators is designed and implemented.
3. The impact of hysteresis control and internal thermal mass on the energy efficiency of domestic refrigerators is investigated.
4. An IoT platform is designed to remotely monitor and control domestic appliances and this platform is to be used to implement the proposed MPC.
5. A DQN agent is trained and used to control hysteresis bands in a single fridge and schedule the operation of multiple fridges during DSR events.

6.2 Recommendations for Further Work

Potential future improvements include:

1- Investigation of the RLS system identification method is presented in other models of refrigerators that have more physical characteristics.
2- Experimental validation for simulation results using DQN and investigating a communication network using Raspberry Pi and domestic fridges.
3- Identifying limitations and improving the proposed scheduling methods by implementing them in a wider scale.
4- Improve the proposed RL and investigate how to add different supply priorities to all refrigerators.
5- Implementation of other types of RL-based approaches to have a more comprehensive evaluation.

6- Considering other types of TCLs and residential heating systems using smart thermostats.
7- Considering more constraints in scheduling section related to human activities such as door opening in refrigerators.
8- IoT based energy management of commercial controllable loads such as commercial refrigerators in super stores using RL.

7 REFERENCES

[1] Department of Energy and Climate Change. "UK renewable energy roadmap." (2011).

[2] Ghadiri, Fatemeh, and Mehdi Rasti. "The effect of selecting proper refrigeration cycle components on optimizing energy consumption of the household refrigerators." Applied thermal engineering 67.1-2 (2014): 335-340.

[3] DoE, U. S. "Office of Electricity delivery and energy reliability." National Electric Transmission Congestion Study—Draft for Public Comment (2014).

[4] Huang, Yantai, Hongjun Tian, and Lei Wang. "Demand response for home energy management system." International Journal of Electrical Power & Energy Systems 73 (2015): 448-455.

[5] Goldman, Charles. "Coordination of energy efficiency and demand response." (2010).

[6] Stamminger, Rainer. "Synergy potential of smart domestic appliances in renewable energy systems." (2009).

[7] Barthel, Claus, and Thomas Götz. "The overall worldwide saving potential from domestic refrigerators and freezers." Wuppertal Institute for Climate, Environment and Energy: Wuppertal, Germany (2012).

[8] Department for Business, E. and I.S. Great B. Department for Business, Energy and Industrial Strategy UK Energy Statistics, HM Government: London, 2018.

[9] Department for Business, E. and I.S. Great B. Department for Business, Energy and Industrial Strategy Energy Consumption in the UK 1970 to 2018, HM Government: London, 2018.

[10] Xu, Zhao, et al. "Design and modelling of thermostatically controlled loads as frequency controlled reserve." 2007 IEEE Power Engineering Society General Meeting. IEEE, 2007.

[11] Sanandaji, Borhan M., He Hao, and Kameshwar Poolla. "Fast regulation service provision via aggregation of thermostatically controlled loads." 2014 47th Hawaii International Conference on System Sciences. IEEE, 2014.

[12] Leading fridge & fridge/freezer brands in the UK 2018 Available online: https://www.statista.com/statistics/437582/leading-brands-of-fridges-and-fridge-freezers-in-the-uk/ (accessed on Oct 20, 2019)

[13] Barthel, Claus, and Thomas Götz. "The overall worldwide saving potential from domestic refrigerators and freezers." Wuppertal Institute for Climate, Environment and Energy: Wuppertal, Germany (2012).

[14] International Energy Agency Key World Energy Statistics 2019; Key World Energy Statistics; OECD, 2019; ISBN 9789264847880.

[15] Schweppe, Fred C., et al. "Homeostatic utility control." IEEE Transactions on Power Apparatus and Systems 3 (1980): 1151-1163.

[16] Ruester, Sophia, et al. "From distribution networks to smart distribution systems: Rethinking the regulation of European electricity DSOs." Utilities Policy 31 (2014): 229-237.

[17] Braithwait, S., and Kelly Eakin. "The role of demand response in electric power market design." Edison Electric Institute (2002): 1-57.

[18] Qdr, Q. J. U. D. E. "Benefits of demand response in electricity markets and recommendations for achieving them." US Dept. Energy, Washington, DC, USA, Tech. Rep 2006 (2006).

[19] Kirby, B. J. "Spinning reserve provided from Responsive Loads." Oak. Ridge National Laboratory (2003).

[20] Ljubljana, “Framework Guidelines on Electricity Balancing.” ACER (2012).

[21] Qdr, Q. J. U. D. E. "Benefits of demand response in electricity markets and recommendations for achieving them." US Dept. Energy, Washington, DC, USA, Tech. Rep 2006 (2006).

[22] Pérez-Lombard, Luis, José Ortiz, and Christine Pout. "A review on buildings energy consumption information." Energy and buildings 40.3 (2008): 394-398.

[23] Yu, William, Tooraj Jamasb, and Michael Pollitt. "Does weather explain cost and quality performance? An analysis of UK electricity distribution companies." Energy Policy 37.11 (2009): 4177-4188.

[24] Bartusch, Cajsa, et al. "Exploring variance in residential electricity consumption: Household features and building properties." Applied Energy 92 (2012): 637-643.

[25] Blom, Inge, Laure Itard, and Arjen Meijer. "Environmental impact of building-related and user-related energy consumption in dwellings." Building and Environment 46.8 (2011): 1657-1669.

[26] Crawley, Drury B., et al. "Contrasting the capabilities of building energy performance simulation programs." Building and environment 43.4 (2008): 661-673.

[27] Firth, Steven, et al. "Identifying trends in the use of domestic appliances from household electricity consumption measurements." Energy and buildings 40.5 (2008): 926-936.

[28] Wood, Georgina, and Marcus Newborough. "Dynamic energy-consumption indicators for domestic appliances: environment, behaviour and design." Energy and buildings 35.8 (2003): 821-841.

[29] Mohsenian-Rad, Amir-Hamed, and Alberto Leon-Garcia. "Optimal residential load control with price prediction in real-time electricity pricing environments." IEEE transactions on Smart Grid 1.2 (2010): 120-133.

[30] Yamamoto, Yoshihiro, et al. "Decision-making in electrical appliance use in the home." Energy Policy 36.5 (2008): 1679-1686.

[31] Torriti, Jacopo. Peak energy demand and demand side response. Routledge, 2015.

[32] Mansouri, Iman, Marcus Newborough, and Douglas Probert. "Energy consumption in UK households: impact of domestic electrical appliances." Applied Energy 54.3 (1996): 211-285.

[33] Kasulis, Jack J., David A. Huettner, and Neil J. Dikeman. "The feasibility of changing electricity consumption patterns." Journal of Consumer Research 8.3 (1981): 279-290.

[34] Bernard, Jean-Thomas, Denis Bolduc, and Nadège-Désirée Yameogo. "A pseudo-panel data model of household electricity demand." Resource and Energy Economics 33.1 (2011): 315-325.

[35] Neenan, Bernard, and Jiyong Eom. "Price elasticity of demand for electricity: a primer and synthesis." Electric Power Research Institute (2008).

[36] Thorsnes, Paul, John Williams, and Rob Lawson. "Consumer responses to time varying prices for electricity." Energy Policy 49 (2012): 552-561.

[37] Office for National Statistics (ONS). "Characteristics of homeworkers." (2014).

[38] Schweppe, Fred C., et al. "Homeostatic utility control." IEEE Transactions on Power Apparatus and Systems 3 (1980): 1151-1163.

[39] Stadler, Michael, et al. "Modelling and evaluation of control schemes for enhancing load shift of electricity demand for cooling devices." Environmental Modelling & Software 24.2 (2009): 285-295.

[40] D. Hirst, “System frequency- A resource for sustainable electricity,” Joint Research Center, November 2011.

[41] Lakshmanan, Venkatachalam, et al. "Domestic refrigerators temperature prediction strategy for the evaluation of the expected power consumption." IEEE PES ISGT Europe 2013. IEEE, 2013.

[42] Lakshmanan, Venkatachalam, et al. "Energy shift estimation of demand response activation on refrigerators—a field test study." 2014 49th International Universities Power Engineering Conference (UPEC). IEEE, 2014.

[43] Stadler, Ingo. "Power grid balancing of energy systems with high renewable energy penetration by demand response." Utilities Policy 16.2 (2008): 90-98.

[44] Lu, Ning. "Grid friendly TM appliances-load-side solution for congestion management." 2005/2006 IEEE/PES Transmission and Distribution Conference and Exhibition. IEEE, 2006.

[45] Postnikov, Andrey, et al. "Modelling of thermostatically controlled loads to analyse the potential of delivering FFR DSR with a large network of compressor packs." 2017 European Modelling Symposium (EMS). IEEE, 2017.

[46] Angeli, David, and Panagiotis-Aristidis Kountouriotis. "A stochastic approach to “dynamic-demand” refrigerator control." IEEE Transactions on control systems technology 20.3 (2011): 581-592.

[47] Hamid, Noor Hayatee Abdul, Mahanijah Md Kamal, and Faieza Hanum Yahaya. "Application of PID controller in controlling refrigerator temperature." 2009 5th International Colloquium on Signal Processing & Its Applications. IEEE, 2009.

[48] Xing, Lijuan, Shizhong Yang, and Qianqian Lu. "Self-tuning fuzzy PID controller for temperature control in Variable Air Volume air conditioning systems." The 2010 IEEE International Conference on Information and Automation. IEEE, 2010.

[49] Huang, Qingbao, Qianzhong She, and Xiaofeng Lin. "Adaptive fuzzy PID temperature control system based on OPC and modbus/TCP protocol." 2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010). Vol. 2. IEEE, 2010.

[50] Becker, M., et al. "Fuzzy control for temperature and humidity in refrigeration systems." IEEE Conference on Control Applications. Vol. 3. 1994.

[51] Angeli, David, and Panagiotis-Aristidis Kountouriotis. "A stochastic approach to “dynamic-demand” refrigerator control." IEEE Transactions on control systems technology 20.3 (2011): 581-592.

[52] Borsche, Theodor S., Juan de Santiago, and Göran Andersson. "Stochastic control of cooling appliances under disturbances for primary frequency reserves." Sustainable Energy, Grids and Networks 7 (2016): 70-79.

[53] Aunedi, Marko, et al. "Economic and environmental benefits of dynamic demand in providing frequency regulation." IEEE Transactions on Smart Grid 4.4 (2013): 2036-2048.

[54] Tindemans, Simon H., Vincenzo Trovato, and Goran Strbac. "Decentralized control of thermostatic loads for flexible demand response." IEEE Transactions on Control Systems Technology 23.5 (2015): 1685-1700.

[55] Dehghanpour, Kaveh, and Saeed Afsharnia. "Electrical demand side contribution to frequency control in power systems: a review on technical aspects." Renewable and Sustainable Energy Reviews 41 (2015): 1267-1276.

[56] Saleh, Ibrahim M., et al. "Impact of demand side response on a commercial retail refrigeration system." Energies 11.2 (2018): 371.

[57] Niro, Glauco, et al. "Large-scale control of domestic refrigerators for demand peak reduction in distribution systems." Electric power systems research 100 (2013): 34-42.

[58] Belman-Flores, J. M., et al. "Energy optimization of a domestic refrigerator controlled by a fuzzy logic system using the status of the door." International Journal of Refrigeration 104 (2019): 1-8.

[59] Serale, Gianluca, et al. "Model predictive control (MPC) for enhancing building and HVAC system energy efficiency: Problem formulation, applications and opportunities." Energies 11.3 (2018): 631.

[60] Carrascal, Edorta, et al. "Optimization of the heating system use in aged public buildings via model predictive control." Energies 9.4 (2016): 251.

[61] Baghina, Nadina, et al. "Predictive control of a domestic freezer for real-time demand response applications." 2012 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe). IEEE, 2012.

[62] Sossan, Fabrizio, et al. "Grey-box modelling of a household refrigeration unit using time series data in application to demand side management." Sustainable Energy, Grids and Networks 5 (2016): 1-12.

[63] Schné, Tamás, Szilárd Jaskó, and Gyula Simon. "Embeddable adaptive model predictive refrigerator control for cost-efficient and sustainable operation." Journal of Cleaner Production 190 (2018): 496-507.

[64] Azzouz, K., Denis Leducq, and D. Gobin. "Performance enhancement of a household refrigerator by addition of latent heat storage." International Journal of Refrigeration 31.5 (2008): 892-901.

[65] Ploug-Srensen, L., J. P. Fredsted, and M. Willatzen. "Improvements in the modeling and simulation of refrigeration systems: Aerospace tools applied to a domestic refrigerator." HVAC&R Research 3.4 (1997): 387-403.

[66] Prata, A.T., Ferreira, R.T.S., Fagotti, F., Todescat, M.L, “Heat transfer in a reciprocating compressor”, International Compressor Engineering Conference at Purdue, West Lafayette, IN, USA, (1997): 605-610.

[67] Hermes, Christian JL, et al. "Prediction of the energy consumption of household refrigerators and freezers via steady-state simulation." Applied Energy 86.7-8 (2009): 1311-1319.

[68] Costanzo, Giuseppe Tommaso, et al. "Grey-box modeling for system identification of household refrigerators: A step toward smart appliances." 2013 4th International Youth Conference on Energy (IYCE). IEEE, 2013.

[69] Schné, Tamás, Szilárd Jaskó, and Gyula Simon. "Dynamic models of a home refrigerator." MACRo 2015 1.1 (2015): 103-112.

[70] Geppert, Jasmin, and Rainer Stamminger. "Analysis of effecting factors on domestic refrigerators' energy consumption in use." Energy Conversion and Management 76 (2013): 794-800.

[71] Goodson, M. P., and C. W. Bullard. Refrigerator/Freezer system modeling. Air Conditioning and Refrigeration Center. College of Engineering. University of Illinois at Urbana-Champaign., 1994.

[72] Grimes, J. W., W. J. Mulroy, and Boyd L. Shomaker. "Effect of usage conditions on household refrigerator-freezer and freezer energy consumption." (1977).

[73] Anjana, H. M. K., P. H. V. Nimarshana, and R. A. Attalage. "Steady state performance variation of domestic refrigerators under different ambient conditions of Sri Lanka." 2015 Moratuwa Engineering Research Conference (MERCon). IEEE, 2015.

[74] Björk, Erik, and Björn Palm. "Performance of a domestic refrigerator under influence of varied expansion device capacity, refrigerant charge and ambient temperature." International Journal of Refrigeration 29.5 (2006): 789-798.

[75] Hermes, Christian JL, and Cláudio Melo. "Assessment of the energy performance of household refrigerators via dynamic simulation." Applied Thermal Engineering 29.5-6 (2009): 1153-1165.

[76] Saidur, Rahman, Haji Hassan Masjuki, and I. A. Choudhury. "Role of ambient temperature, door opening, thermostat setting position and their combined effect on refrigerator-freezer energy consumption." Energy Conversion and Management 43.6 (2002): 845-854.

[77] Sakallı, Özgün, Hüsnü Kerpiççi, and Lütfullah Kuddusi. "A study on optimizing the energy consumption of a cold storage cabinet." Applied Thermal Engineering 112 (2017): 424-430.

[78] Gupta, J. K., M. Ram Gopal, and S. Chakraborty. "Modeling of a domestic frost-free refrigerator." International Journal of refrigeration 30.2 (2007): 311-322.

[79] Zhang, Li, Takeshi Fujinawa, and Michiyuki Saikawa. "Theoretical study on a frost-free household refrigerator-freezer." International Journal of Refrigeration 62 (2016): 60-71.

[80] Harrington, Lloyd, Lu Aye, and Bob Fuller. "Impact of room temperature on energy consumption of household refrigerators: Lessons from analysis of field and laboratory data." Applied energy 211 (2018): 346-357.

[81] Pavithra, D., and Ranjith Balakrishnan. "IoT based monitoring and control system for home automation." 2015 global conference on communication technologies (GCCT). IEEE, 2015.

[82] Vishwakarma, Satyendra K., et al. "Smart energy efficient home automation system using iot." 2019 4th international conference on internet of things: Smart innovation and usages (IoT-SIU). IEEE, 2019.

[83] Mandula, Kumar, et al. "Mobile based home automation using Internet of Things (IoT)." 2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT). IEEE, 2015.

[84] Li, Taoren, et al. "Design of Remote Monitoring System Based on STM32F407 Microcontroller." 2019 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS). IEEE, 2019.

[85] Govindraj, Vignesh, Mithileysh Sathiyanarayanan, and Babangida Abubakar. "Customary homes to smart homes using Internet of Things (IoT) and mobile application." 2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon). IEEE, 2017.

[86] Lokhande, D. G., and Siddiqui Ahmed Mohsin. "Internet of things for ubiquitous smart home system." 2017 1st International Conference on Intelligent Systems and Information Management (ICISIM). IEEE, 2017.

[87] Al-Ali, Abdul-Rahman, et al. "A smart home energy management system using IoT and big data analytics approach." IEEE Transactions on Consumer Electronics 63.4 (2017): 426-434.

[88] Xiao, Zheng, et al. "Design of Home Appliance Control System in Smart Home based on WiFi IoT." 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC). IEEE, 2018.

[89] Hanumanthaiah, Aravind, et al. "Integrated cloud based smart home with automation and remote controllability." 2019 International Conference on Communication and Electronics Systems (ICCES). IEEE, 2019.

[90] Upadhyay, Yuvraj, Amol Borole, and D. Dileepan. "MQTT based secured home automation system." 2016 Symposium on Colossal Data Analysis and Networking (CDAN). IEEE, 2016.

[91] Mitchell, Tom M. "Does machine learning really work?." AI magazine 18.3 (1997): 11-11.

[92] Ayodele, Taiwo Oladipupo. "Types of machine learning algorithms." New advances in machine learning 3 (2010): 19-48.

[93] Kim, Ki-Jo, and Ilias Tagkopoulos. "Application of machine learning in rheumatic disease research." The Korean journal of internal medicine 34.4 (2019): 708.

[94] Barrett, Enda, and Stephen Linder. "Autonomous hvac control, a reinforcement learning approach." Joint European conference on machine learning and knowledge discovery in databases. Springer, Cham, 2015.

[95] Ruelens, Frederik, et al. "Residential demand response of thermostatically controlled loads using batch reinforcement learning." IEEE Transactions on Smart Grid 8.5 (2016): 2149-2159.

[96] Chen, Yujiao, et al. "Optimal control of HVAC and window systems for natural ventilation through reinforcement learning." Energy and Buildings 169 (2018): 195-205.

[97] Schreiber, Thomas, et al. "Application of two promising Reinforcement Learning algorithms for load shifting in a cooling supply system." Energy and Buildings 229 (2020): 110490.

[98] Gupta, Anchal, et al. "Energy-efficient heating control for smart buildings with deep reinforcement learning." Journal of Building Engineering 34 (2021): 101739.

[99] Azuatalam, Donald, et al. "Reinforcement learning for whole-building HVAC control and demand response." Energy and AI 2 (2020): 100020.

[100] Kurte, Kuldeep, et al. "Evaluating the Adaptability of Reinforcement Learning Based HVAC Control for Residential Houses." Sustainability 12.18 (2020): 7727.

[101] Tittaferrante, Andrew, and Abdulsalam Yassine. "Multi-advisor Deep Reinforcement Learning for Thermostatically Controlled Heating in Smart Homes." 2020 IEEE 8th International Conference on Smart Energy Grid Engineering (SEGE). IEEE, 2020.

[102] McKee, Evan, et al. "Deep Reinforcement Learning for Residential HVAC Control with Consideration of Human Occupancy." 2020 IEEE Power & Energy Society General Meeting (PESGM). IEEE, 2020.

[103] Du, Yan, et al. "Intelligent multi-zone residential HVAC control strategy based on deep reinforcement learning." Applied Energy 281 (2021): 116117.

[104] Ljung, Lennart. "System identification." Wiley encyclopedia of electrical and electronics engineering (1999): 1-19.

[105] Constantopoulos, Panos, Fred C. Schweppe, and Richard C. Larson. "ESTIA: A real-time consumer control scheme for space conditioning usage under spot electricity pricing." Computers & operations research 18.8 (1991): 751-765.

[106] Shirazi, Elham, Alireza Zakariazadeh, and Shahram Jadid. "Optimal joint scheduling of electrical and thermal appliances in a smart home environment." Energy Conversion and Management 106 (2015): 181-193.

[107] Bozchalui, Mohammad Chehreghani, et al. "Optimal operation of residential energy hubs in smart grids." IEEE Transactions on Smart Grid 3.4 (2012): 1755-1766.

[108] Kumar, Kandasamy Nandha, Krishnasamy Vijayakumar, and Chaudhari Kalpesh. "Virtual energy storage capacity estimation using ANN-based kWh modelling of refrigerators." IET Smart Grid 1.2 (2018): 31-39.

[109] CAMACHO, EF—BORDONS, and Bordons Alba. "C.: Model Predictive Control." (2004).

[110] Berthold, M. R. "Explorative Data Analysis: from machine learning to discovery support systems." Chemistry Central Journal 3.1 (2009): 1-1.

[111] Söylemez, E.; Alpman, E.; Onat, A. Experimental analysis of hybrid household refrigerators including thermoelectric and vapour compression cooling systems. International Journal of Refrigeration 2018, 95, 93–107. DOI: 10.1016/j.ijrefrig.2018.08.010

[112] Geppert, Jasmin, and Rainer Stamminger. "Analysis of effecting factors on domestic refrigerators' energy consumption in use." Energy Conversion and Management 76 (2013): 794-800.

[113] Harrington, Lloyd, Lu Aye, and Bob Fuller. "Impact of room temperature on energy consumption of household refrigerators: Lessons from analysis of field and laboratory data." Applied energy 211 (2018): 346-357.

[114] Sabegh, MR Zavvar, and C. M. Bingham. "Impact of Hysteresis Control and Internal Thermal Mass on the Energy Efficiency of IoT-Controlled Domestic Refrigerators." 2019 IEEE 7th International Conference on Smart Energy Grid Engineering (SEGE). IEEE, 2019.

[115] Postnikov, Andrey, et al. "Facilitating static firm frequency response with aggregated networks of commercial food refrigeration systems." Applied Energy 251 (2019): 113357.

[116] HOME - ICEAGE REFRIGERATION Available online:

http://www.iceage-hvac.com/Uploads/Product/2014-12-22/5497c67a7dc35.pdf

[117] BITZER // us Available online: https://www.bitzer.de/shared_media/documentation/kt-100-3.pdf (accessed on Oct 25, 2019).

[118] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[119] Sammut, Claude, and Geoffrey I. Webb. Encyclopedia of machine learning and data mining. Springer, 2017.

[120] Bellman, Richard. "Dynamic programming." Science 153.3731 (1966): 34-37.

[121] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.

[122] Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).

[123] Abadi, Martín, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).

[124] Refrigeration How To Guides - Fridges & Freezers | Beko. https://www.beko.co.uk/support/how-to-guides/refrigeration. Accessed 23 Feb. 2022.

[125] “Sunshine Tariff.” Regen, https://www.regen.co.uk/project/sunshine-tariff/. Accessed 15 Mar. 2022.

[126] GEUK | Green Energy UK. https://www.greenenergyuk.com/. Accessed 15 Mar. 2022.

[127] Ovca, Andrej, et al. “Temperatures and Storage Conditions in Domestic Refrigerators - Slovenian Scenario.” Food Control, vol. 123, May 2021, p. 107715. DOI.org (Crossref), https://doi.org/10.1016/j.foodcont.2020.107715.

[128] Electronic Enclosure Cooling Thermoelectric vs. Compressor-Based Air Conditioners | The World Leader in Thermal Management Solutions. https://lairdthermal.com/thermal-technical-library/white-papers/electronic-enclosure-cooling-thermoelectric-vs-compressor-based-air#:~:text=Cooling%20Mode,and%20heat%20load%20is%20smaller. Accessed 27 Apr. 2022.

8 APPENDICES

APENDIX1 INSTRUCTIONS ON SENDING DATA TO GOOGLE SHEETS

1. Login to the Gmail (It is better to create a new google account without two step verification security).
2. Go to the App icon in Top Right Corner and Click on Sheets.
3. Create a New Blank Sheet.
4. The Blank Sheet will be created with an “Untitled Spreadsheet”.
5. Write the names of the columns in the sheet.
6. Select the ‘Tools’ menu and click on “<> Script Editor” option.
7. The new Google Script is created with “Untitled project”.
8. Use the provided Google Script code, then edit the Sheet name and sheet_id in the code. The Sheet ID is accessible from the Sheet URL.
9. Use the case to add new sensors (this example includes the set of four sensors).
10. Go to ‘Publish’ menu and select the ‘Deploy as Web App…’.
11. The “Project version” will be “New”. Select “your email id” in the “Execute the app as” field. Choose “Anyone, even anonymous” in the “Who has access to the app” field. And then Click on “Deploy”.
12. Click on “Review Permissions”. Click on “Advanced” then click on “Go to ‘your_script_name’(unsafe)” Click on “Allow” and this will give the permission to deploy it as web app. The new screen is created with a given link and named as “Current web app URL”. This URL contains Google Script ID.
13. Put Wi-Fi SSID and password. Also, put Google script ID (GAS_ID) in the Arduino code.
14. Add the sensors in the url section of the Arduino code.
15. Upload the Arduino code to the NodeMCU.

Google Script code

```
function doGet(e) {

  Logger.log( JSON.stringify(e) );  // view parameters
```

```
var result = 'Ok'; // assume success

if (e.parameter == 'undefined') {

  result = 'No Parameters';

}

else {

  var sheet_id = '1mrJvHifHRD7gk4efAJcbD_22pTnYPDN9W-a-3iQG3aY';
   // Spreadsheet ID

  var sheet = SpreadsheetApp.openById(sheet_id).getActiveSheet();                    //       get
Active sheet

  var newRow = sheet.getLastRow() + 1;

  var rowData = [];

   var Curr_Date = new Date();

  rowData[0] = Curr_Date;
                                  // Date in column A

   var Curr_Time = Utilities.formatDate(Curr_Date, "Etc/GMT", 'HH:mm:ss');

   rowData[1] = Curr_Time;
                                  // Time in column B

  for (var param in e.parameter) {

   Logger.log('In for loop, param=' + param);

   var value = stripQuotes(e.parameter[param]);

   Logger.log(param + ':' + e.parameter[param]);

   switch (param) {

     case 'sensor1': //Parameter

       rowData[2] = value; //Value in column C

       break;

     case 'sensor2': //Parameter

       rowData[3] = value; //Value in column D

       break;
```

```
      case 'sensor3': //Parameter
        rowData[4] = value; //Value in column E
        break;
      case 'sensor4': //Parameter
        rowData[5] = value; //Value in column F
        break;
      default:
        result = "unsupported parameter";
    }
  }
  Logger.log(JSON.stringify(rowData));
  // Write new row below
  var newRange = sheet.getRange(newRow, 1, 1, rowData.length);
  newRange.setValues([rowData]);
 }
 // Return result of operation
 return ContentService.createTextOutput(result);
}
/**
* Remove leading and trailing single or double quotes
*/
function stripQuotes(value) {
 return value.replace(/^["']|["']$/g, "");
}
//---------------------------------------------
// End of file
```

```
//-----------------------------------------------
```

Arduino Code

```
#include <ESP8266WiFi.h>

#include <WiFiClientSecure.h>

WiFiClientSecure gsclient;

double sensor1;

double sensor2;

double sensor3;

double sensor4;

//const char* ssid = "……";              //replace with our wifi ssid

//const char* password = "……..";        //replace with your wifi password

//const char* ssid = ".....";             //replace with our wifi ssid

//const char* password = "…….";         //replace with your wifi password

const char* ssid = "……";              //replace with our wifi ssid

const char* password = "…….";         //replace with your wifi password


String GAS_ID = "……";//getactivespreadsheetID

const char* fingerprint = "46 B2 C3 44 9C 59 09 8B 01 B6 F8 BD 4C FB 00 74 91 2F EF
F6";

const char* host = "script.google.com";

const int httpsPort = 443;


void setup() {

  delay(1000);

  Serial.begin(115200);

  Serial.println();
```

```
  Serial.print("Connecting to wifi: ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
}
//.....
void loop() {
    sensor1 = 4.68;
    sensor2 = 56.32;
    sensor3 = 101.78;
    sensor4 = 84.17;
    sendData(sensor1, sensor2, sensor3, sensor4);
    delay(1000);

}
//.....
void sendData(double x, double y, double z, double t)
{
  gsclient.setInsecure();
  if (!gsclient.connect("script.google.com", httpsPort)) {
    Serial.println("connection failed");
```

```
  return;
 }
 if (gsclient.verify(fingerprint, host)) {
 } else {
 }
 String string_x     =  String(x, DEC);
 String string_y     =  String(y, DEC);
 String string_z     =  String(z, DEC);
 String string_t     =  String(t, DEC);
 String url = "/macros/s/" + GAS_ID + "/exec?sensor1=" + string_x + "&sensor2=" +
string_y + "&sensor3=" + string_z + "&sensor4=" + string_t;
 gsclient.print(String("GET ") + url + " HTTP/1.1\r\n" +
              "Host: " + host + "\r\n" +
              "User-Agent: BuildFailureDetectorESP8266\r\n" +
              "Connection: close\r\n\r\n");
 while (gsclient.connected()) {
  String line = gsclient.readStringUntil('\n');
  if (line == "\r") {
    break;
  }
 }
 String line = gsclient.readStringUntil('\n');
 Serial.println(line);
 if (line.startsWith("{\"state\":\"success\"")) {
 } else {
 }
```

```
}
```

APPENDIX 2 CLIENT CODE (MICROPYTHON)

```
try:
  import usocket as socket
except:
  import socket
from machine import Pin
import network
from umqtt.simple import MQTTClient
import time
import os
import sys
import socket
import json
import struct
import machine
import onewire
import ds18x20
import esp
esp.osdebug(None)
import gc
gc.collect()
ssid = '…….'
password = '…….'
#staticIP ='192.168.1.13'
#subnet = '255.255.255.0'
```

```
#gateway = '192.168.1.1'
#dns = '194.168.1.1'
TP_Link_address= "192.168.1.12"
THINGSPEAK_Read_CHANNEL_ID = b'……..'
THINGSPEAK_CHANNEL_READ_API_KEY = b'……..'
ds_pin = machine.Pin(2)
ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))
roms = ds_sensor.scan()
romfridge = roms[1]
romambient = roms[0]
station = network.WLAN(network.STA_IF)
station.active(True)
#station.ifconfig((staticIP, subnet, gateway, dns))
station.connect(ssid, password)
count = 0
while station.isconnected() == False:
  pass
#print('Connection successful')
print(station.ifconfig())


def cb(topic, msg):
    global onoff
    onoff = float(str(msg,'utf-8'))
    #print(onoff)
def encrypt(string):
    key = 171
```

```
    result = b"\0\0\0\0"
    for i in string:
        a = key ^ i
        key = a
        result += a.to_bytes((len(bin(a))-2 + 7) // 8, 'big')
    return result
def decrypt(string):
    key = 171
    result = b""
    for i in string:
        a = key ^ i
        key = i
        result += a.to_bytes((len(bin(a))-2 + 7) // 8, 'big')
    return result
def send_hs_command(address, port, cmd):
    data = b""
    tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_sock.connect((address, port))
    tcp_sock.send(encrypt(cmd))
    data = tcp_sock.recv(2048)
    return data
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))
s.listen(5)

while True:
```

```
conn, addr = s.accept()
#print('Got a connection from %s' % str(addr))
request = conn.recv(1024)
request = str(request)
#print('Content = %s' % request)
recievedReq = request.find('/fridgetemperature')
recievedReq1 = request.find('/ambienttemperature')
recievedReq2 = request.find('/power')
ds_sensor.convert_temp()
time.sleep_ms(50)

if recievedReq == 6:
  #print('Reading Fridge Sensor')
  try:
    Tempfridge = ds_sensor.read_temp(romfridge)
    TempfridgeStr = str(Tempfridge)
  except:
    pass
  response = TempfridgeStr
  conn.send('HTTP/1.1 200 OK\n')
  conn.send('Content-Type: text/html\n')
  conn.send('Connection: close\n\n')
  conn.sendall(response)
  conn.close()
  count = count + 1
if recievedReq1 == 6:
```

```
        #print('Reading Ambient Sensor')
        try:
          TempAmb = ds_sensor.read_temp(romambient)
          TempAmbStr = str(TempAmb)
        except:
          pass
        response = TempAmbStr
        conn.send('HTTP/1.1 200 OK\n')
        conn.send('Content-Type: text/html\n')
        conn.send('Connection: close\n\n')
        conn.sendall(response)
        conn.close()
        count = count + 1
      if recievedReq2 == 6:
        #print('Reading power')
        data1 = send_hs_command(TP_Link_address, 9999, b'{"emeter":{"get_realtime":{}}}')
        decrypted_data1 = decrypt(data1[4:]).decode()
        json_data1 = json.loads(decrypted_data1)
        emeter1 = json_data1["emeter"]["get_realtime"]
        power1=emeter1["power"]
        #print(power1)
        power1Str = str(power1)
        response = power1Str
        conn.send('HTTP/1.1 200 OK\n')
        conn.send('Content-Type: text/html\n')
        conn.send('Connection: close\n\n')
```

```
    conn.sendall(response)

    conn.close()

    count = count + 1

  if count == 3:

    randomNum = int.from_bytes(os.urandom(3), 'little')

    myMqttClient = bytes("client"+str(randomNum), 'utf-8')

    client1 = MQTTClient(client_id=myMqttClient, server=b"mqtt.thingspeak.com",
user=b'J487S61H5W6B7NIE', password=b'0Z60O5K6WO410BGH', ssl=False)

    client1.connect()

    subscribeTopic = bytes("channels/{:s}/subscribe/fields/field4/{:s}".format(b'670681',
b'RLJCBZN66R1C9QNQ'), 'utf-8')

    client1.set_callback(cb)

    client1.subscribe(subscribeTopic)

    client1.wait_msg()

    client1.disconnect()

    count = 0

    if onoff == 1:

      send_hs_command(TP_Link_address,                                    9999,
b'{"system":{"set_relay_state":{"state":0}}}')

      #print('TP-Link turned off')

    elif onoff == 2:

      send_hs_command(TP_Link_address,                                    9999,
b'{"system":{"set_relay_state":{"state":1}}}')

      #print('TP-Link turned on')
```

APPENDIX 3 SERVER CODE (ARDUINO)

```
#include <ESP8266WiFi.h>

#include <ESP8266HTTPClient.h>

#include <WiFiClient.h>

#include <ESP8266WiFiMulti.h>

#include <WiFiClientSecure.h>

ESP8266WiFiMulti WiFiMulti;


const char* MY_SSID = "……..";

const char* MY_PWD = "……..";


const char* server = "api.thingspeak.com";

String apiKeyChannel1 ="…….";

String apiKeyChannel4 ="…….";


//Your IP address or domain name with URL path

const char* serverNameIntTempVonShef = "http://82.30.166.126:303/fridgetemperature";

const char* serverNameAmbTempVonShef = "http://82.30.166.126:303/ambienttemperature";

const char* serverNamepowerVonShef = "http://82.30.166.126:303/power";


const char* serverNameIntTempiGenix = "http://89.240.127.106:80/fridgetemperature";

const char* serverNameAmbTempiGenix = "http://89.240.127.106:80/ambienttemperature";

const char* serverNamepoweriGenix = "http://89.240.127.106:80/power";
```

```
const char* serverNameIntTempRussell = "http://82.31.108.103:303/fridgetemperature";

const char* serverNameAmbTempRussell = "http://82.31.108.103:303/ambienttemperature";

const char* serverNamepowerRussell = "http://82.31.108.103:303/power";


String InttemperatureVonShef;

String AmbtemperatureVonShef;

String powerVonShef;


String InttemperatureiGenix;

String AmbtemperatureiGenix;

String poweriGenix;


String InttemperatureRussell;

String AmbtemperatureRussell;

String powerRussell;


const long interval = 60000;


unsigned long previousMillis = 0;

//.....................................................................................................

void setup() {

  Serial.begin(115200);

  Serial.println();

  delay(1000);

  connectWifi();

}
```

```
//.....................................................................................................

void loop() {

  unsigned long currentMillis = millis();


  if(currentMillis - previousMillis >= interval) {

    // Check WiFi connection status

   if ((WiFiMulti.run() == WL_CONNECTED)) {

     InttemperatureVonShef = httpGETRequest(serverNameIntTempVonShef);

     AmbtemperatureVonShef = httpGETRequest(serverNameAmbTempVonShef);

     powerVonShef = httpGETRequest(serverNamepowerVonShef);


     InttemperatureiGenix = httpGETRequest(serverNameIntTempiGenix);

     AmbtemperatureiGenix = httpGETRequest(serverNameAmbTempiGenix);

     poweriGenix = httpGETRequest(serverNamepoweriGenix);


     InttemperatureRussell = httpGETRequest(serverNameIntTempRussell);

     AmbtemperatureRussell = httpGETRequest(serverNameAmbTempRussell);

     powerRussell = httpGETRequest(serverNamepowerRussell);


     Serial.println("Fridge Temperature VonShef: " + InttemperatureVonShef + " *C ");

     Serial.println("Ambient Temperature VonShef: " + AmbtemperatureVonShef + " *C ");

     Serial.println("power VonShef: " + powerVonShef + " W ");


     Serial.println("Fridge Temperature iGenix: " + InttemperatureiGenix + " *C ");

     Serial.println("Ambient Temperature iGenix: " + AmbtemperatureiGenix + " *C ");

     Serial.println("power iGenix: " + poweriGenix + " W ");
```

```
      Serial.println("Fridge Temperature Russell: " + InttemperatureRussell + " *C ");

      Serial.println("Ambient Temperature Russell: " + AmbtemperatureRussell + " *C ");

      Serial.println("power Russell: " + powerRussell + " W ");

      delay(200);

      sendDataToChannel1(InttemperatureVonShef, powerVonShef, InttemperatureiGenix, poweriGenix, InttemperatureRussell, powerRussell, AmbtemperatureVonShef, AmbtemperatureiGenix);

      sendDataToChannel4(AmbtemperatureRussell);

      delay(500);


      // save the last HTTP GET Request

      previousMillis = currentMillis;

    }

    else {

      Serial.println("WiFi Disconnected");

    }

  }

}

//.....................................................................................................

String httpGETRequest(const char* serverName) {

  WiFiClient client;

  HTTPClient http;


  // Your IP address with path or Domain name with URL path

  http.begin(client, serverName);
```

```
  // Send HTTP POST request
  int httpResponseCode = http.GET();

  String payload = "--";

  if (httpResponseCode>0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    payload = http.getString();
  }
  else {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
  // Free resources
  http.end();

  return payload;
}
//.....
void connectWifi()
{
  Serial.print("Connecting to "+*MY_SSID);
  WiFi.begin(MY_SSID, MY_PWD);
  while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
```

```
 Serial.print(".");

 }


 Serial.println("");

 Serial.println("Connected to WiFi Network");

 Serial.println("");

}//end connect.....

void sendDataToChannel1(String a, String b, String c, String d, String e, String f, String g, String h){

 WiFiClient client;

 if (client.connect(server, 80)) { // use ip 184.106.153.149 or api.thingspeak.com

 //Serial.println("WiFi Client connected ");


 String postStr = apiKeyChannel1;

 postStr += "&field1=";

 postStr += String(a);

 postStr +="&field2=";

 postStr += String(b);

 postStr += "&field3=";

 postStr += String(c);

 postStr +="&field4=";

 postStr += String(d);

 postStr +="&field5=";

 postStr += String(e);

 postStr +="&field6=";

 postStr += String(f);

 postStr +="&field7=";
```

```
  postStr += String(g);
  postStr +="&field8=";
  postStr += String(h);
  postStr += "\r\n\r\n";

  client.print("POST /update HTTP/1.1\n");
  client.print("Host: api.thingspeak.com\n");
  client.print("Connection: close\n");
  client.print("X-THINGSPEAKAPIKEY: " + apiKeyChannel1 + "\n");
  client.print("Content-Type: application/x-www-form-urlencoded\n");
  client.print("Content-Length: ");
  client.print(postStr.length());
  client.print("\n\n");
  client.print(postStr);
  }//end if
  client.stop();
}
//.....
.
void sendDataToChannel4(String aa){
 WiFiClient client;
 if (client.connect(server, 80)) { // use ip 184.106.153.149 or api.thingspeak.com
  //Serial.println("WiFi Client connected ");

  String postStr = apiKeyChannel4;
  postStr += "&field1=";
```

```
  postStr += String(aa);
  postStr += "\r\n\r\n";

  client.print("POST /update HTTP/1.1\n");
  client.print("Host: api.thingspeak.com\n");
  client.print("Connection: close\n");
  client.print("X-THINGSPEAKAPIKEY: " + apiKeyChannel4 + "\n");
  client.print("Content-Type: application/x-www-form-urlencoded\n");
  client.print("Content-Length: ");
  client.print(postStr.length());
  client.print("\n\n");
  client.print(postStr);
  }//end if
  client.stop();
}
```

APPENDIX 4 HYSTERESIS BAND CONTROL USING Q-TABLE

```
import numpy as np  # for array stuff and random

#import numpy

from PIL import Image  # for creating visual of our env

import cv2  # for showing our visual live

import matplotlib.pyplot as plt  # for graphing our mean rewards over time

import pickle  # to save/load Q-Tables

from matplotlib import style  # to make pretty charts because it matters.

import time  # using this to keep track of our saved Q-Tables.


style.use("ggplot")  # setting our style!


#upper_temp_real = 2.8

#lower_temp_real = 1.2


upper_temp_virtual = 2.6

lower_temp_virtual = 1.4


a_Fridge = 0.98

b_Fridge = 0.0032

c_Fridge = 0.004


power_usage = 51

min_on = 8

min_off = 14
```

```
HM_EPISODES = 1000

steps = 200

#IN_BOUND_PENALTY = (Tinternal - Tsetpoint)^2

setpoint_reward = 200

#EXCEED_BOUNDS_PENALTY = 100

MIN_ON_AND_OFF_TIME_PENALTY = 300


epsilon = 0.5  # randomness

EPS_DECAY = 0.9999  # Every episode will be epsilon*EPS_DECAY

SHOW_EVERY = 100  # how often to play through env visually.


start_q_table = "qtable-1622045057.pickle"  # if we have a pickled Q table, we'll put the filename of it here.


LEARNING_RATE = 0.1

DISCOUNT = 0.95


class OneFridge:


    def __init__(self):

        self.inTemp = round(np.random.uniform(lower_temp_virtual, upper_temp_virtual), 2)

        self.inTempSave = self.inTemp

        self.s = 1 #np.random.randint(0, 2)

        self.sSave = self.s

        self.minimumon = 0

        self.minimumoff = 0
```

```
        self.setpoint = np.random.choice([upper_temp_virtual, lower_temp_virtual])

        self.setpointprev = self.setpoint

        self.AmbTemp = round(np.random.uniform(23, 24), 2)


    def __str__(self):

        return f"Internal Temp is: {self.inTemp}, fridge state is: {self.s}, cons on is: {self.minimumon}, cons off is: {self.minimumoff}, setpoint is: {self.setpoint}, prevsetpoint is: {self.setpointprev}"


    def __sub__(self, other):

        return (round(self.inTemp-other.setpoint, 1))


    def action(self, choice):

        if choice == 0:

            self.move(s=0)

        elif choice == 1:

            self.move(s=1)


    def move(self,s=False):

        self.s = s

        self.AmbTemp = round(np.random.uniform(21, 22), 2)

        self.inTemp = round((a_Fridge * self.inTempSave) - (b_Fridge * power_usage * self.s) + (c_Fridge * self.AmbTemp), 2)

        self.inTempSave = self.inTemp


        if self.sSave == 0 and self.s == 1:

            self.minimumon = min_on
```

```
        elif self.sSave == 1 and self.s == 0:
            self.minimumoff = min_off

        self.sSave = self.s

        if self.minimumon == 0:
            self.minimumon = 0
        elif self.minimumon >= 0:
            self.minimumon = self.minimumon - 1

        if self.minimumoff == 0:
            self.minimumoff = 0
        elif self.minimumoff >= 0:
            self.minimumoff = self.minimumoff - 1

        self.setpointprev = self.setpoint

        if self.inTempSave >= upper_temp_virtual:
            self.setpoint = lower_temp_virtual
        elif self.inTempSave <= lower_temp_virtual:
            self.setpoint = upper_temp_virtual
        elif self.inTempSave < upper_temp_virtual and self.inTempSave > lower_temp_virtual:
            self.setpoint = self.setpoint

if start_q_table is None:
    # initialize the q-table#
```

```
    q_table = {}
    for setTemp in [lower_temp_virtual, upper_temp_virtual]:
        for delta_inTemp_setpoint in np.arange(-3, 3, 0.1):
            delta_inTemp_setpoint = round(delta_inTemp_setpoint, 1)
            for minon in range(0, min_on):
                for minoff in range(0, min_off):
                    if (minon * minoff) == 0:
                        q_table[(setTemp, delta_inTemp_setpoint, minon, minoff)] = [np.random.uniform(-5, 0) for i in range(2)]
else:
    with open(start_q_table, "rb") as f:
        q_table = pickle.load(f)
#print(q_table[(2.6, -0.9, 6, 13)])


episode_rewards = []


for episode in range(HM_EPISODES):


    Internal_Temps = []
    Ambient_Temp = []
    Fridge_States = []
    upper = []
    lower = []


    VonShef = OneFridge()
```

```
    if episode % SHOW_EVERY == 0:
        #print(f"on #{episode}, epsilon is {epsilon}")
        #print(f"{SHOW_EVERY} ep mean: {np.mean(episode_rewards[-SHOW_EVERY:])}")
        show = True
    else:
        show = False

    episode_reward = 0

    for i in range(steps):
        obs = (VonShef.setpoint, VonShef-VonShef, VonShef.minimumon, VonShef.minimumoff)
        #print(obs)
        if np.random.random() > epsilon:
            # GET THE ACTION
            action = np.argmax(q_table[obs])
        else:
            action = np.random.randint(0, 2)
        # Take the action!
        VonShef.action(action)


        if VonShef.minimumon > 0 and action == 0:
            reward = -MIN_ON_AND_OFF_TIME_PENALTY
        elif VonShef.minimumoff > 0 and action == 1:
            reward = -MIN_ON_AND_OFF_TIME_PENALTY
```

```
    elif VonShef.setpointprev == upper_temp_virtual and VonShef.inTemp >= upper_temp_virtual:

        reward = setpoint_reward

    elif VonShef.setpointprev == lower_temp_virtual and VonShef.inTemp <= lower_temp_virtual:

        reward = setpoint_reward

    else:

        reward = -round(pow((VonShef - VonShef), 2), 2)



    new_obs = (VonShef.setpoint, VonShef-VonShef, VonShef.minimumon, VonShef.minimumoff)  # new observation

    if (VonShef.minimumon * VonShef.minimumoff) == 0:

        max_future_q = np.max(q_table[new_obs])  # max Q value for this new obs

    current_q = q_table[obs][action]  # current Q for our chosen action



    if reward == setpoint_reward:

        new_q = setpoint_reward

    elif reward == -MIN_ON_AND_OFF_TIME_PENALTY:

        new_q = -MIN_ON_AND_OFF_TIME_PENALTY

    else:

        new_q = (1 - LEARNING_RATE) * current_q + LEARNING_RATE * (reward + DISCOUNT * max_future_q)



    q_table[obs][action] = new_q



    #if show:

        #print(VonShef)

        #print(f"step reward is {reward}, action is {action}")
```

```
    episode_reward += reward
    if show:
        Internal_Temps.append(VonShef.inTemp)
        upper.append(upper_temp_virtual)
        lower.append(lower_temp_virtual)
        Fridge_States.append(VonShef.s)
        #print(Fridge_States)
        Ambient_Temp.append(VonShef.AmbTemp)
        #print(Ambient_Temp)


    if reward == -MIN_ON_AND_OFF_TIME_PENALTY:
        break


#if episode == (HM_EPISODES - 1):
    #print(f"episode reward is: {episode_reward}")
    #plt.plot(Internal_Temps)
    #plt.plot(Fridge_States)
    #plt.plot(Ambient_Temp)
    #plt.show()
    #input("Press Enter to continue...")
if episode == (HM_EPISODES-SHOW_EVERY):
    plt.subplot(211)
    plt.plot(Internal_Temps, 'r', label="Internal Temp")
```

```
        plt.plot(Fridge_States, 'b', label="ON/OFF State")
        plt.plot(upper, 'g', linestyle='--')
        plt.plot(lower, 'g', linestyle='--')
        plt.ylabel("Internal Temp (ºC)")
        plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower left',
           ncol=2, mode="expand", borderaxespad=0.)
        #plt.show()
        plt.subplot(212)
        plt.plot(Ambient_Temp, 'k')
        plt.ylabel("Ambient Temp (ºC)")
        plt.xlabel("Time (min)")
        plt.show()
    episode_rewards.append(episode_reward)
    epsilon *= EPS_DECAY

moving_avg = np.convolve(episode_rewards, np.ones((SHOW_EVERY,))/SHOW_EVERY,
mode='valid')

plt.plot([i for i in range(len(moving_avg))], moving_avg)
plt.ylabel("Average reward")
plt.xlabel("Episode number")
plt.show()

with open(f"qtable-{int(time.time())}.pickle", "wb") as f:
    pickle.dump(q_table, f)
```

APPENDIX 5 HYSTERESIS BAND CONTROL USING DQN

```
from gym import Env

from gym.spaces import Discrete, Box, Tuple

import numpy as np

import random

import time

import matplotlib.pyplot as plt  # for graphing our mean rewards over time

from matplotlib import style  # to make pretty charts

# In[16]:

style.use("ggplot")  # setting our style!

upper_temp_virtual = 2.6

lower_temp_virtual = 1.4

a_Fridge = 0.98

b_Fridge = 0.0032

c_Fridge = 0.004

power_usage = 51

min_on = 8

min_off = 14

Each_episode_steps = 500

setpoint_reward = 200

MIN_ON_AND_OFF_TIME_PENALTY = 300

Num_actions = 2 #number of actions

Num_setpoints = 2 #number of Setpoints

Low_delta_TinTset = -4 #min delta Tin and Tsetpoint

High_delta_TinTset = 4 #max delta Tin and Tsetpoint

#plot vars
```

```
Internal_Temps = []

Ambient_Temps = []

Fridge_OnOff_States = []

Step_rewards = []

upper = []

lower = []

# In[17]:

class OneFridge(Env):

   def __init__(self):

      # Actions we can take, 0 and 1

      self.action_space = Discrete(Num_actions)

      # Observation Space(setpoint, Tin - Tsetpoint, minon, minoff)

      self.observation_space = Tuple([Discrete(Num_setpoints),

                              Box(low=np.array([Low_delta_TinTset]),
high=np.array([High_delta_TinTset]), dtype=np.float),

                              Box(low=np.array([0]), high=np.array([min_on]), dtype=np.int),

                              Box(low=np.array([0]),          high=np.array([min_off]),
dtype=np.int)])

      # set start setpoint

      self.setpoint = np.random.choice([upper_temp_virtual, lower_temp_virtual])

      self.setpointprev = self.setpoint

      if self.setpoint == upper_temp_virtual:

         self.setpointstate = 1

      elif self.setpoint == lower_temp_virtual:

         self.setpointstate = 0

      # Set start Internal Temp

      self.inTemp = np.random.uniform(lower_temp_virtual, upper_temp_virtual)
```

```
        self.inTempSave = self.inTemp
        # start delta Tin and Tsetpoint
        self.delta_T= self.inTemp-self.setpoint
        # set start minon and mioff
        self.minimumon = 0
        self.minimumoff = 0
        # Set episode length
        self.episode_length = Each_episode_steps

        # Set start Ambient temp
        self.AmbTemp = np.random.uniform(21, 22)
        # Set start on off state
        self.s = np.random.randint(0, 2)
        self.sSave = self.s

        #define start states
        self.state = (self.setpointstate, self.delta_T, self.minimumon, self.minimumoff)

    def step(self, action):
        # get new ambient Temp
        self.AmbTemp = np.random.uniform(21, 22)
        # Apply action
        self.inTemp = ((a_Fridge * self.inTempSave) - (b_Fridge * power_usage * action) +
(c_Fridge * self.AmbTemp))
        self.inTempSave = self.inTemp
        self.delta_T= self.inTemp-self.setpointprev
        if self.sSave == 0 and action == 1:
```

```
    self.minimumon = min_on
elif self.sSave == 1 and action == 0:
    self.minimumoff = min_off
self.sSave = action
if self.minimumon == 0:
    self.minimumon = 0
elif self.minimumon >= 0:
    self.minimumon = self.minimumon - 1

if self.minimumoff == 0:
    self.minimumoff = 0
elif self.minimumoff >= 0:
    self.minimumoff = self.minimumoff - 1

#update states
self.state = (self.setpointstate, self.delta_T, self.minimumon, self.minimumoff)

# Reduce episode length by 1
self.episode_length -= 1

# Calculate reward
if self.minimumon > 0 and action == 0:
    reward = -MIN_ON_AND_OFF_TIME_PENALTY
elif self.minimumoff > 0 and action == 1:
    reward = -MIN_ON_AND_OFF_TIME_PENALTY
elif self.setpointprev == upper_temp_virtual and self.inTemp >= upper_temp_virtual:
```

```
    reward = setpoint_reward
elif self.setpointprev == lower_temp_virtual and self.inTemp <= lower_temp_virtual:
    reward = setpoint_reward
else:
    reward = -pow((self.inTemp-self.setpointprev), 2)

# Check if episode is done
if self.episode_length <= 0:
    done = True
    Internal_Temps.append(self.inTemp) #Save the plot variables
    Fridge_OnOff_States.append(action)
    Ambient_Temps.append(self.AmbTemp)
    Step_rewards.append(reward)
    upper.append(upper_temp_virtual)
    lower.append(lower_temp_virtual)
elif reward == -MIN_ON_AND_OFF_TIME_PENALTY:
    done = True
    Internal_Temps.append(self.inTemp) #Save the plot variables
    Fridge_OnOff_States.append(action)
    Ambient_Temps.append(self.AmbTemp)
    Step_rewards.append(reward)
    upper.append(upper_temp_virtual)
    lower.append(lower_temp_virtual)
else:
    done = False
    Internal_Temps.append(self.inTemp) #Save the plot variables
```

```
            Fridge_OnOff_States.append(action)
            Ambient_Temps.append(self.AmbTemp)
            Step_rewards.append(reward)
            upper.append(upper_temp_virtual)
            lower.append(lower_temp_virtual)

        #update setpoint
        if self.inTemp >= upper_temp_virtual:
            self.setpoint = lower_temp_virtual
            self.setpointstate = 0
        elif self.inTemp <= lower_temp_virtual:
            self.setpoint = upper_temp_virtual
            self.setpointstate = 1
        elif self.inTemp < upper_temp_virtual and self.inTempSave > lower_temp_virtual:
            self.setpoint = self.setpoint
            self.setpointstate = self.setpointstate

        self.setpointprev = self.setpoint

        # Set placeholder for info
        info = {}

        # Return step information
        return self.state, reward, done, info

    def render(self, mode='human'):
```

```
    global Step_rewards
    global Internal_Temps
    global Ambient_Temps
    global Fridge_OnOff_States
    global upper
    global lower
    if done:
        plt.subplot(311)
        plt.plot(Internal_Temps, 'r', label="Internal Temp")
        plt.plot(Fridge_OnOff_States, 'b', label="ON/OFF State")
        plt.plot(upper, 'g', linestyle='--')
        plt.plot(lower, 'g', linestyle='--')
        #plt.title('VoShef')
        plt.ylabel("Internal Temp (ºC)")
        #plt.xlabel("Time (min)")
        plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower left', ncol=2,
mode="expand", borderaxespad=0.)
        #plt.show()


        plt.subplot(312)
        plt.plot(Step_rewards, 'm')
        plt.ylabel('Step Rewards')
        #plt.xlabel("Time (min)")
        #plt.show()


        plt.subplot(313)
        plt.plot(Ambient_Temps, 'k')
```

```
        plt.xlabel("Time (min)")
        plt.ylabel("Ambient Temp (ºC)")
        plt.show()

        #Internal_Temps = []
        #Ambient_Temps = []
        #Fridge_OnOff_States = []
        #Step_rewards = []

    def reset(self):
        # reset setpoint
        self.setpoint = np.random.choice([upper_temp_virtual, lower_temp_virtual])
        self.setpointprev = self.setpoint
        if self.setpoint == upper_temp_virtual:
            self.setpointstate = 1
        elif self.setpoint == lower_temp_virtual:
            self.setpointstate = 0

        # reset Internal Temp
        self.inTemp = np.random.uniform(lower_temp_virtual, upper_temp_virtual)
        self.inTempSave = self.inTemp
        # reset delta Tin and Tsetpoint
        self.delta_T= self.inTemp-self.setpoint
        # reset minon and mioff
        self.minimumon = 0
        self.minimumoff = 0
```

```
        # reset episode length
        self.episode_length = Each_episode_steps

        # reset start Ambient temp
        self.AmbTemp = np.random.uniform(21, 22)
        # reset on off state
        self.s = np.random.randint(0, 2)
        self.sSave = self.s

        #reset states
        self.state = (self.setpointstate, self.delta_T, self.minimumon, self.minimumoff)
        return self.state



# In[18]:
env = OneFridge()
episodes = 1
for episode in range(1, episodes+1):
    state = env.reset()
    done = False
    score = 0
    while not done:
        action = env.action_space.sample()
        n_state, reward, done, info = env.step(action)
        score+=reward
        print('New State:{} Reward:{} Action:{} Done:{}'.format(n_state, reward, action,
done))
```

```
    print('Episode:{} Score:{}'.format(episode, score))
    env.render()

# In[19]:

#Create a Deep Learning Model with Keras
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
# In[20]:

states = len(env.observation_space)
actions = env.action_space.n
print(states)
print(actions)

# In[21]:

# Deep model (input layer(states), dense layer, dense layer, output layer (actions))
def build_model(states, actions):
    model = Sequential()
    model.add(Flatten(input_shape=(1,states)))
    model.add(Dense(24, activation='relu'))
    model.add(Dense(24, activation='relu'))
    model.add(Dense(actions, activation='linear'))
```

```
    return model
# In[22]:
model = build_model(states, actions)
model.summary()


# In[23]:


#Build Agent with Keras-RL
from rl.agents import DQNAgent
from rl.policy import BoltzmannQPolicy, GreedyQPolicy
from rl.memory import SequentialMemory

# In[26]:

def build_agent(model, actions):
    policy = BoltzmannQPolicy()
    #policy =  GreedyQPolicy()
    memory = SequentialMemory(limit=50000, window_length=1)
    dqn = DQNAgent(model=model, memory=memory, policy=policy,
                   nb_actions=actions, nb_steps_warmup=10, target_model_update=1e-2)
    return dqn

dqn = build_agent(model, actions)
dqn.compile(Adam(lr=1e-3), metrics=['mse'])
```

```
dqn.fit(env, nb_steps=20000, visualize=False, verbose=1)

# In[29]:

Internal_Temps = []
Ambient_Temps = []
Fridge_OnOff_States = []
Step_rewards = []
upper = []
lower = []
env = OneFridge()
states = len(env.observation_space)
actions = env.action_space.n
model = build_model(states, actions)
dqn = build_agent(model, actions)
dqn.compile(Adam(lr=1e-3), metrics=['mse'])
dqn.load_weights('OneFridgeDqn2.h5f')

# In[30]:

scores = dqn.test(env, nb_episodes=1, visualize=False)

plt.subplot(311)
plt.plot(Internal_Temps, 'r', label="Internal Temp")
```

```
plt.plot(Fridge_OnOff_States, 'b', label="ON/OFF State")

plt.plot(upper, 'g', linestyle='--')

plt.plot(lower, 'g', linestyle='--')

#plt.title('VoShef')

plt.ylabel("Internal Temp (ºC)")

#plt.xlabel("Time (min)")

plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower left', ncol=2, mode="expand", borderaxespad=0.)

#plt.show()


plt.subplot(312)

plt.plot(Step_rewards, 'm')

plt.ylabel('Step Rewards')


plt.subplot(313)

plt.plot(Ambient_Temps, 'k')

plt.xlabel("Time (min)")

plt.ylabel("Ambient Temp (ºC)")

plt.show()



# In[31]:



#_ = dqn.test(env, nb_episodes=1, visualize=True)
```

 # In[230]:

#Internal_Temps

#Fridge_OnOff_States

#Step_rewards

#Ambient_Temps

'''del model

del dqn

del env'''

APPENDIX 6 MULTI-REFRIGERATOR SYSTEMS AND DSR USING DQN

```
from gym import Env

from gym.spaces import Discrete, Box, Tuple

import numpy as np

import random

import time

import matplotlib.pyplot as plt  # for graphing our mean rewards over time

from matplotlib import style  # to make pretty charts



# In[16]:



style.use("ggplot")  # setting our style!

#Fridge One

upper_temp_One = 2.5

lower_temp_One = 1.5

a_Fridge_One = 0.98

b_Fridge_One = 0.0032

c_Fridge_One = 0.004

power_usage_One = 51

min_on_One = 6

min_off_One = 12

Num_setpoints_One = 2 #number of Setpoints

Low_delta_TinTset_One = -5 #min delta Tin and Tsetpoint
```

High_delta_TinTset_One = 5 #max delta Tin and Tsetpoint

#Fridge Two

upper_temp_Two = 3

lower_temp_Two = 1.5

a_Fridge_Two = 0.99

b_Fridge_Two = 0.0047

c_Fridge_Two = 0.003

power_usage_Two = 55

min_on_Two = 5

min_off_Two = 15

Num_setpoints_Two = 2

Low_delta_TinTset_Two = -5

High_delta_TinTset_Two = 5

#Fridge Three

upper_temp_Three = 6

lower_temp_Three = 4

a_Fridge_Three = 0.98

b_Fridge_Three = 0.005

c_Fridge_Three = 0.006

power_usage_Three = 50

min_on_Three = 9

min_off_Three = 10

Num_setpoints_Three = 2

Low_delta_TinTset_Three = -5

```
High_delta_TinTset_Three = 5


#constants

Each_episode_steps = 500

setpoint_reward = 200

MIN_ON_AND_OFF_TIME_PENALTY = 1000

Num_actions = 4 #number of actions


#plot vars

Internal_Temps_One = []

Ambient_Temps_One = []

Fridge_OnOff_States_One = []

Step_rewards_One = []

Internal_Temps_Two = []

Ambient_Temps_Two = []

Fridge_OnOff_States_Two = []

Step_rewards_Two = []

Internal_Temps_Three = []

Ambient_Temps_Three = []

Fridge_OnOff_States_Three = []

Step_rewards_Three = []

upper_One = []

lower_One = []

upper_Two = []

lower_Two = []

upper_Three = []
```

```
lower_Three = []

Total_power = []

Total_powerlimit = []



# In[17]:



class ThreeFridge(Env):

   def __init__(self):

      # Actions we can take, 000, 100, 001, 010, 110, 101, 011, 111

      self.action_space = Discrete(Num_actions)

      # Observation Space(setpoint, Tin - Tsetpoint, minon, minoff)

      self.observation_space = Tuple([Discrete(Num_setpoints_One),

                        Discrete(Num_setpoints_Two),

                        Discrete(Num_setpoints_Three),

                        Box(low=np.array([Low_delta_TinTset_One]),
high=np.array([High_delta_TinTset_One]), dtype=np.float),

                        Box(low=np.array([0]),          high=np.array([min_on_One]),
dtype=np.int),

                        Box(low=np.array([0]),          high=np.array([min_off_One]),
dtype=np.int),

                        Box(low=np.array([Low_delta_TinTset_Two]),
high=np.array([High_delta_TinTset_Two]), dtype=np.float),

                        Box(low=np.array([0]),          high=np.array([min_on_Two]),
dtype=np.int),

                        Box(low=np.array([0]),          high=np.array([min_off_Two]),
dtype=np.int),
```

```
                    Box(low=np.array([Low_delta_TinTset_Three]),
high=np.array([High_delta_TinTset_Three]), dtype=np.float),

                    Box(low=np.array([0]),          high=np.array([min_on_Three]),
dtype=np.int),

                    Box(low=np.array([0]),          high=np.array([min_off_Three]),
dtype=np.int)])

    #Fridge one.....

    # set start setpoint fridge one

    self.setpoint_One = np.random.choice([upper_temp_One, lower_temp_One])

    self.setpointprev_One = self.setpoint_One

    if self.setpoint_One == upper_temp_One:

       self.setpointstate_One = 1

    elif self.setpoint_One == lower_temp_One:

       self.setpointstate_One = 0


    # Set start Internal Temp fridge one

    self.inTemp_One = np.random.uniform(lower_temp_One, upper_temp_One)

    self.inTempSave_One = self.inTemp_One

    # start delta Tin and Tsetpoint fridge one

    self.delta_T_One= self.inTemp_One-self.setpoint_One

    # set start minon and mioff fridge one

    self.minimumon_One = 0

    self.minimumoff_One = 0

    # Set start Ambient temp fridge one

    self.AmbTemp_One = np.random.uniform(21, 22)

    # Set start on off state fridge one

    self.s_One = np.random.randint(0, 2)

    self.sSave_One = self.s_One
```

```
# fridge one end.....

#Fridge Two.....
# set start setpoint fridge two
self.setpoint_Two = np.random.choice([upper_temp_Two, lower_temp_Two])
self.setpointprev_Two = self.setpoint_Two
if self.setpoint_Two == upper_temp_Two:
    self.setpointstate_Two = 1
elif self.setpoint_Two == lower_temp_Two:
    self.setpointstate_Two = 0

# Set start Internal Temp fridge two
self.inTemp_Two = np.random.uniform(lower_temp_Two, upper_temp_Two)
self.inTempSave_Two = self.inTemp_Two
# start delta Tin and Tsetpoint fridge Two
self.delta_T_Two= self.inTemp_Two-self.setpoint_Two
# set start minon and mioff fridge Two
self.minimumon_Two = 0
self.minimumoff_Two = 0
# Set start Ambient temp fridge two
self.AmbTemp_Two = np.random.uniform(21, 22)
# Set start on off state fridge two
self.s_Two = np.random.randint(0, 2)
self.sSave_Two = self.s_Two
# fridge two end.....
```

```
#Fridge three.....….....
# set start setpoint fridge three
self.setpoint_Three = np.random.choice([upper_temp_Three, lower_temp_Three])
self.setpointprev_Three = self.setpoint_Three
if self.setpoint_Three == upper_temp_Three:
    self.setpointstate_Three = 1
elif self.setpoint_Three == lower_temp_Three:
    self.setpointstate_Three = 0
# Set start Internal Temp fridge three
self.inTemp_Three = np.random.uniform(lower_temp_Three, upper_temp_Three)
self.inTempSave_Three = self.inTemp_Three
# start delta Tin and Tsetpoint fridge Three
self.delta_T_Three= self.inTemp_Three-self.setpoint_Three
# set start minon and mioff fridge Three
self.minimumon_Three = 0
self.minimumoff_Three = 0
# Set start Ambient temp fridge three
self.AmbTemp_Three = np.random.uniform(21, 22)
# Set start on off state fridge three
self.s_Three = np.random.randint(0, 2)
self.sSave_Three = self.s_Three
# fridge three end.....

# Set episode length
self.episode_length = Each_episode_steps
```

```
    #define start states
    self.state = (self.setpointstate_One,
                  self.setpointstate_Two,
                  self.setpointstate_Three,
                  self.delta_T_One, self.minimumon_One, self.minimumoff_One,
                  self.delta_T_Two, self.minimumon_Two, self.minimumoff_Two,
                  self.delta_T_Three, self.minimumon_Three, self.minimumoff_Three)


def step(self, action):
    # get new ambient Temp fridge one
    self.AmbTemp_One = np.random.uniform(21, 22)
    # get new ambient Temp fridge two
    self.AmbTemp_Two = np.random.uniform(21, 22)
    # get new ambient Temp fridge three
    self.AmbTemp_Three = np.random.uniform(21, 22)


    # Apply action
    if action == 0:
       action_One = 0
       action_Two = 0
       action_Three = 0
    elif action == 1:
       action_One = 0
       action_Two = 0
       action_Three = 1
    elif action == 2:
```

```
            action_One = 1
            action_Two = 0
            action_Three = 0
        elif action == 3:
            action_One = 0
            action_Two = 1
            action_Three = 0
        """elif action == 4:
            action_One = 1
            action_Two = 1
            action_Three = 0
        elif action == 5:
            action_One = 1
            action_Two = 0
            action_Three = 1
        elif action == 6:
            action_One = 0
            action_Two = 1
            action_Three = 1
        elif action == 7:
            action_One = 1
            action_Two = 1
            action_Three = 1"""
        #fridge one.....................................................................................
        self.inTemp_One = ((a_Fridge_One * self.inTempSave_One) - (b_Fridge_One *
power_usage_One * action_One) + (c_Fridge_One * self.AmbTemp_One))
        self.inTempSave_One = self.inTemp_One
```

```
        self.delta_T_One = self.inTemp_One-self.setpointprev_One


        if self.sSave_One == 0 and action_One == 1:
            self.minimumon_One = min_on_One
        elif self.sSave_One == 1 and action_One == 0:
            self.minimumoff_One = min_off_One
        self.sSave_One = action_One


        if self.minimumon_One == 0:
            self.minimumon_One = 0
        elif self.minimumon_One >= 0:
            self.minimumon_One = self.minimumon_One - 1


        if self.minimumoff_One == 0:
            self.minimumoff_One = 0
        elif self.minimumoff_One >= 0:
            self.minimumoff_One = self.minimumoff_One - 1
        #fridge one end.....


        #fridge two.....
        self.inTemp_Two = ((a_Fridge_Two * self.inTempSave_Two) - (b_Fridge_Two *
power_usage_Two * action_Two) + (c_Fridge_Two * self.AmbTemp_Two))
        self.inTempSave_Two = self.inTemp_Two
        self.delta_T_Two = self.inTemp_Two-self.setpointprev_Two


        if self.sSave_Two == 0 and action_Two == 1:
```

```
            self.minimumon_Two = min_on_Two
        elif self.sSave_Two == 1 and action_Two == 0:
            self.minimumoff_Two = min_off_Two
        self.sSave_Two = action_Two

        if self.minimumon_Two == 0:
            self.minimumon_Two = 0
        elif self.minimumon_Two >= 0:
            self.minimumon_Two = self.minimumon_Two - 1

        if self.minimumoff_Two == 0:
            self.minimumoff_Two = 0
        elif self.minimumoff_Two >= 0:
            self.minimumoff_Two = self.minimumoff_Two - 1
        #fridge two end.....

        #fridge three.....
        self.inTemp_Three = ((a_Fridge_Three * self.inTempSave_Three) - (b_Fridge_Three
* power_usage_Three * action_Three) + (c_Fridge_Three * self.AmbTemp_Three))
        self.inTempSave_Three = self.inTemp_Three
        self.delta_T_Three = self.inTemp_Three-self.setpointprev_Three

        if self.sSave_Three == 0 and action_Three == 1:
            self.minimumon_Three = min_on_Three
        elif self.sSave_Three == 1 and action_Three == 0:
            self.minimumoff_Three = min_off_Three
        self.sSave_Three = action_Three
```

```
if self.minimumon_Three == 0:
    self.minimumon_Three = 0
elif self.minimumon_Three >= 0:
    self.minimumon_Three = self.minimumon_Three - 1

if self.minimumoff_Three == 0:
    self.minimumoff_Three = 0
elif self.minimumoff_Three >= 0:
    self.minimumoff_Three = self.minimumoff_Three - 1
#fridge three end.....

#update states
self.state = (self.setpointstate_One,
              self.setpointstate_Two,
              self.setpointstate_Three,
              self.delta_T_One, self.minimumon_One, self.minimumoff_One,
              self.delta_T_Two, self.minimumon_Two, self.minimumoff_Two,
              self.delta_T_Three, self.minimumon_Three, self.minimumoff_Three)

# Reduce episode length by 1
self.episode_length -= 1

# Calculate reward
# fridge one.....
if self.minimumon_One > 0 and action_One == 0:
```

```
            reward_One = -MIN_ON_AND_OFF_TIME_PENALTY

        elif self.minimumoff_One > 0 and action_One == 1:

            reward_One = -MIN_ON_AND_OFF_TIME_PENALTY

        elif self.setpointprev_One == upper_temp_One and self.inTemp_One >= upper_temp_One:

            reward_One = setpoint_reward

        elif self.setpointprev_One == lower_temp_One and self.inTemp_One <= lower_temp_One:

            reward_One = setpoint_reward

        else:

            reward_One = -pow((self.inTemp_One-self.setpointprev_One), 2)

        # fridge one end ..........................................................................


        # fridge two..............................................................................

        if self.minimumon_Two > 0 and action_Two == 0:

            reward_Two = -MIN_ON_AND_OFF_TIME_PENALTY

        elif self.minimumoff_Two > 0 and action_Two == 1:

            reward_Two = -MIN_ON_AND_OFF_TIME_PENALTY

        elif self.setpointprev_Two == upper_temp_Two and self.inTemp_Two >= upper_temp_Two:

            reward_Two = setpoint_reward

        elif self.setpointprev_Two == lower_temp_Two and self.inTemp_Two <= lower_temp_Two:

            reward_Two = setpoint_reward

        else:

            reward_Two = -pow((self.inTemp_Two-self.setpointprev_Two), 2)

        #fridge two end..........................................................................
```

```
        # fridge three.....

        if self.minimumon_Three > 0 and action_Three == 0:

            reward_Three = -MIN_ON_AND_OFF_TIME_PENALTY

        elif self.minimumoff_Three > 0 and action_Three == 1:

            reward_Three = -MIN_ON_AND_OFF_TIME_PENALTY

        elif self.setpointprev_Three == upper_temp_Three and self.inTemp_Three >= upper_temp_Three:

            reward_Three = setpoint_reward

        elif self.setpointprev_Three == lower_temp_Three and self.inTemp_Three <= lower_temp_Three:

            reward_Three = setpoint_reward

        else:

            reward_Three = -pow((self.inTemp_Three-self.setpointprev_Three), 2)

        #fridge three end.....


        reward = reward_One + reward_Two + reward_Three


        if reward >= 180 and reward <= 230:

            reward = 200

        elif reward >= 380 and reward <= 430:

            reward = 200


        # Check if episode is done

        if self.episode_length <= 0:

            done = True

        elif reward_One == -MIN_ON_AND_OFF_TIME_PENALTY:

            done = True
```

```
elif reward_Two == -MIN_ON_AND_OFF_TIME_PENALTY:
    done = True
elif reward_Three == -MIN_ON_AND_OFF_TIME_PENALTY:
    done = True
else:
    done = False

#Save the plot variables
Internal_Temps_One.append(self.inTemp_One)
Fridge_OnOff_States_One.append(action_One)
Ambient_Temps_One.append(self.AmbTemp_One)
Step_rewards_One.append(reward_One)
Internal_Temps_Two.append(self.inTemp_Two)
Fridge_OnOff_States_Two.append(action_Two)
Ambient_Temps_Two.append(self.AmbTemp_Two)
Step_rewards_Two.append(reward_Two)
Internal_Temps_Three.append(self.inTemp_Three)
Fridge_OnOff_States_Three.append(action_Three)
Ambient_Temps_Three.append(self.AmbTemp_Three)
Step_rewards_Three.append(reward_Three)
upper_One.append(upper_temp_One)
lower_One.append(lower_temp_One)
upper_Two.append(upper_temp_Two)
lower_Two.append(lower_temp_Two)
upper_Three.append(upper_temp_Three)
lower_Three.append(lower_temp_Three)
```

```
        Total_power.append((power_usage_One * action_One) + (power_usage_Two *
action_Two) + (power_usage_Three * action_Three))

        Total_powerlimit.append(60)



        #update setpoint fridge one.....

        if self.inTemp_One >= upper_temp_One:

            self.setpoint_One = lower_temp_One

            self.setpointstate_One = 0

        elif self.inTemp_One <= lower_temp_One:

            self.setpoint_One = upper_temp_One

            self.setpointstate_One = 1

        elif self.inTemp_One < upper_temp_One and self.inTempSave_One >
lower_temp_One:

            self.setpoint_One = self.setpoint_One

            self.setpointstate_One = self.setpointstate_One

        self.setpointprev_One = self.setpoint_One

        # fridge one end.....



        #update setpoint fridge two.....

        if self.inTemp_Two >= upper_temp_Two:

            self.setpoint_Two = lower_temp_Two

            self.setpointstate_Two = 0

        elif self.inTemp_Two <= lower_temp_Two:

            self.setpoint_Two = upper_temp_Two

            self.setpointstate_Two = 1
```

```
        elif self.inTemp_Two < upper_temp_Two and self.inTempSave_Two > lower_temp_Two:

            self.setpoint_Two = self.setpoint_Two

            self.setpointstate_Two = self.setpointstate_Two

        self.setpointprev_Two = self.setpoint_Two

        #fridge two end.....

        #update setpoint fridge three.....

        if self.inTemp_Three >= upper_temp_Three:

            self.setpoint_Three = lower_temp_Three

            self.setpointstate_Three = 0

        elif self.inTemp_Three <= lower_temp_Three:

            self.setpoint_Three = upper_temp_Three

            self.setpointstate_Three = 1

        elif self.inTemp_Three < upper_temp_Three and self.inTempSave_Three > lower_temp_Three:

            self.setpoint_Three = self.setpoint_Three

            self.setpointstate_Three = self.setpointstate_Three

        self.setpointprev_Three = self.setpoint_Three

        #fridge three end.....


        # Set placeholder for info

        info = {}


        # Return step information

        return self.state, reward, done, info


    def render(self, mode='human'):
```

```
    global Internal_Temps_One
    global Fridge_OnOff_States_One
    global Ambient_Temps_One
    global Step_rewards_One
    global Internal_Temps_Two
    global Fridge_OnOff_States_Two
    global Ambient_Temps_Two
    global Step_rewards_Two
    global Internal_Temps_Three
    global Fridge_OnOff_States_Three
    global Ambient_Temps_Three
    global Step_rewards_Three
    global upper_One
    global lower_One
    global upper_Two
    global lower_Two
    global upper_Three
    global lower_Three
    global Total_power
    global Total_powerlimit
    if done:
        plt.subplot(231)
        plt.plot(Internal_Temps_One, 'r', label="Int Temp One")
        plt.plot(Fridge_OnOff_States_One, 'b', label="ON/OFF One")
        plt.plot(upper_One, 'g', linestyle='--')
        plt.plot(lower_One, 'g', linestyle='--')
```

```
plt.ylabel("Internal Temp (ºC)")

plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower left', ncol=2, mode="expand", borderaxespad=0.)


plt.subplot(232)

plt.plot(Internal_Temps_Two, 'r', label="Int Temp Two")

plt.plot(Fridge_OnOff_States_Two, 'b', label="ON/OFF Two")

plt.plot(upper_Two, 'g', linestyle='--')

plt.plot(lower_Two, 'g', linestyle='--')

plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower left', ncol=2, mode="expand", borderaxespad=0.)


plt.subplot(233)

plt.plot(Internal_Temps_Three, 'r', label="Int Temp Three")

plt.plot(Fridge_OnOff_States_Three, 'b', label="ON/OFF Three")

plt.plot(upper_Three, 'g', linestyle='--')

plt.plot(lower_Three, 'g', linestyle='--')

plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower left', ncol=2, mode="expand", borderaxespad=0.)


plt.subplot(234)

plt.plot(Step_rewards_One, 'm')

plt.ylabel('Step Rewards')

plt.xlabel("Time (min)")


plt.subplot(235)

plt.plot(Step_rewards_Two, 'm')

plt.xlabel("Time (min)")
```

```
plt.subplot(236)
plt.plot(Step_rewards_Three, 'm')
plt.xlabel("Time (min)")
plt.show()

plt.subplot(131)
plt.plot(Ambient_Temps_One, 'k')
plt.xlabel("Time (min)")
plt.ylabel("Ambient Temp (ºC)")

plt.subplot(132)
plt.plot(Ambient_Temps_Two, 'k')
plt.xlabel("Time (min)")

plt.subplot(133)
plt.plot(Ambient_Temps_Three, 'k')
plt.xlabel("Time (min)")
plt.show()

plt.plot(Total_power, 'y')
plt.plot(Total_powerlimit, 'g', linestyle='--')
plt.xlabel("Time (min)")
plt.ylabel("Total power (W)")
plt.show()
```

```
def reset(self):
    # reset setpoint fridge one.....
    self.setpoint_One = np.random.choice([upper_temp_One, lower_temp_One])
    self.setpointprev_One = self.setpoint_One
    if self.setpoint_One == upper_temp_One:
        self.setpointstate_One = 1
    elif self.setpoint_One == lower_temp_One:
        self.setpointstate_One = 0

    # reset Internal Temp fridge one
    self.inTemp_One = np.random.uniform(lower_temp_One, upper_temp_One)
    self.inTempSave_One = self.inTemp_One
    # reset delta Tin and Tsetpoint fridge one
    self.delta_T_One= self.inTemp_One-self.setpoint_One
    # reset minon and mioff fridge one
    self.minimumon_One = 0
    self.minimumoff_One = 0
    # reset start Ambient temp fridge one
    self.AmbTemp_One = np.random.uniform(21, 22)
    # reset on off state fridge one
    self.s_One = np.random.randint(0, 2)
    self.sSave_One = self.s_One
    # fridge one end.....
    # reset setpoint fridge two.....
    self.setpoint_Two = np.random.choice([upper_temp_Two, lower_temp_Two])
    self.setpointprev_Two = self.setpoint_Two
```

```
if self.setpoint_Two == upper_temp_Two:
    self.setpointstate_Two = 1
elif self.setpoint_Two == lower_temp_Two:
    self.setpointstate_Two = 0


# reset Internal Temp fridge two
self.inTemp_Two = np.random.uniform(lower_temp_Two, upper_temp_Two)
self.inTempSave_Two = self.inTemp_Two
# reset delta Tin and Tsetpoint fridge two
self.delta_T_Two= self.inTemp_Two-self.setpoint_Two
# reset minon and mioff fridge two
self.minimumon_Two = 0
self.minimumoff_Two = 0
# reset start Ambient temp fridge two
self.AmbTemp_Two = np.random.uniform(21, 22)
# reset on off state fridge two
self.s_Two = np.random.randint(0, 2)
self.sSave_Two = self.s_Two
#fridge two end.....
# reset setpoint fridge three.....
self.setpoint_Three = np.random.choice([upper_temp_Three, lower_temp_Three])
self.setpointprev_Three = self.setpoint_Three
if self.setpoint_Three == upper_temp_Three:
    self.setpointstate_Three = 1
elif self.setpoint_Three == lower_temp_Three:
    self.setpointstate_Three = 0
```

```
# reset Internal Temp fridge three
self.inTemp_Three = np.random.uniform(lower_temp_Three, upper_temp_Three)
self.inTempSave_Three = self.inTemp_Three
# reset delta Tin and Tsetpoint fridge three
self.delta_T_Three= self.inTemp_Three-self.setpoint_Three
# reset minon and mioff fridge three
self.minimumon_Three = 0
self.minimumoff_Three = 0
# reset start Ambient temp fridge three
self.AmbTemp_Three = np.random.uniform(21, 22)
# reset on off state fridge three
self.s_Three = np.random.randint(0, 2)
self.sSave_Three = self.s_Three
#fridge three end.....

# reset episode length
self.episode_length = Each_episode_steps

#reset states
self.state = (self.setpointstate_One,
              self.setpointstate_Two,
              self.setpointstate_Three,
              self.delta_T_One, self.minimumon_One, self.minimumoff_One,
              self.delta_T_Two, self.minimumon_Two, self.minimumoff_Two,
              self.delta_T_Three, self.minimumon_Three, self.minimumoff_Three)
```

```
        return self.state



env = ThreeFridge()

episodes = 1

for episode in range(1, episodes+1):

    state = env.reset()

    done = False

    score = 0


    while not done:

        action = env.action_space.sample()

        n_state, reward, done, info = env.step(action)

        score+=reward

        print('New State:{} Reward:{} Action:{} Done:{}'.format(n_state, reward, action,
done))

    print('Episode:{} Score:{}'.format(episode, score))

    env.render()


# In[18]:


#Create a Deep Learning Model with Keras

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten
```

```
from tensorflow.keras.optimizers import Adam


# In[19]:


states = len(env.observation_space)
actions = env.action_space.n
print(states)
print(actions)


# In[20]:


# Deep model (input layer(states), dense layer, dense layer, output layer (actions))
def build_model(states, actions):
    model = Sequential()
    model.add(Flatten(input_shape=(1,states)))
    model.add(Dense(120, activation='gelu'))
    model.add(Dense(32, activation='gelu'))
    model.add(Dense(actions, activation='linear'))
    return model


# In[21]:


model = build_model(states, actions)
model.summary()


# In[22]:
```

```
#Build Agent with Keras-RL
from rl.agents import DQNAgent
from rl.policy import BoltzmannQPolicy, GreedyQPolicy
from rl.memory import SequentialMemory

# In[23]:

def build_agent(model, actions):
    policy = BoltzmannQPolicy()
    #policy =  GreedyQPolicy()
    memory = SequentialMemory(limit=50000, window_length=1)
    dqn = DQNAgent(model=model, memory=memory, policy=policy,
                   nb_actions=actions, nb_steps_warmup=10, target_model_update=1e-2)
    return dqn

dqn = build_agent(model, actions)
dqn.compile(Adam(lr=1e-3), metrics=['mse'])
dqn.fit(env, nb_steps=50000, visualize=False, verbose=1)

#plot vars
Internal_Temps_One = []
Ambient_Temps_One = []
Fridge_OnOff_States_One = []
Step_rewards_One = []
Internal_Temps_Two = []
```

```
Ambient_Temps_Two = []

Fridge_OnOff_States_Two = []

Step_rewards_Two = []

Internal_Temps_Three = []

Ambient_Temps_Three = []

Fridge_OnOff_States_Three = []

Step_rewards_Three = []

upper_One = []

lower_One = []

upper_Two = []

lower_Two = []

upper_Three = []

lower_Three = []

Total_power = []

Total_powerlimit = []

env = ThreeFridge()

states = len(env.observation_space)

actions = env.action_space.n

model = build_model(states, actions)

dqn = build_agent(model, actions)

dqn.compile(Adam(lr=1e-3), metrics=['mae'])

dqn.load_weights('ThreeFridge60Dqn12032330000mae.h5f')


# In[25]:


scores = dqn.test(env, nb_episodes=1, visualize=False)
```

```
plt.subplot(231)

plt.plot(Internal_Temps_One, 'r', label="Int Temp One")

plt.plot(Fridge_OnOff_States_One, 'b', label="ON/OFF One")

plt.plot(upper_One, 'g', linestyle='--')

plt.plot(lower_One, 'g', linestyle='--')

plt.ylabel("Internal Temp (ºC)")

plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower left', ncol=2, mode="expand",
borderaxespad=0.)


plt.subplot(232)

plt.plot(Internal_Temps_Two, 'r', label="Int Temp Two")

plt.plot(Fridge_OnOff_States_Two, 'b', label="ON/OFF Two")

plt.plot(upper_Two, 'g', linestyle='--')

plt.plot(lower_Two, 'g', linestyle='--')

plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower left', ncol=2, mode="expand",
borderaxespad=0.)


plt.subplot(233)

plt.plot(Internal_Temps_Three, 'r', label="Int Temp Three")

plt.plot(Fridge_OnOff_States_Three, 'b', label="ON/OFF Three")

plt.plot(upper_Three, 'g', linestyle='--')

plt.plot(lower_Three, 'g', linestyle='--')

plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower left', ncol=2, mode="expand",
borderaxespad=0.)


plt.subplot(234)
```

```
plt.plot(Step_rewards_One, 'm')
plt.ylabel('Step Rewards')
plt.xlabel("Time (min)")

plt.subplot(235)
plt.plot(Step_rewards_Two, 'm')
plt.xlabel("Time (min)")

plt.subplot(236)
plt.plot(Step_rewards_Three, 'm')
plt.xlabel("Time (min)")
plt.show()

plt.subplot(311)
plt.plot(Ambient_Temps_One, 'k')
plt.xlabel("Time (min)")
plt.ylabel("Amb temp One(ºC)")

plt.subplot(312)
plt.plot(Ambient_Temps_Two, 'k')
plt.xlabel("Time (min)")
plt.ylabel("Amb temp Two(ºC)")

plt.subplot(313)
plt.plot(Ambient_Temps_Three, 'k')
plt.ylabel("Amb temp Three(ºC)")
```

```
plt.xlabel("Time (min)")

plt.show()


plt.plot(Total_power, 'r')

plt.plot(Total_powerlimit, 'g', linewidth=4, linestyle='--')

plt.xlabel("Time (min)")

plt.ylabel("Total power (W)")

plt.show()

# In[27]:


'''del model

del dqn

del env'''
```