REINFORCEMENT LEARNING-BASED EVENT-TRIGGERED ACTIVE BATTERY CELL BALANCING CONTROL FOR ELECTRIC VEHICLE RANGE EXTENSION

by

DAVID FLESSNER

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHATRONIC SYSTEMS ENGINEERING

2024

Oakland University
Rochester, Michigan

Thesis Advisory Committee:

Jun Chen, Ph.D., Chair
Xia Wang, Ph.D.
Manohar Das, Ph.D.

To my love Ava, and to all of our future successes and learning from our failures

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my adviser, Jun Chen, Ph.D. for the extensive mentorship and encouragement. It has been a long and windy road, and Dr. Chen provided invaluable guidance throughout this endeavor. In addition, I would like to thank the committee members Xia Wang, Ph.D. and Manohar Das, Ph.D. for their contributions to this dissertation as well as the members of the ELISA lab for their insights and collaboration.

I would also like to give heartfelt appreciation to my family for their support, my colleagues and friends for their inspiration and creativity, and everyone who has guided me along this path. I could not do it without you.

David Flessner

ABSTRACT

REINFORCEMENT LEARNING-BASED EVENT-TRIGGERED ACTIVE BATTERY CELL BALANCING CONTROL FOR ELECTRIC VEHICLE RANGE EXTENSION

by

DAVID FLESSNER

Adviser: Jun Chen, Ph.D.

Optimal control techniques such as model predictive control (MPC) have been widely studied and successfully applied across a diverse field of applications. However, large computational requirements for these methods result in a significant challenge for embedded applications. While event-triggered MPC (eMPC) is one solution that could address this issue by taking advantage of the prediction horizon, one obstacle that arises with this approach is that the event-trigger policy is complex to design to fulfill both throughput and control performance requirements.

To address this challenge, this thesis proposes to design the event-trigger through training a deep Q network reinforcement learning agent (RLeMPC) to learn the optimal event-trigger policy. This control technique was applied to an active cell balancing controller for range extension of an electric vehicle battery. Simulation results with MPC, eMPC, and RLeMPC control policies are presented. For RLeMPC, multiple iterations of the reward function were tested, and the key improvements and challenges of each are discussed and supported by analysis of simulation data. Specifically, for reward functions defined based on distance driven, challenges of scaling the reward and learning from very delayed rewards are identified. Reward functions based on cell SOC variation performed better especially with the introduction of a mechanism to penalize consecutive, high frequency triggering.

TABLE OF CONTENTS

TABLE OF CONTENTS—Continued

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CVL	Charge Voltage Limit
DNN	Deep Neural Network
DVL	Discharge Voltage Limit
DQN	Deep Q Network
eMPC	Event-Triggered Model Predictive Control
EV	Electric Vehicle
FTP	Federal Test Procedure
MPC	Model Predictive Control
OCP	Optimal Control Problem
RL	Reinforcement Learning
RLeMPC	Reinforcement-Learning Event-Triggered Model Predictive Control
SOC	State of Charge

CHAPTER ONE

INTRODUCTION

1.1 Motivations

Model predictive control (MPC) is an advanced control technique that has been widely studied and successfully applied in many applications, including active battery cell balancing [1]. MPC determines control actions by using a model of the system in combination with a cost function to determine the optimum sequence of control actions that minimizes the cost function over a future time horizon. The capability of handling constraints on the control action and state variables in the formulation of an optimal control problem (OCP) is a key feature that distinguishes MPC as a versatile tool. However, the large magnitude of computations required to solve the OCP at every time step remains a drawback for MPC implementation on embedded systems. Techniques have been investigated for reducing the computational time needed such as explicit MPC [2] and online fast MPC [3].

Another approach, event-triggered MPC (eMPC), triggers the MPC controller to determine a new set of control actions when the defined trigger conditions are met and otherwise, follow the predicted optimal control sequence. The stability of the event-triggered MPC are discussed in detail in papers [4] and [5], which provide essential insights and methodologies for ensuring the stability of eMPC. In general, the stability of event-triggered MPC can be proven by Lyapunov stability theory where the cost function can be selected as the Lyapunov function. In practice, [6, 7] developed an eMPC controller for a DC-DC boost converter and compared to an MPC controller to demonstrate the computational reduction with minimal control performance degradation. Additionally in [8], an eMPC controller was developed for autonomous vehicle path tracking and demonstrated in a simulation environment.

However, the design and calibration of this trigger is not trivial, so to realize the largest benefit from it, a reinforcement learning (RL) agent has been developed and trained to trigger when to solve the OCP for eMPC. This is known as RLeMPC and has been applied in autonomous vehicle path-following problems in [9, 10]. It was found in [9, 10] that RLeMPC can outperform threshold-based eMPC in key metrics related to both throughput and control performance. This work then builds on that RLeMPC formulation by using a deep neural network (DNN) to model the Q-function to determine the event-trigger for eMPC, with a particular focus on the application of active cell balancing for electric vehicle (EV) batteries. Compared to existing works such as [11] and [12] where an RL agent is used to trigger and schedule communications, this particular formulation is unique in the objective to reduce throughput rather than to save communication bandwidth.

On the other hand, the battery cell imbalance is a critical issue that limits the range of electrical vehicles (EV) [13]. In battery packs with multiple cells, the individual cells have state-of-charge (SOC) and terminal voltage variation between them that arises from manufacturing variation and uneven aging among other factors [13–15]. Discharging an individual cell below a minimum voltage results in accelerated degradation of the pack capacity and safety risks, so to prevent these conditions, any individual cell must not be discharged below a discharge voltage limit (DVL). This dynamic results in the total usable energy of the battery pack being limited by the lowest capacity cell when this cell reaches the DVL before any of the other cells while discharging. Similarly, this dynamic also plays out while charging as well where any individual cell should not be charged above a charging voltage level (CVL) to avoid cell damage and safety risks. Therefore, the charging process is then limited to whichever cell has the lowest capacity and charges to the CVL first.

Cell balancing is a technology that has arisen to address these issues [16]. Cell balancing methods can be classified into dissipative and nondissipative methods with the difference being whether the balancing method relies on resistive elements, ie. energy dissipative, or charge transfer, i.e. nondissipative [17]. An additional dichotomy can be drawn between active and passive cell balancing that distinguishes methods that require active control from those that rely on passive circuit elements. This paper focuses on nondissipative, active cell balancing with the goal of reducing the amount of capacity loss of the battery pack by redirecting charge in cells exhibiting voltage imbalance when discharging and charging to maximize the usable battery pack energy.

1.2 Contributions

MPC-based active cell balancing has also been widely studied in the literature. In [18], a reachability analysis was performed to demonstrate the optimal control and range extension of an EV battery pack, and then in [19], an MPC controller was developed achieving close to the maximum range extension. In [20], an MPC controller was developed and validated on a physical bench test property for active cell balancing of a auxiliary power module. The foudation of this work comes from [21] where an MPC controller was developed and evaluated for active cell balancing to extend the range capability of EV batteries. Multiple cost function formulations were evaluated to determine a cost function that increases the range of the vehicle while minimizing throughput. It was concluded in [21] that all the cost function formulations tested resulted in high computation reduction, but the cost function based on minimizing tracking error of each cell voltage was the most robust against model mismatch and load disturbance. Therefore, in this paper, we focus on the development of the tracking MPC into RLeMPC with the addition of an event trigger for the OCP driven by a deep RL agent that determines when to trigger. This development is expected to improve upon the previous MPC-based approach through decreasing required throughput by reducing the amount of

times the OCP must be solved while maintaining similar range extension performance. It is also expected to improve the design of the event-trigger by replacing the conventional threshold-based policy with a DNN.

1.3 Outline

The remainder of the thesis is organized as follows. Chapter 2 provides background of the framework of RL and MPC and how these control strategies can be used for active cell balancing. Chapter 3 describes the broader scope of previous approaches to these problems that this thesis builds from. Chapter 4 highlights the approach and methodology used to develop the RLeMPC agent. Chapter 5 follows by presenting the simulation results from applying these techniques to cell balancing. Finally, Chapter 6 provides the key takeaways and concludes the thesis.

CHAPTER TWO

BACKGROUND

2.1 Preliminary on RL and MPC

2.1.1 Reinforcement Learning

The RL paradigm [22, 23] operates on a state-action framework where at time step t, the state of the environment, s_t, is observed by an agent which then selects an action a_t to take. After the action is taken, a scalar reward is given to the agent depending on the reward function denoted $r(s_t, a_t)$. The goal of the agent is to learn an optimal policy $\pi^* : s \to a$ that maximizes the expected cumulative future rewards which can be described as

$$G = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \tag{2.1}$$

where $r_t = r(s_t, a_t)$ and the scalar $\gamma \in [0, 1]$ is the discount factor that reduces the value of future expected rewards. In order to learn the optimal policy π^*, the agent interacts with the environment to learn which actions to take at given state s_t in order to maximize the expected cumulative reward G.

To measure the value of the agent being in state s, a state value function $V_\pi(s)$ can be defined as the value of the state s under the policy π, which is the expected return starting from s following policy π. This can be expressed as

$$V^\pi(s) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^t r_{t+k} \,\middle|\, s_t = s \right] \tag{2.2}$$

Alternatively, the state-action value function $Q_\pi(s, a)$ of a state-action pair (s, a) can be defined as the expected return starting from s followed by action a and then policy π. This can be expressed as

$$Q^\pi(s, a) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t = s, a_t = a \right] \tag{2.3}$$

Through agent interaction with the environment, the Q-function can be learned, and once it is approximately known, the optimal Q-function $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ is the Q-function with a policy applied to maximize the Q value. This policy π^* can be defined as

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{2.4}$$

The RL agent will then be able to execute the optimal policy π^* by selecting the action that results in the highest Q value for a given state.

To restate, the optimal policy π^* can be found by learning the optimal Q-function $Q^*(s,a)$. The objective of RL training is then to learn $Q^*(s,a)$, which is also referred as as Q-learning. With this method, the RL agent can learn the optimal Q-function exclusively by interacting with the environment. Notably, no model of the system dynamics is required for the agent to learn the optimal policy. However, in exchange for this benefit, ample training time is required for the RL agent to learn Q^*.

2.1.2 Deep Neural Network Based RL

In order to decide an action, the RL agent requires a method to generalize the state-action value function over any state and action. For this purpose, a deep neural network (DNN) was used with the environment state variables as inputs to the network, and the expected, discounted, cumulative future reward of each action, the Q-value, at the given state as the output. This implementation of RL is known as Deep Q-Learning because the DNN models the state-action value function or $Q(s,a)$ [24, 25]. Consequently, the DNN can be more specifically described as a Deep Q Network (DQN).

To train the DQN, double Q-learning, ε-greedy, and batch update methods were used. For each update, a batch of experiences (s_i, a_i, r_i, s_i') of size M is pulled from memory to use. For double Q-learning, two sets of network weights and biases are used, denoted as ϕ for the critic network parameters and ϕ_t for the target network parameters.

Depending on the state, ϕ is used to determine the next action with probability $1-\varepsilon$. Otherwise, a random action is selected to explore for off-policy actions that may provide more reward than the currently learned policy. After the action is taken, ϕ is updated according to a target value y_i. This target is determined with the double Q-learning and batch methods by

$$a_{max} = \arg\max_{a'} Q(s_i', a', \phi) \tag{2.5a}$$

$$y_i = r_i + \gamma Q_t(s_i', a_{max}; \phi_t) \tag{2.5b}$$

For double Q-learning, ϕ is used to select the action a_{max}, and ϕ_t is used to determine the target estimate of the experienced Q-value from taking a_i action in s_i state. These target values y_i are then included in a loss function that compares the target value that was observed to the current critic network estimate. Stochastic gradient descent can be applied to this loss function to determine a new ϕ that minimizes L depending on the gradient of the loss function and the learning rate.

$$L = \frac{1}{2M} \sum_{i=1}^{M} (y_i - Q(s_i, a_i; \phi))^2 \tag{2.6}$$

The new weights and biases ϕ are then set in the behavior network for the agent to select the next action. After a set amount of training time steps, the target network parameters ϕ_t are set to the behavior network parameters ϕ in order to increase the stability of learning and reduce overestimates of Q.

The goal of training the network is to adjust the DQN parameters while the RL agent interacts with the environment to closely approximate the actual Q^* function. Then, this Q^* function, as the optimal policy, can be used to select the action with the highest expected reward.

2.1.3 Event-triggered MPC

Generally, event-triggered MPC (eMPC) builds off a conventional time-triggered MPC controller by adding an event-trigger layer and changing how the control command is computed in the absence of an event. Mathematically, for a system described by the following dynamics

$$\zeta_{t+1} = f(\zeta_t, u_t) \tag{2.7}$$

where $\zeta_t \in \mathbb{R}^n$ is the system state at discrete time t and u_t is the controller output. The MPC controller calculates the optimal control sequence U_t and optimal state sequence ζ_t to minimize a cost function J_{mpc} over a defined prediction horizon p through solving an OCP defined as

$$\min_{U_t} \quad \sum_{k=1}^{p} J_{mpc}^{k}(U_t) \tag{2.8a}$$

$$\text{s.t.} \quad \zeta_t = \hat{\zeta}_t \tag{2.8b}$$

$$\zeta_{t+k} = f(\zeta_{t+k-1}, u_{t+k-1}), \quad 1 \leq k \leq p \tag{2.8c}$$

$$\zeta_{min} \leq \zeta_{t+k} \leq \zeta_{max}, \quad 1 \leq k \leq p \tag{2.8d}$$

$$u_{min} \leq u_{t+k} \leq u_{max}, \quad 1 \leq k \leq p-1 \tag{2.8e}$$

$$\Delta_{min} \leq u_{t+k} - u_{t+k-1} \leq \Delta_{max}, \quad 1 \leq k \leq p-1 \tag{2.8f}$$

where U_t and Z_t are defined as $U_t = \{u_t, u_{t+1}, ..., u_{t+p-1}\}$ and $Z_t = \{\zeta_{t+1}, \zeta_{t+2}, ..., \zeta_{t+p}\}$. The optimization is subject to the current estimate of the state $\hat{\zeta}_t$ (2.8b), subsequent states that only depend on the previous state and control action taken (2.8c), and constraints that are applied to the state and control action (2.8d-f). Conventionally, this OCP is solved at every time step where the first control action u_t is applied then the rest of the control sequence U_t is not used. eMPC in contrast only solves the OCP when the event conditions are met, denoted by γ_{ctrl} where $\gamma_{ctrl} = 1$ when the event conditions are met and otherwise

$\gamma_{ctrl} = 0$. When $\gamma_{ctrl} = 0$, the controller applies the predicted optimal control sequence U_{t_1} computed at the previous trigger at time t_1. This can be described as

$$u = \begin{cases} \text{Solution of (8)} & \text{if } \gamma_{ctrl} = 1 \\ U_{t_1}(k+1) & \text{Otherwise.} \end{cases} \tag{2.9}$$

To summarize, the primary new features that distinguish eMPC from conventional MPC are that the trigger γ_{ctrl} must be changed from a consistent periodic time trigger to event-based and that it applies the predicted optimal control sequence U_t calculated until the next event is triggered. The trigger policy then becomes the focus of the design which can be described as

$$\gamma_{ctrl} = \pi(Z_{t_1}, \hat{\zeta}_t | \theta) \tag{2.10}$$

where Z_{t_1} is the optimal state sequence computed at the last event at time t_1, $\hat{\zeta}_t$ is the current state feedback, and θ is calibration parameters for the trigger. Typically, the event is based on the error between the optimal predicted state calculated at the last event and the current state feedback or estimation such that if the error is large between the state prediction ζ_t from the prediction sequence Z_{t_1} and current actual state $\hat{\zeta}_t$, then trigger the MCP to use the more accurate feedback to determine a new control sequence. This type of event-trigger condition can be described as

$$\left\| \zeta_t - \hat{\zeta}_t \right\| \geq \bar{e} \tag{2.11}$$

where $\bar{e} \in \theta$ is a calibrate-able error threshold. Details for eMPC are described in Algorithm 2.1.

Algorithm 2.1: Event-Triggered MPC [26]

1. 0

(2) **Procedure** *eMPC*

(3) $\quad \hat{\zeta}, k, U_{t_1}, Z_{t_1}, p$

(4) $k \leftarrow k+1$;

(5) $\gamma_{ctrl} \leftarrow$ computing (2.10);

(6) **if** $\gamma_{ctrl} = 1$ **then**

(7) $\quad k \leftarrow 0$;

(8) $\quad (Z_t, U_t) \leftarrow$ Solving OCP (2.16);

(9) $\quad u \leftarrow U_t(1)$;

(10) $\quad U_{t_1} \leftarrow U_t$;

(11) $\quad Z_{t_1} \leftarrow Z_t$;

(12) **end**

(13) **else**

(14) $\quad u \leftarrow U_{t_1}(1)$

(15) $\quad$ **if** $k \leq p$ **then**

(16) $\quad\quad \hat{\zeta} \leftarrow Z_{t_1}(k)$;

(17) $\quad$ **end**

(18) $\quad$ **else**

(19) $\quad\quad \hat{\zeta} \leftarrow Z_{t_1}(p)$;

(20) $\quad$ **end**

(21) **end**

(22) **return** u, U_{t_1}, Z_{t_1}, $\hat{\zeta}$

However, the challenge associated with (2.11) is uncovered then because the analytical form of the MPC closed loop system, especially for nonlinear functions and models is difficult to determine. This results in event-trigger policy designs which are usually problem specific and non-trivial. To solve this challenge, an RL agent with a DQN is proposed to learn the optimal event-trigger policy π^* without a model of the closed-loop system dynamics.

2.2 Problem Formulation

2.2.1 Active Battery Cell Balancing Control

A common equivalent circuit model of a lithium ion cell can be used to model the SOC, relaxation voltage, and terminal voltage of each cell [27, 28]. The cell model can be described as

$$\dot{s}^n = -\eta^n \frac{i^n}{C^n} \tag{2.12a}$$

$$\dot{V}_p^n = -\frac{V_p^n}{R_p^n C_p^n} + \frac{i^n}{C_p^n} \tag{2.12b}$$

$$y^n = V_{oc}^n - V_p^n - i^n R_o^n \tag{2.12c}$$

where the superscript n is the nth cell, s^n is the cell SOC, η^n is the cell coulombic efficiency, C^n is the cell capacity in amp hours, V_p^n is the relaxation voltage over R_p^n, V_{oc}^n is the open circuit voltage, y^n is the terminal voltage, and i^n is the cell current. Positive i^n indicates discharging the battery while negative i^n indicates charging. The variables V_{oc}^n, R_o^n, R_p^n, and C_p^n are all dependent of s^n resulting in a nonlinear cell model.

This model can then be discretized using Euler's method for a model that can be implemented digitally as

$$s_{k+1}^n = s_k^n - \eta^n \frac{T_s}{C^n} i_k^n \tag{2.13a}$$

$$V_{p,k+1}^n = V_{p,k}^n - \frac{T_s}{R_p^n C_p^n} V_{p,k}^n + \frac{T_s}{C_p^n} i_k^n \tag{2.13b}$$

Table 2.1: Cell Imbalance Parameters at 100% SOC

Param.	Unit	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
C	Ah	62.87	60.00	66.61	56.73	61.66
R_p	mΩ	6.40	5.66	5.47	6.68	6.36
C_p	kF	153.7	177.8	175.9	168.9	150.4
R_o	mΩ	1.49	1.27	1.41	1.51	1.53

$$y_k^n = V_{oc,k}^n - V_{p,k}^n - i_k^n R_o^n \tag{2.13c}$$

To simulate imbalance, a probability distribution can be used to apply random variation to the C^n, R_o^n, and C_p^n terms. Table 2.1 lists example variations of these parameters that would later be used for simulation in this paper.

Finally, the SOC and relaxation voltage states can be grouped together as $\zeta^n := [s^n, V_p]^T$ where $.^T$ denotes a matrix or vector transpose. The battery pack state can be written as

$$\zeta_{k+1}^n = f^n(\zeta_k^n, i_k + u_k^n) \tag{2.14a}$$

$$y_k^n = g^n(\zeta_k^n, i_k + u_k^n) \tag{2.14b}$$

where u_k^n is the balancing current applied to the cell as shown in Fig. 2.1. Define $\zeta = [\zeta^1, \zeta^2, ..., \zeta^N]^T$ as the state vector for the battery pack and y to be the terminal voltage of the battery pack, then

$$\zeta_{k+1} = \begin{bmatrix} f^n(\zeta_k^1, i_k + u_k^1) \\ f^n(\zeta_k^2, i_k + u_k^2) \\ \vdots \\ f^n(\zeta_k^N, i_k + u_k^N) \end{bmatrix} \tag{2.15a}$$

$$y_k = \sum_{m=1}^{N} y_k^n = \sum_{m=1}^{N} g^n(\zeta_k^n, i_k + u_k^n) \tag{2.15b}$$

For this study, a 5 cell in series pack is assumed resulting in the pack terminal voltage equal to the sum of the cell terminal voltages. For the active cell balancing power converter, an ideal converter with limits on each balancing current and no charge storage is assumed. This power converter also assumes a direct cell-cell topology meaning that charge from any cell can be directed to any other cell with the rate of charge transfer limited by balancing current limits depicted in Fig. 2.1. These constraints along with the cell model can be formulated as an optimal control problem for MPC at time step k over prediction horizon p as

$$\min_{U_t} \quad \sum_{k=1}^{p} J_{mpc}^{k}(U_t) \tag{2.16a}$$

$$\text{s.t.} \quad \zeta_{k+j+1}^{n} = f^{n}(\zeta_{k+j}^{n}, i_{k+j} + u_{k+j}^{n}),$$
$$0 \leq j \leq p-1, 1 \leq n \leq N \tag{2.16b}$$

$$y_{k+j}^{n} = g^{n}(\zeta_{k+j}^{n}, i_{k+j} + u_{k+j}^{n}),$$
$$1 \leq j \leq p, 1 \leq n \leq N \tag{2.16c}$$

$$u_{min} \leq u_{k}^{n} \leq u_{max}, \quad 1 \leq n \leq N \tag{2.16d}$$

$$\underline{y} \leq y_{k+j}^{n}, \quad 1 \leq j \leq p, 1 \leq n \leq N \tag{2.16e}$$

$$0 = \sum_{i=1}^{N} u_{k}^{n} \tag{2.16f}$$

The cost function chosen for this study is based on minimizing the tracking error between each individual cell to a nominal cell without any imbalance which was studied in [21]. This MPC formulation can be presented mathematically as

$$J_y(u_k) = \sum_{j=1}^{p} (y_{k+j} - y_{k+1}^{0})^T (y_{k+j} - y_{k+1}^{0}) + u_k^T R u_k \tag{2.17}$$

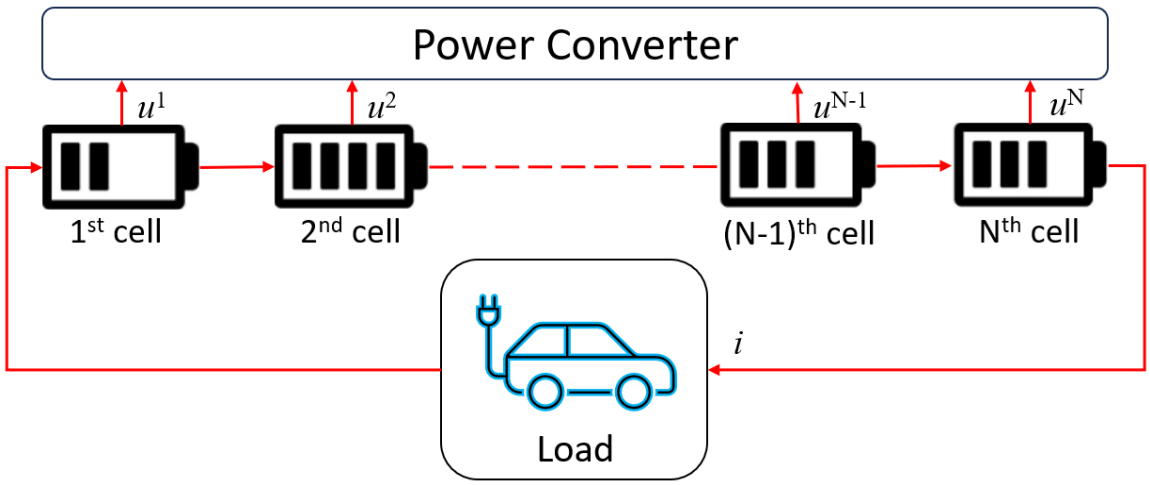

Figure 2.1: Battery pack configuration for active cell balancing.

where y_{k+j} is defined as the vector of cell terminal voltages $y_{k+j} = [y^1_{k+j}, y^2_{k+j}, \ldots y^N_{k+j}]^T$, and R is a positive semi-definitive weighting matrix. The first term penalizes cell voltages that deviate from the nominal cell while the second term penalizes the magnitude of balancing current to reduce resistant heating loses. The nominal cell voltge target is a scalar which leads to needing only the R weighting matrix to tune the cost of the terms.

2.2.2 RLeMPC for Active Cell Balancing

The MPC controller that solves (2.16) periodically is then adapted to eMPC by first integrating an event-trigger into the MPC model using the difference between the predicted voltage of a nominal cell and the measured terminal voltage of each cell compared to a calibrate-able threshold $\bar{e}$ as the trigger condition described in (2.11). When the event is not triggered $\gamma_{ctrl} = 0$, the controller holds the first of the balancing commands of the previous calculated series U_{t_1} instead of using the subsequent elements of U_{t_1}. This modification was used because the future driver power demand is assumed to be unknown and the dynamics of the terminal voltage are relatively slow compared to the controller sample rate of 1s. This implementation of eMPC was studied to compare to a constant time-triggered MPC baseline and an RL-based event-trigger.

In addition to this eMPC implementation, an RL agent was developed to replace the trigger conditions. The RL agent was setup as a DQN agent described above in Section II.B. This agent interacts with environment (2.15) and observes the average and minimum cell voltage, average cell SOC, and pack current demand as state variables as well as the reward. Although each cell voltage is considered as the state in (2.15), only these more general state parameters were used to reduce the number of dimensions of the state for training. During training, the agent will select a random action with probability ε that decays exponentially over time and otherwise, will select the action with the greatest value determined by the critic network with parameters ϕ. This action is the event-trigger for the MCP controller to solve the OCP which then determines a new U_{t_1} and Z_{t_1}.

With the reward function being one of the key design parameters for the RL agent, multiple reward function formulations were defined and tested to explore a few alternatives. First, the reward for each time step was defined as the distance driven over the time step dx in meters subtracted by a flag representing whether or not the trigger was set γ_{ctrl} multiplied by a weighting factor ρ together as

$$r = dx - \gamma_{ctrl} * \rho \tag{2.18}$$

This formulation is a kind of "brute force" reward that includes the key metrics that should be optimized. However, the distance aspect of this reward function was found to be difficult for the agent to learn since all of the driving distance is considered with the range extension portion being a small part of the whole and delayed to the end. These aspects will be discussed further in the results chapter.

Next, to isolate the range extension portion of the distance in the reward function, Eqn 2.19 was defined to not consider any of the distance traveled before the actual range

extension took place.

$$r = \begin{cases} -\gamma_{ctrl} * \rho & \text{if Distance Traveled} < 46 \text{ km} \\ \frac{dx}{1000} - \gamma_{ctrl} * \rho & \text{Otherwise.} \end{cases} \tag{2.19}$$

46 km was defined from the distance that could be driven on the training test cycles without any cell balancing, and dx was divided by a factor of 1000 to scale the units of the distance reward component for an episode between 0-5 to analyze the learning process more easily. Although this function emphasizes the range extension aspect of the distance, an obstacle that remains is that the range extension only occurs at the end of an episode which hampers effective learning at the beginning of the episode from the very delayed reward.

Finally, in an effort to address the much delayed reward, Eqn 2.20 was defined where σ_{SOC} is the standard deviation of the cells of the pack.

$$r = -\sigma_{SOC} - \gamma_{ctrl} * \rho \tag{2.20}$$

The expectation is that (A) σ_{SOC} would correlate well with the range extension since the range extension comes about from reducing cell-to-cell variation and also that (B) the SOC variation would be a more dynamic reward that the agent could observe well ahead of the range extension. In the results chapter, (A) is found to not be true with the conclusion that not all triggers contribute the same to the range extension or simply, not all triggers are equal. This led to a final modification of using an eligibility trace to penalize consecutive triggering.

On each step, if a trigger occurs, the eligibility trace e_t is incremented by 1, otherwise without triggering, e_t decays by λ. This can be described as

$$e_t = \begin{cases} \lambda e_{t-1} & \text{if } \gamma_{ctrl} = 0 \\ \lambda e_{t-1} + 1 & \text{if } \gamma_{ctrl} = 1 \end{cases} \tag{2.21}$$

This trace was then included in the reward function as

$$r = -\sigma_{SOC} - e_t * \rho \tag{2.22}$$

With e_t, the RL agent is encouraged to distribute the triggers more evenly through time to approach periodic trigger behavior but with more to freedom trigger when σ_{SOC} can be improved.

With these RL state and reward formulations, many hyperparamters such as ρ, the structure of the DQN, learning rate, discount factor, and epsilon decay were tuned in a simulation environment while training the agent to find the optimum event trigger policy. Details for training the RLeMPC agent can be found in Algorithm 2.2.

Algorithm 2.2: RL-based Event-Triggered MPC

1. 0
(2) **Procedure** *RLeMPC*
(3) | M, T, dt, γ, N
(4) Initialize ϕ, $\mathscr{D} \leftarrow \emptyset$, $\phi_t \leftarrow \phi$
(5) **for** $j = 0$ *to* $N-1$ **do**
(6) | Initialize s_t, Z_{t_1}, U_{t_1}, $k \leftarrow 0$, $u \leftarrow 0$
(7) | **while** $t \leq T$ **do**
(8) | | **if** *explore* **then**
(9) | | | Sample a_t from $\{0,1\}$
(10) | | **end**
(11) | | **else**
(12) | | | $a_t \leftarrow Q(s_t;\phi)$
(13) | | **end**
(14) | | <Simulate Environment>
(15) | | Execute **eMPC Algorithm 2.1** lines 6-21
(16) | | $\zeta_{t+1} \leftarrow$ Simulate (2.13) using u
(17) | | $s_{t+1} \leftarrow \zeta_{t+1}$
(18) | | $r_t \leftarrow$ (2.18 - 2.22)
(19) | | <End of Environment Simulation>
(20) | | Observe r_t and s_{t+1}
(21) | | Update $\mathscr{D}$ to include (s_t, a_t, r_t, s_{t+1})
(22) | | Sample M experiences from $\mathscr{D}$
(23) | | $\phi \leftarrow$ Perform gradient descent on (2.6)
(24) | | **if** *Target Update Condition is True* **then**
(25) | | | $\phi_t \leftarrow \phi$
(26) | | **end**
(27) | | $s_t \leftarrow s_{t+1}$
(28) | | $t \leftarrow t + dt$
(29) | **end**
(30) **end**

CHAPTER THREE

RELATED WORK

3.1 MPC Throughput Reduction Methods

Multiple different approaches have been taken to address the large amount of throughput required by MPC. A first method is called explicit MPC which is "explicit" because the control law is computed offline for various partitions of the state space, and then this control law can be stored and ran much faster on the controller [2]. The overall components of this method include first determining how to partition the state space with piece-wise affine functions, and then determining the optimal control laws for each partition. With these computed offline, the partitions and control laws are loaded on the controller. The controller then must determine the state and use it to determine which control law to select. These control laws require much less calculations than an implicit MPC quadratic-programming solver since they are usually linear equations that can be used to compute the control actions quickly. In this way, throughput can be traded for memory to store the partition regions and control laws. Determining the partitions is one of the primary challenges for this method especially as the state space dimension increases.

Another approach includes fast MPC where the optimization problem is still solved online, but methods exploiting the structure of the MPC optimization are used to enable much less computations [3]. One core exploitation of a quadratic programming optimization that can be done is related to the block tridiagonal system of linear equations that arises for the interior-point search direction when the variables are ordered properly. Exploiting the structure can result in the scale of the number of computations to be linear instead of cubic with respect to the prediction horizon. Other improvements such as warm-starting where the calculations for each step are initialized by the predictions made

in the previous step and early termination where the accuracy required from the optimization is reduced are additional methods of reducing the computations for MPC.

eMPC and RLeMPC has been already been described in detail but more can be said on particular applications with it. Wang et al. [29] implemented eMPC for control of a buck power converter and measured reduced computational burden and switching losses. Additionally, Li et al. [30] implemented distributed eMPC for multiagent communication systems to reduce frequency of the control law update and reduce power consumption. Finally, RLeMPC was applied for autonomous vehicle path-following [9, 10] where the trade off between the control performance and trigger frequency was analyzed based on the tuning of the RL agent. Also in [31], the formulation of RLeMPC is described and demonstrated on inverted pendulum stability and swing up to save on communication while maintaining control performance.

3.2 Active Cell Balancing Control Methods

In [16], the landscape of battery cell balancing methods was reviewed, and the main five categories of active cell balancing were described in detail: cell bypass, cell-to-cell, cell-to-pack, pack-to-cell, and cell-to-pack-to-cell. [32] described the topologies and control methods required to enable these active cell balancing methods such as capacitor based, inductor based, and converter based. Trade-offs between control complexity, reliability, balancing speed, cost, and efficiency were outlined in the choice of the balancing circuit topology. Many control methods have been explored such as fuzzy control [33], distributed consensus control [34], and constrained optimal control [19, 35–37].

For active cell balancing, a benchmark of the optimal control actions to maximize battery capacity performance was done by [18] where the active cell balancing problem was formulated in a reach-ability analysis. The analysis determined that the optimal balancing actions were constant currents throughout depending on each cells capacity to

minimize balancing losses. The analysis also provides a top benchmark of range extension achievable with a given battery pack.

For real-time control, [19] demonstrated 5% range extension without making assumptions about the drive cycle for a realistic assessment of perforce achievable. Additionally, [35] implemented an auxiliary power module with MPC for active cell balancing of the high voltage battery and charging the low voltage battery. This thesis builds off the work of [37] where multiple MPC cost functions were formulated and tested over steady-state and transient cycles. A cost function based on tracking all of the cells terminal voltages to the predicted trajectory of a nominal cell was the most robust. Many challenges arise with implementing optimal control methods for active cell balancing. In addition to the computational load required by MPC that this thesis addresses, other challenges for implementing an MCP control include battery state estimation [38] and scalability to large battery packs [39].

CHAPTER FOUR

METHODOLOGY

4.1 Approach

One of the key challenges with the RLeMPC control strategy was many hyper-parameters were available for training as shown in Table 4.1. Each of these can significantly impact the performance and interact among the others which made tuning of these hyper-parameters very complex. To avoid a length calibration process of these parameters, the testing focused on identifying and resolving training issues of the reward function to iterate toward a reward function that could be robust and learn well across a wide range of hyper-parameters. Poor performance related to tuning of some of the training hyper-parameters was identified, but the primary motivation was to roughly tune the training hyper-parameters in order to focus on reward function design and iteration.

From iterative improvement, three main reward functions were defined, tested, and analyzed to define obstacles encountered related to the reward function and paths around these obstacles. To assist with analyzing and evaluating the RLeMPC performance, initial evaluations were done with constant frequency MPC and eMPC control strategies to compare with the RLeMPC training and final policies.

4.2 Simulation Environment

Simulations were executed for the training and testing of the active cell balance controls using MATLAB and Simulink with the Reinforcement Learning Toolbox to create and train DQN agents inside of a Simulink environment [40]. As described above, the system being simulated is a battery pack with 5 cells in series and an ideal power converter that can move charge between any cells constrained by balancing current limits for each cell of ± 2 A.

Table 4.1: Key Training Hyper-Parameters

Param.	Description
α	Learning Rate
γ	Discount Factor
Initial ε	Starting probability of taking random actions
ε-decay rate	Value to multiply epsilon by after every time step
Max Episodes	Number of episodes to train for
DQN Architecture	Number and types of layers in DQN
Initial ϕ	Initial values of DQN parameters
M	Mini-batch size
Experience Buffer Length	Length of experience buffer to draw mini-batches from
ρ	Multiplier for trigger penalty
λ	Value to multiply eligibility trace by after every time step

A high-level view of the Simulink model used is shown in Fig 4.1. From left to right the subsystems are vehicle power to battery pack current conversion, voltage reference generation used by the MPC controller, the MPC controller, and finally, the battery pack cell models. The core changes made to this model were through the MPC triggering shown as the trigger to both the voltage reference and MPC subsystems. For eMPC, explicit conditions were defined for the trigger depending on the voltage reference and cell voltage, and for RLeMPC, this top layer was wrapped by an RL layer that would input the MPC trigger action and receive the calculated reward and state observations.

Repeated FTP-72 drive cycle conditions were tested over the sedan EV configuration as used in [21]. The discharge current from the battery pack was scaled from the power demand to assume a larger pack with additional modules in parallel, and a final scaling factor was applied to increase the current draw and reduce simulation time. Starting with all cells fully charged to CVL, the battery was discharged according to the scaled current demand of the vehicle until the first cell reached DVL. The velocity profile and scaled vehicle power demand that was applied to the 5S cell module is shown in Fig

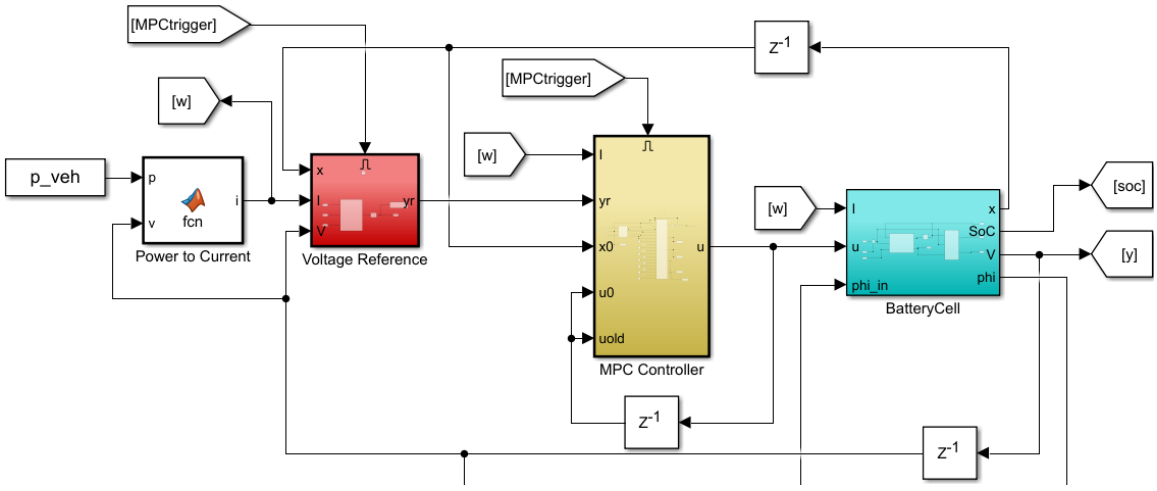

Figure 4.1: Top layer of Simulink model used for virtual analysis.

4.2 where on subsequent cycles, phase 1 of the cycle is repeated for the higher power demands to discharge the battery faster.

Finally, for all the simulations, a constant imbalance was applied across the C^n, R_o^n, C_p^n, and R_p^n cell parameters. To select these parameters, a series of simulations were run that multiplied each of the nominal cell parameters by random factors that were selected from a normal distribution with a mean of 1 and interval between 0.9 and 1.1. From these tests, a set of imbalance factors that resulted in an average active cell balancing range extension benefit for the configuration with the conventional MPC approach was chosen as the constant set of imbalance parameters to use for the rest of the simulations. This was done because the magnitude of the imbalance determines the range extension benefit that can be realized with active cell balancing. With less realizable range extension benefit, the sensitivity of the range extension depending on the control strategy reduces as it changes between MPC, eMPC, and RLeMPC. Future work would include how to generalize these approaches to any distribution of imbalance and quantify the benefit relative to the distribution of cell imbalances, especially for the RL agent which was trained and evaluated using only one set of imbalance parameters for this study.

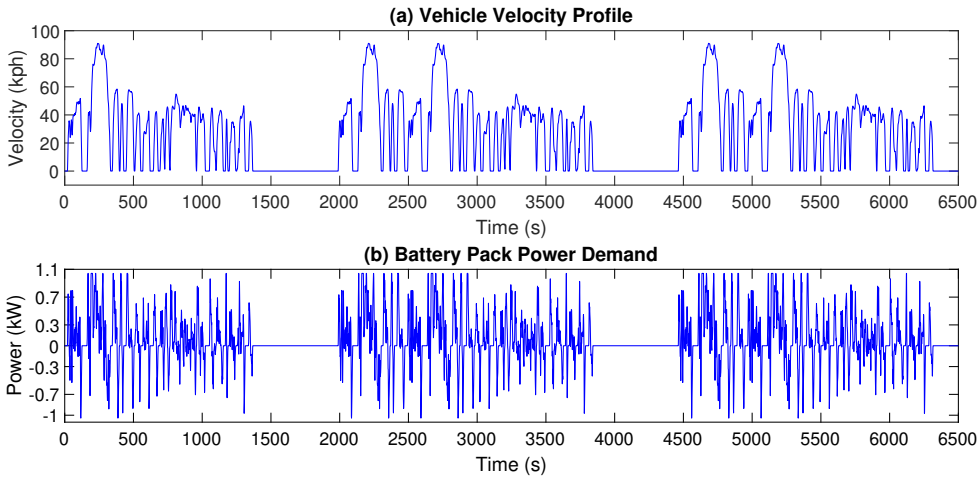

Figure 4.2: Velocity profile for repeated FTP-72 drive cycles and resulting scaled power requested from 5S cell module.

4.3 Evaluation Criteria

The primary performance metrics for this study are the overall driving range and average event-trigger frequency. The purpose of cell balancing in general is to achieve the maximum energy output of the battery which would translate to maximizing the driving range assuming no auxiliary loads. Moreover, average execution frequency is used as an approximation of the computational load with the goal of minimizing it with RLeMPC. In addition, the magnitude of the balancing currents averaged over the drive cycle is considered to approximate the resistant heating losses and referred to as the balancing effort in the sequel.

CHAPTER FIVE

RESULTS

5.1 Results with Constant Trigger Period

Before executing simulations with varying trigger frequencies, the MPC weighting matrix R in Eq. (2.17) was re-calibrated to be more robust toward infrequent triggering. For conventional MPC or eMPC with minimal modeling errors, the weighting of R could be decreased to tune the cost optimization toward lower voltage tracking error of the cells to a nominal cell voltage target at the cost of higher balancing currents. However, for this application where future driver power demand is assumed to be unknown, weighting to prioritize aggressively tracking the reference voltage can result in reduced range extension. The unknown future driver power demand becomes a disturbance in the model prediction which increases as the trigger frequency reduces. Furthermore, the prediction horizon may occasionally be much less than the trigger period resulting in prediction error even if power demand was known ahead of time. Because of these unknown dynamics to the model, increasing the cost of the balancing current magnitude through the weighting matrix R can increase the range performance of the system during infrequent triggering by reducing the response of the controller to a model with large prediction errors.

This dynamic is shown in Fig. 5.1, where the cost weighting of each balancing current magnitude R^n is set to be equal, scaled appropriately, and tested to determine range extension and balancing effort dependencies. For these tests, the value of R^n was varied for transient cycle discharge tests while having a constant trigger period of 5s. Increasing the R^n values leads to less balancing effort which demonstrates the effect of R on the MPC cost optimization. For R^n between 800-4000, the range is extended by 0.1% as compared to when R^n is between 10 to 70. Although a small difference, this result demonstrates that with these model assumptions, larger R^n values can result in larger range extension. This

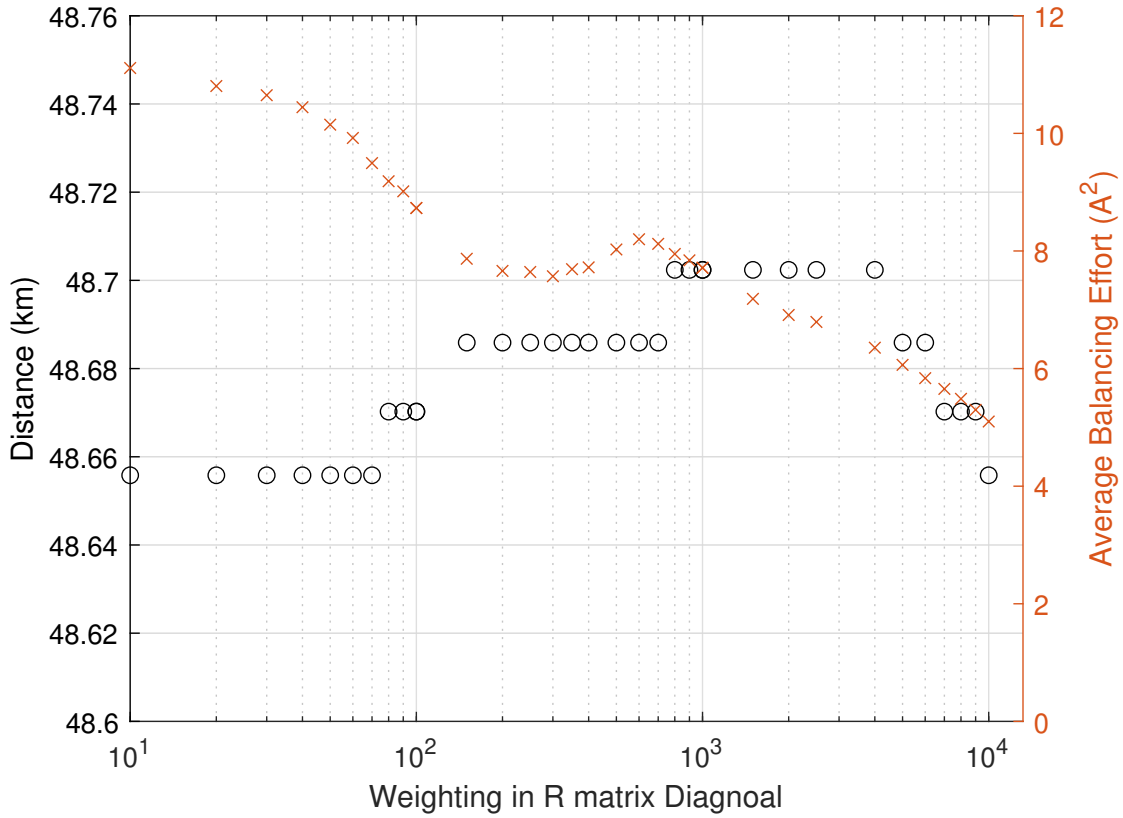

Figure 5.1: R matrix sweep with time-triggered MPC with trigger period of 5 s.

result may not be intuitive since range extension is gained with less balancing effort, but this comes from the previously described modeling discrepancies. Once R is very large, the range extension decreases drastically as the balancing currents are greatly reduced limiting the optimal calibration range for R^n. Fig. 5.2 shows the transient cell balancing performance for the final R calibration and a constant trigger period of 5s.

Once the weighting matrix was calibrated, the first study that was completed was varying a constant trigger frequency. These tests were simulated to understand as a baseline, without any event-trigger, the range extension and balancing current magnitudes with only varying the trigger frequency of the MPC controller. For this purpose, the trigger frequency was set to a constant value starting from 1Hz and decreased between

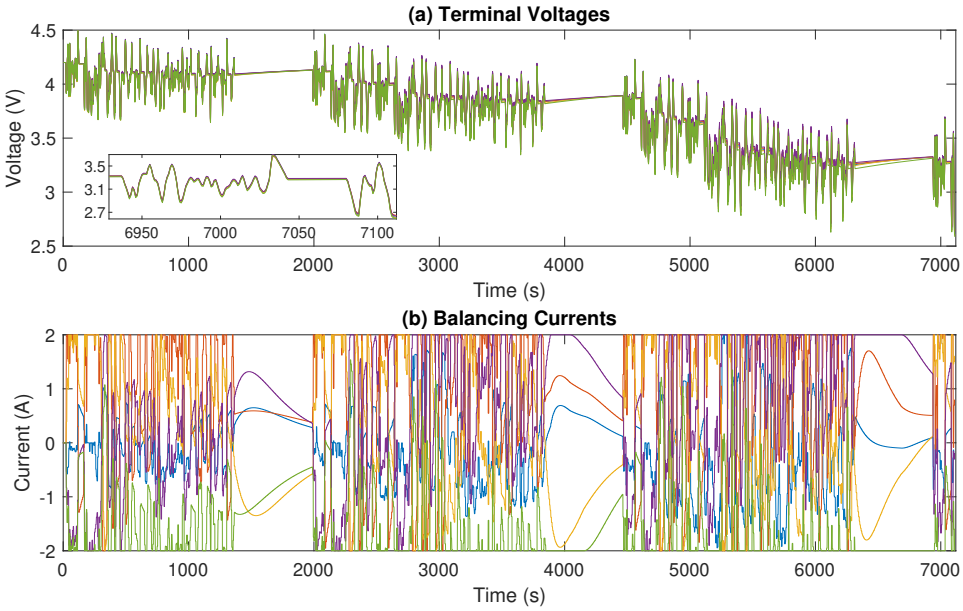

Figure 5.2: Results for 5 s constant trigger MPC cell balancing: terminal voltage and balancing currents.

simulations until 0 triggers occurred. This trigger frequency range demonstrates the maximum and minimum driving ranges achievable. For these tests, the prediction horizon remained as 5 s with only the first element of the control sequence U_t being applied until another trigger occurred. Notably, the time period between triggers can be greater than the prediction horizon as described in Section 2.2.2.

The transient cycle range extension results plotted in Fig. 5.3 indicate that the maximum range of 48.66-48.67 km can be achieved with a constant trigger period of 1 to 700 s or frequency of 1Hz to 1 mHz with varying balancing effort as the trigger period changed. Noticeably, two discrete ranges emerge with the higher trigger frequency tests from 1 Hz down to 1 mHz resulting around this maximum range and tests with frequencies below 1 mHz ending around a minimum range of 46.23-46.24 km. Fig. 5.4 shows effective cell balancing with a constant trigger period of 1000 s and can be compared to Fig. 5.2 to notice the much less busy balancing current which delivered

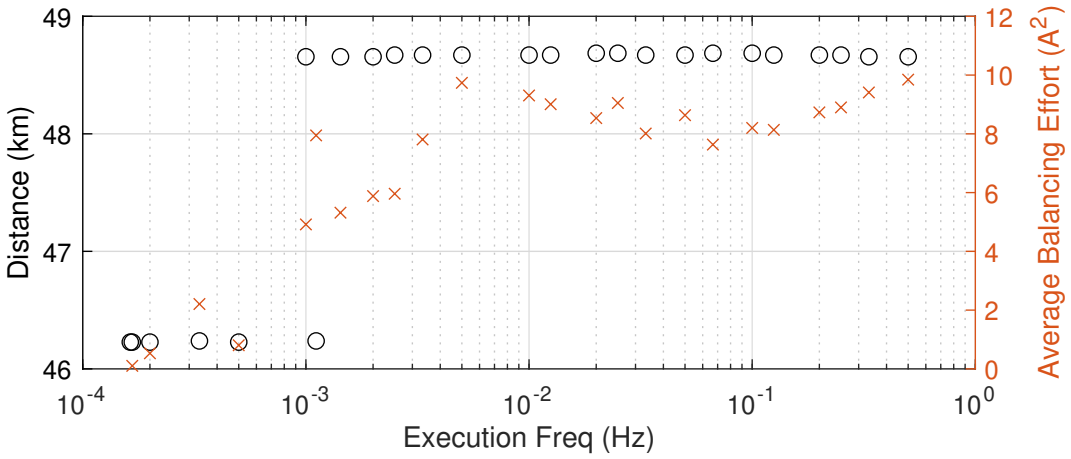

Figure 5.3: Results for varying the constant trigger period of conventional MPC.

nearly the same final driving distance. The discrete driving range levels are attributable to the current scaling that was applied to the transient cycle simulations which leads to these discrete windows emerging for when the DVL is reached.

To expand on this result, Fig. 5.5 shows the vehicle velocity profile along with the transient pack current demand and minimum cell voltage for a simulation that did not reach DVL during the time window where other simulations with poor cell balancing did reach DVL. This occurs on the third repeated cycle where DVL is reached at a pack SOC of 33% and current demand of 280-380 A. The simulations with higher trigger frequencies continued for a fourth cycle until DVL was reached at a pack SOC of 29% and close to 400 A cell current demand. Cell terminal voltages decreased significantly during these high current discharges due to the large internal resistance of the cell. This current scaling creates these discrete final driving ranges such that the first current peak where DVL is reached may be overcome with cell balancing but the second current peak where the rest of the simulations reach DVL cannot be overcome. The second current peak cannot be overcome even with perfectly balanced cells and a significant 29% SOC remaining because of the large current demands. This effect could be smoothed out for a

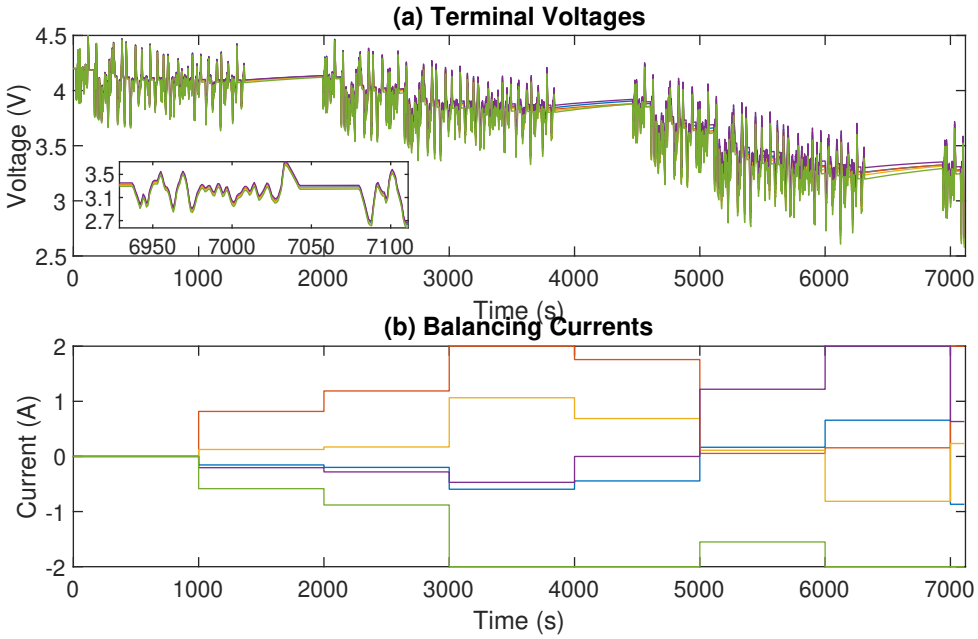

Figure 5.4: Results for 1000 s constant trigger MPC cell balancing: terminal voltage and balancing currents.

more continuous transition with more realistic cell currents at the cost of simulation time. However for this study, especially for testing RLeMPC, low simulation times was necessary to train the RL agent in a reasonable amount of time. These results are sufficient for initial concept demonstration to test if RLeMPC can determine the optimal eMPC trigger policy to overcome the first high power window.

5.2 Results with Threshold Based eMPC

For eMPC, simulations with varying the error threshold were executed to understand if an eMPC approach could outperform the constant trigger frequency MPC controller in terms of range extension, average trigger frequency, and balancing effort. Fig. 5.6 shows as the error threshold increased, the trigger frequency decreased approximately exponentially until saturating at 0-1 trigger per test. Between an error threshold of 1 and 1.5 V, the exponential relationship deteriorates as the cell balancing

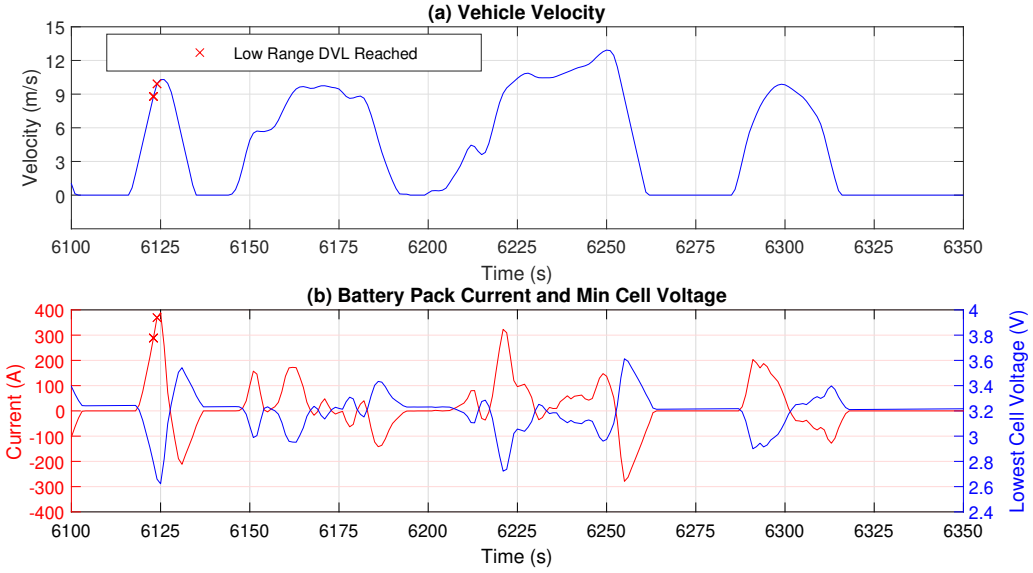

Figure 5.5: Constant trigger velocity and current profiles with red x's marking the conditions where DVL is reached for minimum driving range tests.

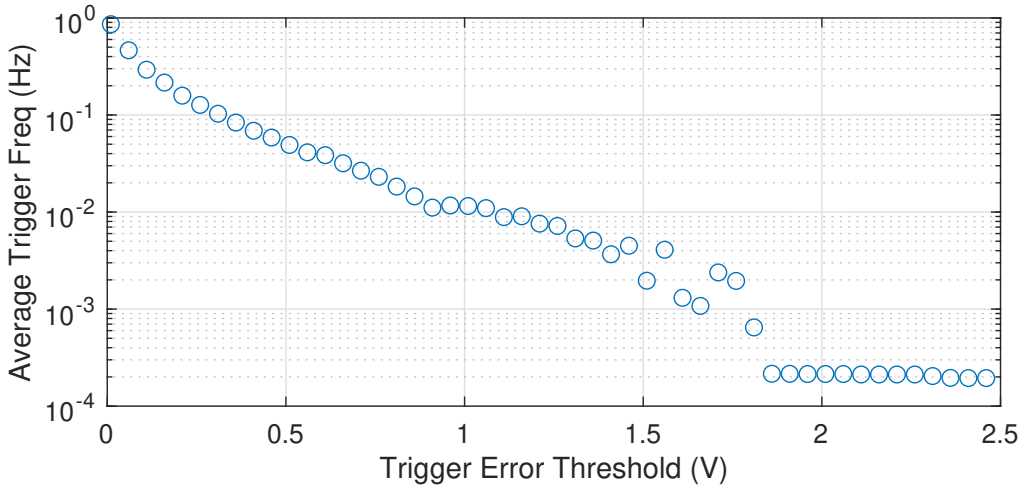

Figure 5.6: Average execution frequency as a function of the error threshold.

fails to avoid DVL during the first very high current peak as illustrated in the large step of the driving range around 10 mHz with Fig. 5.7.

Compared to the constant trigger frequency results, eMPC did not achieve as much range extension as MPC at reduced average triggering frequencies. For example, most of the trigger frequency range between 10 mHz and 1 mHz reached DVL at a driving range of 46.24 km for eMPC. For constant trigger frequency MPC at the same trigger frequency range, DVL was reached at the maximum driving range of 48.66 km. Additionally, the average balancing effort is higher on average and more variable for eMPC. Overall, constant trigger frequency MPC performs better than eMPC over the evaluation criteria for this application and trigger condition highlighting the challenge of implementing an optimal eMPC trigger condition.

Transient results for eMCP with the lowest average event-trigger frequency that achieved 48.66 km are plotted in Fig. 5.8. In this example, the benefit of eMPC is demonstrated in the lack of event-triggers during the standstill portions between repeated cycles when triggering is not required. This control strategy can be effective when the load is 0 or constant, but during the transient portions of the test, the actual cell voltages are much more transient relative to the nominal voltage target computed when the OCP is solved causing excessive triggering. Changing the target voltage to the average actual cell voltage instead of a nominal predicted cell target may be an improvement on the event-trigger policy to account for the modeling errors arising from unpredictable driver power request.

5.3 Initial Results and Tuning with RLeMPC

Beginning with the reward function in Eq. (2.18), initial attempts at training the RL agent were taken. This reward function gave positive reward to any distance traveled in meters for each time step and penalized any triggers taken with a multiplier of ρ to scale the penalty. For this problem, only a small amount of triggers relative to the

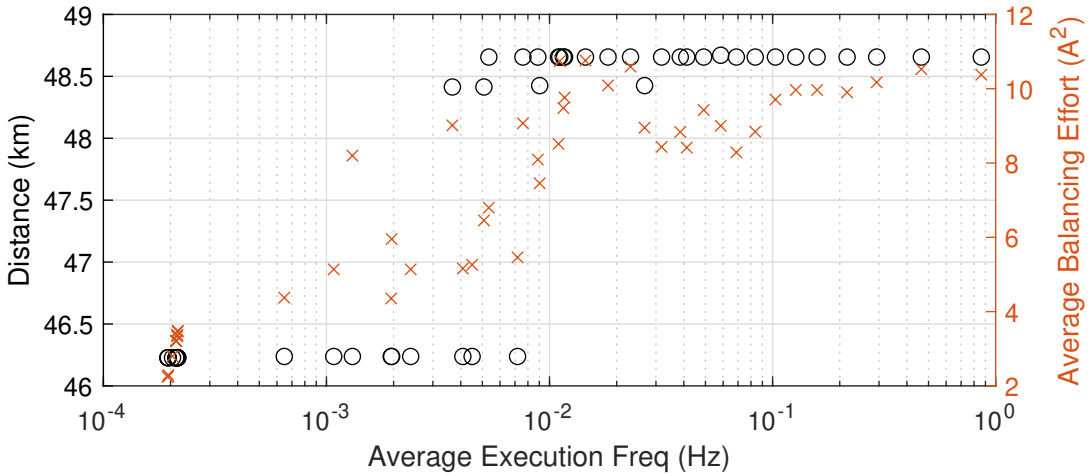

Figure 5.7: Final EV range and balancing effort depending on the average execution period for eMPC.

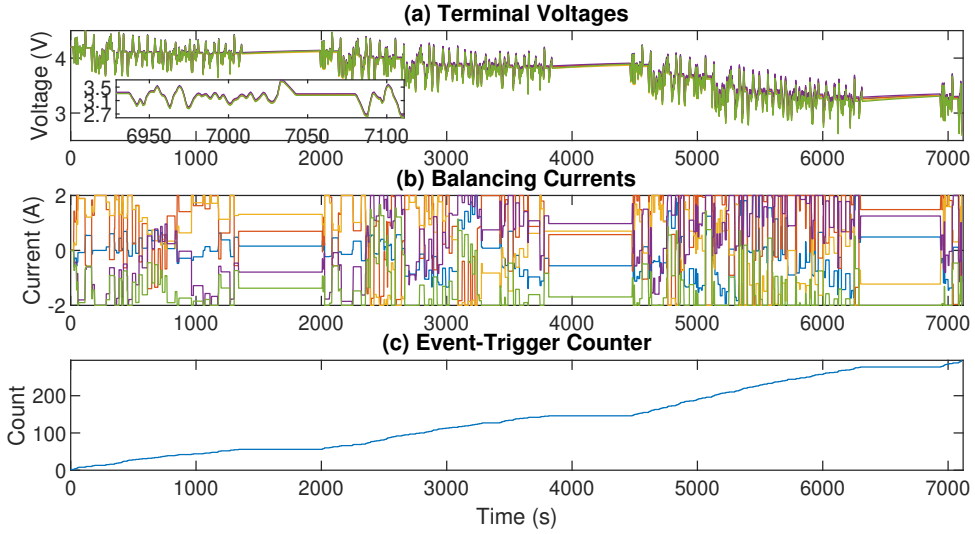

Figure 5.8: Transient results for eMPC with $\bar{e} = 1.31$ V.

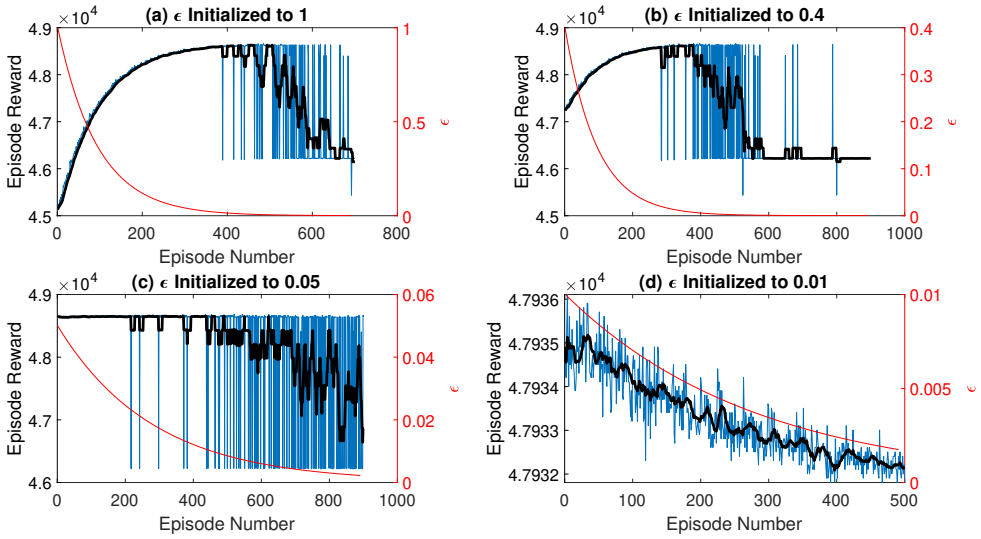

Figure 5.9: Reward and ε-decay during RLeMPC training. Blue: raw episode reward; Black: moving average of episode reward.

potential opportunities to trigger was required to achieve the maximum reward, and any amount of triggers below the minimum necessary resulted in very little range extension benefit. This feature of the pack dynamics and RL reward led the RL agent to tend to converge to sub-optimal policies of triggering very infrequently with no range extension or to triggering much more than required to maximize the range extension benefit. Depending on the scaling of ρ in the reward function, the RL agent would generally learn one of those policies erroneously as the optimal policy even after having experienced the larger driving ranges that were achievable. While training the agent, this local optimum was very difficult to avoid even with a low weighting factor ρ applied to the trigger action in the reward function.

The first challenge addressed with the RL training was the ε-decay exploration method. Typically for training RL agents, ε is initialized around 1 and decayed to a minimum value to explore the environment before transitioning to exploiting the environment to maximize G. For this problem, the maximum final range can be achieved

with very little triggering, so with high ε, and 50% probability of choosing to trigger as a random action, the agent first learned the penalty from triggering because it always achieved the maximum range at the beginning of the training. As ε decayed, the agent transitioned from random actions to following the actions with greater learned Q-value from the DQN which was to not trigger at all since, in the training experience so far, either action would lead to traveling the maximum distance. Eventually, ε decayed enough to where not enough triggers occurred from random actions to achieve the maximum range, and during this transition, it was very difficult for the agent to reverse and learn that triggering more could lead to greater range extension. Overall, since the agent could not learn any distinction between actions for improving driving range extension at the beginning of the training, it biased heavily to not triggering to avoid the trigger penalty. Next, once it did not trigger enough from random actions to overcome the first high current peak without reaching DVL, it could not learn quickly enough that triggering could lead to larger range extension reward. This caused the RL agent to fall into learning a policy of not triggering at all with no driving range extension.

This example of training plays out in Fig. 5.9(a) where the reward correlated with ε from the decrease in trigger frequency until around 400 episodes, after which triggering was not frequent enough to overcome the first high current peak. The switching observed in the episode reward was from going between the high and low final driving ranges with large reward weighting applied to the driving range extension. After this transition, the RL agent settled into a policy of not triggering at all to maximize the reward with no driving range extension.

Training with alternate tunings of the initial ε value and ε-decay rates are shown in Fig. 5.9. First, when ε was not low enough, the agent learned to not trigger falling into the local optimum of 0 triggers per episode as shown in Fig. 5.9(b) and (c). For these training sets, ε was initialized to 40% and 5%, and both resulted in the same policy of no

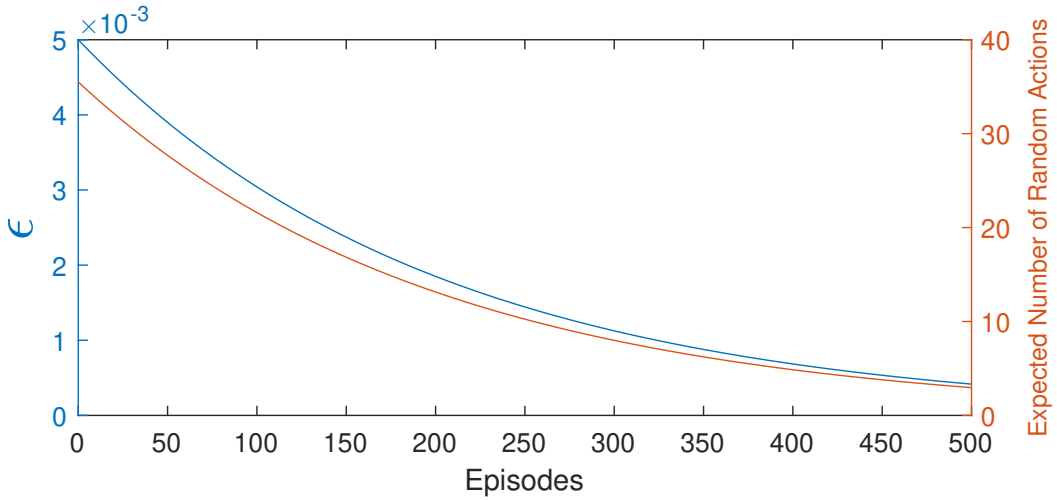

Figure 5.10: Expected ε and number of random actions per episode for new ε-greedy exploration settings of initial ε = 0.005 and decay rate = 7x10^{-7}.

triggering to maximize reward when no driving range extension is achieved. From those training sets, ε of 1-5% was expected to be a better value to begin the training since in that range was where the switching between high and low driving distance ranges began to be observed. However, initializing ε to 1% shown in Fig. 5.9(d) resulted in the reward decreasing with the maximum distance was always achieved but the RL agent learning to trigger more.

5.4 Training with Reduced Exploration

Setting aside the sensitivity of the training to the initial ε value, further tuning was undertaken to explore training sensitivity to other factors with a constant value for ε of 0.005 or 0.5%. The goal for this new exploration strategy was to understand how well the policy could be optimized in the neighborhood of the initial DQN parameters ϕ through restricting the amount of exploration. Assuming 7100 actions per episode, Fig. 5.10 shows the expected evolution of epsilon and the number of random actions per episode for the new initial ε and ε-decay rate settings for the training. Fig. 5.11 shows episode reward over training episodes with varying values of the trigger multiplier ρ.

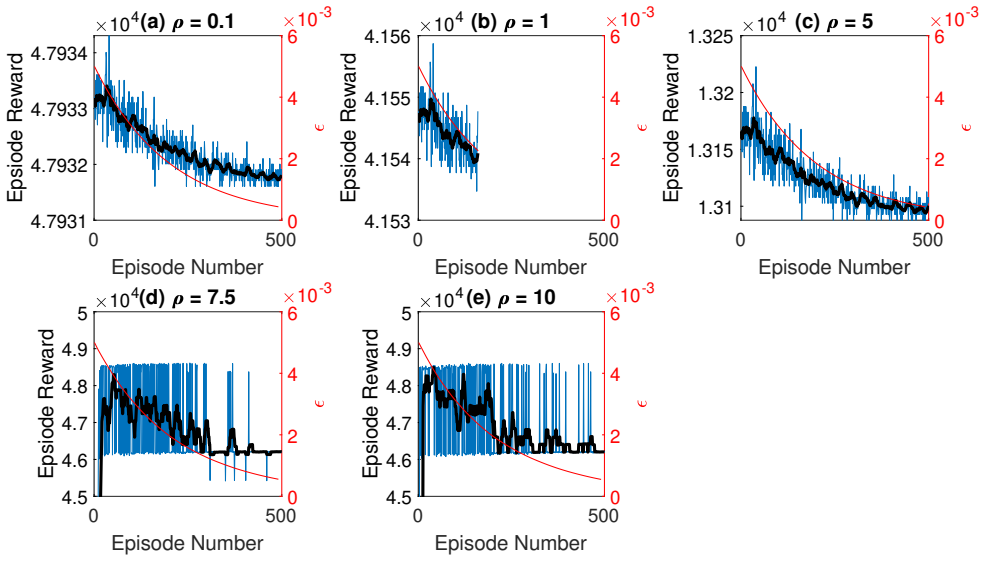

Figure 5.11: Episode reward while training with new ε-decay settings and varying ρ with $\alpha = 1\text{x}10^{-5}$. Training in (b) terminated early, but the initial trend was similar to (a) and (c).

In Fig. 5.11(a), ρ is set to 0.1 such that each trigger receives -0.1 units of reward after taking the trigger action. Fig. 5.11(a)-(c) shows up to a value of $\rho = 5$, the policy learned during the RL agent training converged toward a lower total reward than observed at the beginning of the training. Past $\rho = 5$, for $\rho = 7.5$ and $\rho = 10$ shown in Fig. 5.11(d) and (e), the episode reward during the RL agent training converged toward rewards from not triggering with minimal EV range extension. To further understand why low values of ρ learned toward less reward, the estimated Q-values over each training were analyzed.

To visualize the learning of the Q-values over time, the DQN parameters ϕ were recorded after every 10 episodes, and then the observations from a typical maximum distance episode were used as inputs to each set of saved ϕ to view how the Q-value estimates changed over the training. For example, the Q-value estimates from the initial ϕ and ϕ learned after the first episode for training with $\rho = 7.5$ are presented in Fig. 5.12(a). With these estimated Q-values for either action, the difference between them will result in

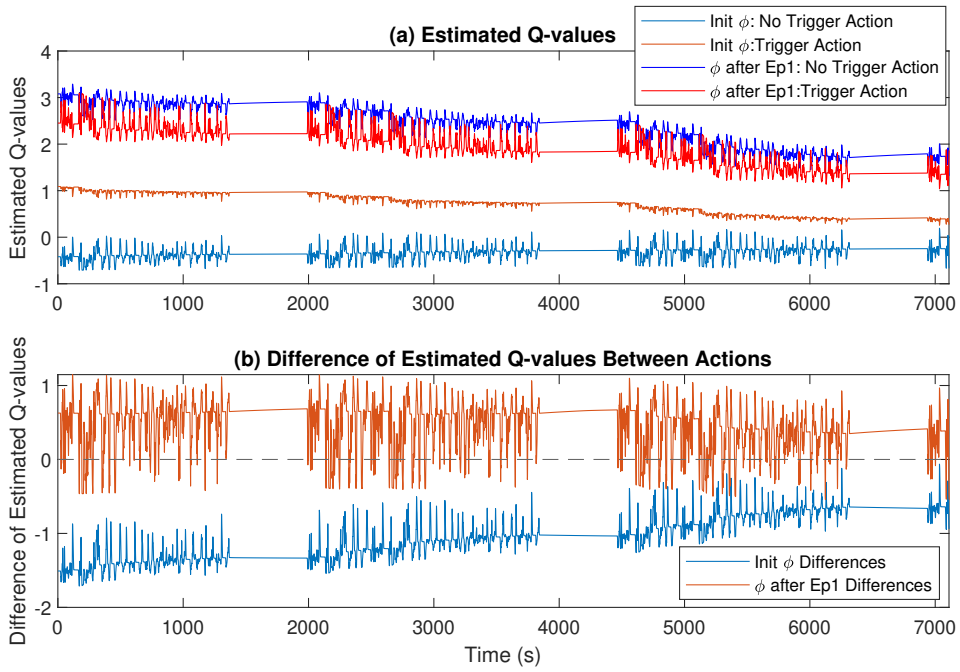

Figure 5.12: (a) Estimated Q-values and (b) the difference of Q-values estimates of the initial ϕ and ϕ learned after first episode of training with $\rho = 7.5$.

one vector that would be positive if not triggering has a higher Q-value and negative if triggering has a higher Q-value as shown in Fig. 5.12(b). With Q-learning, the Q-values estimates are all that are needed to know what on-policy actions the agent will take. This Q-difference vector and the mean and variation of it serve as indicators of the learned policy such that if it is entirely positive for an episode, then no on-policy triggers would occur, and if it is entirely negative, then every on-policy action would be a trigger throughout the episode. Qualitatively, the red trace in Fig. 5.12(b) is expected to be similar to the desired optimal policy such that for the most part, no triggering is selected with some amount of infrequent trigger actions being taken. Visualizing the mean Q-values estimates for each action and the mean and standard deviation of this Q-difference vector for each episode over the training sessions aided analysis of the training of the RL agent to identify issues with the training.

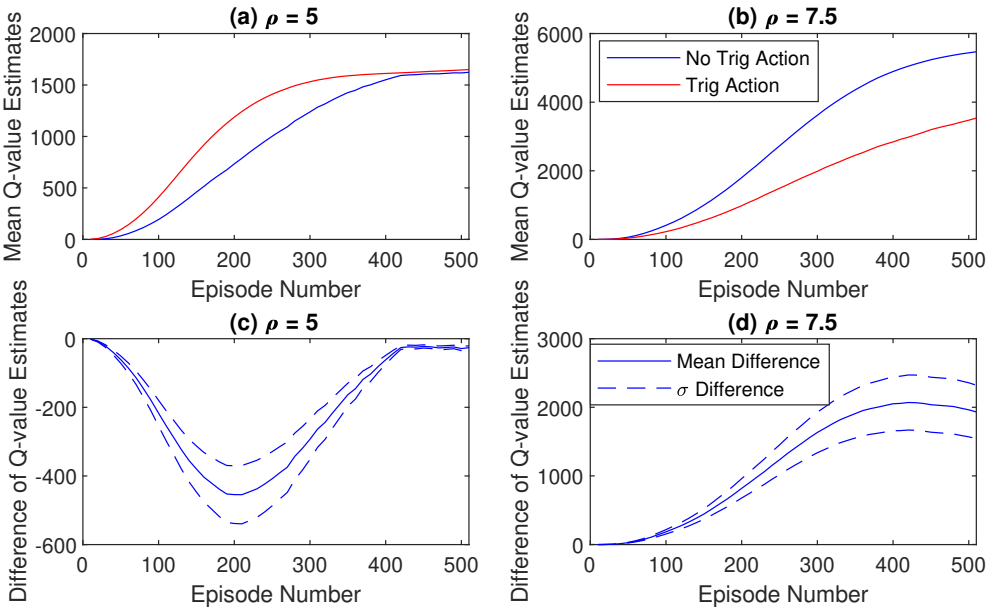

Figure 5.13: Mean Q-value estimates for either action averaged over each episode and mean and standard deviation of difference of estimated Q-values averaged over each episode throughout training with (a) $\rho = 5$ and (b) $\rho = 7.5$.

Fig. 5.13 shows how the mean and difference of Q-value estimates evolved over the training for $\rho = 5$ and 7.5. The initial Q-value estimates predicted that always triggering throughout the episode would result in more reward than always not triggering. For the always triggering case as shown in Fig. 5.13(a) and (c), the RL agent never learned a higher Q-value for the no trigger action. With $\rho = 5$ and receiving rewards much higher than the initial estimate, the learning of Q-value estimates for triggering always outpaced learning the Q-values for not triggering. Fig. 5.13(b) and (d) shows the other side where $\rho = 7.5$ was high enough to overcome the initial estimates toward triggering, but then the estimate for not triggering became so much larger than for triggering that the agent was not able to learn the benefit of triggering.

Fig. 5.13 presents a key issue with this reward function with these cases. In the case of $\rho = 5$, the RL agent learned a policy of always triggering even though it would

receive less reward than the never triggering policy since total distance traveled instead of only range extension distance was considered as a positive reward. To elaborate on the episode rewards expected for either policy, if no triggers were taken for the episode, the total episode reward would be around 46,000 from the minimal distance traveled. For ρ = 5 and a policy of always triggering, the episode reward would come out to 48,000 (from max distance) - 7,000 * 5 = 13,000 reward. However, since this 13,000 reward is much larger than the initial Q-value estimates, the Q-value estimates for triggering increased quickly. Because the agent also learns from the actions being taken, the Q-value estimates for trigger actions were learned much faster than the estimates for no trigger actions from the initial ϕ bias to triggering more. As ε decays, the agent never has an opportunity to experience the maximum reward from never triggering.

This result can be contrasted with ρ = 7.5 where a policy of always triggering would result around 48,000 - 7,000 * 7.5 = -4,500 episode reward which would be much less than the initial Q-value estimates. Because that reward would be much less than the initial estimates, the Q-value estimates for no trigger actions were able to be learned as the better actions to take, but the range extension benefit of the trigger actions was very challenging to learn because it was a very small portion of the overall distance traveled. To motivate the agent to gain that small portion of range extension reward by triggering, ρ must be small compared to the range extension, but decreasing the value of ρ to get in that range was limited from the training gravitating to the previous case of always triggering. Overall, including the total distance as positive reward in the reward function restricts how much trigger actions could be devalued to achieve range extension without very deliberate definition of the initial ϕ and exploration strategy over the training.

Another key hyper-parameter of training the RL agent is the learning rate α. With the same ε-greedy tuning as before and ρ = 7.5, four values of learning rates were tested as presented in Fig. 5.14. For reference, Fig. 5.14(b) was run with the same α and training

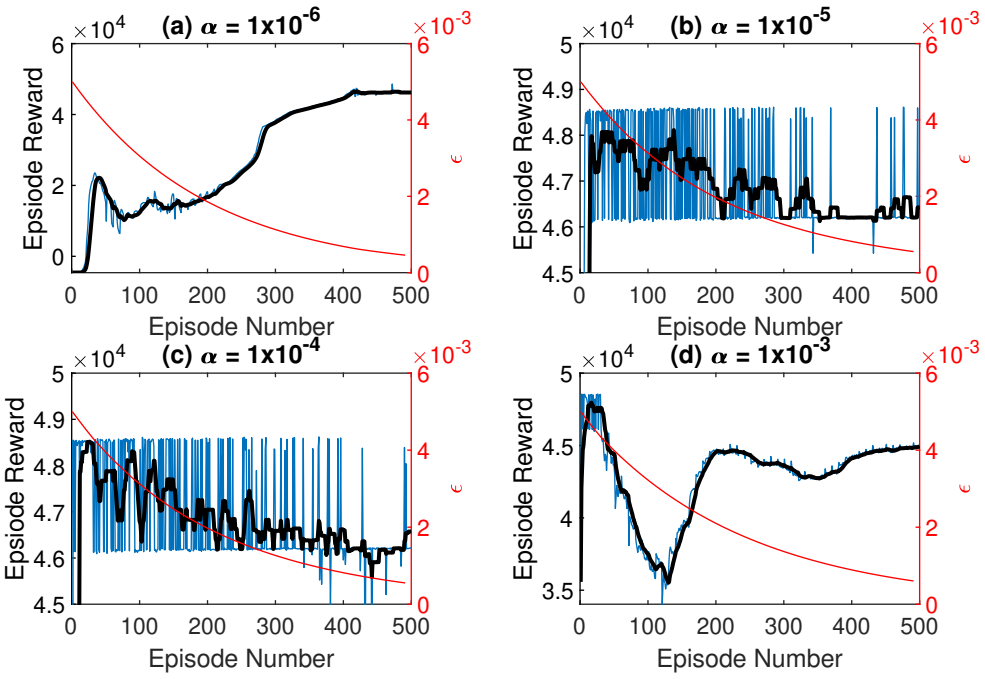

Figure 5.14: Episode reward over training sessions for varying learning rates and $\rho = 7.5$.

parameters as the previous $\rho = 7.5$ test in Fig. 5.11(d). From there, α was decreased by a factor of 10 as shown in Fig. 5.14(a) and increased twice by a factor of 10 as shown by Fig. 5.14(c) and (d). Corresponding trigger periods are presented in Fig. 5.15 and Q-value metrics are presented in Fig. 5.16.

Since the test with $\alpha = 1\text{x}10^{-6}$ was still far from convergence of Q-values with the length of the training, the results $\alpha = 1\text{x}10^{-4}$ and $\alpha = 1\text{x}10^{-3}$ were more interesting to analyze the Q-value metrics further. $\alpha = 1\text{x}10^{-4}$ followed a very similar patter of episode reward and trigger period as $\alpha = 1\text{x}10^{-5}$, but the Q-value estimates did not separate nearly as much. For $\alpha = 1\text{x}10^{-5}$, the max difference of Q-value estimates was nearly 2000 as shown in Fig. 5.13(b), but for increased α shown in Fig. 5.16(c), the max difference of Q-value estimates was only around 50. Fig. 5.16(d) with $\alpha = 1\text{x}10^{-3}$ shows similar behavior with Q-value estimates more stable than $\alpha = 1\text{x}10^{-4}$. Notably, the training with $\alpha = 1\text{x}10^{-3}$ did not converge to an always trigger or never trigger policy, but

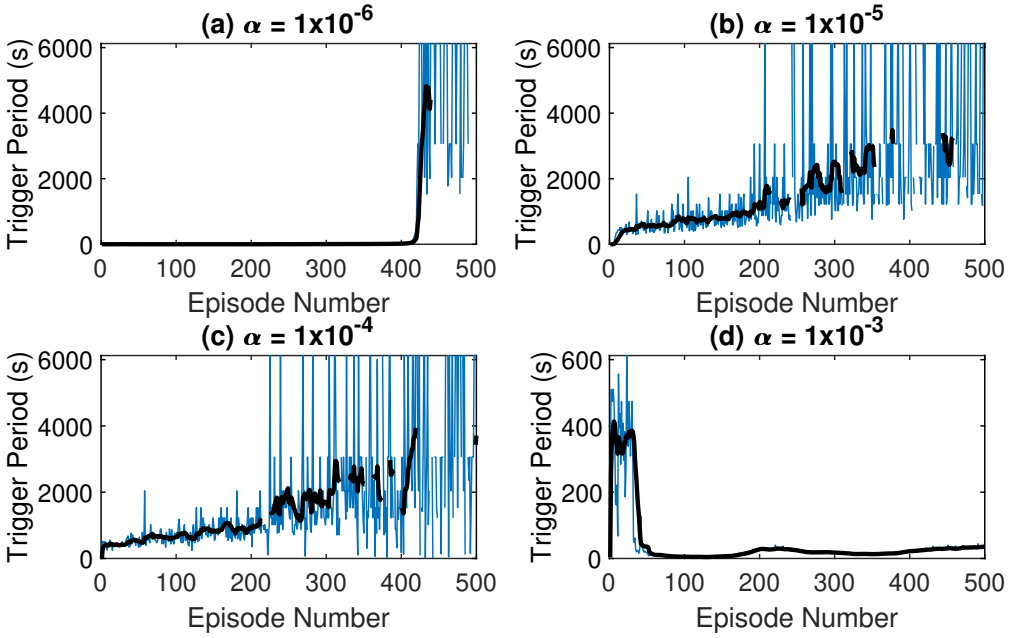

Figure 5.15: Average trigger period over training sessions for varying learning rates and ρ = 7.5.

the policy remained sub-optimal with average trigger period below 100 s while not achieving any range extension.

Overall, while increasing α showed better convergence of the Q-value estimates from a rough scale, the issue with using total distance in the reward function remained as a critical obstacle to the training. Without extending the training time and careful selection of the initial ϕ and ε-greedy parameters, the range of policies that this reward function could learn were limited. With this issue highlighted, new reward functions without the total distance included were explored.

5.5 Reward Function Based on Range Extension

To address the issue with including the total distance as positive reward, Eq. (2.19) was used as the new reward function to emphasize the range extension component of the optimization goal by creating more separation in the reward between the minimum and maximum distance. This new reward function maintains the trigger penalty for each

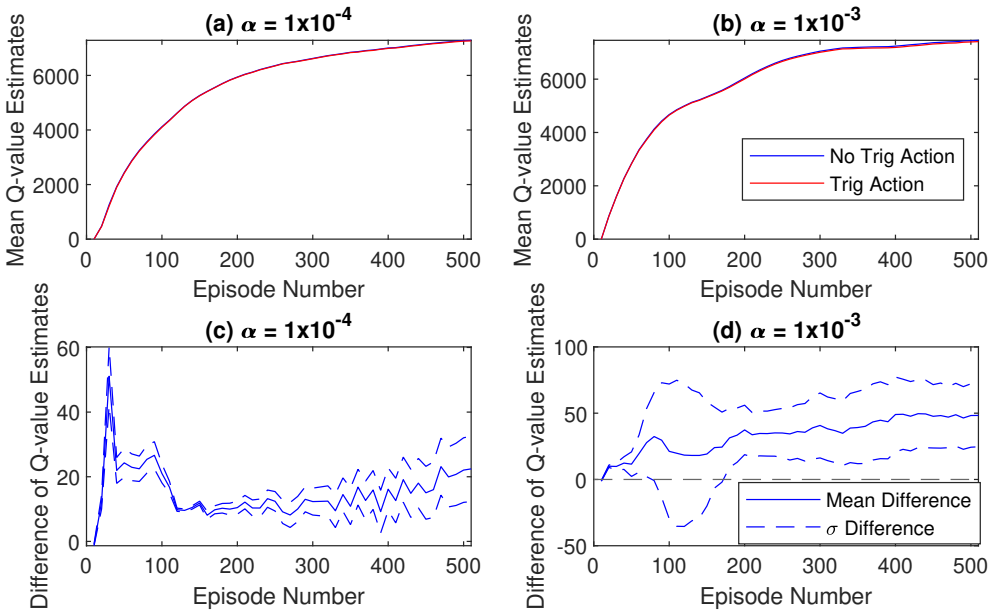

Figure 5.16: Mean and difference of Q-values over training sessions for α = 1x10^{-4} and 1x10^{-3}. Q-value difference is positive when no trigger return is expected to be higher than triggering.

trigger action, but instead of rewarding the distance traveled in units of meters after each state, it only rewards the distance traveled in units of km after a minimum distance threshold has been exceeded. The minimum distance threshold was set to 46 km to give a slight positive episode reward of 0.2 for tests without any range extension and a larger positive reward of 2.6 for maximum range extension. The key difference to the previous reward function is now the distance component increases more than 10x from min to max range extension when before the reward for max range extension would have only increased 5% from the min reward. To explore training this model, three different learning rates and three different values of ρ were tested.

For the learning rate tests shown in Fig. 5.17, each training session converged to minimal triggers with minimal range resulting in around 0.2 total episode reward. Noticeably, as α increased from $1x10^{-8}$ to $5x10^{-7}$, the initial transient of the episode

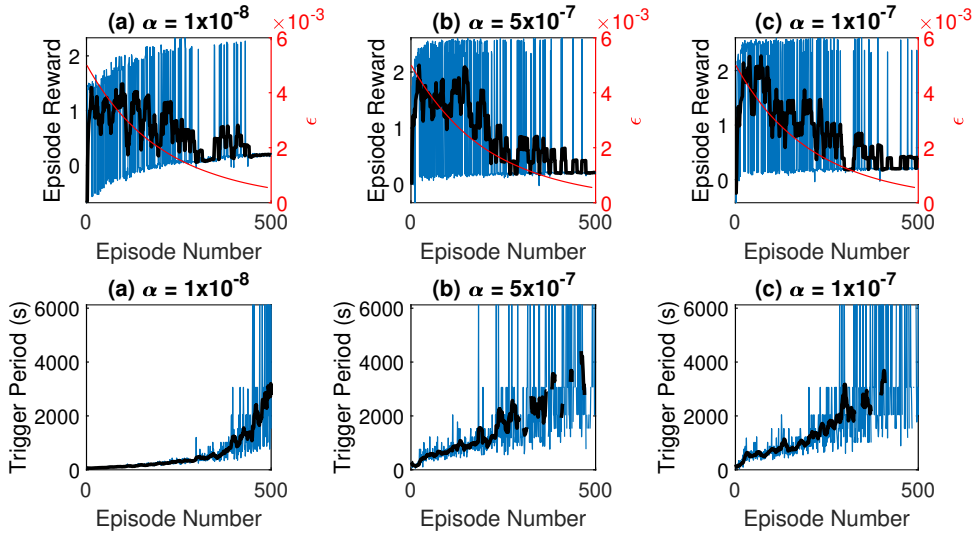

Figure 5.17: Episode reward and trigger period over training results with range extension reward function and varying learning rate with constant $\rho = 0.01$.

reward curve occurred much faster. Additionally, Fig. 5.18 displays the mean Q-values for each action and the mean and standard deviation of the differences between the Q-values of the actions throughout each episode. The learning of the Q-value estimates was drastically different for $\alpha = 1x10^{-8}$ as compared to the other two tests with the higher learning rates demonstrating the significant affect that the learning rate can have on the training. Additionally, the standard deviation of the differences of the Q-values collapsed to the mean for the higher learning rate tests indicating that the Q-values did not change significantly depending on the state. This can further be shown in Fig. 5.19 that presents the initial Q-values and the Q-values learned after 200 episodes, and after 200 episodes, the Q-value estimates for either action were constant across the episode.

Overall, these results indicated that $\alpha = 5x10^{-7}$ or greater would be required to roughly converge the Q-values estimates sufficiently. Additionally, ρ was too high for the benefit of triggering to be learned, so $\alpha = 5x10^{-7}$ was carried forward to test lower values of ρ.

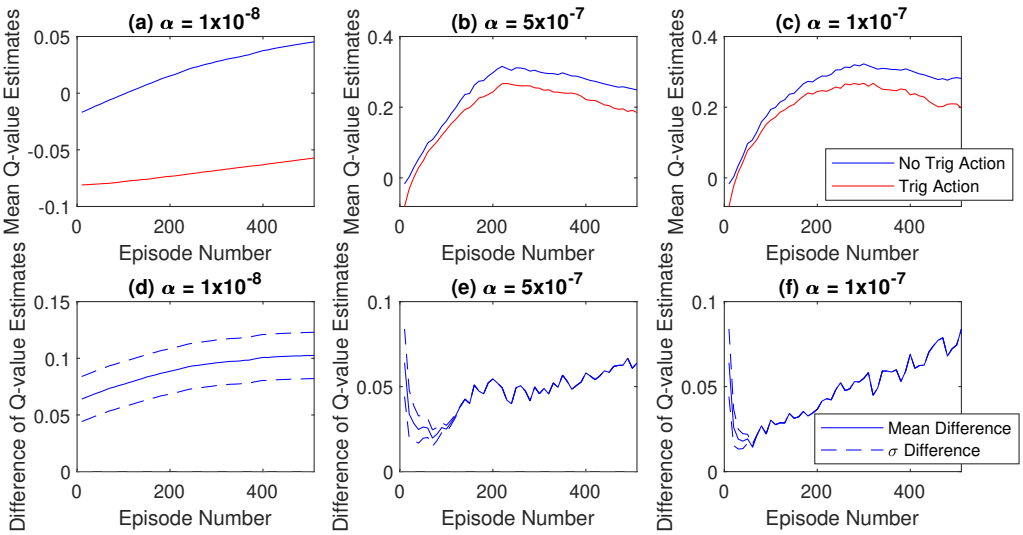

Figure 5.18: Mean and difference of Q-value estimates over training with range extension reward function and varying learning rate with constant $\rho = 0.01$. Q-value difference is positive when no trigger return is expected to be higher than triggering.

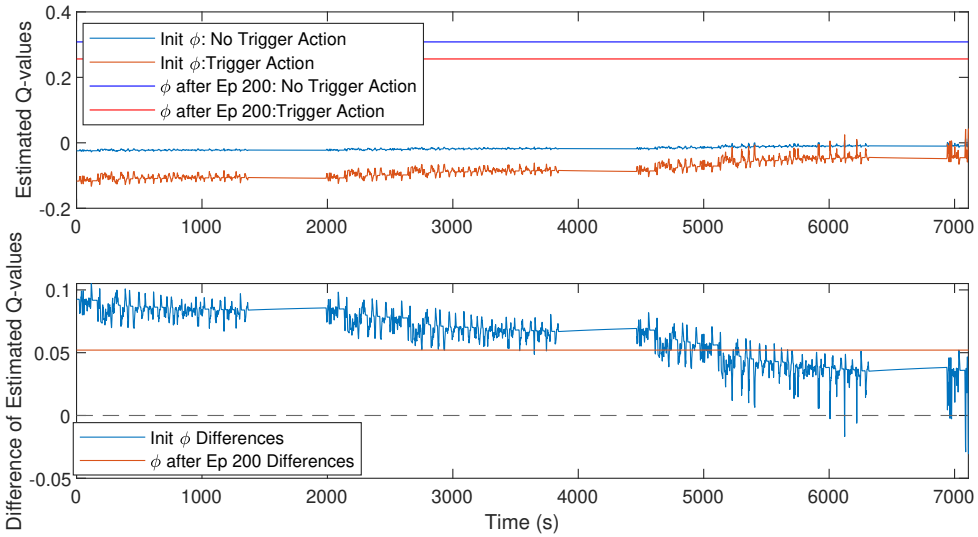

Figure 5.19: Q-values estimates of initial DQN parameters and after 200 episodes over typical episode states with $\rho = 0.01$ and $\alpha = 5\text{x}10^{-7}$.

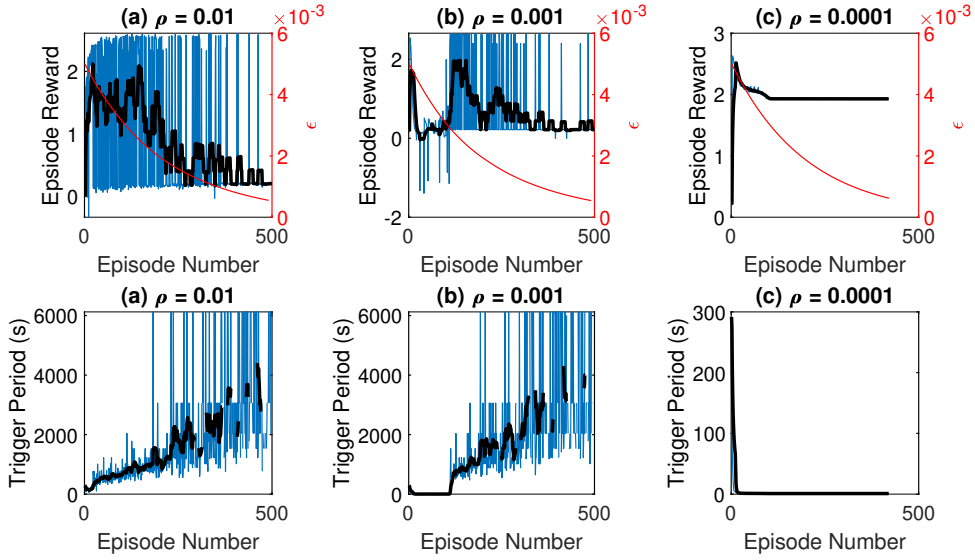

Figure 5.20: Episode reward and trigger period over training with range extension reward function and varying ρ with constant $\alpha = 5\text{x}10^{-7}$.

To motivate the RL agent to trigger more frequently, ρ was reduced from 0.01 to 0.001, then finally to 0.0001 as shown in Fig. 5.20. With $\rho = 0.001$ shown in Fig. 5.20(b) and (e), the agent began to attempt to trigger more frequently close to 1 s/trigger for the first 100 episodes before sharply changing course to almost 1000 s/trigger and finally converging to no triggers at all. Fig. 5.21 shows the Q-value metrics, where for the $\rho = 0.001$ test, the mean of the differences of the Q-values was around 0 during the low trigger period episodes before favoring not triggering again shown in Fig. 5.21(e). With ρ even lower to 0.0001, the mean of the differences dropped below 0 after the first 100 episodes and continued decreasing, triggering very frequently as shown in Fig. 5.20(f) and Fig. 5.21(f).

The results between the tests of $\rho = 0.001$ and $\rho = 0.0001$ in Figures 5.20 and 5.21 demonstrated significant improvement in learning for low values of ρ over the previous reward function. For the test with $\rho = 0.001$, assuming an episode length of 7,000 s and always triggering, the trigger reward would be +2.6 (max range extension) - 7,000 triggers

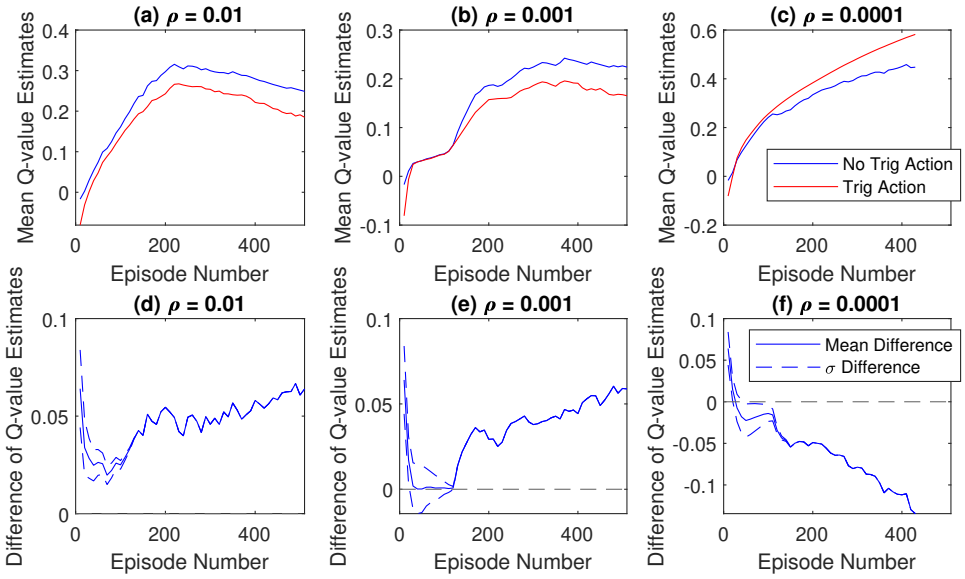

Figure 5.21: Mean and difference of Q-value estimates over training with range extension reward function and varying ρ with constant $\alpha = 5\text{x}10^{-7}$.

* -0.001 = -4.4 reward. Compared to not triggering throughout the episode with 0 trigger penalty, the reward would come out to +0.2 from the min range extension. If only these extremes are considered, not triggering at all would then be the better policy. With ρ = 0.0001 and always triggering, a reward of +2.6 - 7,000 triggers * -0.0001 = +1.9 would be received. Since this +1.9 is greater than the no trigger, no range extension reward of +0.2, the agent is now motivated to choose always triggering as the optimal policy. This was an improvement from the previous reward function because now the episode reward for always triggering must be higher than not triggering for the agent to learn higher Q-values for triggering actions. This feature of the reward function allows a greater range of ρ to be tested to balance between range extension and triggering.

However, a further issue remains since the agent was still learning the extremes of choosing either action for the entire epsiode. This is where the issue of delayed rewards comes in as shown by the standard deviation of the Q-value differences in Fig. 5.21 and 5.19. The Q-value estimates were insensitive to changes in observations across the

episodes resulting in all actions falling to trigger or not trigger throughout the episodes. To accurately learn Q-values of the observations early on in the episode for the far delayed rewards, much longer training with well tuned parameters would be required.

5.6 Reward Function Based on SOC Standard Deviation

To address the challenge of learning long delayed rewards, Eq. (2.22) was used as the reward function to rely on a parameter with faster dynamics that would correlate with the overall goal of driving range extension. The parameter chosen for this purpose was SOC standard deviation σ_{SOC}. The intermediate goal of cell balancing is to reduce the SOC variation across the cells which leads to the final range extension, and this SOC variation exhibits faster dynamics that can be controlled through time with much less time delay than the final range extension. This reward function was defined to penalize σ_{SOC} of the cells across the battery pack with the assumption that real-time SOC of each cell was available. To aid the agent in learning to minimize σ_{SOC}, σ_{SOC} was substituted for the minimum SOC in the RL states that the agent observed.

Additional analysis on previous constant trigger and eMPC results was done to study the relationships between MPC trigger frequency, SOC variation, and range extension. As mentioned previously, for this strategy to be effective for increasing EV range, SOC variation quantified by σ_{SOC} must correlate with EV range extension and trigger frequency. The overall relationship is expected to be as the trigger frequnecy increases, σ_{SOC} decreases, and as a result, EV range increases. Fig. 5.22 shows these relationships over a range of constant MPC trigger frequencies. Overall, the expected relationship was demonstrated with $\Sigma\sigma_{SOC}$ as the sum of σ_{SOC} over all time steps for each test decreasing and maximum distance achieved as trigger frequency increases. With constant triggering, this data indicates that $\Sigma\sigma_{SOC}$ values under about 70 could guarantee maximum range extension with a trigger frequency of 0.01 Hz or 100 s/trigger.

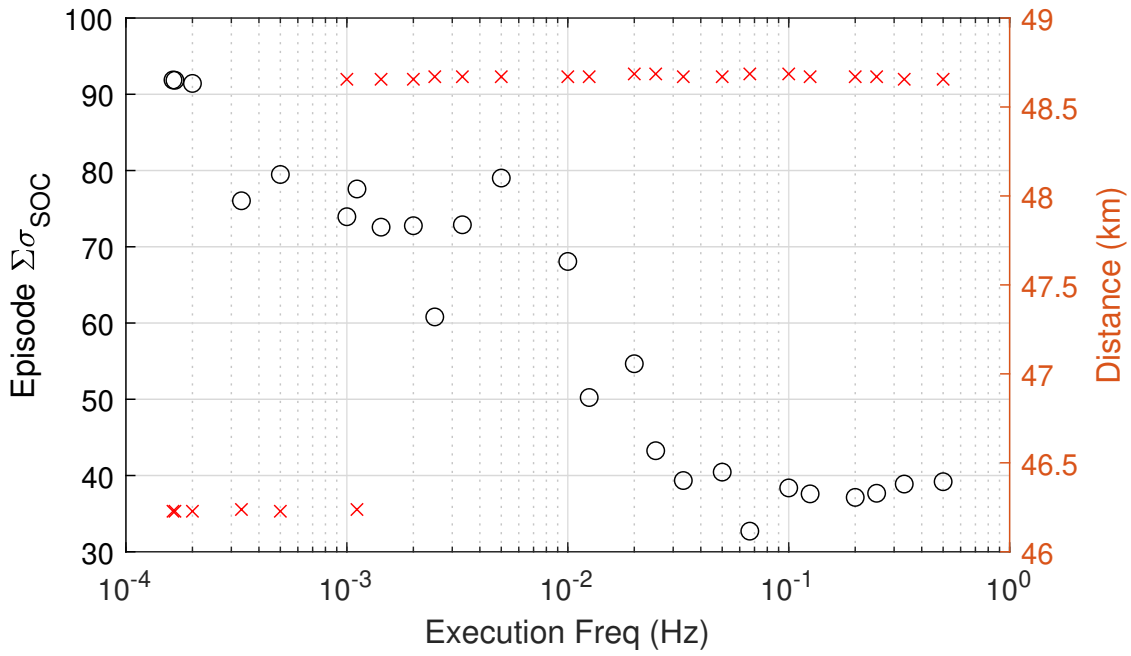

Figure 5.22: Sum of σ_{SOC} over each test and final EV range vs MPC trigger frequency with constant MPC trigger frequency for each test.

Additionally, similar results for the previous eMPC testing are shown in Fig. 5.23 to understand how these relationships are affected by variable trigger frequencies. With eMPC, the value of $\Sigma\sigma_{SOC}$ required to guarantee higher values of range extension was about 70 which aligns with the constant trigger results. However, the trigger frequency required for that level of $\Sigma\sigma_{SOC}$ was now increased to 0.027 Hz or 35 s/trigger. Also observed in the eMPC results was a peak in $\Sigma\sigma_{SOC}$ around 0.01 Hz as well as a peak in balancing effort at that same trigger frequency from the previous eMPC results shown in Fig. 5.7 indicating an instability of the eMPC controller around those frequencies.

First attempting to use the reward function Eq. (2.20) penalizing σ_{SOC} in the RLeMPC scheme, the relationship between trigger frequency and $\Sigma\sigma_{SOC}$ broke apart as shown in Fig. 5.24. The x-axis was scaled to highlight the narrow range that the RL agent explored inside of compare to the constant trigger frequency and eMPC results. Looking

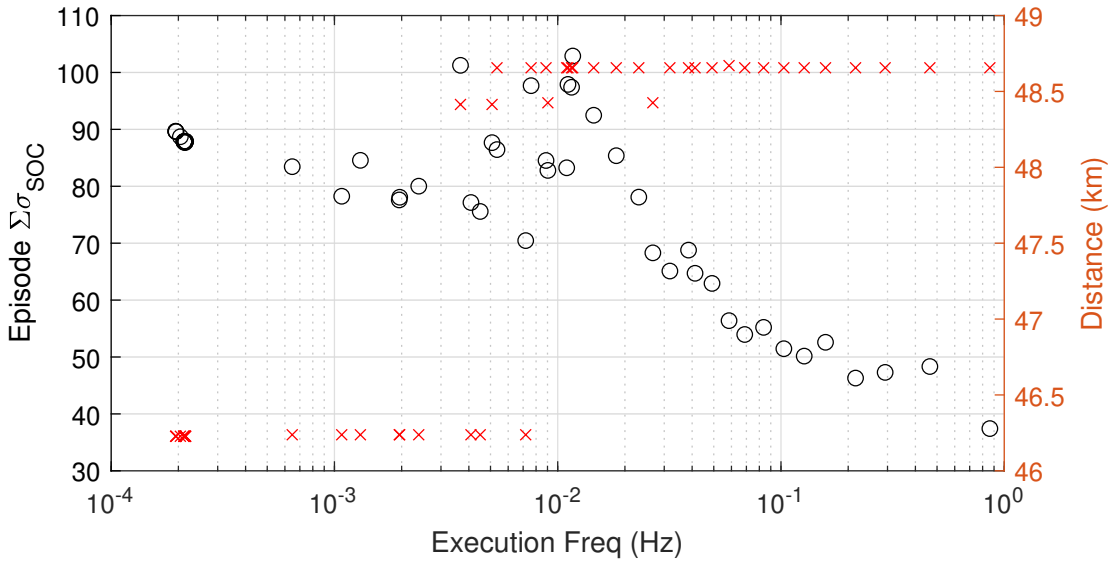

Figure 5.23: Sum of σ_{SOC} over each test and final EV range vs MPC trigger frequency with eMPC.

closer at the high trigger frequency tests from the RLeMPC training that resulted in high $\Sigma\sigma_{SOC}$, the reason for the breakdown in the relationship became clear because very infrequent triggering would occur for the majority of the test with small windows of very frequent triggering as shown in Fig. 5.25.

To encourage more evenly distributed triggering, an eligibility trace was introduced to penalize consecutive triggers. This trace is described with the introduction of the reward function Eq. (2.22) and is demonstrated in Fig. 5.26. For example at the first data tips at 4931 s, the trigger penalty is the value of the eligibility trace multiplied by ρ of 2. Since the eligibility trace was near zero before the trigger, it increased to 1 from the trigger and multiplied by ρ for a trigger penalty of about 2. At the data tips at 4434 s, the eligibility trace has not fully decayed yet, resulting in an a higer value of about 1.01 being multiplied by ρ for a penalty that is 1% larger than the previous penalty at 4391 s. Although that penalty increase was rather small, the extra penalty still came after 43 s since the previous trigger, and trigger periods smaller than that would receive even more

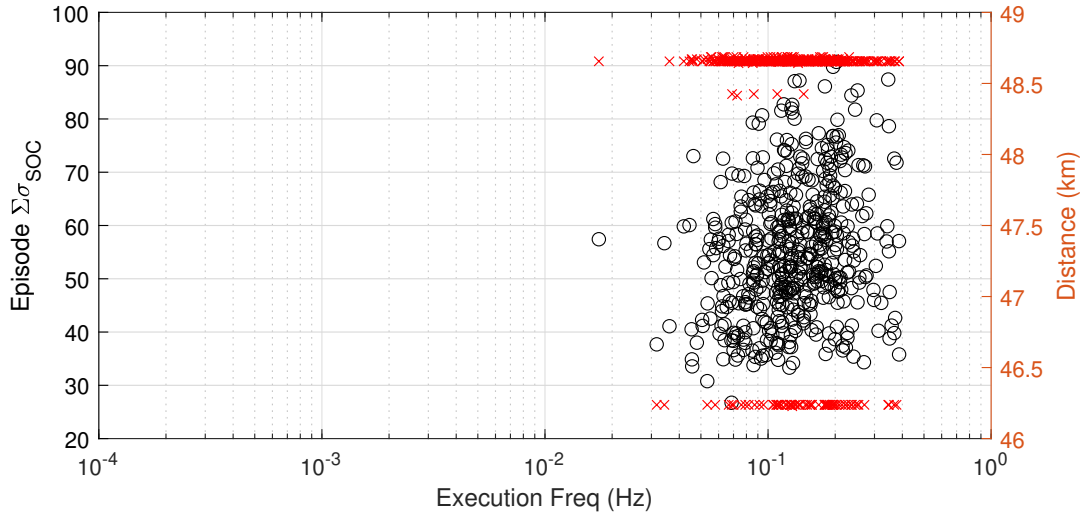

Figure 5.24: Sum of σ_{SOC} over each test and final EV range vs MPC trigger frequency with RLeMPC. X-axis scaled to highlight narrow range compared to constant frequency and eMPC results.

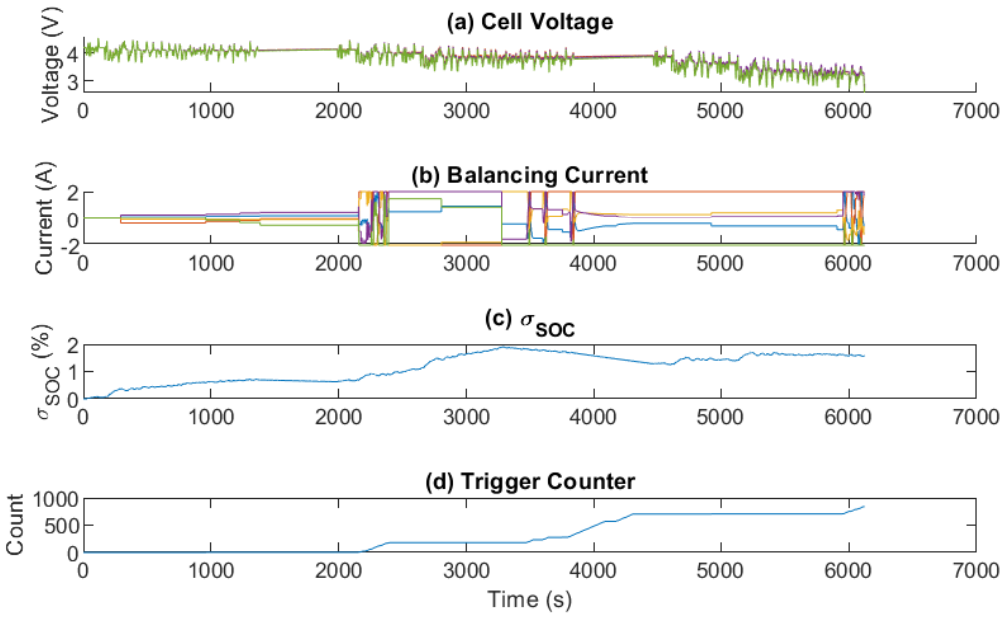

Figure 5.25: Final RLeMPC episode of initial training with σ_{SOC} reward function. (a) cell voltages, (b) balancing currents, (c) σ_{SOC}, and (d) trigger count over time.

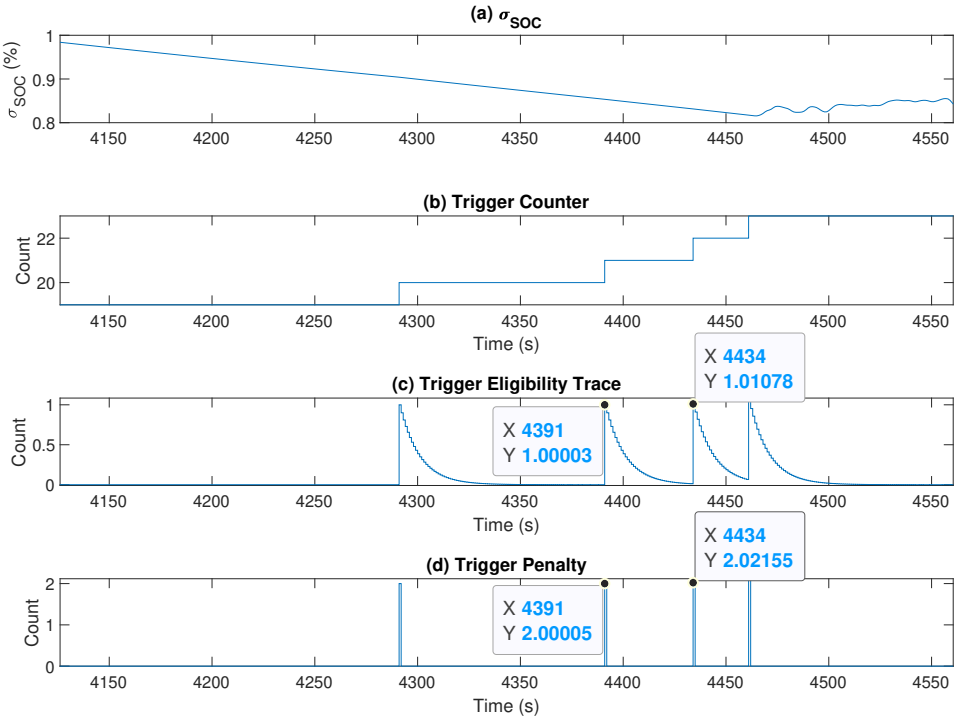

Figure 5.26: Example of how eligibility trace impacts trigger penalty with $\rho = 2$ and $\lambda = 0.9$.

penalty. The goal is to motivate a more consistent trigger period unless a decrease in σ_{SOC} would be expected to offset the penalty.

This eligibility trace was also included in the RL states that the agent observed substituting the pack current demand state. The substitution was done to prevent increasing the dimension of the RL state with the expectation that the current demand would be reflected sufficiently through the average cell voltage and SOC states.

The first results training the RLeMPC agent with the eligibility trace is shown in Fig. 5.27. The trigger frequencies now center around 0.01 Hz which was around where constant trigger MPC exhibited $\Sigma\sigma_{SOC} < 70$. Most notable was the improvement in correlation between trigger frequency and $\Sigma\sigma_{SOC}$. Although still containing much variability, the relationship of increasing frequency and decreasing $\Sigma\sigma_{SOC}$ was more

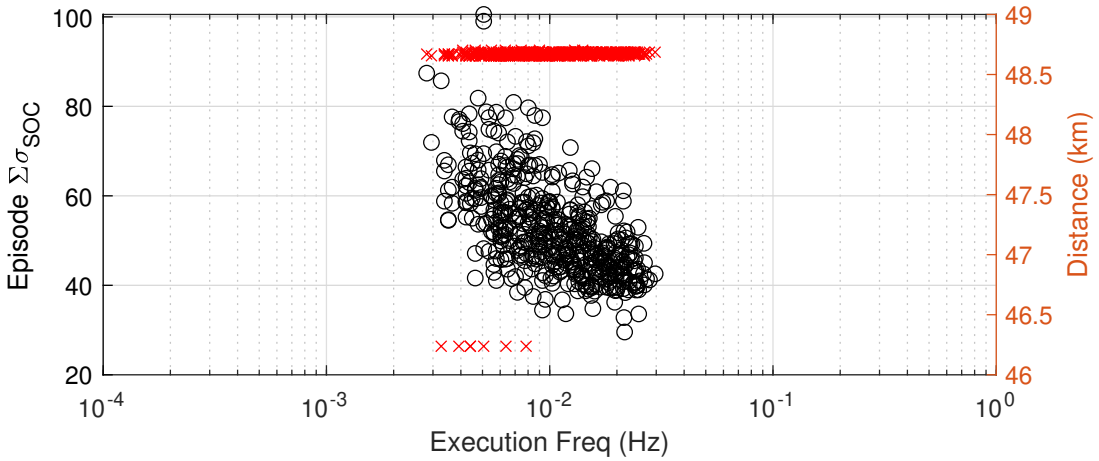

Figure 5.27: Episode $\Sigma\sigma_{SOC}$ and distance vs trigger frequency for RLeMPC with σ_{SOC} reward function and eligibility trace. $\rho = 2$ and $\lambda = 0.9$.

apparent than without the eligibility trace. Much of the variability could be attributed to the learning process since the agent must have experience with the eligibility trace to learn the penalty from consecutive triggers.

From the final learned ϕ with $\rho = 2$, Q-value sensitivity to the value of the eligibility trace and σ_{SOC} is presented in Fig. 5.28 where the z-axis is the difference between the Q-value estimates for the no trigger action and the trigger action with positive difference indicating that no trigger action has higher estimated Q-value. The agent clearly learned that a larger value of the eligibility trace will bring a larger penalty for the trigger action since the Q-value difference increased, favoring not triggering even more, as the eligibility trace value increased. However, from the large value of ρ, the agent learned a final policy of not triggering at all throughout the episode. For the particular observations in Fig. 5.28, no trigger actions had higher Q-value estimates throughout since the entire surface is positive. Also clear in Fig. 5.28 was that no sensitivity of the Q-values to σ_{SOC} was learned from the trigger penalty far outweighing the σ_{SOC} penalty. To motivate the agent to trigger more frequently to reduce σ_{SOC}, lower values for ρ were tested.

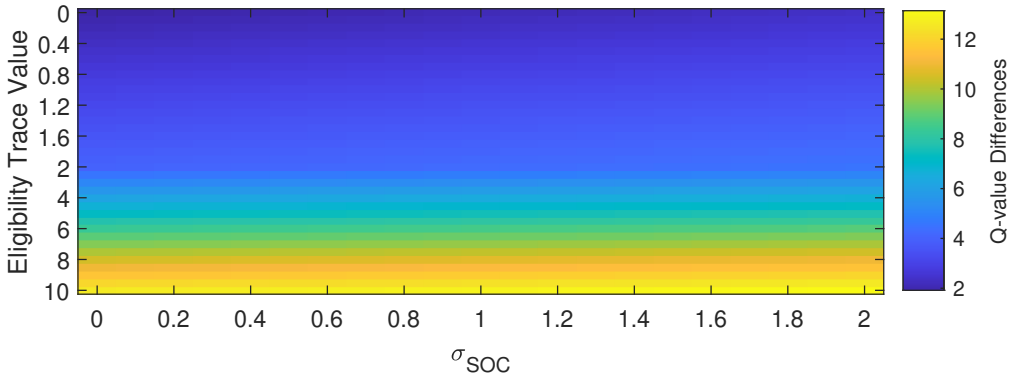

Figure 5.28: Q-value differences over the value of the eligibility trace and σ_{SOC} with "typical" mid-episode values for average SOC and voltage observations with $\rho = 2$ and $\lambda = 0.9$. Q-value difference = Q for no trigger action - Q for trigger action (positive means no trigger is greedy action).

Overall results for testing with decreased ρ including reward, trigger period, and $\Sigma\sigma_{SOC}$ are shown in Figures 5.29 - 5.31. The increase of reward shown in Fig. 5.29(a) and (b) with $\rho = 2$ and 0.2 respectively was primarily a result of decreasing trigger period which is clear from the average trigger period increasing and $\Sigma\sigma_{SOC}$ increasing shown in Figures 5.30 - 5.31(a) and (b). A similar trend happens across these figures in subplots (c) and (d) with $\rho = 0.02$ and 0.002 respectively, except that now, the reward decreased with increasing trigger period and increasing $\Sigma\sigma_{SOC}$. This occurred since ρ was not large enough to increase the total reward by only increasing the trigger period without improving σ_{SOC} as seen in Fig. 5.31(c) and (d).

To understand why $\Sigma\sigma_{SOC}$ was not improving, further analysis was done beginning with Fig. 5.32 to show the results of $\Sigma\sigma_{SOC}$ over trigger frequencies for the ρ values tested. As ρ decreased, episodes with higher trigger frequency were observed as expected, especially for the case of $\rho = 0.002$. However, also apparent, was that the correlation between $\Sigma\sigma_{SOC}$ and trigger frequency broke down again. With the training and exploration process, a decent amount of variability in the correlation was expected, but the result in Fig. 5.32(d) showed that the eligibility trace was not effective with enforcing the

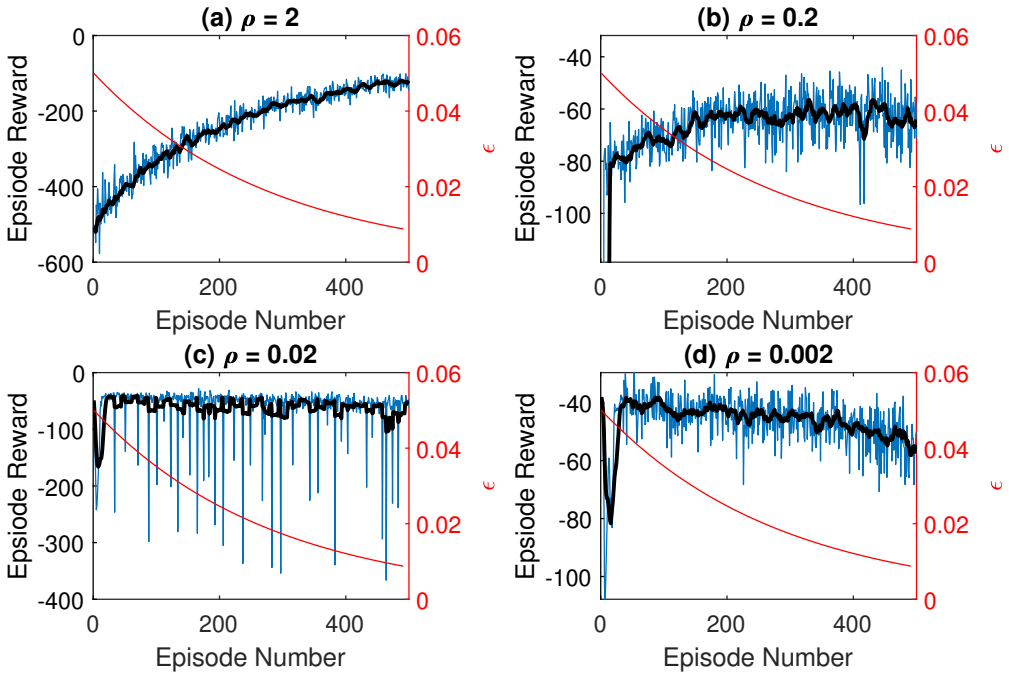

Figure 5.29: Episode reward over training with varying ρ and discount = 0.9.

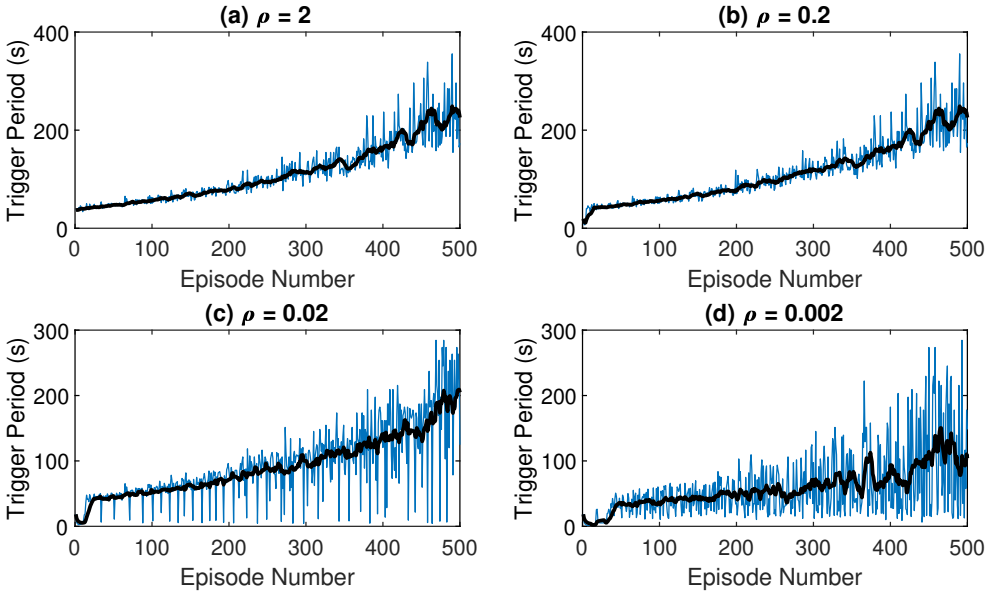

Figure 5.30: Average trigger period over training with varying ρ and discount = 0.9.

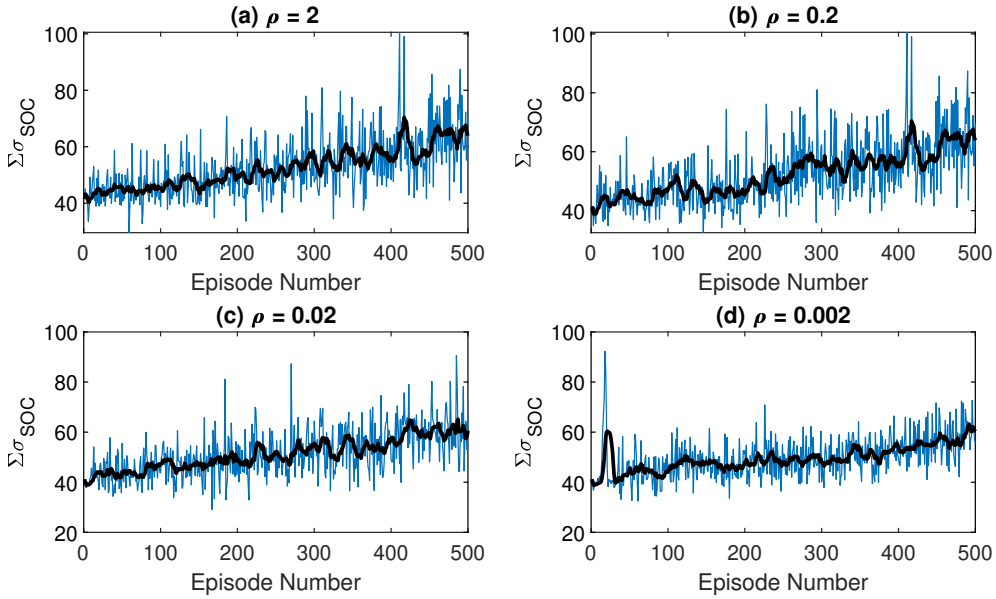

Figure 5.31: $\Sigma\sigma_{SOC}$ over training with varying ρ and discount = 0.9.

correlation. Including the eligibility trace into the reward function added the complexity of coordinating yet another calibration because as ρ decreases, λ must be increased as well to maintain the effectiveness of the eligibility trace to even out the distribution of triggers. To demonstrate that, further testing with decreased decay rate was done with ρ = 0.002, but before that, the Q-value sensitivity to the eligibility trace and σ_{SOC} was visualized to show the decreasing sensitivity to the eligibility trace.

Fig. 5.33 shows the sensitivity of the Q-values to the eligibility trace value and σ_{SOC} for a "typical" average SOC and voltage observation in the middle of the episode. Very apparent was that from ρ = 2 to ρ = 0.2, the gradient of the Q-value sensitivity dramatically rotated 90 degrees to be much more sensitive to σ_{SOC} than to the eligibility trace. With the decreased value of ρ, σ_{SOC} was had much more impact on the received reward every step. However, it was still unexpected to completely remove the sensitivity to the eligibility trace. With ρ of 0.02 and 0.002 in Fig. 5.33(c) and (d), the sensitivity to σ_{SOC} remained, but here some sensitivity to the eligibility trace was retained with no

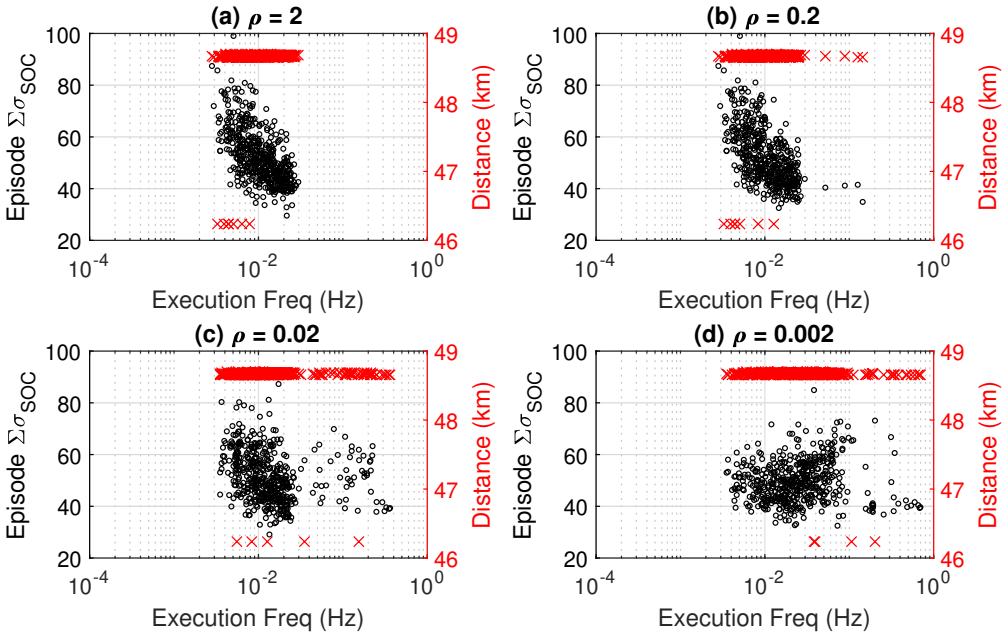

Figure 5.32: Episode $\Sigma\sigma_{SOC}$ and distance vs trigger frequency for RLeMPC with elignibility trace with varying ρ. discount = 0.9.

trigger action having a larger Q-value estimate for higher values of the eligibility trace at any given σ_{SOC}. The gradient with respect to the eligibility trace was still much less than the gradient with respect to σ_{SOC}.

To highlight the final policy with $\rho = 0.002$ and $\lambda = 0.9$, Fig. 5.34 shows transient data for a test using the final learned ϕ. No triggers were taken until over 2000 s into the episode when the trigger frequency increased drastically. The trigger frequency decreased during the 0 power demand pause before ramping up again during the next transient cycle and terminating at the minimum range extension distance. Even though the average trigger frequency for considering the entire test was around 0.22 Hz or 4.5 s/trigger, it was not enough triggering to decrease the SOC variation enough to overcome the minimum distance. The benefit of the eligibility trace was still apparent by limiting the maximum trigger frequency during the high trigger frequency portions of the episode even with a low value for ρ although very high trigger frequencies were still observed.

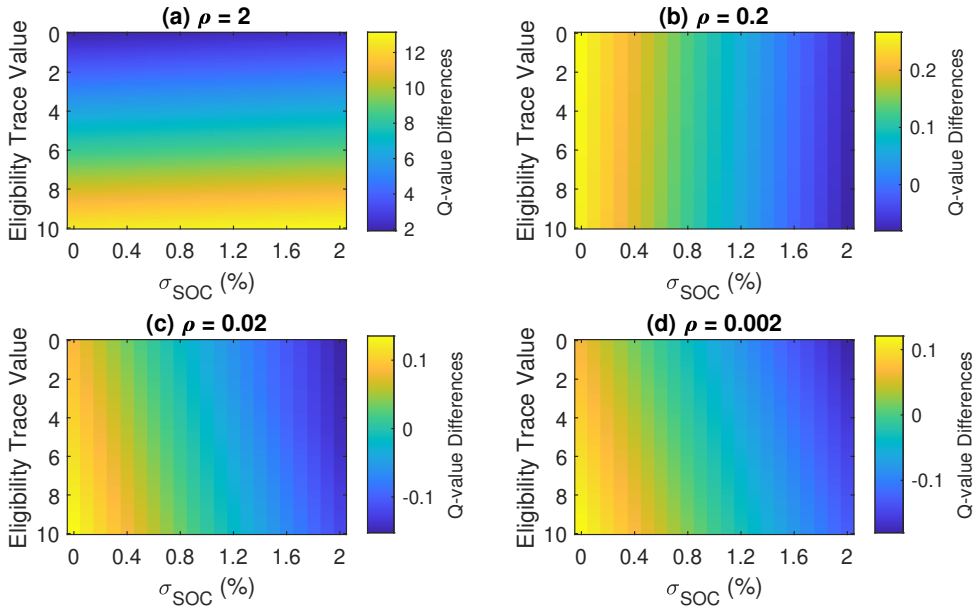

Figure 5.33: Episode $\Sigma\sigma_{SOC}$ and distance vs trigger frequency for RLeMPC with elignibility trace with varying ρ. discount = 0.9.

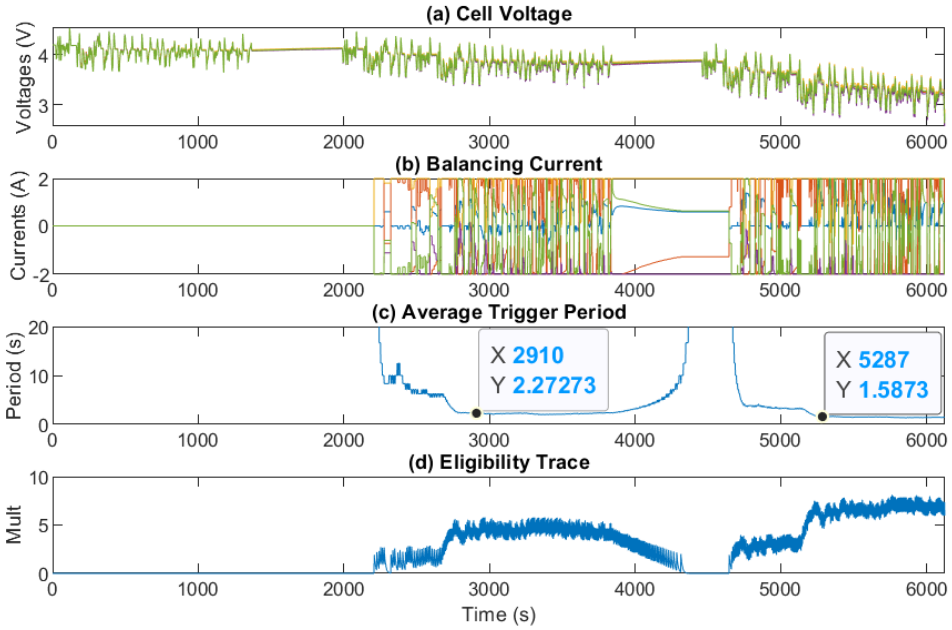

Figure 5.34: On-policy episode with final policy learned with ρ = 0.002 and eligibility trace decay = 0.9

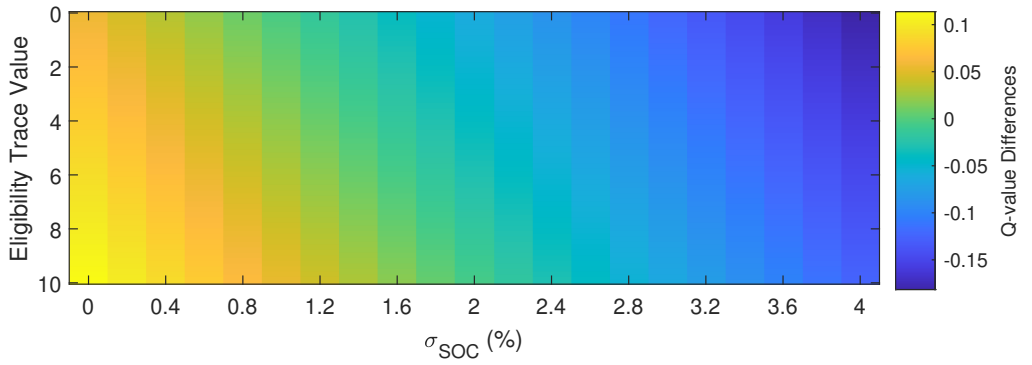

Figure 5.35: Episode $\Sigma\sigma_{SOC}$ and distance vs trigger frequency for RLeMPC with eligibility trace with varying ρ. discount = 0.9.

Increasing the eligibility trace decay rate from 0.9 to 0.95 was the final test that was executed to demonstrate how increased decay rate could improve the final policy. The episode reward, trigger period, and $\Sigma\sigma_{SOC}$ over the training evolved very similarly to the case with the decay rate of 0.9, and the Q-value estimate sensitivities from the final learned ϕ is shown in Fig. 5.35 being very similar to the ϕ learned previously in Fig. 5.33(d). The core differences become clear when comparing tests using the final learned ϕ on-policy from Fig. 5.34 to Fig. 5.36. During the high trigger frequency periods of the test, the trigger period increased from 2 to about 3.5 s at the first high frequency portion and 1.5 to 2.5 s at the second high frequency portion with < 0.5% increase in $\Sigma\sigma_{SOC}$.

Further tuning of the training parameters including the reward function weightings of ρ and λ would be required to demonstrate the full performance achievable with this reward function, but the improvements demonstrated over the previous reward functions should enable much better performance. For example, although the training failed to learn a policy better than a low constant trigger frequency, the final episode, shown in Fig. 5.37, showed that the agent was exploring such policies with very good performance.

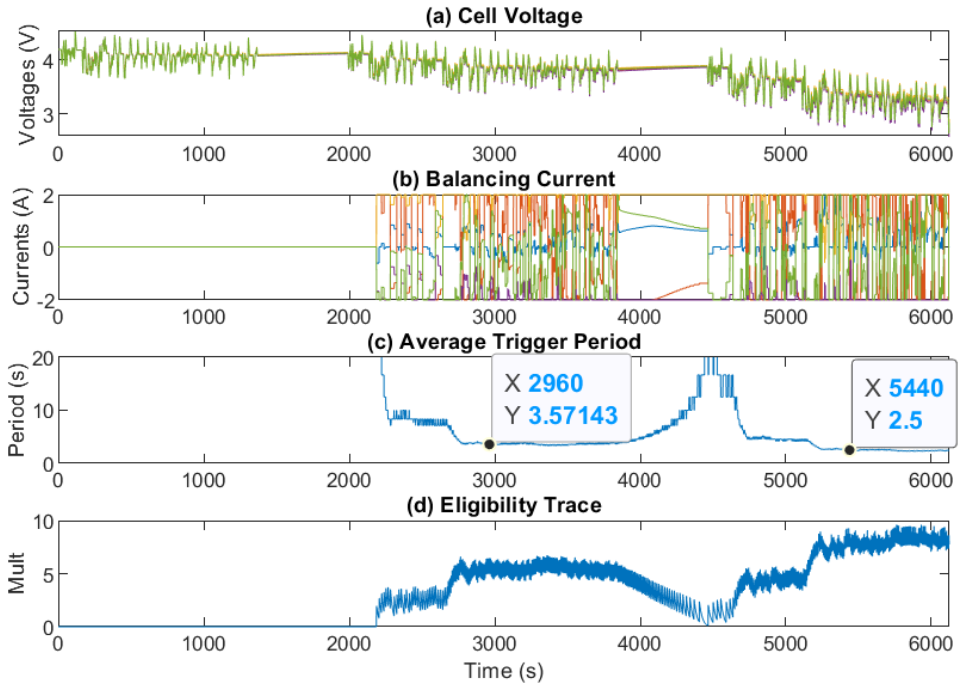

Figure 5.36: On-policy episode with final policy learned with $\rho = 0.002$ and $\lambda = 0.95$

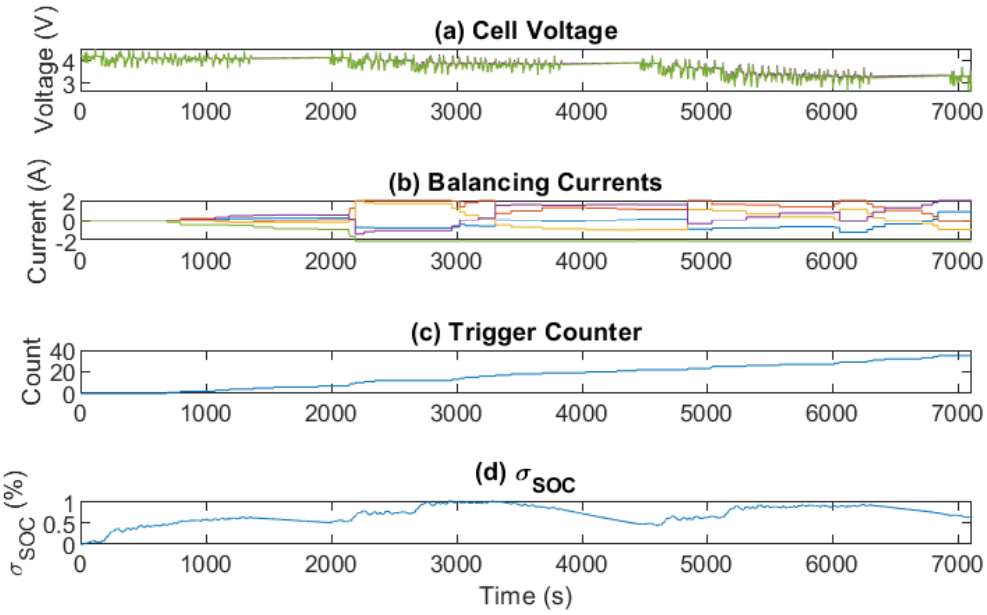

Figure 5.37: Final training episode with $\rho = 0.002$ and $\lambda = 0.95$.

CHAPTER SIX

CONCULSIONS AND FUTURE WORK

6.1 Conclusions

Three model predictive control (MPC) strategies, namely time-triggered MPC, event-triggered MPC (eMPC), and reinforcement learning-based MPC (RLeMPC), for active cell balancing, were formulated and tested in a simulation environment to reduce computational requirements relative to a baseline MPC controller while maintaining the EV range extension benefit. MPC with reduced constant period triggering and eMPC control methods demonstrated significant computational load reduction with an average trigger period increase to 1000 s and 187 s respectively compared to the 1 s trigger period of the baseline MPC. Only a negligible decrease of EV range extension of 0.03% from the maximum achievable range was the penalty for this significant decrease in throughput. From scaling of the current demand, the discharge voltage limit was met at very high current demands and significant energy remaining in the battery pack shaping the EV range extension results of the simulations.

For RLeMPC, three different reward functions were tested with key issues and improvements identified. Overall, the key challenges identified with this control strategy included training sensitivities to many tuning parameters including the learning rate and the exploration strategy, defining a metric to capture range extension, and learning from delayed rewards. A few of the training parameters such as the ε-greedy exploration settings and the learning rate were tested with to understand sensitivities and define rough values to explore issues from the reward functions, but optimizing these hyper-parameters was never sufficiently explored. The first reward function based on total distance driven was shown to be difficult to train since the range extension distance was only a small portion of the overall distance, and the reward function based on range extension distance

corrected this issue. However, a further more challenging issue of learning from very delayed rewards remained. To address that, the final reward function was based on σ_{SOC} to learn from less delayed rewards and included the addition of an eligibility trace to penalize the rate of triggering. This reward function demonstrated promising initial results from overcoming the previously described issues and showing capability to improve performance of the RLeMPC agent with a more refined calibration process.

6.2 Future Work

The primary future work stream would be to use a hyper-parameter tuning tool with the RLeMPC agent and σ_{SOC} based reward function to find the hyper-parameters that could maximize the capability of this strategy. Much improvement would be expected from further coordinated tuning of only ρ and λ, and even more could come from better values for the training hyper-parameters. Other future work items would include training and generalizing to various cell imbalance parameters since this work only focused on one set throughout, scaling cell currents to more realistic values for more continuous range extension results, and increasing the number of cells to study how well this method could be applied to larger packs.

APPENDIX A

MATLAB CODE FOR RL TRAINING

A.1 Code for Training RL Agent

```
clc; clear all; close all;
tic

DistBasedReward = 1;
RawDistBasedReward = 0;
SOCVarBasedReward = 1; %DistBasedReward = 1 for SOC reward
fixedMultTrigPenalty = 0;
trigPenaltyDecayRate = 0.95;
OldState = 0;
triggerCountMult = 0.002;
load_params; %start small with 5 cells
fixedPhiSel = 1;
phiIn = rand([nCell,4])/5+0.9;
load('ACB_Extension_Baseline_2Alim_CurrScale3.mat','phiIn');
fixedPhi = phiIn{10}; %param.phi

%%% LPVMPC specific cals
Wyy = zeros(2,1);
Wyy(1) = 0; % weights on SOC tracking
Wscale = 10;
Wyy(2) = 0.1/5*p/Wscale; % weights on voltage tracking
Wy = diag(repmat(Wyy,nCell,1));
Wu = 1e1*ones(nu,nu); %0e-1*ones(nu,nu);
Wd = 1e1*ones(nu,nu); %1e3*ones(nu,nu);
```

```
25 We = 1e2;
26
27 %%
28 rng(3564); %init weights
29
30 obsInfo = rlNumericSpec([4 1]); %CHANGED FOR nCELLS
31 actInfo = rlFiniteSetSpec(0:1);
32
33 env = rlSimulinkEnv(...
34 "acb_mpc_track_RLadded_HPC_DistanceBased",...
35 "acb_mpc_track_RLadded_HPC_DistanceBased/RL Trigger",...
36 obsInfo,actInfo);
37
38 net = [featureInputLayer(4,... %dependings on obs space
39           'Normalization','none','Name','myobs')
40       fullyConnectedLayer(4,'Name','value')
41       reluLayer()
42 %       fullyConnectedLayer(4)
43       fullyConnectedLayer(2)];
44
45 critic = ...
46 rlQValueRepresentation(net,obsInfo,actInfo,...
47 'Observation',{'myobs'});
48 critic.Options.LearnRate = 0.00001;
49 critic.Options.GradientThreshold = 1;
```

```
50
51 agent = rlDQNAgent(critic);
52
53 agent.AgentOptions.UseDoubleDQN=true;
54 agent.AgentOptions.TargetUpdateMethod="periodic";
55 agent.AgentOptions.TargetUpdateFrequency=500;
56 agent.AgentOptions.ExperienceBufferLength=150000;
57 agent.AgentOptions.DiscountFactor=0.95;
58 agent.AgentOptions.MiniBatchSize=256;
59 EpsDecay = 0.0000005;
60 EpsMin = 0.0002;
61 agent.AgentOptions.EpsilonGreedyExploration.EpsilonMin = ...
62 EpsMin;
63 agent.AgentOptions.EpsilonGreedyExploration.EpsilonDecay = ...
64 EpsDecay;
65 EpsOld = 0.05; %this has to change to reset epsilon!!!
66 agent.AgentOptions.EpsilonGreedyExploration.Epsilon = EpsOld;
67
68 fprintf('start')
69 %%
70 maxEp =500;
71 epInterval = 10;
72 optsTrain = rlTrainingOptions("MaxStepsPerEpisode",20000, ...
73     "ScoreAveragingWindowLength",3, ...
74     "MaxEpisodes",epInterval, ...
```

```
75     "StopTrainingCriteria", ...
76     "AverageSteps","StopTrainingValue",20000,"Plots","none");
77
78 NN{1} = getLearnableParameters(critic);
79 %%
80 info{1} = train(agent,env,optsTrain);
81
82 %%
83 NN{2} = getLearnableParameters(agent);
84
85 EpsNew(1) = EpsOld;
86 triggerCount = [];
87 TfinalEp = [];
88 DfinalEp = [];
89 sumSOCstd = [];
90 avgSOCstd = [];
91 phiCheck = zeros(5,4,1);
92 tint(1) = 0;
93
94 m=1;
95 i=1;
96 for k = 1:epInterval
97     triggerCount = [triggerCount, ...
98     info{1,i}.SimulationInfo(k,1).triggerCounter....
99     signals.values(end)];
```

```
100     TfinalEp = [TfinalEp, ...
101     info{1,i}.SimulationInfo(k,1).triggerCounter.time(end)];
102     sumSOCstd = [sumSOCstd, ...
103     sum(std(info{1,i}.SimulationInfo(k,1).soc.signals....
104     values'))];
105     avgSOCstd = [avgSOCstd,...
106     mean(std(info{1,i}.SimulationInfo(k,1).soc.signals....
107     values'))];
108     phiCheck(:,:,m) = ...
109     info{1,i}.SimulationInfo(k,1).phi.signals.values(:,:,1);
110     DfinalEp = [DfinalEp, d_veh(TfinalEp(end)+1)/1000]; %km
111     m=m+1;
112 end
113
114 for i =2:maxEp/epInterval
115     fprintf("percent done\n")
116     disp(i/(maxEp/epInterval)*100)
117     totSteps = info{i-1}.TotalAgentSteps(end);
118     for j = 1:totSteps
119         EpsNew(i) = EpsOld * (1-EpsDecay);
120         if EpsNew(i)<EpsMin
121             EpsNew(i) = EpsMin;
122         end
123         EpsOld = EpsNew(i);
124     end
```

```
125     EpsNew(i);
126     agent.AgentOptions.EpsilonGreedyExploration.Epsilon =...
127     EpsNew(i);
128
129     info{i} = train(agent,env,optsTrain);
130     NN{i+1} = getLearnableParameters(agent);
131     m=1;
132     for k = 1:epInterval
133         triggerCount = [triggerCount, ...
134         info{1,i}.SimulationInfo(k,1).triggerCounter....
135         signals.values(end)];
136         TfinalEp = [TfinalEp, ...
137         info{1,i}.SimulationInfo(k,1).triggerCounter.time(end)];
138         sumSOCstd = [sumSOCstd, ...
139         sum(std(info{1,i}.SimulationInfo(k,1).soc.signals....
140         values'))];
141         avgSOCstd = [avgSOCstd, ...
142         mean(std(info{1,i}.SimulationInfo(k,1).soc.signals....
143         values'))];
144         phiCheck(:,:,m) = ...
145         info{1,i}.SimulationInfo(k,1).phi.signals.values(:,:,1);
146         DfinalEp = [DfinalEp, d_veh(TfinalEp(end)+1)/1000]; %km
147         m=m+1;
148     end
149
```

```
150     if i<maxEp/epInterval
151         info{i} = rmfield(info{i},'SimulationInfo')
152     end
153 
154     tint(i) = toc/60
155 end
156 
157 %% Test Processing
158 %neuron plotting
159 m = 1;
160 n = 1;
161 for i = 1:2
162     if i ==2
163         n = 3;
164     end
165     for j =1:4
166         for k = 1:size(NN{1,1}{n,1},1)
167             neuron(m,1) = NN{1,1}{n,1}(k,j);
168             m = m+1;
169         end
170     end
171 end
172 
173 for h = 2:length(NN)
174     m = 1;
```

```
175     n = 1;
176     for i = 1:2
177         if i ==2
178             n = 3;
179         end
180         for j =1:4
181             for k = 1:size(NN{1,h}.Critic{n,1},1)
182                 neuron(m,h) = NN{1,h}.Critic{n,1}(k,j);
183                 m = m+1;
184             end
185         end
186     end
187 end
188 
189 fprintf("neuron\n")
190 disp(neuron)
191 fprintf("EpsNew\n")
192 disp(EpsNew)
193 
194 %% reward plotting
195 reward = [];
196 
197 m = 1;
198 for i = 1:length(info)
199     reward = [reward; info{1,i}.EpisodeReward];
```

```
200 end
```

A.2 Code for Definition of Active Cell Balancing

```
1 %%%%%%% cell params %%%%%%%
2
3 % parameters from He et al "State-of-charge estimation of
4 % the lithium-ion battery using an adaptive extended Kalman
5 % filter based on an improved Thevenin model," IEEE
6 % Transactions on Vehiclar Technology,
7 % Vol. 60, No. 4, May 2011.
8
9 nCell = 5;
10 Ca = 62;%Ah Original value: 100; modified to 62 Ah for bolt ev
11 eta_col = 1;
12 sec_per_hour = 3600;
13 Ca = Ca*sec_per_hour; % As
14
15 max_v = 4.2;
16 min_v = 2.6;
17 cell_scaling = 69/max_v;
18 table_x_ro = 0.1:0.1:0.9;
19 table_y_ro = [0.0226, 0.0225, 0.0225, 0.0225,...
20 0.0223, 0.0222, 0.0221, 0.0222, 0.0222]+0.001;
21 table_y_ro = table_y_ro/cell_scaling;
22 table_x_p = 0.1:0.1:1;
23 table_y_p = [0.1, 0.1, 0.0987, 0.099, 0.0994, 0.097,...
```

```
0.0941, 0.0996, 0.0999, 0.1025; 10000, 10000, 9909,...
10010, 10009, 9903,9921, 10210, 9666, 9987];
table_y_p(1,:) = table_y_p(1,:)/cell_scaling;
table_y_p(2,:) = table_y_p(2,:)*cell_scaling;

min_OCV = 40;
k1 = -13*(69-min_OCV)/12/cell_scaling;
k2 = 25*(69-min_OCV)/12/cell_scaling;
k3 = min_OCV/cell_scaling;

rng(3556);
phi = rand([nCell,4])/5+0.9;

param.Ca = Ca;
param.eta_col = eta_col;
param.table_x_ro = table_x_ro;
param.table_y_ro = table_y_ro;
param.table_x_p = table_x_p;
param.table_y_p = table_y_p;
param.phi = phi;
param.k1 = k1;
param.k2 = k2;
param.k3 = k3;
param.cell_scaling = cell_scaling;
param.nCell = nCell;
```

```
49
50 clear Ca eta_col sec_per_hour cell_scaling
51 clear table_x_ro table_y_ro table_y_ro table_x_p...
52 table_y_p phi k1 k2 k3 min_OCV
53
54 %%% simulation setting and input currents
55 load('ws_result.mat')
56 nCycle = 3*4;
57 Tfinal = dataOut(end,1)*nCycle; %2474; (Changed from og 1400)
58 Ts = 1;
59 Ts_log = 1;
60 % eta = 0.7;
61 power_lim = 25;
62 SoC0 = 1;
63 p_veh(:,2) = min(power_lim,max(-power_lim,p_veh(:,2)));
64 constant_c = 0;
65 %constant_c = 10;
66 if constant_c > 0
67     current_scale = 5;
68     % upper bound on balancing current (Amp)
69     uu = 1; %0.5; % upper bound on balancing current (Amp)
70     % lower balancing current (Amp)
71     ul = -uu;
72     % upper bound on change rate of balancing current (Amp)
73     duu = uu; %0.5;
```

```
74      % lower bound on change rate of balancing current (Amp)
75      dul = -duu;
76  else
77      current_scale = 5;
78      % upper bound on balancing current (Amp)
79      uu = 2;
80      % lower balancing current (Amp)
81      ul = -uu;
82      % upper bound on change rate of balancing current (Amp)
83      duu = 10;
84      % lower bound on change rate of balancing current (Amp)
85      dul = -duu;
86  end
87
88  %%% MPC setting
89  x0 = repmat([SoC0;0],nCell,1);
90  x0mpc = repmat([SoC0;0],nCell,1);
91  u0 = zeros(nCell,1);
92  nx = 2*nCell;
93  nu = nCell;
94  ny = 2*nCell;
95  p = 5;
96
97  %%% inpput constraints
98  % upper bound on balancing current (Amp)
```

```
% uu = 10; %0.5;
% lower balancing current (Amp)
% ul = -uu;
% upper bound on change rate of balancing current (Amp)
% duu = 10; %0.5;
% lower bound on change rate of balancing current (Amp)
% dul = -duu;

yl = repmat(repmat([-1; min_v],nCell,1),p,1);
yu = repmat(10*ones(ny,1),p,1);
uu = repmat(uu*ones(nu,1),p,1);
ul = repmat(ul*ones(nu,1),p,1);
duu = repmat(duu*Ts*ones(nu,1),p,1);
dul = repmat(dul*Ts*ones(nu,1),p,1);

ur = zeros(nu*p,1);
```

A.3 Code to Simulate Agent On-Policy from Learned DQN Parameters

```

newCritic = critic;
fixedPhiSel = 1;
simOpts = rlSimulationOptions("MaxSteps",10000);

% agentOptions = agent.AgentOptions;
% agentOptions.EpsilonGreedyExploration.EpsilonMin = 0.0000001;
% agentOptions.EpsilonGreedyExploration.Epsilon = 0.8;
```

```
% agent.AgentOptions = agentOptions;

for k = 1:1
    phiIn{k} = rand([nCell,4])/5+0.9;
    fixedPhi = param.phi; %phiIn{k};
    for i = length(NN):length(NN) %1:length(NN)
        if i ==1
            newModParam = setLearnableParameters...
            (newCritic,NN{i});
        else
            newModParam = ...
            setLearnableParameters(newCritic,NN{1,i}.Critic);
        end
        setCritic(agent,newModParam);
        experience{i,k} = sim(env,agent,simOpts);
        oneSecT{i,k} = experience{i,k}.SimulationInfo.phi.time;
        phi{i,k} = ...
        experience{i,k}.SimulationInfo.phi.signals.values;
        expTFinal(i,k) = experience{i,k}.SimulationInfo....
        tout(end);
        triggerout(i,k) = experience{i,k}.SimulationInfo. ...
        triggerCounter.signals.values(end);
        veh_idx(i,k) = find(expTFinal(i,k) == v_veh(:,1));
        distout(i,k) = d_veh(veh_idx(i))/1000

```

```
34          i/length(NN)
35      end
36  end
```

REFERENCES

[1] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: An engineering perspective," *The International Journal of Advanced Manufacturing Technology*, vol. 117, no. 5, pp. 1327–1349, 2021.

[2] P. Tøndel, T. A. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit mpc solutions," *Automatica*, vol. 39, no. 3, pp. 489–497, 2003.

[3] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010.

[4] H. Li and Y. Shi, "Event-triggered robust model predictive control of continuous-time nonlinear systems," *Automatica*, vol. 50, no. 5, pp. 1507–1513, 2014.

[5] L. Yu, Y. Xia, and Z. Sun, "Robust event-triggered model predictive control for constrained linear continuous system," *International Journal Of Robust And Nonlinear Control*, vol. 29.

[6] R. Badawi and J. Chen, "Performance evaluation of event-triggered model predictive control for boost converter," in *2022 IEEE Vehicle Power and Propulsion Conference*, (Merced, CA), November 1–4, 2022.

[7] R. Badawi and J. Chen, "Enhancing enumeration-based model predictive control for dc-dc boost converter with event-triggered control," in *2022 European Control Conference*, (London, UK), July 12–15, 2022.

[8] Z. Zhou, C. Rother, and J. Chen, "Event-triggered model predictive control for autonomous vehicle path tracking: Validation using CARLA simulator," *IEEE Transactions on Intelligent Vehicles*. Accepted for publication April 2023.

[9] J. Chen, X. Meng, and Z. Li, "Reinforcement learning-based event-triggered model predictive control for autonomous vehicle path following," in *American Control Conf.*, (Atlanta, GA), June 8–10, 2022.

[10] F. Dang, D. Chen, J. Chen, and Z. Li, "Event-triggered model predictive control with deep reinforcement learning," *IEEE Transactions on Intelligent Vehicles*. Accepted for Publication, October 2023.

[11] D. Baumann, J.-J. Zhu, G. Martius, and S. Trimpe, "Deep reinforcement learning for event-triggered control," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 943–950, 2018.

[12] A. S. Leong, A. Ramaswamy, D. E. Quevedo, H. Karl, and L. Shi, "Deep reinforcement learning for wireless sensor scheduling in cyber–physical systems," *Automatica*, vol. 113, p. 108759, 2020.

[13] J. Chen and Z. Zhou, "Battery cell imbalance and electric vehicles range: Correlation and NMPC-based balancing control," in *2023 IEEE International Conference on Electro Information Technology*, (Romeoville, IL), May 18–20, 2023.

[14] M. Dubarry, N. Vuillaume, and B. Y. Liaw, "Origins and accommodation of cell variations in li-ion battery pack modeling," *International Journal of Energy Research*, vol. 34, no. 2, pp. 216–231, 2010.

[15] J. Chen, Z. Zhou, Z. Zhou, X. Wang, and B. Liaw, "Impact of battery cell imbalance on electric vehicle range," *Green Energy and Intelligent Transportation*, vol. 1, pp. 1–8, December 2022.

[16] Z. B. Omariba, L. Zhang, and D. Sun, "Review of battery cell balancing methodologies for optimizing battery pack performance in electric vehicles," *IEEE Access*, vol. 7, pp. 129335–129352, 2019.

[17] M. Einhorn, W. Roessler, and J. Fleig, "Improved performance of serially connected li-ion batteries with active cell balancing in electric vehicles," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 6, pp. 2448–2457, 2011.

[18] F. S. J. Hoekstra, H. J. Bergveld, and M. Donkers, "Range maximisation of electric vehicles through active cell balancing using reachability analysis," in *2019 American Control Conference (ACC)*, pp. 1567–1572, IEEE, 2019.

[19] F. S. Hoekstra, L. W. Ribelles, H. J. Bergveld, and M. Donkers, "Real-time range maximisation of electric vehicles through active cell balancing using model-predictive control," in *2020 American Control Conference*, (Denver, CO), pp. 2219–2224, July 1–3, 2020.

[20] M. Preindl, "A battery balancing auxiliary power module with predictive control for electrified transportation," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 8, pp. 6552–6559, 2017.

[21] J. Chen, A. Behal, and C. Li, "Active battery cell balancing by real time model predictive control for extending electric vehicle driving range," *IEEE Transactions on Automation Science and Engineering*. Accepted June 2023.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*. MIT Press, 2018.

[23] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[25] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[26] J. Chen and Z. Yi, "Comparison of event-triggered model predictive control for autonomous vehicle path tracking," in *IEEE Conf. Control Technology and Applications*, (San Diego, CA), August 8–11, 2021.

[27] Z. Pei, X. Zhao, H. Yuan, Z. Peng, and L. Wu, "An equivalent circuit model for lithium battery of electric vehicle considering self-healing characteristic," *Journal of Control Science and Engineering*, vol. 2018, 2018.

[28] J. Wehbe and N. Karami, "Battery equivalent circuits and brief summary of components value determination of lithium ion: A review," in *2015 Third International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, (Beirut, Lebanon), pp. 45–49, 2015.

[29] B. Wang, J. Huang, C. Wen, J. Rodriguez, C. Garcia, H. B. Gooi, and Z. Zeng, "Event-triggered model predictive control for power converters," *IEEE transactions on industrial electronics*, vol. 68, no. 1, pp. 715–720, 2020.

[30] L. Li, P. Shi, R. K. Agarwal, C. K. Ahn, and W. Xing, "Event-triggered model predictive control for multiagent systems with communication constraints," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 5, pp. 3304–3316, 2019.

[31] D. Baumann, J.-J. Zhu, G. Martius, and S. Trimpe, "Deep reinforcement learning for event-triggered control," in *2018 IEEE Conference on Decision and Control (CDC)*, (Miami, FL, USA), pp. 943–950, 2018.

[32] T. Duraisamy and D. Kaliyaperumal, "Active cell balancing for electric vehicle battery management system," *International Journal of Power Electronics and Drive Systems*, vol. 11, no. 2, p. 571, 2020.

[33] Y.-S. Lee and M.-W. Cheng, "Intelligent control battery equalization for series connected lithium-ion battery strings," *IEEE Transactions on Industrial electronics*, vol. 52, no. 5, pp. 1297–1307, 2005.

[34] Q. Ouyang, J. Chen, J. Zheng, and H. Fang, "Optimal cell-to-cell balancing topology design for serially connected lithium-ion battery packs," *IEEE transactions on sustainable energy*, vol. 9, no. 1, pp. 350–360, 2017.

[35] M. Preindl, "A battery balancing auxiliary power module with predictive control for electrified transportation," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 8, pp. 6552–6559, 2017.

[36] J. Liu, Y. Chen, and H. K. Fathy, "Nonlinear model-predictive optimal control of an active cell-to-cell lithium-ion battery pack balancing circuit," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14483–14488, 2017.

[37] J. Chen, A. Behal, and C. Li, "Active cell balancing by model predictive control for real time range extension," in *2021 IEEE Conference on Decision and Control*, (Austin, TX, USA), December 13–15, 2021.

[38] S. Sepasi, R. Ghorbani, and B. Y. Liaw, "Improved extended kalman filter for state of charge estimation of battery pack," *Journal of Power Sources*, vol. 255, pp. 368–376, 2014.

[39] Q. Ouyang, W. Han, C. Zou, G. Xu, and Z. Wang, "Cell balancing control for lithium-ion battery packs: A hierarchical optimal approach," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5065–5075, 2019.

[40] The MathWorks Inc., "Deep Q-Network (DQN) Agents." Accessed: 2024-01-29.