JAMES COOK UNIVERSITY

COLLEGE OF SCIENCE & ENGINEERING

EG4012

Electronic Systems and Internet of Things Engineering

OPTIMISING AND DEPLOYING CONTROL SCHEMES FOR REGIONAL MICROGRIDS IN FIRST NATIONS COMMUNITIES

Jai Wilson

Thesis submitted to the College of Science and Engineering

in partial fulfilment of the requirements for the degree of

Bachelor of Engineering with Honours

(Electronic Systems and Internet of Things Engineering)

6 October 2023

Statement of Access

I, the undersigned, author of this work, understand that James Cook University may make this thesis available for use within the University Library and, via the Australian Digital Theses network, for use elsewhere.

I understand that, as an unpublished work, a thesis has significant protection under the Copyright Act and I do not wish to place any further restriction on access to this work.

_____ _____

Jai Wilson Date

Declaration of Sources

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

_____ _____

Jai Wilson Date

Abstract

The power industry is undergoing a once in a generation change, where political, economic and environmental factors are pushing the transition to renewable energy. The idea of a microgrid, where local renewable energy generation can save a community money and increase energy security, has gained traction from both academia and government agencies. Jumbun, a Fringe of Grid First Nations community in Far North Queensland, can benefit from the economic gain that a microgrid provides. This project assessed the feasibility of developing a microgrid Energy Management System (EMS) to reduce Jumbun's electricity costs, while also assessing the feasibility of deploying an EMS to a microgrid controller.

MATLAB's model predictive control (MPC) toolbox was used to develop an economic MPC EMS to reduce the energy cost for a small community. MATLAB coder was then used to deploy the developed EMS to a Raspberry Pi 4 as a proof of concept for deploying custom EMS to external embedded systems, acting as a microgrid controller.

Simulating the MPC EMS showed its success at reducing a community's electricity costs. It was seen that the MPC EMS can reduce a community's electricity cost further than a rule-based logic approach can, so long as two conditions are met. 1) An accurate forecaster is needed to forecast future load, Solar PV and energy price approximately 15 hours in advance. 2) The location's electricity costs must vary throughout the day. Throughout this process it was determined that the MPC EMS is better suited for reducing the costs of supplying electricity from the utility's perspective, rather than reducing the electricity bills of a community. Limitations in the modelling process were noted and discussed. It is recommended that the MPC EMS design and simulation process be conducted again when considering the presented limitations before determining the suitability for field deployment.

This project also found that using MATLAB coder to deploy an EMS to external embedded systems is feasible, and the deployed EMS yielded the same results as when simulated on the host computer verifying the deployment process. Future works have been outlined for testing the deployed EMS with a Real Time Digital Simulator to observe the EMS performance with a simulated microgrid and to develop the communications framework required for field deployment.

Acknowledgements

I would firstly like to thank both of my supervisors, Dr. Yang Du and Dr. Alan Louis for their assistance in getting me through this thesis.

Yang, your endless guidance through the technical and academic challenges of this thesis has been phenomenal. You have had unwavering willingness to answer my constant emails of how to write better and I am forever grateful.

Alan, thank you for coming up with a thesis topic which has both challenged and interested me. I am forever indebted to your knowledge of how the MPC toolbox works and to also help me see the bigger picture in how this research is meaningful for more than just an undergraduate thesis.

I would also like to thank Mitchell Tap and Jake Anderson for your constant mentoring and proofreading abilities. Hopefully one day I can return the favour for you both.

Finally, to my friends and family who have put up with me saying “Sorry, I have to work on my thesis” for the past eight months. Thank you for the constant support through the whole process. I promise it has been worth it.

Contents

List of Figures

List of Tables

List of Acronyms

AER – Australian Energy Regulator

BESS – Battery Energy Storage System

CHIL – Control Hardware in the Loop

CUT – Controller Under Test

DER – Distributed Energy Resources

DNSP – Distribution Network Service Provider

EMS – Energy Management System

ESS – Energy Storage System

FoG – Fringe of Grid

HIL – Hardware in the Loop

kW - kilowatt

kWh – kilowatt hour/s

MILP – mixed integer Linear Programming

MIMO – Multiple Input Multiple Output

MPC – model Predictive Control

MSS – minimum Service Standard

PCC – Point of Common Coupling

PLC – Programmable Logic Controller

PV – Photovoltaic

QEJP – Queensland Energy and Jobs Plan

RTDS – Real Time Digital Simulator

SAIDI – System Average Interruption Duration Index

SAIFI – System Average Interruption Frequency Index

SOC – State of Charge

1 Introduction

The power industry is undergoing a once in a generation change. Political, economic, and environmental factors have shaped the future of energy usage to incorporate cleaner, smarter solutions. This is evident with the release of the Queensland Energy and Jobs Plan (QEJP) in late 2022, showing that the future electricity grid in Queensland will incorporate a range of renewable energy generation technologies [1]. By 2035, the Queensland Government plans to introduce 25 GW of renewable energy generation capacity across the state, incorporating solar photovoltaic (PV) and wind energy generation. There is also large investment in battery, hydrogen and pumped hydro technologies to be introduced as part of the QEJP.

1.1 Jumbun Community Description

Small communities in regional Queensland have expressed their willingness to integrate renewable energy into their communities to help preserve the environment for future generations. One such community is Jumbun, a grid-connected First Nations community south of Tully, as shown in Figure 1-1. Jumbun Limited, the local Jumbun charity, undertook a community workshop to take control of their future, aiming to increase their use of green energy in the community while also increasing their energy security during bad weather events.

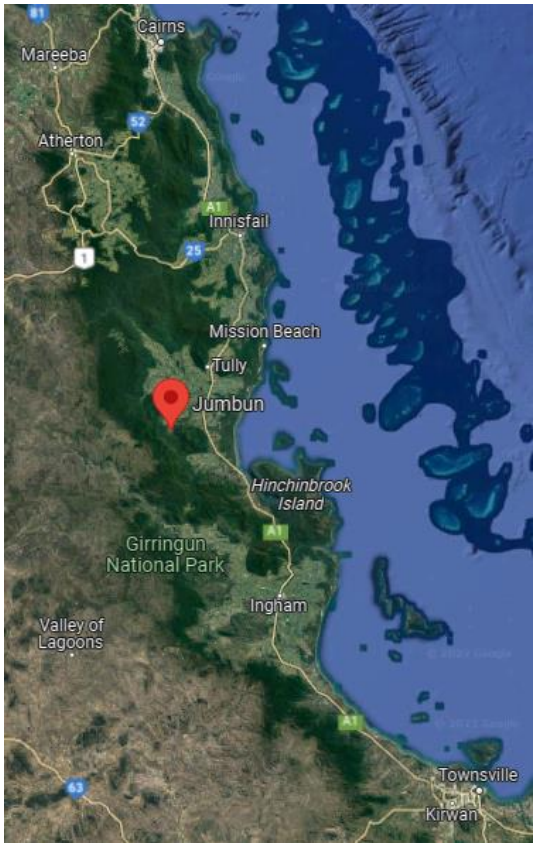

Figure 1-1 Jumbun Location

The community hall, which is run by Jumbun Limited, is the epicentre of the community, providing employment opportunities, Centrelink services and community initiatives. As the electricity grid is the only power supply in the community, there is a financial incentive to install local renewable generation to minimise electricity costs for Jumbun Limited. Reduction in electricity costs would allow Jumbun Limited to put more money back into funding community initiatives.

Jumbun experience poor electrical reliability due to their geographical location, as they are situated approximately 35 km in circuit length from the zone substation. Jumbun is referred to as a "Fringe of Grid (FoG)" location, as they are geographically remote, distant to the nearest population centres and located at the end of the electricity network [2]. FoG locations often experience poor electrical reliability as any upstream issues directly affect their electricity supply. Outages may be more frequent and last longer in duration due to powerline length, environmental factors and added callout distance for nearby powerline workers [2]. The Australian Energy Regulator (AER) provides two statistics which measure the reliability performance of Distribution Network Service Providers (DNSP) in Australia [3]:

1. System Average Interruption Duration Index (SAIDI), which measures the duration of an outage.
2. System Average Interruption Frequency Index (SAIFI), which measures the amount of outages.

Currently, Jumbun's SAIDI and SAIFI statistics are worse than allowable limits according to the Queensland government's Minimum Service Standard (MSS) [4]. Jumbun's geographical location is also prone to cyclones, which often cause electrical disruptions lasting several days, as restoration efforts during these times are often challenging. Jumbun needs a reliable electricity supply which is robust to upstream outages, which can also provide energy security during poor weather events.

1.2 Challenges Integrating Renewable Energy

Adding renewable energy generation to Jumbun will greatly reduce carbon emissions and electricity costs however, it will also create issues to the existing electricity distribution network. Renewable energy generation, such as solar PV and wind, rely heavily on weather conditions, which are inherently intermittent and unpredictable [5]. As wind speed and sunlight availability vary, the energy production from renewable sources also varies, which causes issues when balancing energy generation and demand in the outer electricity distribution grid. In addition, varying renewable energy generation causes voltage and frequency fluctuations, introducing unwanted power quality concerns. Instantaneous energy is

needed to absorb excess power from renewable energy or to export power in times of reduced output from renewable energy [5].

Interestingly, energy storage systems (ESS) have emerged as a solution to allow higher penetration of renewable generation sources. During times of high renewable energy production, the electricity grid may not always be able to accept the output from renewable energy generation. Consequently, power output from renewable energy must be curtailed, which effectively wastes energy [6]. In the short term, ESS can be used to import or export power on demand to smooth the "peaks" and "troughs" caused by intermittent renewable generation [6], [7]. ESS can also store energy for longer periods to shift when energy made from distributed renewable generation sources is used, i.e. in higher load times where it may be more effective [6].

The traditional electricity grid consists of centralised control of centralised generation assets. As the penetration of renewable energy generation and ESS increases, effectively controlling these energy resources at the distribution grid level becomes increasingly more difficult [8]. Microgrid technologies are becoming popular to assist with the integration of these distributed energy resources (DER), while also improving electricity reliability.

1.3 Microgrids

The US Department of Energy describes microgrids as "A microgrid is a group of interconnected loads and distributed energy resources within clearly defined electrical boundaries that acts as a single controllable entity with respect to the grid. A microgrid can connect and disconnect from the grid to enable it to operate in both grid-connected or island mode." [9]

Microgrids will consist of renewable energy generation, energy storage and loads, which can either operate with the main utility grid or operate autonomously in an islanded format. The microgrid can operate in both grid-connected and island mode through the operation of a Point of Common Coupling (PCC) with the main utility grid. Microgrids offer vast benefits by helping to increase the penetration of DER by balancing local energy generation and consumption, minimising energy losses through long distance transmission, but also by providing energy security during times of network outages [10]. A diagram of a typical microgrid topology is shown in Figure 1-2. Literature has been investigating the benefits of microgrids for many years now, with notable papers [10] and [11] being released in 2001 and 2004 respectively, which mainly focussed on the benefits of using microgrids for distributed generation while also explaining preliminary case studies alongside research and development efforts.

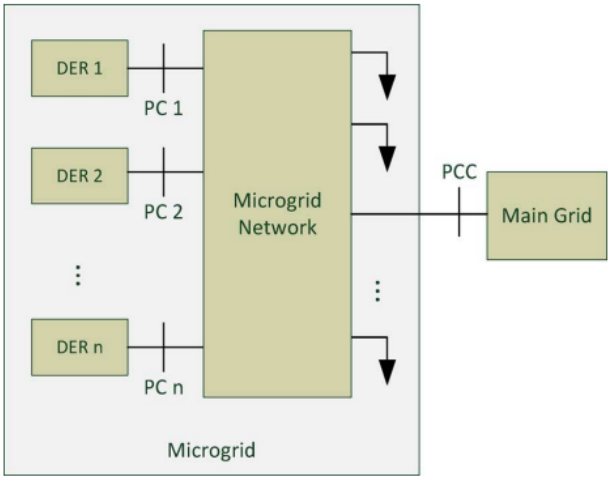

Figure 1-2 Microgrid Network Diagram [12]

Microgrids will typically have a central controller, which governs the operation and dispatch of DER within the microgrid. IEEE released standards governing the specification of microgrid controllers and the testing of microgrid controllers in [13] and [14], respectively. Energy Management Systems (EMS) extend basic control of microgrids. EMS can control both energy supply and demand, while also optimising the dispatching schedules of DER to realise energy and cost savings, and respecting microgrid technical constraints [15].

Introducing microgrid technologies to the Jumbun community presents vast opportunities by:

- Reducing Jumbun's carbon footprint by using renewable energy sources.
- Realising financial savings through utilisation of local electricity generation. This creates a secondary benefit of allowing the Jumbun Limited charity to use the financial savings on electricity to fund more community initiatives.
- Increasing Jumbun's electrical reliability through a microgrid's ability to operate islanded from the electricity grid.

1.4 Research Objectives

This thesis focuses on introducing a microgrid with an energy management system to the Jumbun community. In grid connected mode, the microgrid EMS can optimise the use of DER to minimise interaction with the main utility grid, reducing Jumbun's electricity costs. Jumbun's electrical reliability will also be improved as the microgrid can operate without a grid connection during electrical outages.

To achieve this objective, this project aims to investigate:

- The feasibility of an energy management system to help minimise Jumbun's interaction with the utility grid by maximising the use of local electrical generation

and energy storage. This will bring benefits such as financial savings to the Jumbun community and can also avoid infrastructure upgrades to supply the FoG community.

- The development of the EMS in widely used software for embedded hardware platforms.
- The feasibility and practicality of deploying an EMS to embedded hardware platforms.

1.4.1 Project Scope

This thesis only focusses on the grid connected operation of the microgrid. In islanded mode, there is less to optimise from an economic point of view as the main economic contributor to electricity supply, the electricity grid, is not present.

This thesis only focusses on linear relationships between DERs and loads within the microgrid. Even though nonlinear modelling may yield more accurate results, the complexity is too great for an undergraduate thesis.

No forecasting models have been included in this thesis. This is inclusive of load, solar PV and electricity price.

Protection of the microgrid has also not been considered in this thesis.

Even though these considerations are outside the scope of these works, they can certainly be included in future iterations of the research.

2 Literature Review

2.1 Microgrid Concept and Benefits

The microgrid concept was introduced to increase penetration and more effectively control DER when connected to the electricity grid. The commonly accepted definition of a microgrid was presented in 2012 by the U.S. Department of Energy from Section 1.3 [9].

Microgrid reviews conducted in [16], [17] agree that microgrids must have three distinct characteristics to align with this definition: microgrids must be identifiable and distinct from the rest of the electricity grid, microgrids must have an overarching controller to orchestrate DER and loads to act as a single controllable entity, and the microgrid must be able to operate with or without the electricity grid. The overarching controller to orchestrate DER and loads is the focal point of this research.

Microgrid benefits were discussed in the early to mid 2000's in [11], [18], [19]. The microgrid controller can control DER and loads locally, in a de-centralised sense, which can solve the control issues facing electricity grids presented in Section 1.2. This decentralised control (from the perspective of the electricity grid) allows microgrid benefits to be realised locally while also benefitting the distribution grid by balancing local electrical generation and consumption to allow increased penetration of distributed energy resources in the electricity grid [19].

A unique benefit presented by microgrids is the ability to operate disconnected, also known as islanded, from the main electricity grid. This helps to increase electrical reliability as a microgrid is still able to operate if the main electricity grid has an outage. FoG areas are prone to electrical outages, as any upstream outage directly affects these locations [20]. Using microgrids in poor electrical reliability locations, such as FoG locations, can greatly improve electrical reliability and allow these locations to still function in basic daily living.

2.2 Microgrid Control

The IEEE standard in [13] states that a microgrid control system should have real time control and energy management functions that can:

- Operation in grid-connected and islanded modes
- Automatic transition from grid-connected to islanded mode by providing a managed transition to islanded mode for microgrid loads during abnormal bulk power system conditions and planned interruptions of the system.
- Resynchronization and reconnection from islanded mode to grid-connected mode
- Energy management to optimize both real and reactive power generation and consumption.

- Ancillary services provision, support of the grid, and participation in the energy market and/or utility system operation, as applicable

A vastly documented method of control for microgrids is hierarchical control, as found in [12], [17], [21]–[24]. [12] explains that hierarchical control is necessary for microgrids to operate reliably locally while also maintaining control of interactions with the electricity grid. Hierarchical control levels are:

1. Primary control; used to control and stabilise voltage and frequency in the microgrid.
2. Secondary control; corrects voltage and frequency deviations from primary control.
3. Tertiary control; manages power flow between the microgrid and the main grid and the management of multiple microgrids.

Energy management systems and optimal control, which is the focus of thesis, in conjunction with hierarchical control are mentioned in [12], [17], [21], [23], [24], however it is not clear where EMS and optimal control lies in the hierarchy. [12], [21], [24] Define EMS to lie in the second layer, while [17], [23], believe that it lies in the third layer. This inconsistency of where the EMS lies in the control hierarchy is noted in [24].

This thesis focuses on the EMS and optimal operation of the microgrid along with the electricity grid, therefore the EMS and control systems that will be further discussed in this thesis will focus on the relevant levels of the control hierarchy (secondary/tertiary control). It is assumed that primary control will be handled by another control system for this project.

2.3 Energy Management Systems

Energy management in a microgrid typically consists of finding the optimal dispatch of DERs so that certain objectives are achieved [25]. This role is typically fulfilled by an EMS. A review of microgrid EMS is provided in [26], discussing that a microgrid EMS is a software that optimally controls DER in a microgrid to supply its load. The microgrid EMS receives relevant information about the operational characteristics of the microgrid such as load and DER forecasts, electricity market prices and technical constraints within the microgrid. The EMS then uses this information to control and schedule DER outputs to meet certain technical and economic criteria.

EMS were reviewed in [15], stating that there are two types of EMS: centralised and distributed. Centralised EMS consists of a central controller which acquires all information about the microgrid, such as DER outputs, cost functions and other relevant data. The centralised EMS will then make an optimal decision to govern the control of the microgrid. In a distributed EMS, a central controller will receive information from distributed local controllers of DER in a microgrid which all propose a current and future demand or generation

request. The microgrid central controller then bargains with local controllers for optimal operation while meeting local objectives.

Limitations of the centralised EMS are discussed in [27], explaining that centralised optimisation often requires additional computing power. This also raises the point that centralised EMS needs information about all DERs and loads in the microgrid which is not always feasible, as sometimes generation assets belong to different entities. Customers may also not want to disclose energy use or generation due to privacy.

2.4 Optimisation Methods for Energy Management Systems

A review provided in [28] defines optimisation methods as either online or offline optimisation. Offline optimisation is when the optimiser has a priori knowledge of the future system. This is where system information such as future loads, DER generations, and pricing signals are all known to the optimiser. Online methods do not have this perfect knowledge, where load and generation changes over time meaning that real time optimisation takes place. Often, offline methods are often used as an unattainable benchmark method to compare online methods to.

There are various optimisation methods that can be used for microgrid control, which have not been investigated in this paper due to the scope of this project. Authors in [15] review programming algorithms along with different optimisation methods, identifying their strengths and weaknesses.

2.5 Rule-based Logic EMS

Rule-based logic control is a simple control algorithm. Control system operation is determined by a finite set of rules and is typically controlled by a programmable logic controller (PLC), however it can be controlled from other devices too. It is up to the control systems engineer to correctly determine the systems operation under each rule, and how the system operation changes depending on system conditions. Rule-based logic with PLC has been used in embedded systems, transportation, production plants and power systems [29]. It is a simple yet robust method for control systems.

A study in [30] compared a rules based EMS and an optimisation based EMS which was tested in a microgrid testbed facility in Canberra. The study's goals were to evaluate the practical implementation of a commonly used method (rule-based) and a more modern method (optimisation based) with a real time simulator. The rule-based method followed a state machine system that represented use of different genset units, charging and discharging of the battery and curtailment of PV systems. The rule-based method had less computational time,

however the optimisation based method reduced operating costs per kWh of energy delivered more than the rule-based method did.

A study in [31] implemented a rule-based energy dispatch microgrid control scheme which was tested with comprehensive hardware-in-the-loop testing with a real time simulator. Developed control methodologies were heavily dependent on battery State of Charge (SOC). This paper found that the rule-based method was successful at maintaining battery SOC to a desired level, while also successfully using the battery for peak shaving. There was no comparison to other control methods.

A rule-based EMS is proposed in [32] for an isolated microgrid in China. The EMS was able to start and stop generators (diesel and wind turbine) and control an ESS. The EMS aimed at using the wind turbine generation as much as possible. Experiments take place to understand if the EMS can be verified, however experimental processes are not explained.

2.6 Model Predictive Control EMS

Model predictive control (MPC) has emerged as an advanced control method for microgrids which is supported by many literature articles. MPC is a type of optimisation-based control method that uses a process model to predict future system states due to both current and predicted future system inputs (control actions) [33]. MPC was first used in the early 1970's by Shell Oil engineers [34]–[36], however has recently proven to be successful in other industries such as refrigeration, power production plants, transportation networks and microgrids (as mentioned in [37]). MPC has attracted attention as an advanced control scheme as it can handle multiple input multiple output (MIMO) systems, while still operating within technical constraints of system behaviour [38].

A model predictive control algorithm contains three key components: 1) a predictive model; 2) a cost function, and 3) a solving algorithm [39]. At each time step, the MPC algorithm optimises a sequence of system inputs (control actions) so that it minimises the cost function over a pre-defined time period, known as the prediction horizon. The amount of control actions which are optimised within the prediction horizon is known as the control horizon. The cost function compares the predicted system output with a specified reference trajectory. From the optimal sequence of control actions that is chosen, only the first control action is executed. This process is repeated for each time step. A simple diagram of the MPC diagram is shown in Figure 2-1.

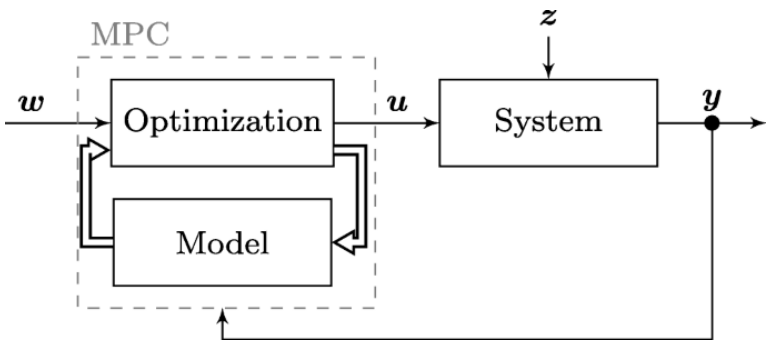

Figure 2-1 Model Predictive Control Diagram [40]

Any advanced microgrid EMS always involves multiple inputs and outputs to the control scheme, and technical constraints need to be obeyed to ensure reliable operation, which is why MPC lends itself to microgrid EMS. MPC is also quite attractive for EMS as it considers the impacts of current control actions on the future states of the system. MPC allows the anticipation of variations in DER power outputs, market prices and demands, which allow more effective control of DER [12].

A comprehensive review of using MPC for controlling microgrids is presented in [39]. This overview states that MPC for microgrids exists on two levels: converter level MPC which dictates the behaviour of power converters, or grid level MPC, which controls system level operation. Hierarchical controls with MPC were also mentioned, explaining that optimisation problems for microgrid operation sit in the tertiary layer. This paper mainly mentioned distributed MPC, MPC for islanded microgrids, and MPC for networked microgrids.

A review presented by Babayomi et al. in [41] explains the opportunities for MPC in microgrids in secondary and tertiary layers. This mentioned that MPC can either be implemented as a centralised strategy where the control is located at a central controller, or a decentralised strategy where the control is located at each local controller. Babayomi et al. also mention that seven optimisation techniques exist for autonomous microgrids: mixed integer linear programming, nonlinear programming, mixed integer quadratic programming, fuzzy prediction interval model, mixed logic dynamic, quadratic programming, and mixed integer bilinear programming.

2.6.1 Mixed Integer Linear Programming with MPC

Linear programming is an optimisation method which includes maximising, or minimising, a linear objective function subject to linear equality, and inequality constraints [42]. Mixed integer linear programming (MILP) is the same as linear programming, except a subset of variables are constrained to being integers. The MILP approach has been used widely in literature for microgrid EMS applications with MPC.

A MILP modelling approach was used in [43] with the rolling horizon strategy (similar to MPC) for a microgrid in Chile. The goals of the EMS were to reduce diesel usage, deliver generation set points to DER, control a water pump to maintain water levels in a tank, and send demand side management signals to customers. The MILP method with rolling horizon was compared to a regular unit commitment approach. It was found that the MILP model could reduce costs compared to the regular unit-commitment approach.

A MILP modelling approach was also used in [37] with MPC aiming to increase battery utilisation during high demand times to reduce electricity usage from the external grid (electricity market prices are provided), and increase utilisation of local energy generation. A simulation was conducted comparing the opposed MPC method with a reinforcement learning method from another paper. It was found that the MPC method yielded better results than the reinforcement learning method.

A MILP MPC approach is used in [44], which compared the MPC approach with a rule-based logic approach to minimise electricity bills of commercial buildings. The considered microgrid is only of a building scale (building with solar, PV, grid connection). MPC was found to reduce electricity bills by up to 34% compared to rule-based logic. However, time of use prices, power converter efficiencies and time-variant constraints are used.

Parisio et al. extended their own studies on using MILP for MPC in microgrids in [45]. Parisio at al. compared a heuristic method, single MILP method, MILP MPC method and a benchmark method (un-realistic, perfect future knowledge system). It was found that the MILP MPC was only slightly worse than the proposed benchmark method. The control system was verified on an experimental microgrid in Athens, finding that the MILP MPC method was able to successfully reduce running costs of the microgrid.

2.6.2 Distributed MPC methods

Multiple past works have suggested that distributed MPC is an effective EMS, where an MPC controller is located at each local element in the microgrid. Works completed in [46] used the distributed MPC method for a Shanghai microgrid. The authors in [46] mentioned that implementing distributed convex non-linear optimisation is faster than centralised mixed integer programming. Also, the distributed algorithm allows for more plug-and-play functionality as the structure of the microgrid adapts (adding or removing DER). The proposed method was compared to heuristic methods and a centralised MPC method. It was found that the distributed method was 1.2% worse than the centralised method, however computational times for the centralised method were much longer (3-12 minutes longer).

A distributed economic MPC (EMPC) method is proposed in [47], aiming to economically optimise the microgrid by utilising local generation, reduce power exchange with the grid and

extend the lifetime of the ESS. The distributed EMPC is a cooperative model, where sub-systems cooperate to achieve system-wide performance. This once again found that the distributed EMPC was much faster than centralised EMPC, while yielding similar results.

A distributed EMPC is also presented in [48]. However, this scheme was more customer focussed by creating a microgrid power market (MPM) as an economic incentive for microgrid users. This had its own logic control scheme for adjusting the MPM price. This method used dissipativity theory, which helped to benefit microgrid users while also maintaining performance levels of the microgrid.

2.7 Industrial Implementations and Embedded Applications

Works conducted in [49] found that Hardware in the Loop (HIL) approach is an effective technique to evaluate a microgrid EMS. HIL testing involves embedding a physical controller into a real time simulation system. A focus of this thesis is deploying an EMS to an embedded device which can act as a microgrid controller. If future works continue beyond this thesis, the next testing stage would be using control hardware in the loop (CHIL) testing. This uses a controller under test (CUT) which is interfaced with a real time simulator sending digital or analogue signals [49], [50]. This creates a closed-loop simulation where control schemes can be verified in real time before field deployment, as shown in Figure 2-2 below.

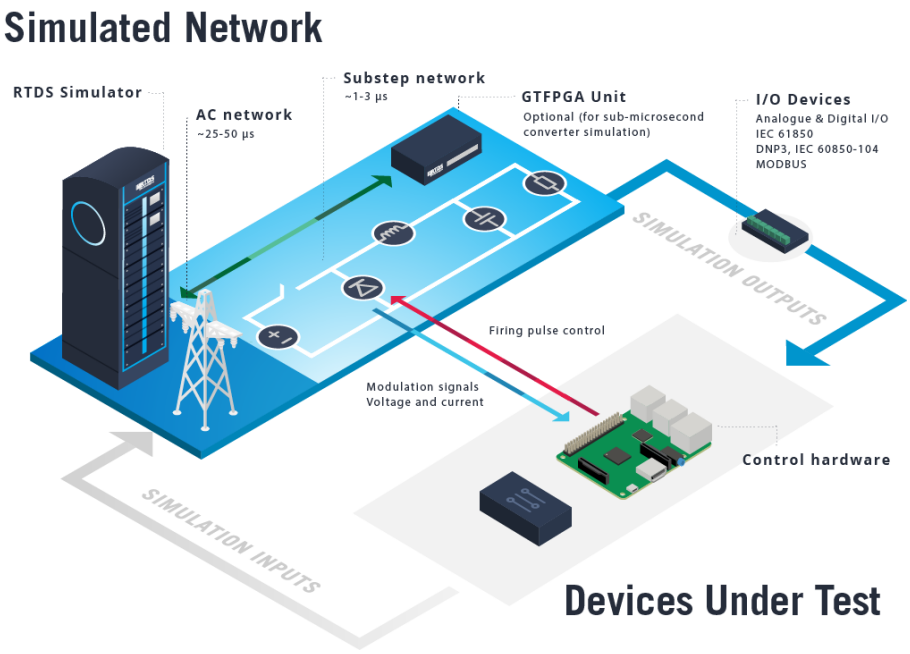

Figure 2-2 Control Hardware In The Loop (CHIL) Simulation Setup [51]

A suitable device for using CHIL testing is a Real Time Digital Simulator (RTDS) [52]. The RTDS can run highly detailed electromagnetic transient simulation in real time, with average

simulation time steps between 25 and 50 microseconds [53]. CHIL testing with the RTDS allows control schemes to be evaluated and developed in a safe environment before field deployment. The high speed simulation timesteps are as close to reality as can be feasibly achieved, allowing for high accuracy testing approaches.

A real time digital simulator from RTDS was used in [54] to test an MPC control scheme to optimise power flow between battery energy storage systems (BESS) distributed in a DC microgrid. The real time simulation verified the control scheme and showed the proper interaction between the model predictive controller and the DC-DC converter controllers.

A real time simulator from Opal Real Time is also used in [31] for a CHIL simulation of a rule-based logic EMS. The microgrid control scheme was deployed to a Raspberry Pi which was interfaced with the real time simulator through a TCP/IP socket.

2.8 Literature Gaps

The above literature review shows that there are gaps in research when investigating an MPC EMS using linear programming, such as MILP. When a MPC EMS using MILP is considered, there has been no mention of using the Model Predictive Control Toolbox in MATLAB or deploying the EMS to embedded hardware. The only mentioned use of MATLAB was in [45], however the Model Predictive Control Toolbox was not used. Thus, this project proposes to formulate an EMS using MPC with a MILP optimisation approach which will be deployed to embedded hardware for future CHIL testing with an RTDS. Deploying the EMS to embedded hardware shows the next step required to deploy an EMS to the field for microgrid operation.

3 Methodology

This chapter describes the methodology used to conduct the modelling and simulations of this project. For this project, an economic MPC EMS was formulated in MATLAB, similar to the approach in [37], using the Model Predictive Control Toolbox. This was simulated to understand the capabilities and suitability of the EMS at reducing the cost of supplying power to a community. Following this, the EMS was deployed to a Raspberry Pi 4, which will act as a proof of concept for an embedded system acting as a microgrid controller.

This chapter will take the following layout. First, MATLAB modelling approaches will be discussed in detail to describe the formulation of the MPC EMS. MATLAB is a suitable software for designing the MPC EMS as it has an MPC Toolbox, which is widely accessible and thoroughly documented. Adding to this, the process of deploying the EMS to the Raspberry Pi 4 will be discussed.

3.1 Linear (Implicit) MPC Design

The MATLAB MPC toolbox has two main types of MPC controllers which can be used: Linear MPC controllers and Non-Linear MPC controllers. MATLAB refers to an implicit MPC as a traditional, linear MPC where all equations used are linear and a standard quadratic program is solved. The implicit MPC controller model was initially chosen for this project as this modelling approach will only consider linear models like those specified in the above literature review.

This thesis used a Non-Linear MPC approach for the EMS, even though the specified problem and approach was linear. This is due to issues with the Linear MPC toolbox in MATLAB which are described in Section 3.1.3.

3.1.1 Plant Model Design

This section describes the plant model design which was used for the Implicit MPC controller. Model predictive controllers use a “plant model” as shown in Figure 2-1. This is a mathematical model of the system and is used to predict how control actions will affect the overall system, which allows the optimal sequence of control actions to be computed. To begin with, the plant model only considered the BESS in the microgrid. This was modelled using discrete-time state-space modelling which takes the below form:

$$\boldsymbol{State\ equation}:$$

$$X_{k+1} = Ax_k + Bu_k$$

$$\boldsymbol{Output\ equation}:$$

$$y_k = Cx_k + Du_k$$

Equation 3-1 Linear Discrete-Time State-Space General Form

State-space modelling is a process which uses state variables, system inputs and system outputs to describe the dynamics of a system. This system is a discrete-time system, where the state variables are updated at each sampling time. The variables in Equation 3-1 are described as:

x_k: state variables matrix at timestep k

y_k: Output variables matrix at timestep k

u_k: Input variables matrix at timestep k

A: State matrix

B: Input-to-state matrix

C: State-to-output matrix

D: Feedthrough matrix

State-space modelling was used to model the charging and discharging dynamics of the BESS with Equation 3-2:

$$\boldsymbol{State\ equation}:$$

$$SOC_{k+1} = SOC_k + \left(\frac{100\%}{Battery\ size} \times timestep\right) \times battery\ power\ input_k$$

$$\boldsymbol{Output\ Equation}:$$

$$SOC_k = SOC_k$$

Equation 3-2 Plant Model for Linear MPC

Where battery power input is measured in kW, battery size is measured in kWh and the sample time is measured in hours.

In this simple example, SOC is the only state variable and the only input for the system is the power input to the battery. Therefore, the only output to the system is the SOC. MATLAB

state space command requires an output equation, which is why it has been declared above even though it may seem redundant.

3.1.2 Implicit MPC Model and Simulation

The Implicit MPC controller was designed using the plant model specified above. The plant model is used to estimate the controller state and predict future plant outputs to calculate the optimal sequence of control actions.

MATLAB's implicit MPC controller is designed as a reference tracking MPC. Reference tracking MPC's optimise a sequence of control actions to regulate system outputs to a specified reference value. In this scenario, the MPC regulates the State of Charge of the battery to the specified reference value of 100%. The Implicit MPC controller was designed with the properties and constraints in Table 3-1 and Table 3-2 respectively.

Table 3-1 MPC Properties for Implicit MPC

MPC Property	Value
Prediction Horizon	10 steps
Control Horizon	2 steps
Timestep	1 hour
Output Reference	100%
Initial SOC	20%

Table 3-2 MPC Constraints for Implicit MPC

MPC Constraint	Value
Battery Discharge Limit	-50
Battery Charge Limit	50
Battery Discharge Rate of Change Minimum	-10 per step
Battery Charge Rate of Change Maximum	10 per step
SOC Minimum Limit	0 %
SOC Maximum Limit	100 %

The Implicit MPC controller was simulated in MATLAB. The goal of the MPC is to find the most optimal sequence of control actions to regulate the system outputs at the specified reference value in Table 3-1, while also obeying the specified constraints in Table 3-2.

The MATLAB script for the Implicit MPC is provided in Appendix A.

3.1.3 Implicit Model Limitations

The use of the implicit MPC controller was not continued past the foundational stages explained above. Linear MPC controllers in MATLAB have limited functionality where they can only be used for reference tracking. This is outlined in [55] where Mathworks explain the optimisation used to determine the optimal system inputs which are used in the MPC algorithm for the Implicit MPC controller.

To summarise [55], linear MPC models in MATLAB only use a standard cost function which minimises the difference between current system outputs and a specified reference value. Finding this difference and minimising it allows the MPC to regulate system outputs at the specified reference value. This standard cost function cannot be edited, meaning that linear MPC controllers in MATLAB can only be use for reference tracking.

As the standard cost function can't be edited, following some of the approaches outlined in the literature review aren't possible for linear MPC controllers in MATLAB. For instance, the approach in [37] is an economic MPC which minimises a cost function relating to the money spent on power in the prediction horizon. This type of economic MPC does not regulate system outputs to a specified reference value, rather it optimises control actions to minimise a specified performance criterion [56]. Hence, this type of approach is not possible for linear MPC controllers in MATLAB.

There are other parameters which are also used in the standard cost function, as explained in [55], however they are irrelevant in this situation.

As modifying the standard cost function is not possible for linear MPC controllers in MATLAB, a non-linear MPC controller was designed which will be outlined in section **Error! Reference source not found.** below.

3.2 Non-Linear Economic MPC for Microgrid Control

A non-linear MPC controller was used for this project instead of a linear MPC controller. This was due to the limited capabilities of the linear MPC controller as mentioned in section 3.1.3 above. MATLAB's non-linear MPC controller was designed for more complex problems and therefore has extra functionality that the linear MPC controller does not have. Non-linear MPC controllers allow a custom cost function and custom equality and inequality constraints.

Even though a non-linear MPC controller object is designed and used in MATLAB, the modelling process is still linear. Non-linear controllers aren't constrained to only being used for non-linear problems hence, they can still handle linear problems.

An economic MPC EMS was designed as the main simulation of this paper to answer the presented research questions in section 1.4. The economic MPC aims to find the most optimal

sequence of control actions to minimise the amount of money paid for energy in that period for a microgrid.

3.2.1 Plant Model for Economic MPC

The plant model for the economic MPC was a discrete time model which used 3 states, 2 outputs and 1 input. This was declared using the *nlmpc* MATLAB object function. The state equations for the system are represented in Equation 3-3 with the units of each variable represented in

. A summary diagram of the system can be seen in Figure 3-1.

The plant model for the economic MPC used net load, price of energy and battery SOC as state variables. This was declared as a function in a separate file which was in the same working directory as the main simulation script. The net load is the difference between load and the power produced by a renewable generator, in this case a solar PV system, and is described with Equation 3-4. When net load is positive, it represents that load is greater than solar PV. When it is negative, it represents the solar PV system generating power which is greater than the current load.

Battery power input is used as the input to the system and is shown in Equation 3-5. When this value is positive, the battery is charging. When it is negative, the battery is discharging.

Battery SOC and net grid usage were used as output variables. This was also declared as a function in a separate file. Net grid usage represents the difference between the battery power input and the net load, which is detailed in Equation 3-6. If the net grid is positive, power is flowing into the microgrid. Likewise, if net grid is negative, the microgrid is exporting power to the outer grid. Two output variables are needed to specify two reference trajectories for load and price, respectively. The reference profiles were used in conjunction with the cost function, which is described in section 3.2.2.

The MATLAB scripts for the state equation and output equation are in Appendix B and C, respectively.

State Equations:

$$SOC_{k+1} = SOC_{k+1} = SOC_k + \left(\frac{100\%}{Battery\ size} \times timestep\right) \times battery\ input_k$$

$$Net\ Load_{k+1} = Net\ Load_k$$

$$Price_{k+1} = Price_k$$

Output Equations:

$$SOC_k = SOC_k$$

$$Net\ Grid_k = Net\ Load_k + battery\ input_k$$

Equation 3-3 Plant Model for Microgrid Economic MPC

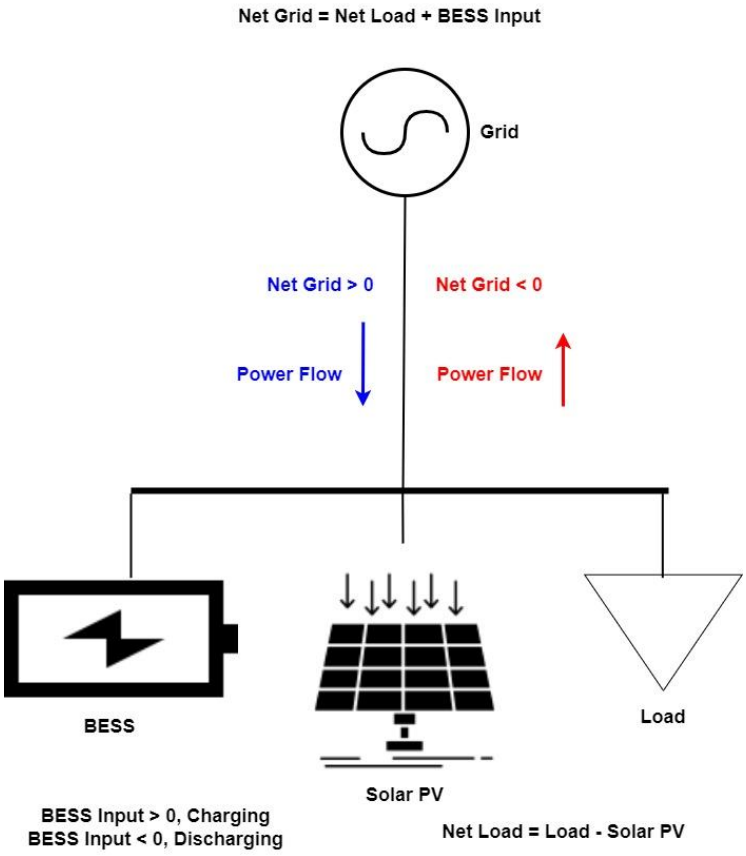

Figure 3-1 Microgrid Model for Economic MPC

Table 3-3 Economic MPC Plant Model Units

Variable	Units
SOC	% (Percentage)
Battery size	kWh (Kilowatt-hours)
Battery Input	kW (Kilowatts)
Timestep	Hrs (Hours)
Net Load	kW (Kilowatts)
Price	c/kWh (Cents per Kilowatt-Hour)
Net Grid	kW (Kilowatts)

$$Net\ Load_k = \begin{cases} Net\ Load > 0, & Load > Solar\ PV \\ Net\ Load < 0, & Load < Solar\ PV \end{cases}$$

Equation 3-4 Net Load Representation

$$Battery\ Input_k = \begin{cases} Battery\ Input > 0, & battery\ is\ charging \\ Battery\ Input < 0, & battery\ is\ discharging \end{cases}$$

Equation 3-5 Battery Input Representation

$$Net\ Grid_k = \begin{cases} Net\ Grid > 0, & Power\ Flowing\ into\ the\ microgrid \\ Net\ Grid < 0, & Power\ flowing\ out\ of\ the\ microgrid \end{cases}$$

Equation 3-6 Net Grid Representation

Reference Profiles

Reference profiles have been used in this thesis for both the net load and energy price states. The reference profiles consist of 24 values, which specify the value for these states for each hour of the day. These reference profiles are used to represent what net load and energy price would look like for a small microgrid in a real world environment. These profiles are used with the MPC EMS to understand the performance of the EMS when presented with some primitive values representing a microgrid's energy consumption and cost of energy.

During the simulation, the net load and price states update accordingly depending on the time of the simulation. The reference profiles for net load and energy price are shown in Figure 3-2 and Figure 3-3 respectively for each timestep throughout the day.

The net load profile follows the typical "Duck Curve" load profile which is often seen when analysing energy consumption in the modern day. Energy consumption moderately drops in

the early hours of the morning (2am to 4am) and picks up slightly as people tend to wake up at around 6am. The net load becomes negative as the solar PV system supports the load and generates excess power. A peak demand period is seen just after 6pm which is where solar stops producing and people use the most energy. Load then slowly reduces after the peak demand period.

The energy price profile follows a similar profile to what is specified in [37]. This profile is comparable to the load profile and also what is seen at the Australian utility scale, for example the NEM spot pricing dashboard in [57]. Electricity is very cheap in the middle of the day when renewable energy is producing, but more expensive in the evening when other non-renewable forms of electricity come online to supply large loads.

Both the net load and energy price profiles shown in Figure 3-2 and Figure 3-3 only show the change in net load and price for 1 day. These profiles repeat when the simulation lasts for multiple days. For an actual community, the net load profile will change every day, however the overall shape of the profile remains relatively consistent. This is the same case for energy price.

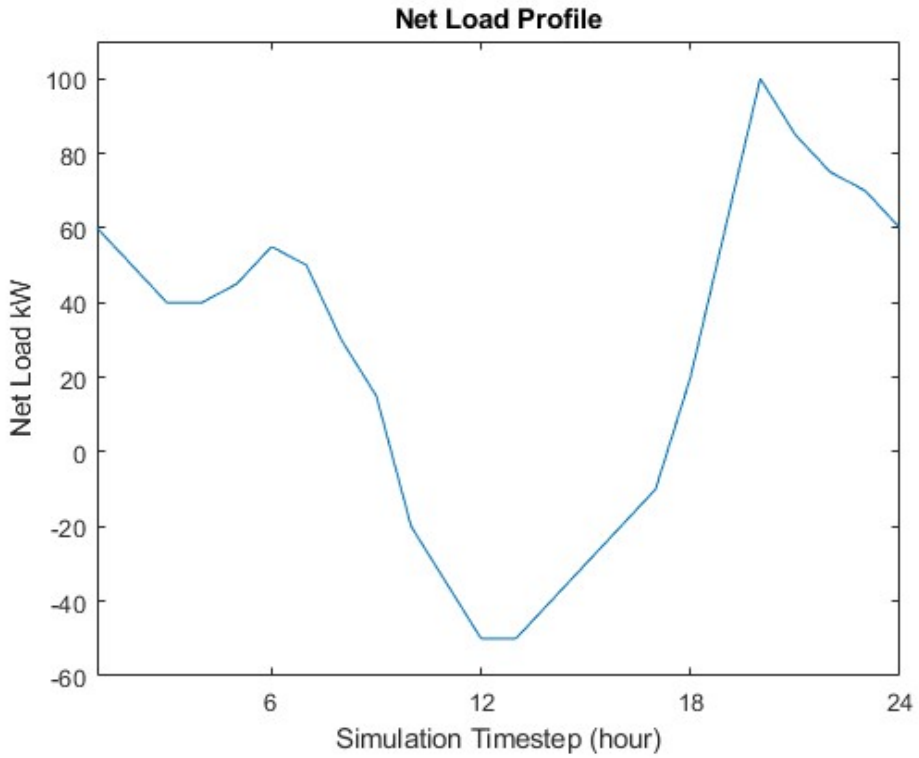

Figure 3-2 Load Profile

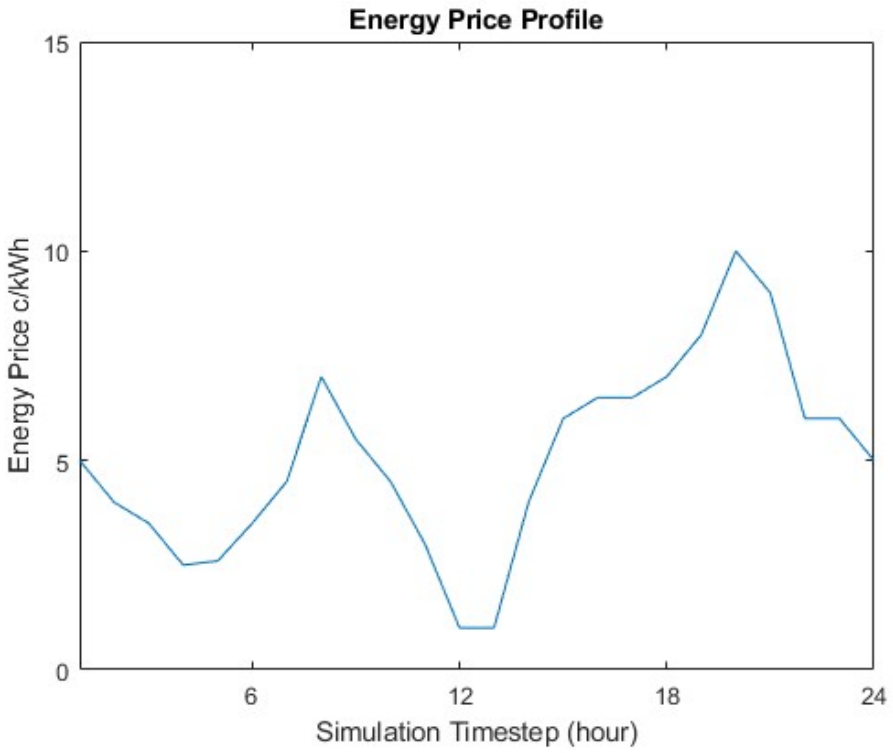

Figure 3-3 Price Profile

3.2.2 Cost Function

The goal of the economic MPC is to use the battery in a way, which minimises the amount of money paid for electricity. Two factors determine the amount of money paid for electricity: the net grid demand and the price of electricity. Equation 3-6 above has shown that the battery input directly affects the microgrid's demand from the outer grid. The MPC needs to judge when the best time is economically to charge and discharge the battery to minimise the amount of money paid for the simulation.

A custom cost function was declared, which incorporated the net grid demand and the price of electricity to determine the sequence of control actions that will reduce the cost function as much as possible. This cost function assumed perfect knowledge of future price and future net load for the prediction horizon of the MPC. This is shown in Equation 3-7

$$Cost_k = \sum_{k=1}^{10} \begin{cases} (net\ grid_k)(price_k), & grid_k > 0 \\ (net\ grid_k)(1), & grid_k < 0 \end{cases}$$

Equation 3-7 Cost Function

The cost function calculates the amount of money spent in the timestep by multiplying the net grid demand by the price of power for the timestep when the grid demand is positive. However, this is different when the net grid demand is negative i.e., when solar PV is producing and is exporting power to the grid. When the microgrid is exporting power to the outer grid the

amount of energy is only multiplied by 1c/kWh. It is common procedure for an export tariff to be much less than an import tariff.

The MATLAB code for the cost function can be seen in Appendix D.

Model Limitations

Section 3.2.2 mentions that the MPC EMS has perfect knowledge of future loads and energy price for the prediction horizon. This is a limitation of the model. Naturally, it is impossible to know what future loads will be however, a good estimation can be made. Electricity usage patterns tend to be very similar each day as can be seen from the Ergon Energy network demand monitor in Figure 3-4 from [58]. Even though this figure is on a state-wide scale, the same principle applies for a small microgrid. Using a perfect knowledge of future loads is a limitation however for a simple proof of concept project, it is not unreasonable. A more effective model would incorporate load, energy price and PV forecasting however, this is out of the scope of this project as mentioned in Section 1.4.1.

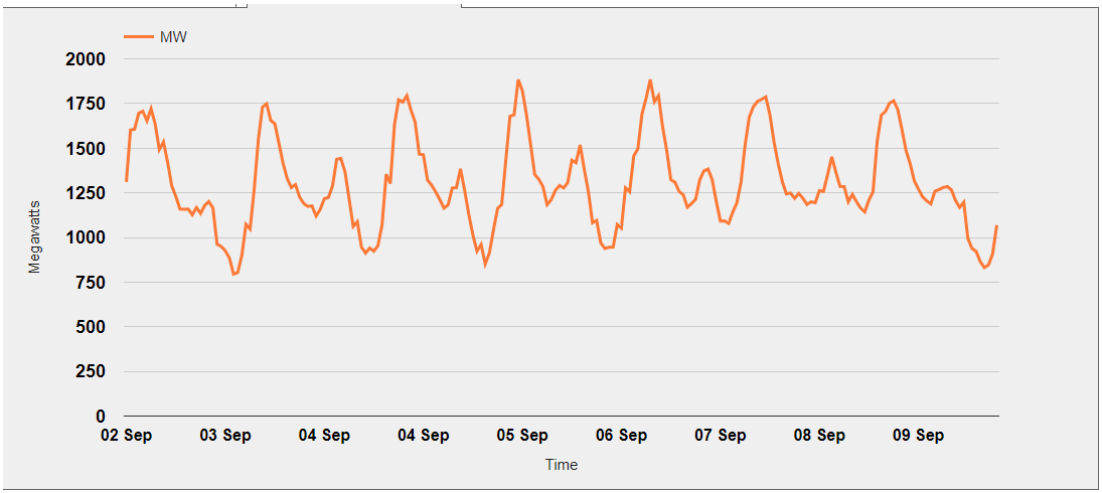

Figure 3-4 Ergon Energy Network Demand Monitor

Perfect knowledge of future electricity pricing is also considered. This is not as unrealistic as perfect knowledge of future loads as time based tariff pricing has been and is still being used in the utilities space.

3.2.3 Simulation Parameters and Constraints

A simulation was conducted to investigate the performance of the economic MPC EMS. The *nlmpcmove* function is used to compute the next optimal control action. This function takes in 6 values: the *nlmpc* object, current states, last control actions, reference values, measured disturbances and optional parameters.

The reference values are the values for net load and energy price, for the specified prediction horizon i.e., from the current timestep to the next *p* steps (where *p* is the prediction horizon). Typically, the references are values which the MPC attempts to regulate the output values at. However, the reference parameter has been used to pass forward the future state of the system to the cost function to optimise for.

As there are no measured disturbances in this model, this field was declared as an empty array.

There was only one optional parameter specified which was the simulation timestep length. This value is passed into the state function which is declared as a standalone function in a separate file. The functions of the MATLAB toolbox require the optional parameter to be declared in the state function, output function and *nlmpcmove* function even though it is only used once, hence it is also declared in the *nlmpcmove* function.

The simulation parameters and constraints are specified in Table 3-4 and Table 3-5, respectively.

Table 3-4 Economic MPC Simulation Parameters

Parameter	Value	Units
Timestep Length	1	Hours
Prediction Horizon	10-20 (changes depending on simulation)	Timesteps
Control Horizon	2-10 (changes depending on simulation)	Timesteps
Simulation Length	100	Timesteps
Battery Size	1000	kWh

Table 3-5 Economic MPC Simulation Constraints

Simulation Constraints		Value
State Limits	SOC State Max	100
	SOC State Min	0
	Load State Max	1×10^6
	Load State Min	-1×10^6
	Price State Max	1×10^6
	Price State Min	-1×10^6
Input Limits	BESS Power Input Max	50
	BESS Power Input Min	-50, -100

	BESS Power Input Rate of Change Max	1×10^6
	BESS Power Input rate of Change Min	-1×10^6
Output Limits	SOC Output Max	100
	SOC Output min	0
	Grid Output Max	1×10^6
	Grid Output Min	-1×10^6

The economic MPC was simulated with the above constraints as the base testing scenario and used a 1 MWh (1000 kWh) battery for the simulation. Parameters and constraints which show multiple values indicate where multiple values were tested to observe the change in behaviour of the system in later simulations.

Some of the constraints shown in Table 3-5 are either 1×10^6 or -1×10^6. For the simulation, these values did not need to be declared and could have been treated as treated as positive and negative infinity. However, there was concern that not declaring these values may have caused issues when generating C code, as it was unknown how the MATLAB infinity variable would translate to C during code generation. Therefore, the values were set as high numbers which would not interfere with code generation or the simulation.

The code used in the to simulate the MPC EMS can be seen in Appendix E.

3.3 Deployment to Raspberry Pi Hardware

This section describes the processes used to deploy the non-linear economic MPC EMS to a Raspberry Pi 4 using MATLAB coder. This process generates C code from the MATLAB script which ran the economic MPC. The generated C code was transferred to the Raspberry Pi 4, compiled and run. Deploying the EMS to the Raspberry Pi 4 hardware proved that EMS which are developed in MATLAB can be deployed to embedded systems which can act as microgrid controllers. This demonstrates the feasibility of developing an EMS in MATLAB which can be deployed to more robust hardware for advanced testing.

3.3.1 Raspberry Pi 4 Hardware

A Raspberry Pi is a small, low cost computer which can plug into an external monitor to be used [59]. Raspberry Pi's run on a Linux operating system and are used in many situations such as education, commercial industry and even startup businesses [60].

A Raspberry Pi 4 was an accessible embedded system which could run the generated C code from MATLAB. Deploying the code to the Raspberry Pi hardware acted as a proof of concept

for deploying custom EMS developed in MATLAB to embedded systems which can act as microgrid controllers.

Ideally, a robust embedded system or PLC is used to control large scale power systems, such as microgrids. Typically, these devices are extremely expensive, more than $10,000 Australian Dollars, which is out of budget for an undergraduate thesis. A Raspberry Pi computer is typically less than $150, and therefore has much less functionality than sophisticated embedded systems, meaning that a Raspberry Pi would not likely be used for controlling a microgrid in the field. However, using a Raspberry Pi for this project still shows the proof of concept for deploying EMS to embedded systems, while also providing developmental knowledge for future works.

Figure 3-5 below shows the Raspberry Pi 4 computer, while Figure 3-6 below shows a size comparison of a Raspberry Pi 4 to a person's hand. A summary of key technical specification is shown in Table 3-6

Figure 3-5 Raspberry Pi 4 Computer

Figure 3-6 Raspberry Pi 4 Size Comparison

Table 3-6 Summary of Raspberry Pi 4 Technical Specifications

Raspberry Pi Component	Details
CPU	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
RAM	2 GB
I/O	2 x USB 3.0
	2x USB 2.0
	Gigabit Ethernet
	2 x Micro HDMI
	4-pole stereo audio and composite video port
Storage	Micro-SD card slot for loading operating system and data storage
Power Supply	5V DC via USB-C connector
	5V DC via GPIO header

3.3.2 Simulation Changes

There are three main changes to the MATLAB script before code generation for the Raspberry Pi hardware takes place. Firstly, the *GetCodeGenerationData* function is called. Then, *nlmpcmove* is replaced with *nlmpcmoveCodeGeneration*, which takes different input arguments to the standard *nlmpcmove* function to allow code generation. Finally, the *fprintf* function is added to output result variables to a text file.

The *GetCodeGenerationData* function is called before the simulation loop begins. This function has 4 input variables: the *nlmpc* object, initial states, initial control action and optional

parameters. It returns two variables: *coreData* and *onlineData*. The *coreData* variable is a large struct which houses all the data which is normally stored in the *nlmpc* object. This is information such as constraints, variable weights and names of external functions which are used (state function, output function, cost function). It is essentially the *nlmpc* object but stored as a struct variable. The *nlmpc* object is proprietary to MATLAB meaning that it is not possible to represent it in C. Whereas struct variables can be represented in C which is why the coreData variable is needed for code generation. The coreData struct is constant and it does not change throughout the simulation.

The *onlineData* variable has a slightly different purpose. It is also a struct variable and it provides information such as reference values, optional parameters, optimal sequence of control actions and the state trajectories for the prediction horizon. This struct stores the information which updates with each time the MPC controller computes a control action, hence the "online" in the name of *onlineData*.

Once the *coreData* and *onlineData* structs have been generated from the *GetCodeGenerationData* function, they need to be manually declared, not from the *GetCodeGenerationData* function. This is because the *GetCodeGenerationData* function is not supported for MATLAB code generation. This is explained further in 3.3.3.

Then, the *nlmpcmoveCodeGeneration* function is called which using the MPC to compute the optimal control action. This function takes in 4 arguments: *coreData*, current states, last control action and *onlineData*. It returns the optimal control action and updates onlineData for the next iteration of the simulation.

Finally, the *fprintf* function was added in the MATLAB code which writes output variables to a text file. Before the code generation process, all visualisations of results were completed using MATLAB's inbuilt *plot* function. The *plot* function is not supported by code generation however, the *fprintf* function is. Therefore, the Raspberry Pi can write all data to a text file, which can be converted to a csv file and then visualised on the host computer using other tools such as python or excel.

The MATLAB script which was generated as C code can be seen in Appendix F.

3.3.3 Code Generation

After the script changes in 3.3.2 were conducted, the code generation process took place. This process converts all the MATLAB data handling and MPC processes to C code. For this process, some functions aren't compatible with MATLAB code generation. One of these is the *GetCodeGenerationData* function. This is an issue as the variables which are generated from *GetCodeGenerationData* are needed as inputs to *nlmpcmoveCodeGeneration*.

Consequently, the *coreData* and *onlineData* variables need to be manually declared. The MATLAB code used to manually declare these variables is then generated to C code so that the system running the C code can declare the *coreData* and *onlineData* variables without the *GetCodeGenerationData* function.

The *codegen* function is used to generate the C code with specific function inputs to generate code for a Raspberry Pi. The *packNGo* function is then used to package the code into a compressed zipped file. The example in [61] shows this process. The generated code is then transferred to the Raspberry Pi, where it is then built with the *gcc* command on the Raspberry Pi Terminal. Building the code generates an executable which is then used to run the MPC on the Raspberry Pi.

The code which was used to generate the MATLAB code as C code is shown in Appendix G.

3.4 Rule-based Logic EMS for Comparison

A rule-based logic EMS was also developed in MATLAB to act as a comparison against the MPC EMS. The rule-based logic EMS was taken from [44], where a MILP MPC EMS was compared to a rule-based logic EMS. Figure 3-7 below shows the rule-based EMS which was developed in MATLAB. This EMS did not use any external MATLAB toolboxes and was formulated with a series of *if* statements, as the EMS below also has been.

The code for the rule-based logic EMS can be seen in Appendix H.

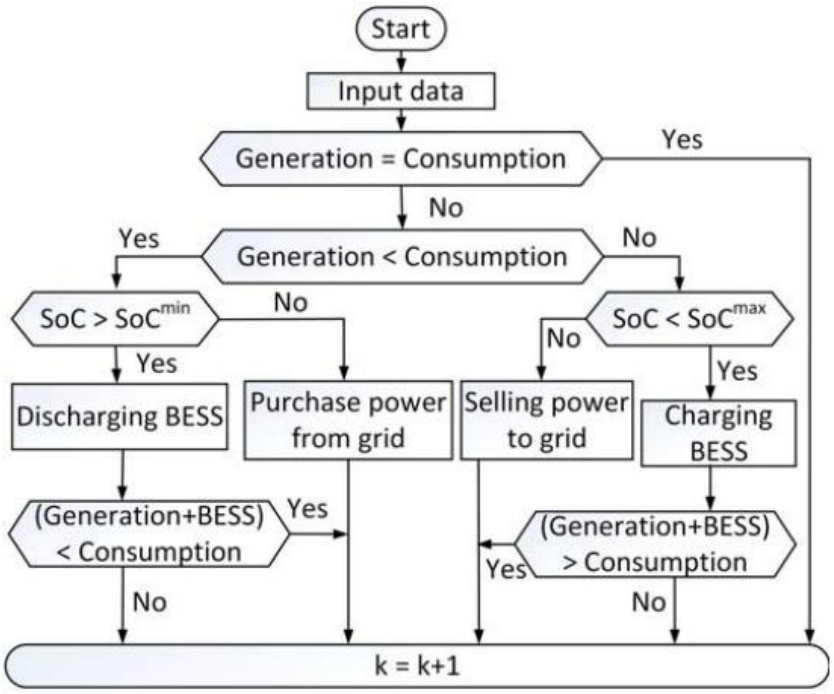

Figure 3-7 Rule-based Logic EMS from [44]

4 Results and Discussion

This chapter visualises the performance of the MPC EMS, which was designed using MATLAB's Model Predictive Control Toolbox. In some instances, this is compared to the rule-based EMS to illustrate the difference in performance between the two control schemes. The results of the MPC EMS which ran on the Raspberry Pi 4 are shown for one scenario as a proof of concept for industry deployment. Future works are discussed for advancing this project to the next stage of testing before field deployment.

The simulations shown in this chapter were conducted to answer the following research questions, which were also stated above in Section 1.4:

- The feasibility of an energy management system to help minimise Jumbun's interaction with the utility grid by maximising the use of local electrical generation and energy storage. This will bring benefits such as financial savings to the Jumbun community and can also avoid infrastructure upgrades to supply the FoG community.
- The development of the EMS in widely used software for embedded hardware platforms.
- The feasibility and practicality of deploying an EMS to embedded hardware platforms.

To answer these questions research questions, a range of simulations were conducted. The following hypotheses were formed:

1. Increasing the prediction horizon length will increase the MPC's ability to find the most economic use of the battery and minimise the amount of money paid for electricity. This assumes that future forecasting is sufficiently accurate.
2. Increasing the control horizon will also reduce the money paid for electricity, but the increase in performance will eventually plateau.
3. The MPC will outperform the rule-based logic EMS in both a varying load scenario and a flatter load scenario.

A range of scenarios were simulated to show the performance of the developed EMS's. Most of the simulations used the net load and price profiles specified in Figure 3-2 and Figure 3-3, however, a flat price profile was also used in one simulation.

4.1 Nonlinear Economic MPC Results

4.1.1 Standard Performance without BESS

A standard performance without any BESS was used to quantify the benefits of using a BESS, and then using the MPC EMS as well. If there is no BESS, the total energy taken from the grid is 3655 kWh. The net grid demand is 2635 kWh when solar exports are included. The total

amount of money paid for electricity $203.25, this includes the condition where solar exports are rewarded at 1c/kWh.

4.1.2 Varying Prediction and Control Horizon

The MPC algorithm has two fundamental parameters: the prediction horizon and the control horizon. The prediction horizon, *p*, determines how far ahead the MPC algorithm forecasts the system performance. The control horizon, *m*, determines how many free control moves within the prediction horizon that the optimiser computes for the MPC controller, where *m* is always less than or equal to *p*. When *m* is less than *p*, the last computed optimal control action is held constant for the remainder of the prediction horizon.

The prediction horizon was set to 10 timesteps, which represents 10 hours in simulation time. This was used as a starting point to measure the system's performance.

The control horizon was varied between 2 and 10 steps and is shown below in Table 4-1. A control horizon length of 1 was not used as it does not allow enough degrees of freedom for the MPC to optimise. The simulation scenario in Table 4-1 used a charge and discharge limit of 50 kW for the battery in the microgrid.

The simulation lasted for 100 hours, which is just over 4 days, and the battery size was 1 MWh (1000 kWh). The main metrics which are analysed are total net grid demand in kilowatt hours (kWh) and energy cost in dollars ($). The total net grid demand is the addition of all imported power, minus the exported power. The energy cost is calculated as c/kWh from Figure 3-3 when electricity is imported from the grid and is paid back at 1 c/kWh when electricity is exported to the grid.

Table 4-1 Varying Control Horizon for Charge and Discharge Limit of 50 kW

Prediction Horizon (Steps)	Control Horizon (Steps)	Net Grid Demand (kWh)	Energy Cost ($ Dollars)
10	2	1,735	101.71
10	3	1,735	104.42
10	4	1,735.01	103.62
10	5	1,735.09	101.71
10	6	1,735	102.77
10	7	1,735	104.69
10	8	1,735	106.41
10	9	1,735.09	106.37
10	10	1,737.48	106.99

Table 4-1 shows the difference in MPC performance as the control horizon is varied. Each of the scenarios above have very similar performance metrics to each other. Each control horizon length tested uses almost the exact same net grid demand but has slightly different total energy costs. This indicates that varying the control horizon does not greatly change the overall system behaviour. The behaviour of the MPC EMS with control horizon length of 2 steps and 9 steps are shown below in Figure 4-1 and Figure 4-2, respectively. These diagrams prove that the overall behaviour of the MPC EMS is extremely similar even though the control horizon has changed significantly.

The results in Table 4-1 have demonstrated that increasing the control horizon did not conclusively decrease the cost of energy for the simulation. The energy cost almost always increased when the control horizon increased (excluding the control horizon lengths form 2-4). This may be caused from the overall system changing at every timestep.

As mentioned, MATLAB's MPC toolbox is designed for reference tracking, where the reference that the MPC is trying to regulate system outputs to rarely changes. The MPC used in this scenario optimises based on minimising a performance metric, which is the cost of electricity. At each timestep of the simulation, the future net load and energy price changes from the previous timestep. Therefore, the optimal solution in the previous timestep may not be optimal anymore when new information is presented to the MPC. This could be why the overall performance does not increase when the control horizon does too.

Currently, the MPC is using equal weighting between each of the steps in the control horizon. There is a limit in how much future data can help the MPC EMS makes an optimal decision in the present time. It is unclear if this future knowledge limit has been reached in this simulation, as this EMS uses perfect future knowledge of the system i.e., forecasting future net load and pricing. Further investigation is needed to find this limit to find the optimal control horizon for the MPC EMS.

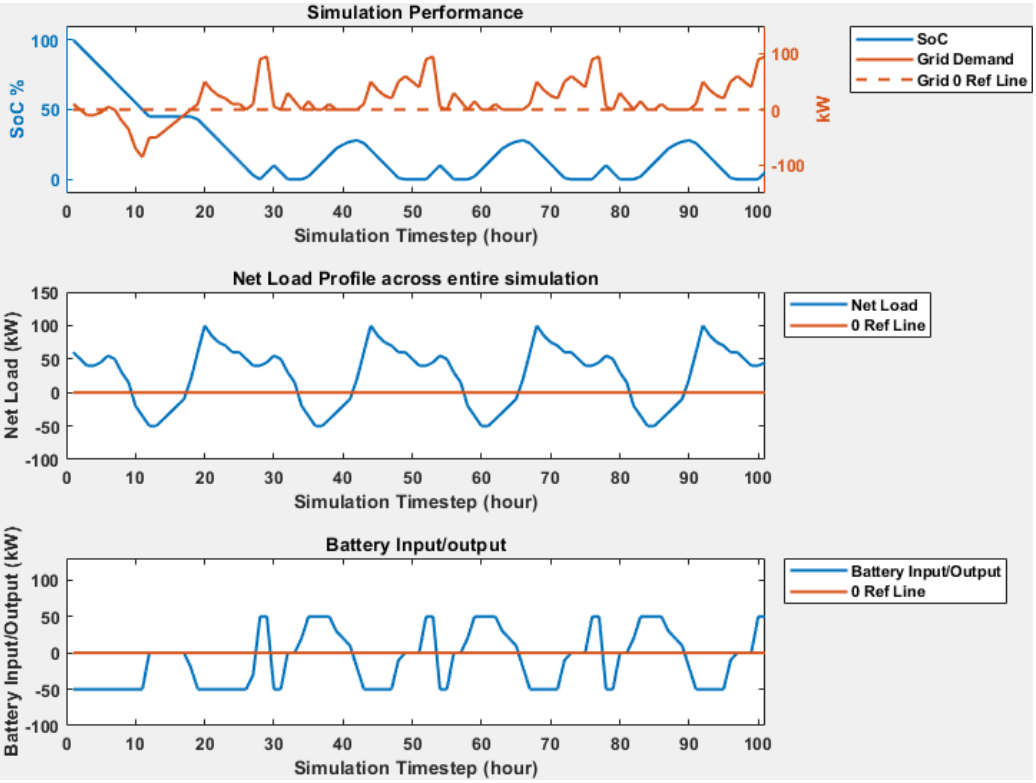

Figure 4-1 MPC Performance with a 50 kW Battery and Control Horizon Length of 2 Steps

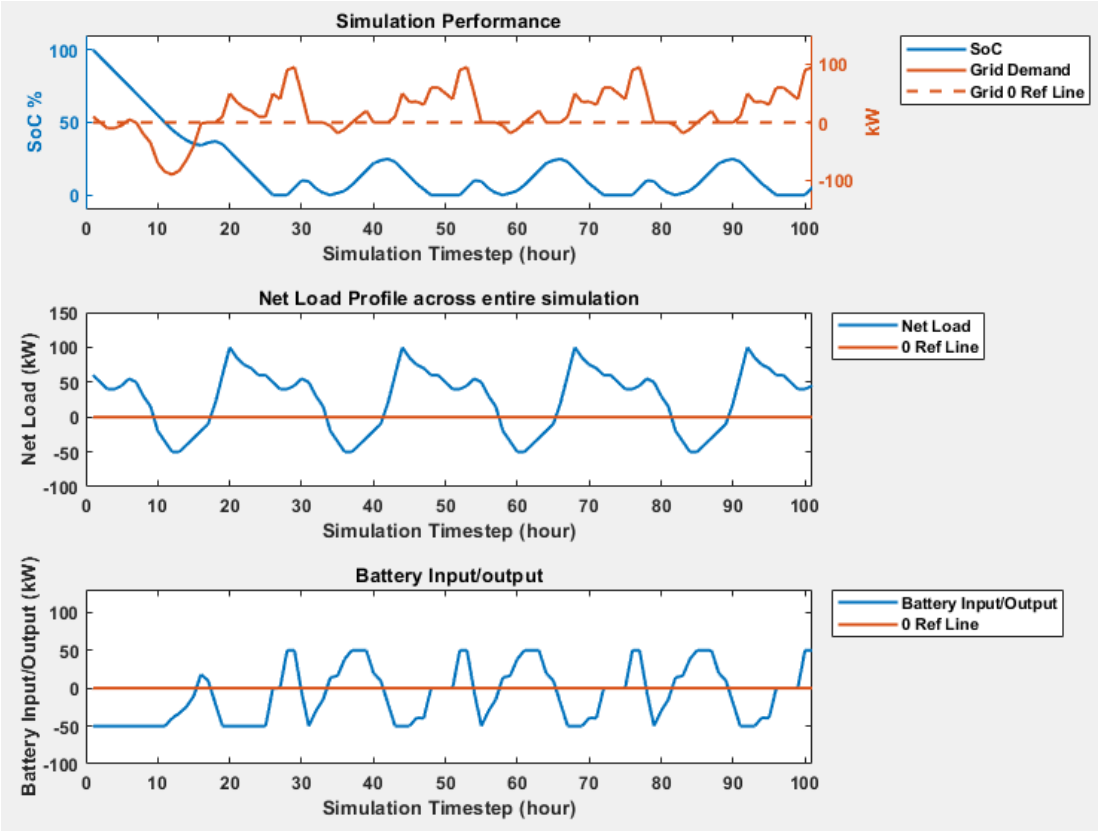

Figure 4-2 MPC Performance with a 50 kW Battery and Control Horizon Length of 9 Steps

Comparison to Rule-based Logic

The rule-based logic EMS was simulated with the same 50 kW charge and discharge limitation that the MPC EMS had. Its resulting behaviour can be seen in Figure 4-3. The rule-based logic EMS performs better than the MPC EMS in this scenario as the rule-based EMS draws 1635 kWh from the grid and the total energy price is $91.11.

Table 4-2 Rule-based Logic EMS Performance for Charge and Discharge Limit of 50 kW

Net Grid Demand (kWh)	Energy Cost (Dollars $)
1,635	91.11

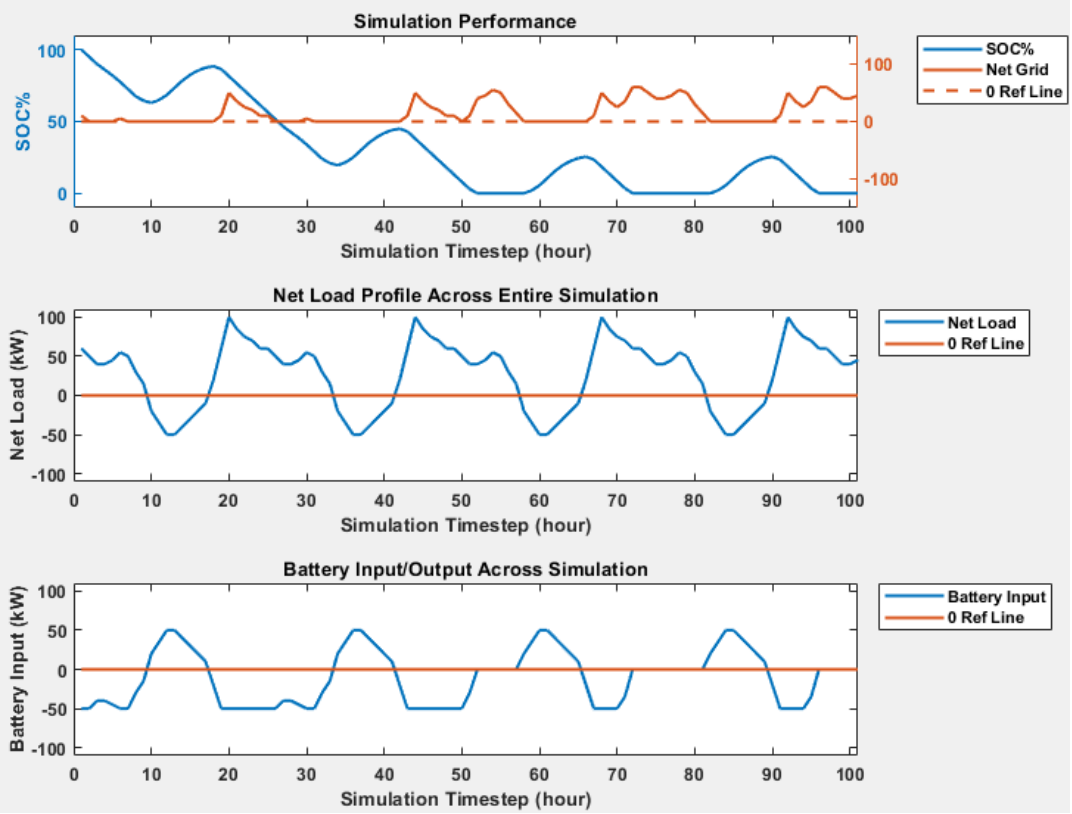

Figure 4-3 Rule-based Logic EMS with 50 kW Battery Limit

The rule-based logic EMS will always charge the battery when there is excess solar unless the battery is already full. However, the MPC EMS will only charge the battery when it determines that it is the most economical solution. These behaviours mentioned can be seen when analysing timesteps 0 to 20 on Figure 4-1, Figure 4-2 and Figure 4-3. In both MPC EMS scenarios (Figure 4-1 and Figure 4-2), the MPC has determined that it is more economically optimal to either do nothing, or a mixture of discharging and charging, in these early timesteps of the simulation when solar is available. This is because there is still a high enough SOC for the specified prediction horizon, so the value is not seen in charging the battery up during this

period. Whereas the rule-based logic utilised the available solar as much as possible when it was available in this same period. The consequence of the MPC EMS not charging the battery early is then seen over the next two days, where the MPC EMS depletes the battery much earlier than the logic EMS (~30th timestep for MPC EMS compared to the 50th timestep for logic EMS).

It is evident that the MPC EMS is trying to optimise for economy when it performs a small charging event each morning when there is a small dip in net load and price. The MPC has seen that even though there is not excess solar, it is more economical to charge from the grid when there is a quick dip in price and load. The rule-based logic EMS is not able to see this information, hence it only charges when solar is available.

The above simulations have relatively similar behaviour towards the end of the simulation, where the battery charges from the excess solar and then supplies the load where possible. As mentioned above, the biggest difference in behaviour is seen early in the simulation, in timesteps 0-20, where the MPC EMS does not see the economic benefit in charging from the excess solar. This is most likely due to the prediction horizon of the MPC, where it cannot "see" far enough ahead to understand the economic benefits of charging while the SOC is still so high.

To improve this, the prediction horizon was increased from 10 to 15 steps, with the control horizon kept at 2 steps. The behaviour of this system can be seen below in Figure 4-4. This system can reduce energy costs to $88.18, which is less than the rule-based logic EMS for the same scenario. These simulations show that when the prediction horizon is increased, the MPC EMS retains a higher SOC in the battery for longer compared to then a shorter prediction horizon was used in Figure 4-1 and Figure 4-2.

Table 4-3 MPC EMS with Increased Prediction Horizon for CHarge and Discharge Limit of 50 kW

MPC EMS			
Prediction Horizon	Control Horizon	Net Grid Demand (kWh)	Energy Cost (Dollars $)
15	2	1,735	88.18

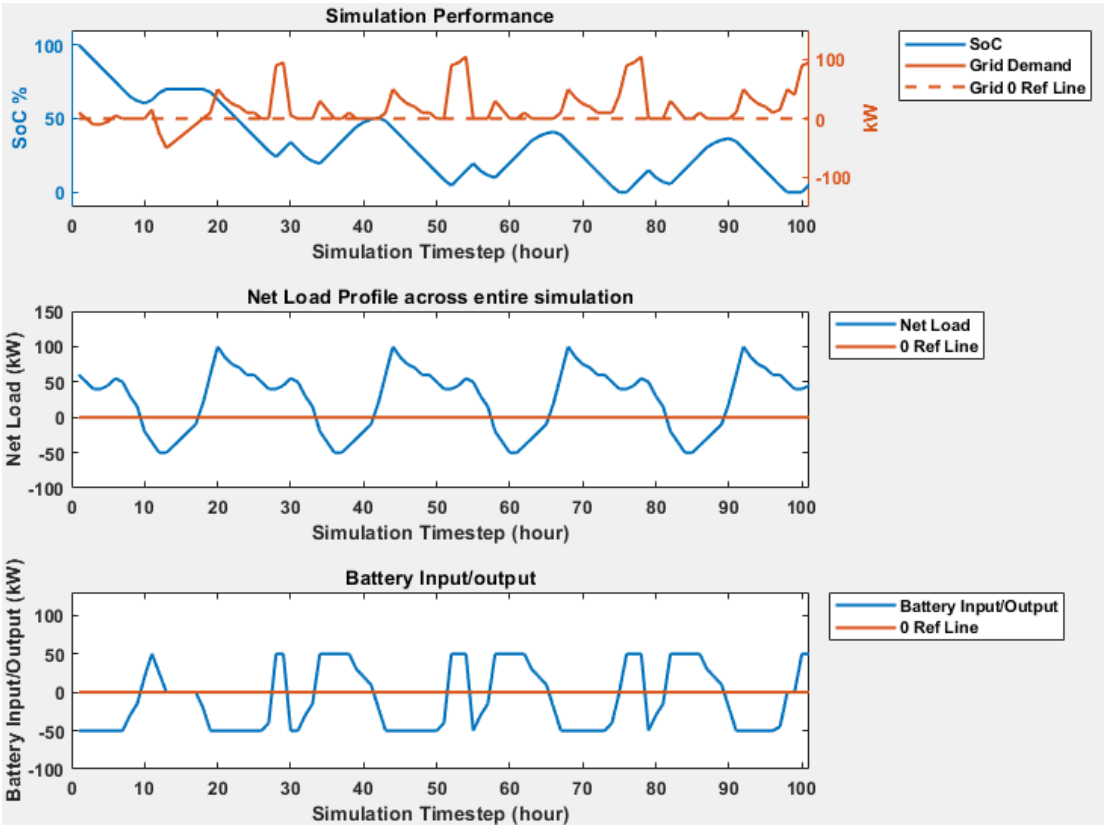

Figure 4-4 MPC Performance with 50 kW Battery Limit and Prediction Horizon of 15 steps

Increasing Battery Discharge to 100 kW

The above simulations used a battery charge and discharge limit of 50 kW. As a peak load of 100 kW is seen in the simulation, the simulation parameters were changed to allow a maximum discharge of 100 kW for both the logic EMS and the MPC EMS. To keep the comparison between the EMS fair, the charging rate was kept at 50 kW. The logic EMS only charges when solar is available, and in this scenario the peak excess solar generation is 50 kW. Therefore, it would be an unfair comparison if the charging limit is increased above 50 kW for the MPC EMS as the logic EMS will never be able to do this.

Table 4-4 MPC and Rule-Based Logic EMS Comparison for 100 kW Discharge Scenario

MPC EMS				Rule-Based Logic EMS	
Prediction Horizon	Control Horizon	Grid Demand (kWh)	Energy Price (Dollars $)	Grid Demand (kWh)	Energy Price (Dollars $)
10	2	1,735	72.32	1,635	76.51
15	2	1,735	61.17		

Table 4-4 shows that the MPC EMS can perform better than the logic EMS when the maximum discharge rate can fully cover the load at 100 kW. Both scenarios for the MPC EMS use more electricity from the grid compared to the logic EMS, however they both cost less money to service the load than the logic EMS. This shows that the MPC EMS can use more information allowing it to make a more economically optimal decision in when to use power form the grid. The performance of the MPC EMS using a 15 step prediction horizon is shown below in Figure 4-5.

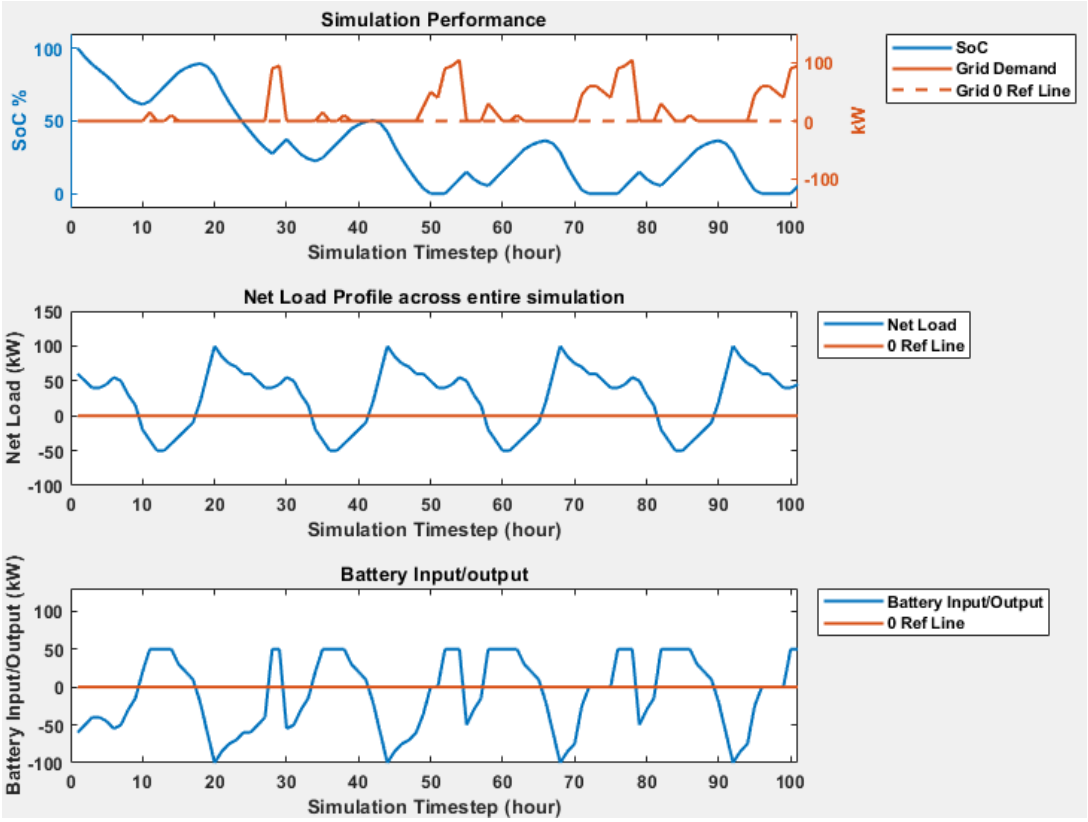

Figure 4-5 MPC Performance with Prediction Horizon of 15 Steps and Discharge Limit of 100 kW

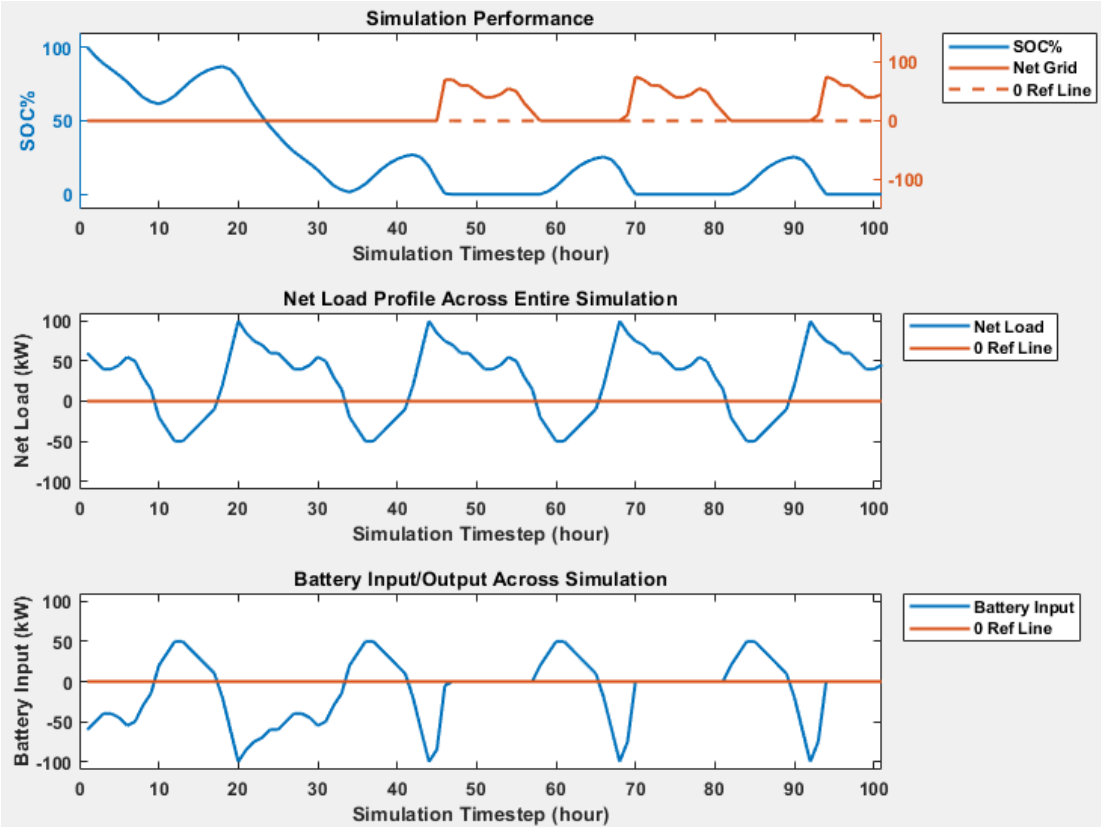

Figure 4-6 Rule-Based Logic EMS With 100 kW Discharge Limit

Figure 4-5 and Figure 4-6 show the performance of the MPC and rule-based logic EMS when the discharge limit is increased to 100 kW. As the MPC EMS has had its prediction horizon increased to 15 steps, it now has enough forward knowledge to charge in the first period where solar is available, which the MPC EMS has not done in previous simulations with smaller prediction horizons. Also, the MPC EMS sees that there is value in charging the battery a small amount when there is a dip in the load in the early morning, which the rule-based logic EMS cannot see. In total, Figure 4-5 shows the importance in the first day of the simulations for the overall performance of the MPC EMS in this simulation.

Simulations Summary, Limitations and Considerations

The above simulations show that the MPC EMS can theoretically reduce electricity costs further than the rule-based logic EMS if the prediction horizon is set far enough forward. However, this MPC EMS assumed perfect knowledge of future loads and future pricing signals. To implement an economic MPC EMS in industry, it would need to be combined with highly accurate forecasters to achieve this same predictability. This shows the benefit that accurate future forecasting provides. Additionally, the further forward that something is forecasted, the less likely that the forecast is true. This simulation has assumed equal weighting across all future timesteps when forecasting information is provided. Both factors need to be considered to determine the suitability of the Economic MPC EMS before field deployment.

As well, Table 4-1 does not conclusively show that the performance of the MPC EMS improves as the control horizon increases. Future work is required to determine the optimal control horizon to allow the MPC EMS to find the optimal control action for the present timestep.

Other simulation limitations were noted when comparing the MPC and rule-based logic EMS. Firstly, only one net load and varying price profile are compared above. Other profiles were not able to be tried due to time constraints with the project. Therefore, it is unclear if the chosen profiles help or hinder the performance of both EMS. Further development with legitimate values of load, solar PV and price profiles should be simulated to understand the suitability and limitations of both EMS. Secondly, both EMS have hard constraints of 0% SOC and 100% SOC. The 0% SOC constraint was reached multiple times throughout the simulations from both EMS. This prevented the MPC EMS from acting optimally and minimising the cost function further. The MPC EMS should be simulated using a battery size which is likely to be used for Jumbun to understand the EMS performance with a deployment ready microgrid.

Overall, the above mentioned points should be considered when analysing the comparison between the MPC EMS and the rule-based logic EMS. The comparison between EMS may not be viewed as a true comparison. There are certainly areas where this simulation can be improved, and should be improved, before determining the suitability of deployment to the field.

4.1.3 Simulation with a Flat Price Profile

The simulations conducted in Chapter 4.1 have used a varying net load and a varying energy price profile. However, remote fringe of grid communities in Queensland currently experiences a flat energy price tariff across the entire day.

To evaluate the limitations of the MPC EMS, a flat price profile of 5c/kWh was used for both the MPC EMS and the rule-based logic EMS. The varying net load profile used in Chapter 4.1 was also used for this comparison. It was hypothesised that simulating with a flat price profile the MPC EMS will outperform the logic EMS, however it will not do as well as previous scenarios with a varying price profile. Table 4-5 below shows the performance of both EMS with the flat price profile.

Table 4-5 MPC and Rule-Based EMS Performance Comparison for Flat Price Profile

MPC EMS				Rule-Based Logic EMS	
Prediction Horizon	Control Horizon	Grid Demand (kWh)	Energy Price (Dollars $)	Grid Demand (kWh)	Energy Price (Dollars $)

10	2	1,635	83.35	1,635	81.75
10	5	1,680.2	93.46		
15	2	1,635	81.75		
15	5	1,652.7	82.67		

The table above shows that when there is a flat price profile, the MPC EMS is only able to match the performance of the logic EMS at best. This is understandable, as when there is a flat price profile the benefit of charging the battery at cheaper times and discharging at expensive times is removed. The cheapest way to operate the microgrid is to charge the battery when it is free (when solar is available) and discharge when there is load. There is no further optimisation that can take place.

In fact, once the prediction horizon is increased far enough ahead, the MPC EMS behaviour is the same as the logic EMS behaviour. This is shown below in Figure 4-7 and Figure 4-8, where Figure 4-7 shows the MPC EMS with a 20 step prediction horizon and 2 step control horizon, and Figure 4-8 shows the rule-based logic EMS.

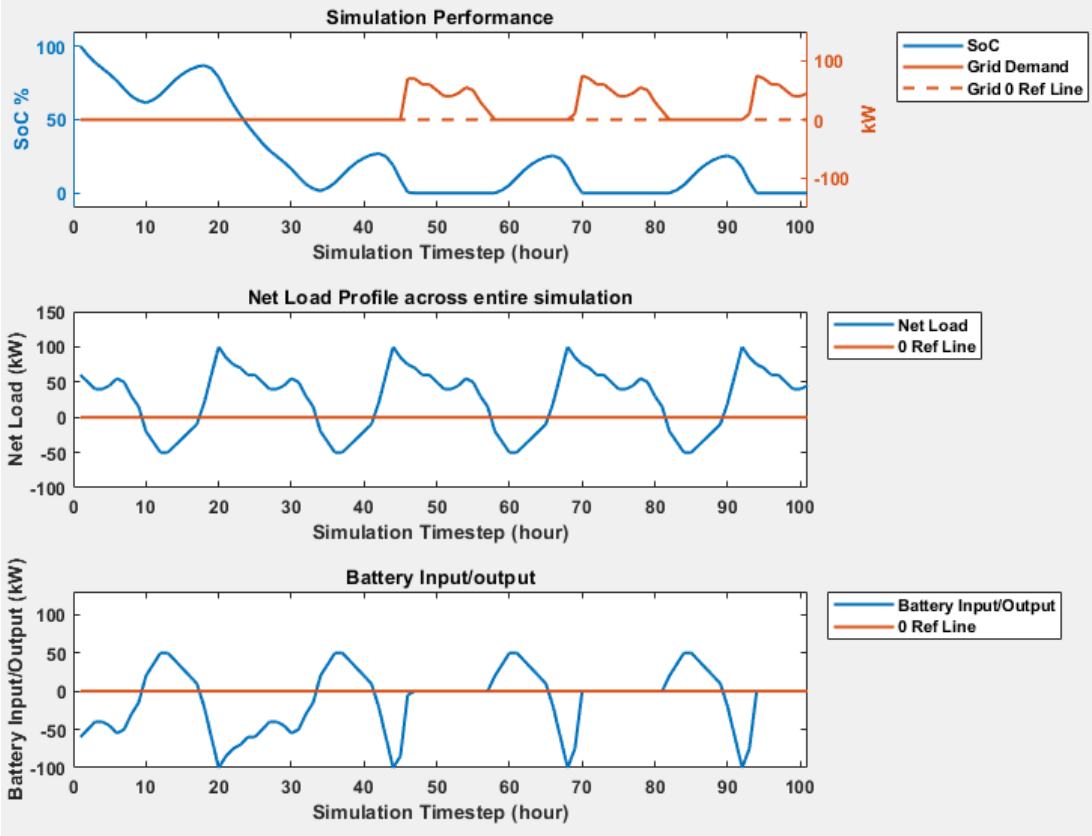

Figure 4-7 MPC EMS Behaviour with Flat Price Profile and 20 Step Prediction Horizon

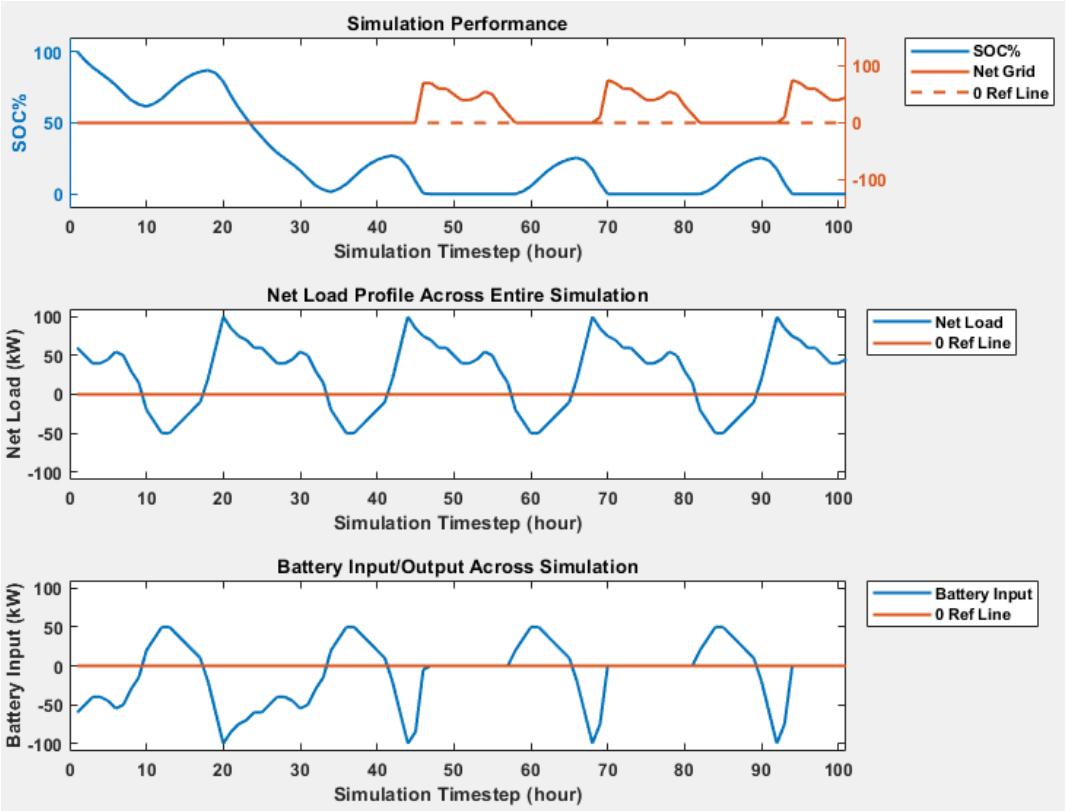

Figure 4-8 Logic EMS Behaviour with Flat Price Profile

Simulations Summary, Limitations and Considerations

The simulation above has shown that when a flat price profile has been used, the MPC EMS will eventually achieve the same behaviour as the rule-based logic EMS. Without a varying price profile, the economic benefit in charging the battery form the grid during cheap periods and discharging during expensive periods is removed. The most economical solution is charging the battery when solar is available and no power from the external grid is needed. This shows that the MPC EMS may not be suitable for reducing the costs of electricity for Jumbun, as Jumbun is charged at a flat rate of electricity.

The comparison between the EMS should once again be explored further with a different battery size and a different net load profile. In Figure 4-7 and Figure 4-8, the battery is depleted for large periods from roughly timesteps 45 to 55 and roughly timesteps 70 to 80. This constrains both EMS from using the battery at all and reducing the cost of electricity. The simulation should be conducted with different battery sizes and different net load profiles to investigate if any differences in performance arise between the two EMS for a flat price profile. Doing so will allow more accurate conclusions to be made to determine if the MPC EMS is suitable for fringe of grid locations, like Jumbun, in Queensland.

4.1.4 MPC EMS Suitability for Utility Scale

Currently, this project has aimed at reducing electricity costs for a community. However, a better use of the MPC EMS is to reduce the cost of supplying electricity to a community from the utility/electricity retailer perspective. The customers of electricity retailers in Queensland are charged at a flat rate, however electricity retailers and utilities purchase the power form generators at varying price rates. This project has shown that the MPC EMS performs best when a varying price rate is applied, which shows that the MPC EMS will reduce costs for retailers to supply electricity to customers. This can potentially have future effects where customers electricity bills may be reduced.

4.2 Deployment to Raspberry Pi Hardware

A large part of this thesis was also looking at developing the hardware deployment of the MPC EMS to embedded hardware and developing a testing framework. As described in Chapter 3.3, the MPC from the MATLAB environment was converted to C code by using MATLAB coder, which generates C code from MATLAB code.

The simulation which was deployed to the Raspberry Pi Hardware used the following parameters described in Table 4-6. Each of the below parameters (except for the load and price profiles) were manually declared in the coreData object before code deployment to the Raspberry Pi.

As MATLAB plotting functions are not available with the MATLAB code generation, the variables which are normally plotted in MATLAB were output to a text file on the raspberry pi. This file was transferred back to the host computer and visualised and plotted in Microsoft Excel.

Table 4-6 Raspberry Pi MPC Parameters

Parameter	Value
Prediction Horizon	10
Control Horizon	5
Load Profile	Varying Profile from Figure 3-2
Price Profile	Varying Profile from Figure 3-3
Battery Charge Limit	50 kW
Battery Discharge Limit	50 kW

Prior to deployment, the MPC was run using the *nlmpcmoveCodeGeneration* function in MATLAB to ensure that the function produced the same output as *nlmpcmove*, which was used for the simulations.

It was observed that the behaviour of the system was slightly different when using *nlmpcmoveCodeGeneration* as compared to using *nlmpcmove*. This slight change in behaviour is unexpected and is not described in MATLAB documentation. Even though the two functions produce slightly different results, the overall behaviour of the system is extremely similar and it does not affect the overall usability of the MPC EMS.

The two figures below show the difference in behaviour between *nlmpcmoveCodeGeneration* and *nlmpcmove*, in Figure 4-9 and Figure 4-10 respectively. Using the *nlmpcmoveCodegeneration* function, the microgrid drew 1735 kWh of energy from the grid which had a total cost of $101.31, whereas using the *nlmpcmove* function drew 1735.09 kWh of energy from the grid and had a total cost of $101.71. This shows that the behaviour is indeed slightly different, however it is an extremely small difference which does not affect the MPC's usability.

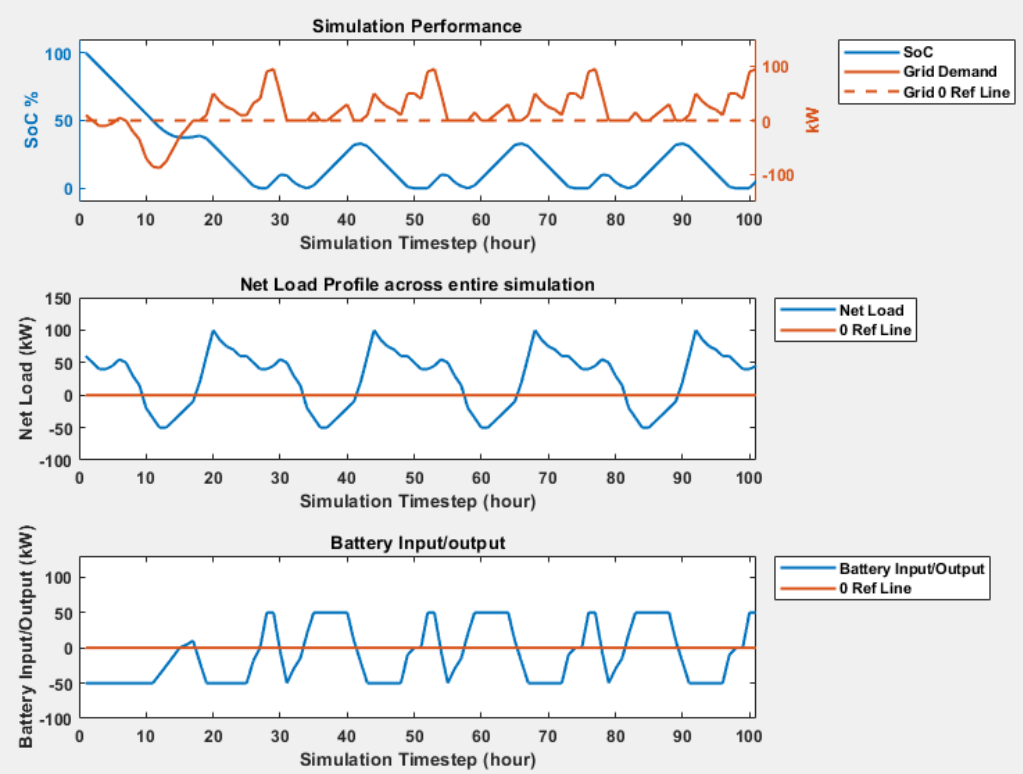

Figure 4-9 Behaviour When Using *nlmpcmoveCodeGeneration* Function

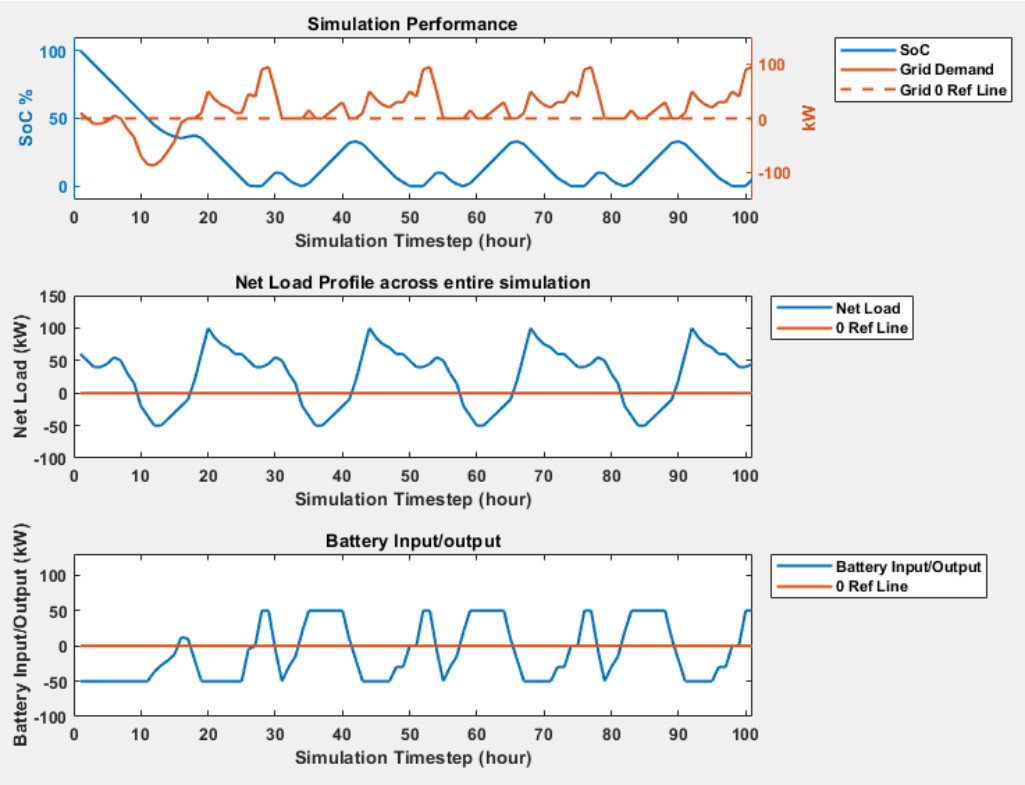

Figure 4-10 Behaviour when Using *nlmpcmove* Function

The Excel graphs below show the performance of the MPC which ran on the Raspberry Pi 4. As is shown in Figure 4-11 , the system behaviour is the same as Figure 4-9 which ran on the host machine in the MATLAB environment, which verifies that the MPC is running correctly on the Raspberry Pi 4.

The time to compute each MPC control action was recorded on the Raspberry Pi 4 to understand the field deployment suitability of this application, with details of this found in

Table 4-7. The time to compute each control action is visualised in Figure 4-12. The results in Table 4-7 show that it took the Raspberry Pi an average of 0.04 seconds to compute the next control action for the system. This is extremely quick and is certainly suitable to deploy this application, considering that this project is theoretically calling the MPC to compute the next control action every hour.

However, the simplicity of the calculations in this project should also be considered when analysing the average computation time. The constraints, cost functions, state and output equations are all linear and non-complex problems which are trivial calculations for a computer. Creating an industry ready MPC will incorporate more complex, potentially non-linear calculations which will almost certainly increase the computation time needed for the MPC. Though, an industry suitable microgrid controller will also have much higher processing

power compared to the Raspberry Pi 4. These factors will need to be considered and assessed before industry deployment.

Table 4-7 Raspberry Pi Computational Time Summary

Average Time for Computation (Seconds)	Minimum Time for Computation (Seconds)	Maximum Time for Computation (Seconds)
0.04	0.00713	0.13549

The simulation performance of the MPC running on the Raspberry Pi is shown below in Figure 4-11, which shows the same performance as the MATLAB simulation in Figure 4-9. This verifies that the code was both compiled and running correctly.

Simulating the MPC on the Raspberry Pi 4 proves the feasibility of developing a MPC EMS in a MATLAB environment and deploying them to embedded systems to be used in the field.

Figure 4-11 MPC Performance on Raspberry Pi 4

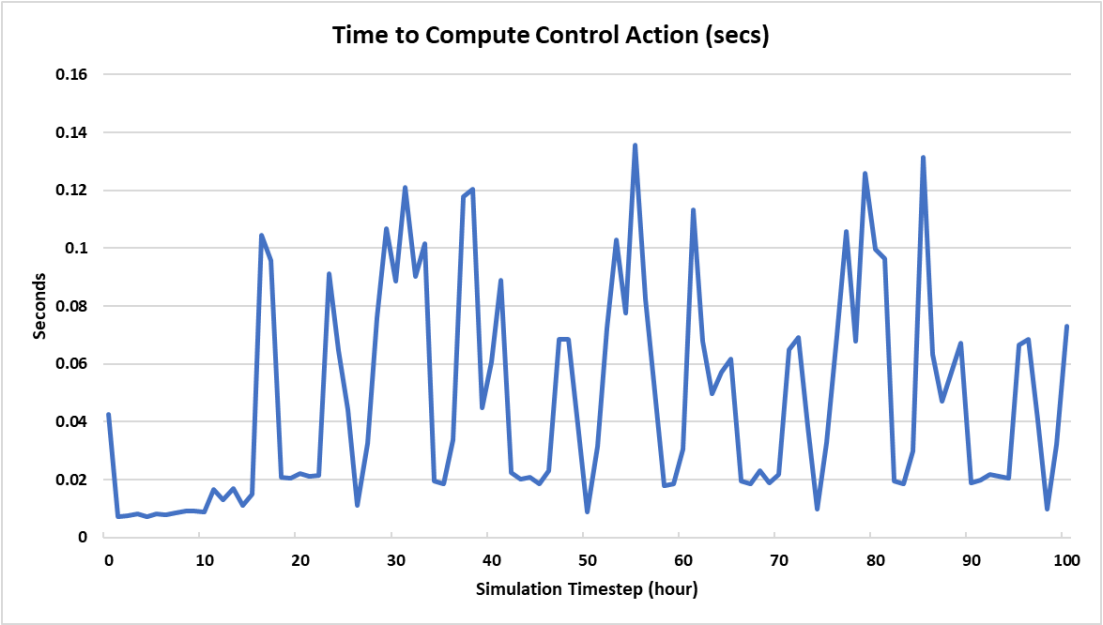

Figure 4-12 Time to Compute Control Action on Raspberry Pi 4

4.3 Future Work Required for Real Time Testing

Even though this thesis has shown the proof of concept for deploying an EMS to an external embedded system, further development is still required to develop an industry ready testing framework for a deployed EMS. Firstly, the microgrid controller must be interfaced with a simulation which can represent the dynamics of a microgrid as close to reality as possible. This will return reliable results to better understand the suitability of a developed EMS. Secondly, a suitable communication platform needs to be developed to enable two way communication between the microgrid and the microgrid controller.

In this project the "microgrid" was simulated on the EMS host device, where the microgrid's state and output variables were updated by running the state and output equations on the EMS host device. In an actual microgrid, the state data will be sent to the microgrid controller running the EMS at regular intervals, then the microgrid controller will send back control actions for the battery to perform. A diagram of this can be seen below in Figure 4-13.

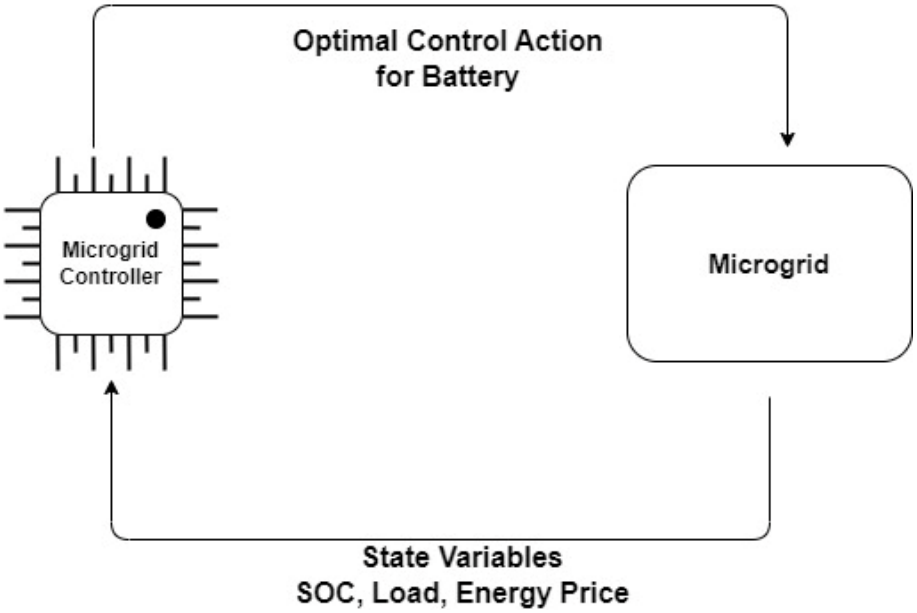

Figure 4-13 Microgrid Control Field Implementation Diagram

To achieve the system shown in Figure 4-13, the microgrid controller running the MPC EMS should be interfaced with a Real Time Digital Simulator (RTDS). An RTDS can run dynamic simulations at timesteps between 25 and 50 microseconds, which is currently as close to real time as possible. The microgrid controller running the MPC EMS will control a microgrid being simulated inside the RTDS. Doing so will produce accurate and reliable results pertaining to how the EMS will perform if it is deployed in the field, without the dangers of having to test on an actual microgrid.

Using the RTDS also helps develop the communication that is required for deploying an EMS to the field. The system shown in Figure 4-13 requires two-way communication between the microgrid and the microgrid controller. The microgrid needs to send state data, and the microgrid controller needs to send a control action to the battery. The RTDS supports a variety of communication protocols which are used in industry and would be suitable for this application [62].

Developing the required communication interface will also require development on the embedded system acting as the microgrid controller. MATLAB's code generator generates code in C; therefore the chosen device will need to be able to run C code. Additionally, the network interface will need to be implemented on the microgrid controller and developed in C. There is an option to develop the communication interface in MATLAB and then generate the code for this as well, which will remove the need for development in the microgrid controller. However, currently it is not known to the author if MATLAB supports code generation for the available communication protocols and would therefore need to be investigated before this direction is taken.

5 Conclusion

When comparing the performance of the two EMS (MPC and rule-based logic), it was found that the MPC EMS can outperform the rule-based logic EMS so long as two conditions are met: 1) an accurate forecaster for around 15 hours ahead of the current time is required and 2) electricity price must vary throughout the day. However, it was noted that the MPC EMS needs to be investigated further as there are many limitations with the comparison which was presented above. More testing needs to take place to know with certainty that the MPC EMS can decrease electricity costs further than the rule-based logic approach can.

Firstly, the results in this project show that the MPC EMS begins to reduce electricity costs further than the rule-based logic approach when a prediction horizon of 15 steps is used with a varying price profile. This project assumed that the MPC has perfect knowledge of the future states of the system (i.e., future load, future PV and future energy price). To get the same results in the field, a forecaster which can accurately forecast future system states around 15 hours ahead is required. This shows the importance of developing accurate forecasters.

Secondly, a varying price profile is required for this economic MPC to outperform a rule-based logic approach. Electricity cost was used as a key metric in the cost function for this project. The results in this project showed that when a static energy price profile is used, such as what is currently experienced for customers on the Ergon Energy network, the rule-based logic approach is the most economic EMS. Therefore, the economic MPC EMS is most likely unsuitable for reducing customers electricity bills.

Limitations in the simulation and comparison between the MPC EMS and the rule-based logic EMS were discussed. Due to time constraints, only one net load profile and one battery size were evaluated. Also, it was identified that the comparison between the MPC EMS and the rule-based logic EMS may not be a fair comparison. Time constraints greatly hindered developing more robust testing methodologies which may have revealed different results and conclusions.

It was discussed that the MPC EMS is better suited at reducing the cost of supplying electricity to customers from the utility/retailer perspective. Electricity retailers are charged by electricity generators at varying price rates, which the MPC has shown to reduce the cost of supplying electricity. The suitability of a utility using a MPC EMS for a microgrid to reduce the cost of supplying communities by storing power when it cheap and discharging when it is expensive may be a future use for this application.

Positive results were found during the EMS development process. This thesis proved that MPC EMS can be developed in a Mathworks environment, which is a widely accessible tool

globally. This is extremely beneficial to utilities or microgrid installers, as it allows these entities to design and understand the EMS which is controlling their microgrid. This may be more beneficial than purchasing an off-the-shelf product, where the microgrid installer/manager may not fully understand the inner workings of an externally designed EMS.

This project has also shown that a MPC EMS can be deployed from MATLAB to an external embedded device, a Raspberry Pi 4, which can function as a microgrid controller. It was proven that the EMS running on the external device will yield the same results as the EMS running in MATLAB. It was also shown that it took the embedded system an average of 0.04 seconds to compute control actions. Microgrid optimisations, such as the MPC EMS presented, typically run in the order of minutes to hours. Deploying the EMS to target hardware has shown the feasibility for successful field operation.

6 References

[1] Department of Energy and Public Works, 'Queensland Energy and Jobs Plan'. Queensland Government, Sep. 28, 2022. Accessed: Mar. 24, 2022. [Online]. Available: https://media.epw.qld.gov.au/files/Queensland_Energy_and_Jobs_Plan.pdf

[2] Ergon Energy, 'Community Microgrid Feasibility Study FAQ's'. Jun. 01, 2022. Accessed: Apr. 30, 2023. [Online]. Available: https://www.ergon.com.au/__data/assets/pdf_file/0008/993905/Community-Microgrid-FAQs.pdf

[3] Australian Energy Regulator, 'Distribution Reliability Measures Guideline'. Nov. 14, 2018. Accessed: May 04, 2023. [Online]. Available: https://www.aer.gov.au/system/files/AER%20-%20Distribution%20Reliability%20Measures%20Guideline%20-%20Version%201%20-%2014%20November%202018%20%28updated%2020%20November%202018%29.pdf

[4] Energy and Public Works, 'Minimum service standards reporting'. Accessed: May 04, 2023. [Online]. Available: https://www.business.qld.gov.au/industries/mining-energy-water/energy/electricity/regulation-licensing/minimum-standards

[5] X. Liang, 'Emerging Power Quality Challenges Due to Integration of Renewable Energy Sources', *IEEE Transactions on Industry Applications*, vol. 53, no. 2, pp. 855–866, Mar. 2017, doi: 10.1109/TIA.2016.2626253.

[6] J. P. Barton and D. G. Infield, 'Energy storage and its use with intermittent renewable energy', *IEEE Transactions on Energy Conversion*, vol. 19, no. 2, pp. 441–448, Jun. 2004, doi: 10.1109/TEC.2003.822305.

[7] A. Gabash and P. Li, 'Active-Reactive Optimal Power Flow in Distribution Networks With Embedded Generation and Battery Storage', *IEEE Transactions on Power Systems*, vol. 27, no. 4, pp. 2026–2035, Nov. 2012, doi: 10.1109/TPWRS.2012.2187315.

[8] M. H. Bellido, L. P. Rosa, A. O. Pereira, D. M. Falcão, and S. K. Ribeiro, 'Barriers, challenges and opportunities for microgrid implementation: The case of Federal University of Rio de Janeiro', *Journal of Cleaner Production*, vol. 188, pp. 203–216, Jul. 2018, doi: 10.1016/j.jclepro.2018.03.012.

[9] United States of America Department of Energy, '2012 DOE Microgrid Workshop Report', Chicago, 2012.

[10] 'Microgrids', Center for Climate and Energy Solutions. Accessed: Mar. 25, 2023. [Online]. Available: https://www.c2es.org/content/microgrids/

[11] R. H. Lasseter and P. Paigi, 'Microgrid: a conceptual solution', in *2004 IEEE 35th Annual Power Electronics Specialists Conference (IEEE Cat. No.04CH37551)*, Jun. 2004, pp. 4285-4290 Vol.6. doi: 10.1109/PESC.2004.1354758.

[12] D. E. Olivares *et al.*, 'Trends in Microgrid Control', *IEEE Transactions on Smart Grid*, vol. 5, no. 4, pp. 1905–1919, Jul. 2014, doi: 10.1109/TSG.2013.2295514.

[13] 'IEEE Standard for the Specification of Microgrid Controllers', *IEEE Std 2030.7-2017*, pp. 1–43, Apr. 2018, doi: 10.1109/IEEESTD.2018.8340204.

[14] 'IEEE Standard for the Testing of Microgrid Controllers', *IEEE Std 2030.8-2018*, pp. 1–42, Aug. 2018, doi: 10.1109/IEEESTD.2018.8444947.

[15] M. F. Zia, E. Elbouchikhi, and M. Benbouzid, 'Microgrids energy management systems: A critical review on methods, solutions, and prospects', *Applied Energy*, vol. 222, pp. 1033–1055, Jul. 2018, doi: 10.1016/j.apenergy.2018.04.103.

[16] A. Hirsch, Y. Parag, and J. Guerrero, 'Microgrids: A review of technologies, key drivers, and outstanding issues', *Renewable and Sustainable Energy Reviews*, vol. 90, pp. 402–411, Jul. 2018, doi: 10.1016/j.rser.2018.03.040.

[17] S. Parhizi, H. Lotfi, A. Khodaei, and S. Bahramirad, 'State of the Art in Research on Microgrids: A Review', *IEEE Access*, vol. 3, pp. 890–925, 2015, doi: 10.1109/ACCESS.2015.2443119.

[18] R. H. Lasseter, 'MicroGrids', in *2002 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No.02CH37309)*, Jan. 2002, pp. 305–308 vol.1. doi: 10.1109/PESW.2002.985003.
[19] N. Hatziargyriou, H. Asano, R. Iravani, and C. Marnay, 'Microgrids', *IEEE Power and Energy Magazine*, vol. 5, no. 4, pp. 78–94, Jul. 2007, doi: 10.1109/MPAE.2007.376583.
[20] N. Ertugrul, 'Battery storage technologies, applications and trend in renewable energy', in *2016 IEEE International Conference on Sustainable Energy Technologies (ICSET)*, Nov. 2016, pp. 420–425. doi: 10.1109/ICSET.2016.7811821.
[21] M. Yazdanian and A. Mehrizi-Sani, 'Distributed Control Techniques in Microgrids', *IEEE Transactions on Smart Grid*, vol. 5, no. 6, pp. 2901–2909, Nov. 2014, doi: 10.1109/TSG.2014.2337838.
[22] J. M. Guerrero, J. C. Vasquez, J. Matas, L. G. de Vicuna, and M. Castilla, 'Hierarchical Control of Droop-Controlled AC and DC Microgrids—A General Approach Toward Standardization', *IEEE Transactions on Industrial Electronics*, vol. 58, no. 1, pp. 158–172, Jan. 2011, doi: 10.1109/TIE.2010.2066534.
[23] A. Bidram and A. Davoudi, 'Hierarchical Structure of Microgrids Control System', *IEEE Transactions on Smart Grid*, vol. 3, no. 4, pp. 1963–1976, Dec. 2012, doi: 10.1109/TSG.2012.2197425.
[24] L. Ahmethodzic and M. Music, 'Comprehensive review of trends in microgrid control', *Renewable Energy Focus*, vol. 38, pp. 84–96, Sep. 2021, doi: 10.1016/j.ref.2021.07.003.
[25] D. E. Olivares, C. A. Cañizares, and M. Kazerani, 'A centralized optimal energy management system for microgrids', in *2011 IEEE Power and Energy Society General Meeting*, Jul. 2011, pp. 1–6. doi: 10.1109/PES.2011.6039527.
[26] W. Su and J. Wang, 'Energy Management Systems in Microgrid Operations', *The Electricity Journal*, vol. 25, no. 8, pp. 45–60, Oct. 2012, doi: 10.1016/j.tej.2012.09.010.
[27] W. Shi, X. Xie, C.-C. Chu, and R. Gadh, 'Distributed Optimal Energy Management in Microgrids', *IEEE Transactions on Smart Grid*, vol. 6, no. 3, pp. 1137–1146, May 2015, doi: 10.1109/TSG.2014.2373150.
[28] S. Phommixay, M. L. Doumbia, and D. Lupien St-Pierre, 'Review on the cost optimization of microgrids via particle swarm optimization', *Int J Energy Environ Eng*, vol. 11, no. 1, pp. 73–89, Mar. 2020, doi: 10.1007/s40095-019-00332-1.
[29] J. Zaytoon and B. Riera, 'Synthesis and implementation of logic controllers – A review', *Annual Reviews in Control*, vol. 43, pp. 152–168, Jan. 2017, doi: 10.1016/j.arcontrol.2017.03.004.
[30] M. Restrepo, C. A. Cañizares, J. W. Simpson-Porco, P. Su, and J. Taruc, 'Optimization- and Rule-based Energy Management Systems at the Canadian Renewable Energy Laboratory microgrid facility', *Applied Energy*, vol. 290, p. 116760, May 2021, doi: 10.1016/j.apenergy.2021.116760.
[31] C. Sun *et al.*, 'Design and Real-Time Implementation of a Centralized Microgrid Control System With Rule-Based Dispatch and Seamless Transition Function', *IEEE Transactions on Industry Applications*, vol. 56, no. 3, pp. 3168–3177, May 2020, doi: 10.1109/TIA.2020.2979790.
[32] Z. Xu, P. Yang, Z. Chen, C. Yang, Q. Zheng, and C. Zheng, 'Energy management strategy for medium-voltage Isolated microgrid', in *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, Nov. 2015, pp. 000080–000085. doi: 10.1109/IECON.2015.7392079.
[33] Z. Zhang *et al.*, 'Advances and opportunities in the model predictive control of microgrids: Part I–primary layer', *International Journal of Electrical Power & Energy Systems*, vol. 134, p. 107411, Jan. 2022, doi: 10.1016/j.ijepes.2021.107411.
[34] S. J. Qin and T. A. Badgwell, 'A survey of industrial model predictive control technology', *Control Engineering Practice*, vol. 11, no. 7, pp. 733–764, Jul. 2003, doi: 10.1016/S0967-0661(02)00186-7.
[35] W. R. Sultana, S. K. Sahoo, S. Sukchai, S. Yamuna, and D. Venkatesh, 'A review on state of art development of model predictive control for renewable energy applications',

Renewable and Sustainable Energy Reviews, vol. 76, pp. 391–406, Sep. 2017, doi: 10.1016/j.rser.2017.03.058.
[36] O. Babayomi, Z. Zhang, T. Dragicevic, J. Hu, and J. Rodriguez, 'Smart grid evolution: Predictive control of distributed energy resources—A review', *International Journal of Electrical Power & Energy Systems*, vol. 147, p. 108812, May 2023, doi: 10.1016/j.ijepes.2022.108812.
[37] I. Prodan and E. Zio, 'A model predictive control framework for reliable microgrid energy management', *International Journal of Electrical Power & Energy Systems*, vol. 61, pp. 399–409, Oct. 2014, doi: 10.1016/j.ijepes.2014.03.017.
[38] D. Q. Mayne, 'Model predictive control: Recent developments and future promise', *Automatica*, vol. 50, no. 12, pp. 2967–2986, Dec. 2014, doi: 10.1016/j.automatica.2014.10.128.
[39] J. Hu, Y. Shan, J. M. Guerrero, A. Ioinovici, K. W. Chan, and J. Rodriguez, 'Model predictive control of microgrids – An overview', *Renewable and Sustainable Energy Reviews*, vol. 136, p. 110422, Feb. 2021, doi: 10.1016/j.rser.2020.110422.
[40] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, 'Review on model predictive control: an engineering perspective', *Int J Adv Manuf Technol*, vol. 117, no. 5, pp. 1327–1349, Nov. 2021, doi: 10.1007/s00170-021-07682-3.
[41] O. Babayomi *et al.*, 'Advances and opportunities in the model predictive control of microgrids: Part II–Secondary and tertiary layers', *International Journal of Electrical Power & Energy Systems*, vol. 134, p. 107339, Jan. 2022, doi: 10.1016/j.ijepes.2021.107339.
[42] MathWorks, 'Linear Programming', Solve linear optimization problems. Accessed: Apr. 10, 2023. [Online]. Available: https://au.mathworks.com/discovery/linear-programming.html
[43] R. Palma-Behnke *et al.*, 'A Microgrid Energy Management System Based on the Rolling Horizon Strategy', *IEEE Transactions on Smart Grid*, vol. 4, no. 2, pp. 996–1006, Jun. 2013, doi: 10.1109/TSG.2012.2231440.
[44] R. K. Bonthu, R. P. Aguilera, H. Pham, M. D. Phung, and Q. P. Ha, 'Energy Cost Optimization in Microgrids Using Model Predictive Control and Mixed Integer Linear Programming', in *2019 IEEE International Conference on Industrial Technology (ICIT)*, Feb. 2019, pp. 1113–1118. doi: 10.1109/ICIT.2019.8754971.
[45] A. Parisio, E. Rikos, and L. Glielmo, 'A Model Predictive Control Approach to Microgrid Operation Optimization', *IEEE Transactions on Control Systems Technology*, vol. 22, no. 5, pp. 1813–1827, Sep. 2014, doi: 10.1109/TCST.2013.2295737.
[46] Y. Zheng, S. Li, and R. Tan, 'Distributed Model Predictive Control for On-Connected Microgrid Power Management', *IEEE Transactions on Control Systems Technology*, vol. 26, no. 3, pp. 1028–1039, May 2018, doi: 10.1109/TCST.2017.2692739.
[47] Y. Jia, Z. Y. Dong, C. Sun, and G. Chen, 'Distributed Economic Model Predictive Control for a Wind–Photovoltaic–Battery Microgrid Power System', *IEEE Transactions on Sustainable Energy*, vol. 11, no. 2, pp. 1089–1099, Apr. 2020, doi: 10.1109/TSTE.2019.2919499.
[48] X. Zhang, J. Bao, R. Wang, C. Zheng, and M. Skyllas-Kazacos, 'Dissipativity based distributed economic model predictive control for residential microgrids with renewable energy generation and battery energy storage', *Renewable Energy*, vol. 100, pp. 18–34, Jan. 2017, doi: 10.1016/j.renene.2016.05.006.
[49] J. Wang, Y. Song, W. Li, J. Guo, and A. Monti, 'Development of a Universal Platform for Hardware In-the-Loop Testing of Microgrids', *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2154–2165, Nov. 2014, doi: 10.1109/TII.2014.2349271.
[50] RTDS Technologies, 'Power Hardware In The Loop Simulation (PHIL)', May 2018. Accessed: Apr. 15, 2023. [Online]. Available: https://knowledge.rtds.com/hc/en-us/article_attachments/360049451353/RTDS_PHIL_Report-2-1.pdf
[51] RTDS Technologies, 'Meticulously engineered models for high-fidelity HIL testing', POWER ELECTRONICS HIL. Accessed: Apr. 15, 2023. [Online]. Available: https://www.rtds.com/applications/power-electronics-hil/

[52] ‘RTDS Technologies’, RTDS Technologies. Accessed: Apr. 20, 2023. [Online]. Available: https://www.rtds.com/
[53] *De-Risking Microgrids: Hardware-in-the-Loop Testing with the RTDS Simulator*, (Jul. 11, 2020). Accessed: Apr. 20, 2023. [Online Video]. Available: https://youtu.be/ZuYeQiCsokk
[54] T. Morstyn, B. Hredzak, and V. G. Agelidis, ‘Dynamic optimal power flow for DC microgrids with distributed battery energy storage systems’, in *2016 IEEE Energy Conversion Congress and Exposition (ECCE)*, Sep. 2016, pp. 1–6. doi: 10.1109/ECCE.2016.7855059.
[55] ‘Optimization Problem - MATLAB & Simulink - MathWorks Australia’. Accessed: Sep. 06, 2023. [Online]. Available: https://au.mathworks.com/help/mpc/ug/optimization-problem.html
[56] ‘Economic MPC - MATLAB & Simulink - MathWorks Australia’. Accessed: Sep. 30, 2023. [Online]. Available: https://au.mathworks.com/help/mpc/ug/economic-mpc.html
[57] ‘NEM data dashboard’. Accessed: Sep. 09, 2023. [Online]. Available: https://aemo.com.au/energy-systems/electricity/national-electricity-market-nem/data-nem/data-dashboard-nem
[58] Network, ‘Electricity network demand’. Accessed: Sep. 09, 2023. [Online]. Available: https://www.ergon.com.au/network/manage-your-energy/managing-electricity-demand/peak-demand/electricity-network-demand
[59] ‘What is a Raspberry Pi?’, Raspberry Pi Foundation. Accessed: Sep. 30, 2023. [Online]. Available: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/
[60] ‘Raspberry Pi - About us’, Raspberry Pi. Accessed: Sep. 30, 2023. [Online]. Available: https://www.raspberrypi.com/about/
[61] ‘Deploy Generated C Code to External Hardware: Raspberry Pi Examples - MATLAB & Simulink - MathWorks Australia’. Accessed: Sep. 30, 2023. [Online]. Available: https://au.mathworks.com/help/coder/ug/deploy-generated-c-code-to-external-hardware-raspberry-pi-examples.html
[62] RTDS Technologies, ‘Communication Protocols - RTDS Technologies’, RTDS Technologies. Accessed: Apr. 20, 2023. [Online]. Available: https://www.rtds.com/technology/communication-protocols/

7 Appendices

7.1 Appendix A: Implicit MPC Code

```
clc;
clear;
battery_size = 1000;
A = [1];
B = [100/(battery_size)];
C = [1];
D = [0];
Ts = 10;
batt_plant = ss(A,B,C,D, Ts);

%set names
batt_plant.InputName = {'Power Draw'};
batt_plant.OutputName = {'SoC'};
batt_plant.StateName = {'SoC'};

%set units
batt_plant.InputUnit = {'kW'};
batt_plant.OutputUnit = {'%'};
batt_plant.StateUnit = {'%'};

batt_plant = setmpcsignals(batt_plant, 'MV', 1, 'MO', 1);

batt_plant;

old_status = mpcverbosity('off');
mpcobj = mpc(batt_plant);

get(mpcobj)

mpcobj.PredictionHorizon = 5;
mpcobj.ControlHorizon = 2;
mpcobj.MV.Min = -50;
mpcobj.MV.Max = 50;
mpcobj.ManipulatedVariables(1).RateMin = -10; %minimum ramp rate
mpcobj.ManipulatedVariables(1).RateMax = 10; % maximum ramp rate

mpcobj.OutputVariables(1).Min = 0;
mpcobj.OutputVariables(1).Max = 100;

mpcstateobj = mpcstate(mpcobj); % sets MPC state object variable
mpcstateobj.Plant = 20; % sets the initial state to 20%
r = 100; % sets the reference (goal) variable to 100

t = 0:Ts:300; % sets the simulation length
N = length(t); % number of iterations
y = zeros(N, 1); % empty y array
u = zeros(N, 1); % empty x array

for i = 1:N
    y(i) = mpcstateobj.Plant; % current state of plant
    u(i) = mpcmove(mpcobj,mpcstateobj,y(i),r); % input from movement
end

yyaxis left
```

```
plot(y);
axis([0 35 -10 130])
hold on;
yyaxis right
plot(u);
axis([0 35 -10 130])
hold on;
legend('SOC %', "Battery Power kW")
```

7.2 Appendix B: State Equation for Non-Linear MPC Code

```
function x1 = BatterySoCStateFcnV4(x, u_in, Ts)
batt_size = 1000;
x1 = zeros(3, 1);
x1(1) = x(1) + (100*Ts/batt_size) * u_in; %SOC
x1(2) = x(2); %Load
x1(3) = x(3); % Price
```

7.3 Appendix C: Output Equation for Non-Linear MPC Code

```
function y = BatterySoCOutputFcnV5(x, u, Ts)
y = zeros(2, 1);
y(1) = x(1); %SOC
y(2) = x(2) + u; %Grid Demand
```

7.4 Appendix D: Cost Function for Non-Linear MPC Code

```
function J = EnergyCostFunctionV3(x, u, e, data, Ts)

load_vec = data.References(:, 2);
grid_vec = load_vec + u([2:end]);

first_grid = x(1, 2) + u(1);
J = 0;
for k = 1:data.PredictionHorizon + 1
    if k == 1
        if first_grid > 0
            J = J + first_grid * x(1, 3);
        elseif first_grid < 0
            J = J + first_grid * 1;
        end
    elseif grid_vec(k-1) <= 0
        J = J + grid_vec(k-1) * 1;
        testvar = grid_vec(k-1) * 1;
    else
        J = J + grid_vec(k-1) * data.References(k-1, 1);
        testvar = grid_vec(k-1) * data.References(k-1, 1);
    end
end
```

7.5 Appendix E: Economic MPC Code

```
%% Clear Workspace
clc;
clear;
close all;
format long;

%% Specify Nonlinear MPC object with 3 states, 2 outputs and 1 inputs
nlobj = nlmpc(3, 2, 1);

%specify state names
nlobj.States(1).Name = 'SoC';
nlobj.States(2).Name = 'Load';
nlobj.States(3).Name = 'Price';

%Manipulated variable (input) name
nlobj.MV(1).Name = 'Battery Power (kW)';

%Output Variables Name
nlobj.OutputVariables(1).Name = 'SoC';
nlobj.OutputVariables(2).Name = 'Grid Power';


%% Specify State and Output Functions
nlobj.Model.StateFcn = 'BatterySoCStateFcnV4';
nlobj.Model.IsContinuousTime = false;
nlobj.Model.OutputFcn = "BatterySoCOutputFcnV5";


nlobj.Optimization.CustomCostFcn = "EnergyCostFunctionV3";
nlobj.Optimization.ReplaceStandardCost = true;

%% Specify profiles
flatprice = [0; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5;
5; 5; 5; 5;];
varying_price = [0; 5; 4; 3.5; 2.5; 2.6; 3.5; 4.5; 7; 5.5; 4.5; 3; 1; 1;
4; 6; 6.5; 6.5; 7; 8; 10; 9; 6; 6; 5;];

used_price = varying_price;

flatter_load = [0 0; 50 1;  50 2; 50 3; 50 4; 50 5; 50 6; 50 7; 30 8; 15
9; -20 10; -35 11; -50 12; -50 13; -40 14; -30 15; -20 16; -10 17; 20 18;
50 19; 50 20; 50 21; 50 22; 50 23; 50 24;];
varying_load = [0 0; 60 1;  50 2; 40 3; 40 4; 45 5; 55 6; 50 7; 30 8; 15
9; -20 10; -35 11; -50 12; -50 13; -40 14; -30 15; -20 16; -10 17; 20 18;
60 19; 100 20; 85 21; 75 22; 70 23; 60 24;];
no_solar = [0 0; 60 1;  50 2; 40 3; 40 4; 45 5; 55 6; 50 7; 30 8; 15 9; -
20 10; 5 11; -50 12; -10 13; -10 14; -30 15; 0 16; -10 17; 20 18; 60 19;
100 20; 85 21; 75 22; 70 23; 60 24;];

used_load = varying_load;


%% specify sample time, prediction horizons
Ts = 1; % 1 hour
nlobj.Ts = Ts; % Sample time
nlobj.PredictionHorizon = 10; % Prediction horizon
nlobj.ControlHorizon = 2;  % Control Horizon
```

```
nloptions = nlmpcmoveopt;
nloptions.Parameters = {Ts}; % add sample time as a parameter for the
simulation% Control horizon

%% Specify Constraints
nlobj.States(1).Min = 0;
nlobj.States(1).Max = 100;
nlobj.ManipulatedVariables(1).Min = -50;
nlobj.ManipulatedVariables(1).Max = 50;
nlobj.OutputVariables(1).Min = 0;
nlobj.OutputVariables(1).Max = 100;
nlobj.Model.NumberOfParameters = 1;
nlobj.ManipulatedVariables.RateMin = -1000;
nlobj.ManipulatedVariables.RateMax = 1000;

%set the below min and maxes to ensure that NaN values are not sent over
%TCP
nlobj.OutputVariables(2).Min = -1000000;
nlobj.OutputVariables(2).Max = 1000000;
nlobj.States(2).Min = -1000000;
nlobj.States(2).Max = 1000000;
nlobj.States(3).Min = -1000000;
nlobj.States(3).Max = 1000000;

%% Specify closed loop simulations parameters

%specify dummy variables for validatefcns line
x0 = [20; 100; 0.05]; %dummy x array for validate fcns command
u0 = [15]; %dummy input
yref = [100 0]; %dummy references
validateFcns(nlobj, x0, u0, [], {Ts});

t = 0:100; %simulation length

u_vars = zeros(1, length(t)); %array to store input variables
mv_vars = zeros(1, length(t));
x_vars = zeros(3, length(t)); %array to store states
output_states = zeros(2, length(t)); %array to store outputs
load_result = zeros(1, length(t)); %array to store load at each timestep
for graphing

timings = zeros(1, length(t));
zero_reference = zeros(1, length(t));

%% Initial conditions for simulation

x_vars(1, 1) = 100; %initial SoC
x_vars(2, 1) = 100; %Initial Load
x_vars(3,1) = used_price(1); %initial price
u_vars(1, 1) = 0; %initial control input
mv_vars(1,1) = 0;

%% Code Generation Initialisation
[coreData, onlineData] = getCodeGenerationData(nlobj, x_vars(:, 1),
mv_vars(1), {Ts});

%% Closed Loop Simulation

for i = 1:length(t) %i is the end time of the hour
```

```
    index_vector = [i+1:nlobj.PredictionHorizon+i+1];
    %gets the indexes needs for indexing the laod and price vectors

    for z = 1:5
        if isempty(index_vector(index_vector>=26*z)) == 0
            index_vector(index_vector>=26*z) =
2+index_vector(index_vector>=26*z);
            %to avoid the issue of the 0 index in matlab, this was my
            % work around
        end
        check_vec = mod(index_vector, 26); %used for debugging
    end

    load2_vector = used_load(mod(index_vector, 26));
    price2_vector = used_price(mod(index_vector, 26));

    refs(:, 1) = price2_vector(2:end);
    refs(:, 2) = load2_vector(2:end);

    %updates the price state
    x_vars(3, i) = price2_vector(1);
    x_vars(2, i) = load2_vector(1);
    load_result(i) = load2_vector(1);

    onlineData.ref = [price2_vector(:), load2_vector(:)];

    %computes the optimal control action
    if i == 1
        tstart = tic;
        [u_vars(:, i), testseq, new_info] = nlmpcmove(nlobj, x_vars(:, i),
0, refs, [], nloptions);
        tstop = toc(tstart);
        timings(i) = tstop;
    else
        tstart = tic;
        [u_vars(:, i), testseq, new_info] = nlmpcmove(nlobj, x_vars(:, i),
u_vars(:, i-1), refs, [], nloptions);
        tstop = toc(tstart);
        timings(i) = tstop;
    end

    %calculates the updated state if control action is applied
    x_vars(:, i+1) = BatterySoCStateFcnV4(x_vars(:, i), u_vars(i), Ts); %
this is the state for the next time step
    output_states(:, i) = BatterySoCOutputFcnV5(x_vars(:, i), u_vars(i),
Ts);

end


%% Visaulisations
main_figure = figure;
main_figure.Position(2:4) = [300 800 600];
tl = tiledlayout(3, 1);
nexttile
yyaxis left
plot(x_vars(1, :), 'LineWidth', 1.5);
axis([0 length(t) -10 110])
```

```
hold on;
xlabel('Simulation Timestep (hour)', 'FontWeight','bold')
ylabel('SoC %', 'FontWeight','bold')
a = get(gca, 'XTickLabel');
set(gca,'XTickLabel',a,'FontWeight','bold')
yyaxis right
hold on;
plot(output_states(2, :), 'LineWidth', 1.5);
axis([0 length(t) -150 150])
ylabel('kW', 'FontWeight','bold')
a = get(gca, 'XTickLabel');
set(gca,'XTickLabel',a,'FontWeight','bold')
title("Simulation Performance")
hold on;
plot(zero_reference, 'LineWidth', 1.5)
hold on;
legend('SoC', 'Grid Demand', 'Grid 0 Ref Line')
legend('Location','northeastoutside')
ax = gca(main_figure);
ax.XAxis.FontWeight = 'bold';

nexttile
plot(load_result, 'LineWidth', 1.5)
axis([0 length(t) -100 150]);
title('Net Load Profile across entire simulation')
xlabel('Simulation Timestep (hour)')
ylabel('Net Load (kW)')
a = get(gca, 'XTickLabel');
set(gca,'XTickLabel',a,'FontWeight','bold')
hold on;
plot(zero_reference, 'LineWidth', 1.5)
hold on;
legend("Net Load", "0 Ref Line")
legend('Location','northeastoutside')
ax = gca(main_figure);
ax.XAxis.FontWeight = 'bold';
ax.YAxis.FontWeight = 'bold';

nexttile
plot(u_vars, 'LineWidth', 1.5);
axis([0 length(t) -100 130]);
title('Battery Input/output')
xlabel('Simulation Timestep (hour)')
ylabel('Battery Input/Output (kW)')
a = get(gca, 'XTickLabel');
set(gca,'XTickLabel',a,'FontWeight','bold')
hold on;
plot(zero_reference, 'LineWidth', 1.5)
hold on;
legend("Battery Input/Output", "0 Ref Line")
legend('Location','northeastoutside')
ax = gca(main_figure);
ax.XAxis.FontWeight = 'bold';
ax.YAxis.FontWeight = 'bold';

disp("Grid demand is: ")
sum(output_states(2, :))

total_price = 0;
```

```
for k = 1:length(t)
    if output_states(2, k) > 0
        total_price = total_price + output_states(2, k) * x_vars(3, k);
    elseif output_states(2, k) < 0
        total_price = total_price + output_states(2, k) * 1;
    end
end
disp("money paid is: ")
total_price

disp("Average time is")
mean(timings)

load_figure = figure;
plot(used_load(1:25));
axis([2 25 -60 110]);
xlabel("Hour in day");
ylabel("Net Load kW");
title("Net Load Profile");
xticks([7, 13, 19, 25]);
xticklabels({'6', '12', '18', '24'});

price_figrue = figure;
plot(used_price(1:end));
axis([2 25 0 15]);
xlabel("Hour in day");
ylabel("Energy Price c/kWh");
title("Energy Price Profile")
xticks([7, 13, 19, 25]);
xticklabels({'6', '12', '18', '24'});
```

7.6 Appendix F: Code Deployment

```
function [] = fullcodegenV2()

%% making coreData again
new_coreData.Style = 'General';
new_coreData.Ts = 1;
new_coreData.OutputWeights = [1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0; 1 0;
1 0; ];
new_coreData.MVWeights = zeros(10, 1);
new_coreData.MVRateWeights = 0.1 * ones(10, 1);
new_coreData.ECRWeight = 100000;
new_coreData.OutputMin = [0 -1000000; 0 -1000000; 0 -1000000; 0 -1000000;
0 -1000000; 0 -1000000; 0 -1000000; 0 -1000000; 0 -1000000; 0 -1000000;];
new_coreData.OutputMax = [100 1000000; 100 1000000; 100 1000000; 100
1000000; 100 1000000; 100 1000000; 100 1000000; 100 1000000; 100 1000000;
100 1000000; ];
new_coreData.StateMin = [0-1000000     -1000000; 0  -1000000     -1000000;
0      -1000000     -1000000; 0  -1000000     -1000000; 0  -1000000     -
1000000; 0   -1000000     -1000000; 0  -1000000     -1000000; 0  -1000000
       -1000000; 0  -1000000     -1000000; 0  -1000000     -1000000;];
new_coreData.StateMax = [100     1000000      1000000; 100 1000000
       1000000; 100 1000000      1000000; 100 1000000      1000000; 100
       1000000      1000000; 100 1000000      1000000; 100 1000000
       1000000; 100 1000000      1000000; 100 1000000      1000000; 100
       1000000      1000000];
new_coreData.MVMin = -50 * ones(10, 1);
new_coreData.MVMax = 50 * ones(10, 1);
new_coreData.MVRateMin = -1000 * ones(10, 1);
new_coreData.MVRateMax = 1000 * ones(10, 1);
new_coreData.Uscale = 1;
new_coreData.Yscale = [1; 1];
new_coreData.Xscale = [1;1;1];
new_coreData.MVscale = 1;
new_coreData.OVMinECR = ones(10, 2);
new_coreData.OVMaxECR = ones(10, 2);
new_coreData.MVMinECR = zeros(10, 1);
new_coreData.MVMaxECR = zeros(10, 1);
new_coreData.MVRateMinECR = zeros(10, 1);
new_coreData.MVRateMaxECR = zeros(10, 1);
new_coreData.hasxscale = false;
new_coreData.hasyscale = false;
new_coreData.p = 10;
new_coreData.imv = 1;
new_coreData.imd = [];
new_coreData.iud = [];
new_coreData.nmv = 1;
new_coreData.nmd = 0;
new_coreData.nud = 0;
new_coreData.nx = 3;
new_coreData.ny = 2;
new_coreData.nu = 1;
new_coreData.npara = 1;
new_coreData.replacecost = true;
new_coreData.isct = false;
new_coreData.strStateFcn = "BatterySoCStateFcnV4" ;
new_coreData.strOutputFcn = "BatterySoCOutputFcnV5";
new_coreData.strCostFcn = "EnergyCostFunctionV3";
new_coreData.strEqConFcn = char("");
```

```
new_coreData.strIneqConFcn = char("");
new_coreData.strJacobianStateFcn = char("");
new_coreData.strJacobianOutputFcn = char("");
new_coreData.strJacobianCostFcn = char("");
new_coreData.strJacobianEqConFcn = char("");
new_coreData.strJacobianIneqConFcn = char("");
new_coreData.Iz2u = [1 0 0 0 0; 0 1 0 0 0; 0 0 1 0 0; 0 0 0 1 0; 0 0 0 0
1; 0 0 0 0 1; 0 0 0 0 1; 0 0 0 0 1; 0 0 0 0 1;0 0 0 0 1];
new_coreData.Iu2z = [1 0 0 0 0 0 0 0 0 0; 0 1 0 0 0 0 0 0 0 0; 0 0 1 0 0 0
0 0 0 0; 0 0 0 1 0 0 0 0 0 0; 0 0 0 0 1 0 0 0 0 0];
new_coreData.usesuboptimalsolution = false;
new_coreData.isLTV = false;
new_coreData.isQP = false;
new_coreData.solveroptionsConstraintTolerance = 1.000000000000000e-06;
new_coreData.solveroptionsMaxIterations = 400;
new_coreData.solveroptionsObjectiveLimit = -1.000000000000000e+20;
new_coreData.solveroptionsOptimalityTolerance = 1.000000000000000e-06;
new_coreData.solveroptionsScaleProblem = false;
new_coreData.solveroptionsStepTolerance = 1.000000000000000e-06;


new_onlineData.ref = [0,0];
new_onlineData.MVTarget = 0;
new_onlineData.Parameters = {1};
new_onlineData.X0 = [100 100 0; 100 100 0; 100 100 0; 100 100 0; 100 100
0; 100 100 0; 100 100 0; 100 100 0; 100 100 0; 100 100 0];
new_onlineData.MV0 = 10 * zeros(10, 1);
new_onlineData.Slack0 = 0;

%% Specify profiles

varying_price = [0; 5; 4; 3.5; 2.5; 2.6; 3.5; 4.5; 7; 5.5; 4.5; 3; 1; 1;
4; 6; 6.5; 6.5; 7; 8; 10; 9; 6; 6; 5;];

used_price = varying_price;


varying_load = [0 0; 60 1;  50 2; 40 3; 40 4; 45 5; 55 6; 50 7; 30 8; 15
9; -20 10; -35 11; -50 12; -50 13; -40 14; -30 15; -20 16; -10 17; 20 18;
60 19; 100 20; 85 21; 75 22; 70 23; 60 24;];


used_load = varying_load;
Ts = 1; % 1 hour

%% Specify closed loop simulations parameters


t = 0:100; %simulation length

u_vars = zeros(1, length(t)); %array to store input variables
mv_vars = zeros(1, length(t));
x_vars = zeros(3, length(t)); %array to store states
output_states = zeros(2, length(t)); %array to store outputs
load_result = zeros(1, length(t)); %array to store load at each timestep
for graphing

timings = zeros(1, length(t));
zero_reference = zeros(1, length(t));
```

```
%% Initial conditions for simulation

x_vars(1, 1) = 100; %initial SoC
x_vars(2, 1) = 100; %Initial Load
x_vars(3,1) = used_price(1); %initial price
u_vars(1, 1) = 0; %initial control input
mv_vars(1,1) = 0;

%% Closed Loop Simulation

file_id = fopen("MPCOutput.txt", "w");
fprintf(file_id, "%s,%s,%s,%s,%s,%s\n", "Battery SoC (%)", "Net Grid Power
(kW)", "Net Load (kW)", "Price of power (c/kWh)", "Battery Input (kW)",
"Time for Calculation (secs)");
fclose(file_id);


for i = 1:length(t) %i is the end time of the hour

    index_vector = [i+1:10+i+1]; % cost function v3

    for z = 1:5
        if isempty(index_vector(index_vector>=26*z)) == 0
            index_vector(index_vector>=26*z) =
2+index_vector(index_vector>=26*z);
        end
        check_vec = mod(index_vector, 26);
    end

    load2_vector = used_load(mod(index_vector, 26));
    price2_vector = used_price(mod(index_vector, 26));


    %updates the price state
    x_vars(3, i) = price2_vector(1); %cheanged here
    x_vars(2, i) = load2_vector(1); %changed here
    load_result(i) = load2_vector(1); %changed here
    new_onlineData.ref = [price2_vector(2:end), load2_vector(2:end)'];

    %computes the optimal control action
    if i == 1
        stateData = x_vars(:, i);
        lastCntrl = 0;
        tstart=tic;
        [mv_vars(i), new_onlineData, new_info] =
nlmpcmoveCodeGeneration(new_coreData, stateData, lastCntrl,
new_onlineData);
        tend = toc(tstart);
    else
        statedata=x_vars(:, i);
        lastCntrl=mv_vars(:, i-1);
        tstart = tic;
        [mv_vars(i), new_onlineData, new_info] =
nlmpcmoveCodeGeneration(new_coreData, statedata, lastCntrl,
new_onlineData);
        tend = toc(tstart);
    end
```

```
    %calculates the updated state if control action is applied
    x_vars(:, i+1) = BatterySoCStateFcnV4(x_vars(:, i), mv_vars(i), Ts); %
this is the state for the next time step
    output_states(:, i) = BatterySoCOutputFcnV5(x_vars(:, i), mv_vars(i),
Ts);
    file_ID = fopen("MPCOutput.txt", "a");
    fprintf(file_ID, "%.5f,%.5f,%.5f,%.5f,%.5f,%.5f\n", x_vars(1, i),
output_states(2,i), x_vars(2,i), x_vars(3,i), mv_vars(i),tend);
    fclose(file_ID);
end
end
```

7.7 Appendix G: Code Deployment

```
clc;
clear;
close all;

func = 'fullcodegenV2';
funcOutput = 'picodegenv2';

hwObj = coder.hardware('Raspberry Pi');

Cfg = coder.config('lib','ecoder',false);
Cfg.Hardware = hwObj;
Cfg.GenCodeOnly = true;
Cfg.GenerateReport = true;
Cfg.DynamicMemoryAllocation = 'off';

codegen('-config',Cfg,func,'-o',funcOutput);


myBuildInfoFile = 'codegen/lib/fullcodegenV2/buildInfo.mat';
load(myBuildInfoFile);
packNGo(buildInfo);

movefile ./codegen/lib/fullcodegenV2/examples/main.c
movefile ./codegen/lib/fullcodegenV2/examples/main.h
```

7.8 Appendix H: Rule-based Logic EMS Code

7.8.1 EMS Code

```
%% Clear Variables

close all;
clc;
clear;

%% Declare simulation constants
flatprice = [0; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5; 5;
5; 5; 5; 5; 5;];
varying_price = [0; 5; 4; 3.5; 2.5; 2.6; 3.5; 4.5; 7; 5.5; 4.5; 3; 1; 1;
4; 6; 6.5; 6.5; 7; 8; 10; 9; 6; 6; 5;];

used_price = varying_price;

flatter_load = [0 0; 50 1;  50 2; 50 3; 50 4; 50 5; 50 6; 50 7; 30 8; 15
9; -20 10; -35 11; -50 12; -50 13; -40 14; -30 15; -20 16; -10 17; 20 18;
50 19; 50 20; 50 21; 50 22; 50 23; 50 24;];
varying_load = [0 0; 60 1;  50 2; 40 3; 40 4; 45 5; 55 6; 50 7; 30 8; 15
9; -20 10; -35 11; -50 12; -50 13; -40 14; -30 15; -20 16; -10 17; 20 18;
60 19; 100 20; 85 21; 75 22; 70 23; 60 24;];
no_solar = [0 0; 60 1;  50 2; 40 3; 40 4; 45 5; 55 6; 50 7; 30 8; 15 9; -
20 10; 5 11; -50 12; -10 13; -10 14; -30 15; 0 16; -10 17; 20 18; 60 19;
100 20; 85 21; 75 22; 70 23; 60 24;];

used_load = flatter_load;


Ts = 1;
```

```
t = 0:100;
batt_size = 1000;
charge_lim = 50;
discharge_lim = -100;
disSOClim = 0 - ((discharge_lim * 100 * Ts)/batt_size); % if soc below
this violation then
chaSOClim = 100-((charge_lim * 100 * Ts)/batt_size);


x_vars = zeros(4, length(t));
u_vars = zeros(1, length(t));
x_vars(1, 1) = 100; %initial SoC
x_vars(2, 1) = 100; %Initial Load
x_vars(3,1) = used_price(1); %initial price
x_vars(4,1) = 100; %initial grid power
u_vars(1,1) = 0;



%% Logic functions

for i = 1:length(t)
        index_vector = [i+1:10+i];
    %gets the indexes needs for indexing the laod and price vectors

    for z = 1:5
        if isempty(index_vector(index_vector>=26*z)) == 0
            index_vector(index_vector>=26*z) =
2+index_vector(index_vector>=26*z);
            %to avoid the issue of the 0 index in matlab, this was my
            %get around
        end
        check_vec = mod(index_vector, 26); %used for debugging
    end
    current_load = used_load(mod(index_vector, 26));
    current_price = used_price(mod(index_vector, 26));
    current_load = current_load(1);
    x_vars(2, i) = current_load;
    if x_vars(2, i) ~= current_load
        disp('Error')
    end

    x_vars(3, i) = current_price(1);
    current_price = current_price(1);
    if current_load < 0 %load < 0 therefore charge

        if x_vars(1, i) >= 100
            u_vars(i) = 0;
            [x_vars(1, i+1), x_vars(4, i)] = do_calcs(x_vars(:, i),
u_vars(i), Ts, batt_size);
            continue;
        end
        if current_load >= charge_lim * -1 % if is under the max that batt
can handle
            if x_vars(1, i) > chaSOClim % if we are within the charge lims
                input_needed = (100 - x_vars(1, i)) * (batt_size/(100 *
Ts));
                %finds what is needed to charge in one timestep
```

```
                if current_load * -1 > input_needed
                    % if load is larger than what is needed to charge in a
                    % timestep, just do what is needed
                    u_vars(i) = input_needed;
                else
                    % if laod is less than whats needed, just charge at
the
                    % load
                    u_vars(i) = current_load * -1;
                end
            elseif x_vars(1, i) <= chaSOClim %if SOC is below the SOClim
                if current_load * -1 > charge_lim
                    u_vars(i) = charge_lim;
                else
                    u_vars(i) = current_load * -1;
                end
            end
            % u_vars(i) = current_load * -1;
        elseif current_load <= charge_lim * -1 % if wont be able to supply
the entire load
            if x_vars(1, i) > chaSOClim
                input_needed = (100 - x_vars(1, i)) * (batt_size/(100 *
Ts));
                if charge_lim > input_needed
                    u_vars(i) = input_needed;
                else
                    u_vars(i) = current_load * -1;
                end
            end
        end
    elseif current_load > 0
        if x_vars(1, i) <= 0
            u_vars(i) = 0;
            [x_vars(1, i+1), x_vars(4, i)] = do_calcs(x_vars(:, i),
u_vars(i), Ts, batt_size);
            continue;
        end
        if current_load < discharge_lim * -1 %
            if x_vars(1, i) < disSOClim % might need to be chaSOClim
                input_needed = (0 - x_vars(1, i)) * (batt_size/(100 *
Ts));
                if current_load * -1 < input_needed
                    u_vars(i) = input_needed;
                elseif current_load * -1 > input_needed
                    u_vars(i) = current_load * -1;
                end
            elseif x_vars(1, i) >= disSOClim
                u_vars(i) = current_load * -1;
            end
        elseif current_load >= discharge_lim * -1
            if x_vars(1, i) < disSOClim
                input_needed = (0 - x_vars(1, i)) * (batt_size/(100 *
Ts));
                if discharge_lim < input_needed
                    u_vars(i) = input_needed;
                else
                    u_vars(i) = discharge_lim;
                end
            else
```

```
                u_vars(i) = discharge_lim;
            end
        end
    end
    [x_vars(1, i+1), x_vars(4, i)] = do_calcs(x_vars(:, i), u_vars(i), Ts,
batt_size);




end

total_cost = 0;
for k = 1:length(t)
    if x_vars(4, k) > 0
        total_cost = total_cost + x_vars(4, k) * x_vars(3, k);
    elseif x_vars(4, k) < 0
        total_cost = total_cost + x_vars(4, k) * 1;
    end
end

disp("Grid is: ")
sum(x_vars(4, :))
disp("money spent is: ")
total_cost


f = figure;
f.Position(2:4) = [300 800 600];
tiledlayout(3, 1)
nexttile
yyaxis left;
plot(x_vars(1, :), 'LineWidth', 1.5)
axis([0 length(t) -10 130])
xlabel("Simulation Timestep (hour)")
ylabel("SOC%")
a = get(gca, 'XTickLabel');
set(gca,'XTickLabel',a,'FontWeight','bold')
yyaxis right
hold on;
plot(x_vars(4, :), 'LineWidth', 1.5)
axis([0 length(t) -150 150])
hold on;
plot(zeros(1, length(t)), 'LineWidth', 1.5)
title("Simulation Performance")
legend("SOC%", "Net Grid", "0 Ref Line")
legend('Location','northeastoutside')


nexttile;
plot(x_vars(2, :), 'LineWidth', 1.5)
axis([0 length(t) -110 110])
title("Net Load Profile Across Entire Simulation")
xlabel("Simulation Timestep (hour)")
ylabel("Net Load (kW)")
a = get(gca, 'XTickLabel');
set(gca,'XTickLabel',a,'FontWeight','bold')
hold on;
plot(zeros(1, length(t)), 'LineWidth', 1.5)
```

```
legend("Net Load", "0 Ref Line ")
legend('Location','northeastoutside')

nexttile
plot(u_vars, 'LineWidth', 1.5)
axis([0 length(t) -110 110])
title("Battery Input/Output Across Simulation")
xlabel("Simulation Timestep (hour)")
ylabel("Battery Input (kW)")
a = get(gca, 'XTickLabel');
set(gca,'XTickLabel',a,'FontWeight','bold')
hold on;
plot(zeros(1, length(t)), 'LineWidth', 1.5)
legend("Battery Input", "0 Ref Line")
legend('Location','northeastoutside')
```

7.8.2 Supplementary Calculations Code

```
function [x1, x4] = do_calcs(x_vars, u_vars, Ts, batt_size)

x1 = LogicStateFcn(x_vars, u_vars, Ts, batt_size);
x4 = x_vars(2) + u_vars;
```

7.8.3 Logic State Function Code

```
function x1 = LogicStateFcn(x, u_in, Ts, batt_size)
x1 = x(1) +  (100*Ts/batt_size) * u_in;
```