EVERLASTING

Electric Vehicle Enhanced Range, Lifetime And Safety Through INGenious battery management

D7.5 – Validation report of self-learning SoH algorithm on lithium-ion battery module

February 2021

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 713771

Public

PROJECT SHEET	
Project Acronym	**EVERLASTING**
Project Full Title	Electric Vehicle Enhanced Range, Lifetime And Safety Through INGenious battery management
Grant Agreement	**713771**
Call Identifier	H2020-GV8-2015
Topic	GV-8-2015: Electric vehicles' enhanced performance and integration into the transport system and the grid
Type of Action	Research and Innovation action
Project Duration	54 months (01/09/2016 – 28/02/2021)
Coordinator	VLAAMSE INSTELLING VOOR TECHNOLOGISCH ONDERZOEK NV (BE) - *VITO*
Consortium Partners	COMMISSARIAT A L ENERGIE ATOMIQUE ET AUX ENERGIES ALTERNATIVES (FR) - *CEA* SIEMENS INDUSTRY SOFTWARE SAS (FR) - *Siemens PLM* TECHNISCHE UNIVERSITAET MUENCHEN (DE) - *TUM* TUV SUD BATTERY TESTING GMBH (DE) - *TUV SUD* ALGOLION LTD (IL) - *ALGOLION LTD* RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN (DE) - *RWTH AACHEN* LION SMART GMBH (DE) - *LION SMART* TECHNISCHE UNIVERSITEIT EINDHOVEN (NL) - *TU/E* VOLTIA AS (SK) - *VOLTIA* VDL ENABLING TRANSPORT SOLUTIONS (NL) – *VDL ETS*
Website	www.everlasting-project.eu

DELIVERABLE SHEET	
Title	**D7.5 – Validation report of self-learning SoH algorithm on lithium-ion battery module**
Related WP	WP7 (Increased Reliability)
Lead Beneficiary	VITO
Author(s)	Chris Hermans (VITO) Fred Spiessens (VITO) Carlo Manna (VITO) Klaas De Craemer (VITO)
Reviewer(s)	Alexander Blömeke (RWTH) Sebastian Ludwig (TUM)
Type	Report
Dissemination level	PUBLIC
Due Date	M54
Submission date	February 26, 2021
Status and Version	Final, V1.0

REVISION HISTORY			
Version	**Date**	**Author/Reviewer**	**Notes**
V0.1	24/12/2020	Chris Hermans (VITO) Researcher	First draft
V0.2	23/02/2021	Fred Spiessens (VITO) Researcher	Update V0.1
V0.3	26/02/2021	Carlo Manna (VITO) Researcher	Update V0.2
V0.4	08/03/2021	Klaas De Craemer (VITO) Researcher	Update V0.3
V0.5	09/03/2021	Carlo Mol (VITO)	Update V0.4
V0.6	11/03/2021	Carlo Manna (VITO) Researcher	Update V0.5
V0.7	11/03/2021	Alexander Blömeke (RWTH) WP Leader	Peer review
V0.8	12/03/2021	Sebastian Ludwig (TUM) WP Leader	Peer review
V1.0	31/03/2021	Carlo Mol (VITO) Coordinator	Submission to the EC

DISCLAIMER

The opinion stated in this report reflects the opinion of the authors and not the opinion of the European Commission.

All EVERLASTING consortium members are committed to publish accurate information and take the greatest care to do so. However, the EVERLASTING consortium members cannot accept liability for any inaccuracies or omissions nor do they accept liability for any direct, indirect, special, consequential or other losses or damages of any kind arising out of the use of this information.

ACKNOWLEDGEMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 713771

EXECUTIVE SUMMARY

Determining the State of Charge/State of Health (SOC/SOH) of a battery is a research topic that has received considerable attention. This is mainly attributed to the fact that indirect techniques are preferred over direct techniques (such as the prevalent Coulomb counting), the latter suffering from severe divergence issues due to its integrating character. Indirect techniques, however, rely on a model to calculate the hidden SOC from direct measurements such as current and voltage. Creating effective and practical reverse modelling techniques, however, is not a trivial endeavour.

This is because of:

(1) The fact that the dynamics of the battery can be complex and can be captured only partially by an equivalent circuit model. When using a more complex model, practicality is sacrificed for performance.

(2) The battery dynamics (and thus, the model parameters) are changing over time, and estimating model parameters and a hidden (correlated) state simultaneously can result in unstable behaviour.

Driven by recent developments in machine learning, model-free techniques such as Artificial Neural Networks (ANN) and Gaussian mixture models have been used to mitigate the modelling challenge. Here, however, most physical insight into the problem is omitted, resulting in a reduced data-efficiency and non-intuitive solution. A new step forward in recent literature is the use of structured neural networks (SNN), where a simple equivalent circuit model (ECM) is used, the parameters of which, however, are resulting from a nonlinear model-free approach.

In this work, we have expanded onto this two-fold:

(1) We have introduced the notion of Structured Recurrent Neural Networks (SRNN), in which we leverage the idea of a structured neural network (which like ANNs, has mainly been used as a direct technique) into the realm of indirect SOC/SOH estimation techniques. We do this by turning the Artificial Neural Network into the form of a Recurrent Neural Network. This brings its own set of challenges, for which we apply solutions from the appropriate branch of machine learning research. This provides a suitable alternative to methods such as dual Kalman filters commonly used in literature to estimate the time-varying battery parameters in the SOH estimation.

(2) We introduce a very general, universally applicable method to augment any base model – such as an ECM, in which each parameter has a physical meaning and is interpretable – with a black-box extension that is able to capture structured leftover unexplained behaviour that was not captured by the physical model.

All the methods introduced in this work are fully compatible with the usage of Kalman filters. Furthermore, by designing the structure of our method even allows for further integration of the Kalman filter into the optimisation of the SRNN, but this remains a topic for further research in the future.

TABLE OF CONTENTS

List of Abbreviations and Acronyms

ACRONYM	DEFINITION
ANN	Artificial Neural Networks
BP	Back Propagation
BMS	Battery Management System
CPU	Central Processing Unit
CAN	Controller Area Network
CNN	Convolutional Neural Network
DKF	Dual Kalman Filter
ECM	Equivalent Circuit Model
EKF	Extended Kalman Filter
ELM	Extreme Learning Machine
GIL	Global Interpreter Lock
GPU	Graphics Processing Unit
JKF	Joint Kalman Filter
LSTM	Long Short Term Memory
MSE	Mean Squared Error
MLP	Multilayer Perceptron
MLP	Multi-Layer Perceptron
OCV	Open Circuit Voltage
RNN	Recurrent Neural Network
RC	Resistor-Capacitor
SOC	State of Charge
SOH	State of Health
SNN	Structured Neural Networks
SRNN	Structured Recurrent Neural Networks
UKF	Unscented Kalman Filter

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Determining the State of Charge/State of Health (SOC/SOH) of a battery is a research topic that has received considerable attention. This is mainly attributed to the fact that indirect techniques are preferred over direct techniques (such as the prevalent Coulomb counting), the latter suffering from severe divergence issues due to its integrating character. Indirect techniques, however, rely on a model to calculate the hidden SOC from direct measurements such as current and voltage. Creating effective and practical reverse modelling techniques however, is not a trivial endeavour.

This deliverable will describe the State of Charge/State of Health (SOC/SOH) estimation efforts performed by VITO, with regards to the EVERLASTING project.

The document is structured as follows:

- in the first section, we will briefly cover the taxonomy of approaches to state of charge estimation, and how our own estimation methods fit within this context.
- the second section will cover the way our data was collected, and the different datasets that were used to train and test our different methods.
- the third section will give a basic overview of how the Kalman filter works, and what our battery model looks like.
- the fourth section covers our contributions, how a new structured error term can be added to this model to capture additional nonlinear behaviour not captured by the simplified model.

To explain a broader and less technical audience, the definitions and the methods State of Charge and State of Health (SOC/SOH) estimations, the EVERLASTING project has published the following white papers:

- D8.3: BMS Functions
- D8.5: SoC Definition
- D8.6: Evaluation of SoC Accuracy
- D8.7: Definition of SOH.
- D8.8: Cell Testing and Estimation of SOH.

All white papers are available on: https://everlasting-project.eu/results/deliverables-reports/.

1 Related Work & Taxonomy

In this section, we will provide a rough taxonomy of the different methods that are currently available for SOC and SOH estimation in the field of battery systems and how our current methods fit within this larger framework.

We will make the primary categorisation based on the method's requirement of retaining a state.

Approaches that do not retain a state reduce the problem to a curve-fitting approach. However, without incorporation into a larger adaptive (stateful) framework, they cannot capture any of the nonlinear behaviours of a dynamic battery system[1].

Approaches that do retain a state can be described in terms of a dynamic system, and at each timestep they determine the next state based on the information of the previous state, combined with the information collected by the sensors.

1.1 Stateless Methods

In this section, we will cover the stateless curve-fitting models. It should be noted that these methods require the explicit presence of an accurate measurement of the SOC within the training set, which in normal circumstances is the hidden variable that we are trying to predict. Thus, we would only be able to train these models on short sequences for regions where we can trust the SOC estimates produced in test lab conditions. This limits their practical use when trying to collect and use larger amounts of training data.

In this section, we will also include the methods that combine a simple or extended RC (Resistor-Capacitor) model with a direct estimation of the OCV (open circuit voltage) curve. We know the OCV is equal to the terminal voltage when there are no polarisation effects or voltage drops on the internal impedance. The OCV is then directly related to the battery SOC [1].

1.1.1 Low-Dimensional Functions

In a later section, we will take a closer look at the specific training data used to train our algorithms. For now, it suffices to say that a quick visual inspection of the data shows us that the manifold relating our input parameters to the state of charge is fairly smooth. This opens up the possibility of fitting a low-dimensional function to the data. Examples in the literature include (partially) linear models [20][4], logarithmic models [20][4], exponential models [4][6][7][8][17], polynomial models [8][6][10][11], sums of sines [8], combinations of these, or potentially any other curve that is sufficiently low in numbers of parameters. Other authors have opted to employ more classical methods from statistics to perform this task [10].

1.1.2 Machine Learning

Many estimators are available when we look to the domain of machine learning to fit the training data. Support vector machines are a plausible candidate [5][14][15], but non-differentiable estimators such as gradient boosted trees or random forests are not an option[2]. Similarly, simple linear models will

[2] This is for two reasons: (1) Non-differentiable functions are not usable when we employ the Extended Kalman Filter. This can be overcome by swithing to the Unscented Kalman Filter. (2) But also, these functions lack sufficient continuity, which is a requirement to have a realistic modeling of a physical process.

fail due to the highly nonlinear nature of the data. Another option consists of ensemble methods, such as bags of extreme learning machines (ELMs).

1.1.3 Deep Learning

An alternative to the parametrised curve-fitting approach consists of fitting a neural network to the inputs/SOC manifold. The main difficulty lies in choosing a suitable architecture, which combines an appropriate model capacity to generalise well into the test cases. Some preliminary experiments of our own seem to suggest that a simple feedforward neural network architecture seems to be sufficient to get decent results under the known limitations of the stateless methods. This has been confirmed by literature, where the results of the trained model are either combined with the extended [15] or the unscented [17] Kalman filter to perform adaptive SOC estimation.

1.1.4 Structured Neural Networks

A fairly recent development in the world of deep learning is the usage of so-called structured neural networks [1], which is a hybrid form of ANN that replaces part (or all) of the black box components in a feedforward neural network by a structure that corresponds at least partially to a physical model. Parameter estimation of this model is performed traditionally using backpropagation (BP) and an optimiser of choice (e.g. stochastic gradient descent).

1.2 Stateful Methods

Whereas the previous methods did not retain any state by themselves, they usually need to be contained within a system that does so to be sufficiently adaptive to real-world situations. Such approaches that do retain a state can be described in terms of a dynamic system, and at each timestep, they determine the next state based on the information of the previous state, combined with information collected by the sensors.

1.2.1 Coulomb Counting

In the most basic scheme, called Coulomb counting, we use a highly accurate measurement device to register the currents sent to the battery and simply integrate the State of Charge based on this computation [19]. As long as the sequences are short and the sensor is of high quality, the estimates made by this approach are considered accurate. However, a very accurate sensor is too expensive to include in a conventional battery management system and thus only present in lab conditions. Additionally, for longer sequences (data collected over more than a week), small inaccuracies will build up over time, rendering the SOC estimate no longer trustworthy.

1.2.2 Kalman Filter

The Kalman Filter, also known as linear quadratic estimation, is an iterative linear algorithm that combines the information from predictions made by a linear model with a series of measurements coming from a noisy sensor. Both sources of information are assumed to be subjected to Gaussian noise with a known prior distribution, represented by their respective covariance matrices.

The Extended Kalman Filter (EKF) [20] removes the restriction of modelling the process using a *linear* dynamic system. Instead, we can employ any state transition function, as long as it is differentiable, and we still have a way to compute a reasonable covariance matrix.

The Unscented Kalman Filter (UKF) [20] additionally removes the requirement for the transition function to be differentiable by generating a set of so-called sigma points from the initial Gaussian

distributions and fitting a new set of Gaussian distributions on the projections of these points under the transition function.

The Dual Kalman Filter (DKF) [22] is another compatible extension, where we are using two parallel Kalman filters to update both state and parameter estimates simultaneously. In the context of battery management systems, the DKF is used to update the resistance and capacity values, which serve as indicators of the general state of health (SOH) of the battery. The DKF is a less computationally intensive alternative to the Joint Kalman filter, which extends the state space to also include the parameters[3].

The battery model we employ for our dynamic system is an equivalent circuit model (ECM). More specifically, it is a 1st order RC Circuit with a nonlinear variable voltage source dependent on the state of charge. For more information on the Kalman Filters and our battery model, we refer to sections 1.2.2 and 3.2.

1.2.3 Structured Recurrent Neural Networks

In this work, we introduce a novel way to capture the nonlinear behaviour that remains when using a relatively simple quasi-linear dynamic system such as the 1st order RC circuit that formed the basis of our work. We extend this battery model with a structural error term, which we try to learn based on the available training data.

In theory, you could use any estimator to approximate this nonlinear behaviour, as long as you have complete knowledge about the evolution of the true state vector, which is already extremely hard to achieve in perfect lab conditions. In practice, due to ageing effects, temperature differences, chemical effects and simple acts of randomness, the actual nonlinear effects might even be sufficiently different from those of the reference battery cell(s) that a practical system will most likely benefit from online retraining. Keeping this in mind, we opt for a slightly different approach, which restricts our list of potential estimators. We will illustrate this in more detail in section 4.4, when we present our hybrid approach that combines a Kalman filter with a structured neural network.

[3] If we have state space of dimension M, and a parameter space of dimension N, the DKF will have two covariance matrices of MxM and NxN, while the JKF will have a single covariance matrix of (M+N)x(M+N).

2 DATA COLLECTION

The original title of deliverable D7.5 was "Validation report of self-learning SoH algorithm on full electric bus". The title has been changed to "Validation report of self-learning SoH algorithm on lithium-ion module" because the source of the dataset, used for the validation of the SoH algorithm developed in Task2.3, has been changed. The main reason for this change of source of datasets is that some electrical and performance information of the cell is needed to build the model. Due to confidentiality reasons, it was not easy to collect the needed data on cells used in electric buses which are owned and operated by partners from outside the consortium. Consequently, and for consistency reasons, the same battery cell that was used in the other EVERLASTING research tasks was also selected for Task7.6 to run the SOH validation tests on the battery modules level.

The battery cell selected in the EVERLASTING project is an 18650 high energy lithium-ion cell from LG Chem (INR18650 MJ1). According to the manufacturer datasheet, it has a nominal capacity of 3.5 Ah and specific energy of 259.6 $Whkg^{-1}$. It has an ohmic resistance of around 0.04 Ohm, and the recommended operating voltage window varies between 2.5 and 4.2 V.

In the previous deliverable D2.4 "Self-Learning Algorithms for SOC/SOH Estimation", data from individual battery cells were used. But in a real application such as an electric vehicle, a battery pack consisting of many battery cells in series will be used. Therefore, for this validation, a 16S1P battery module was assembled from the same cells and connected to a battery tester machine. The ~60V module has a 4x4 shape with a 4 mm gap between the cells, provided by a 3D printed support structure. The cells capacity was initially balanced (i.e. brought on the same level), then shortened and scaled-down driving profiles were applied, followed by constant current charging and regularly interrupted to run a few checkup tests.

The battery module was cycled for about 4 months using the PEC battery tester in the VITO battery lab, and the total voltage, current and temperature were logged with at least 0.5 s sample intervals. A BMS, depicted in Figure 2, was also connected to the cells, allowing the recording of individual cell voltages and temperatures. The BMS sends these measurements over CAN-bus to the battery tester machine, which includes it in the test log, but the BMS did not perform additional balancing.

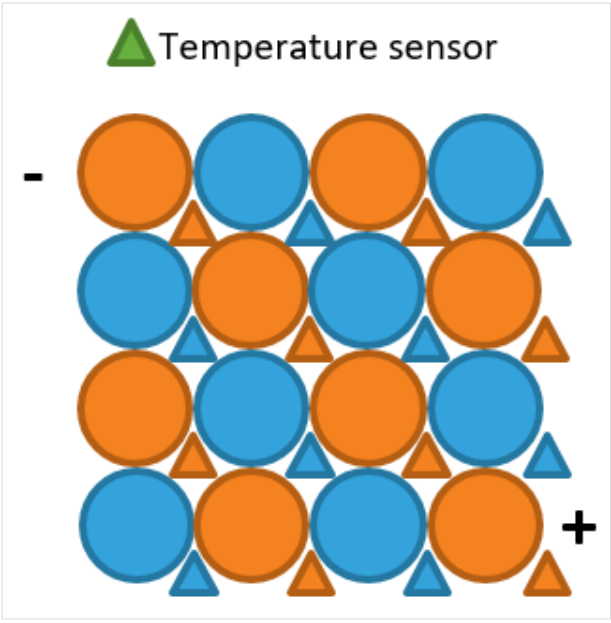

Figure 1: Schematic view of the structure of the 16S1P battery module

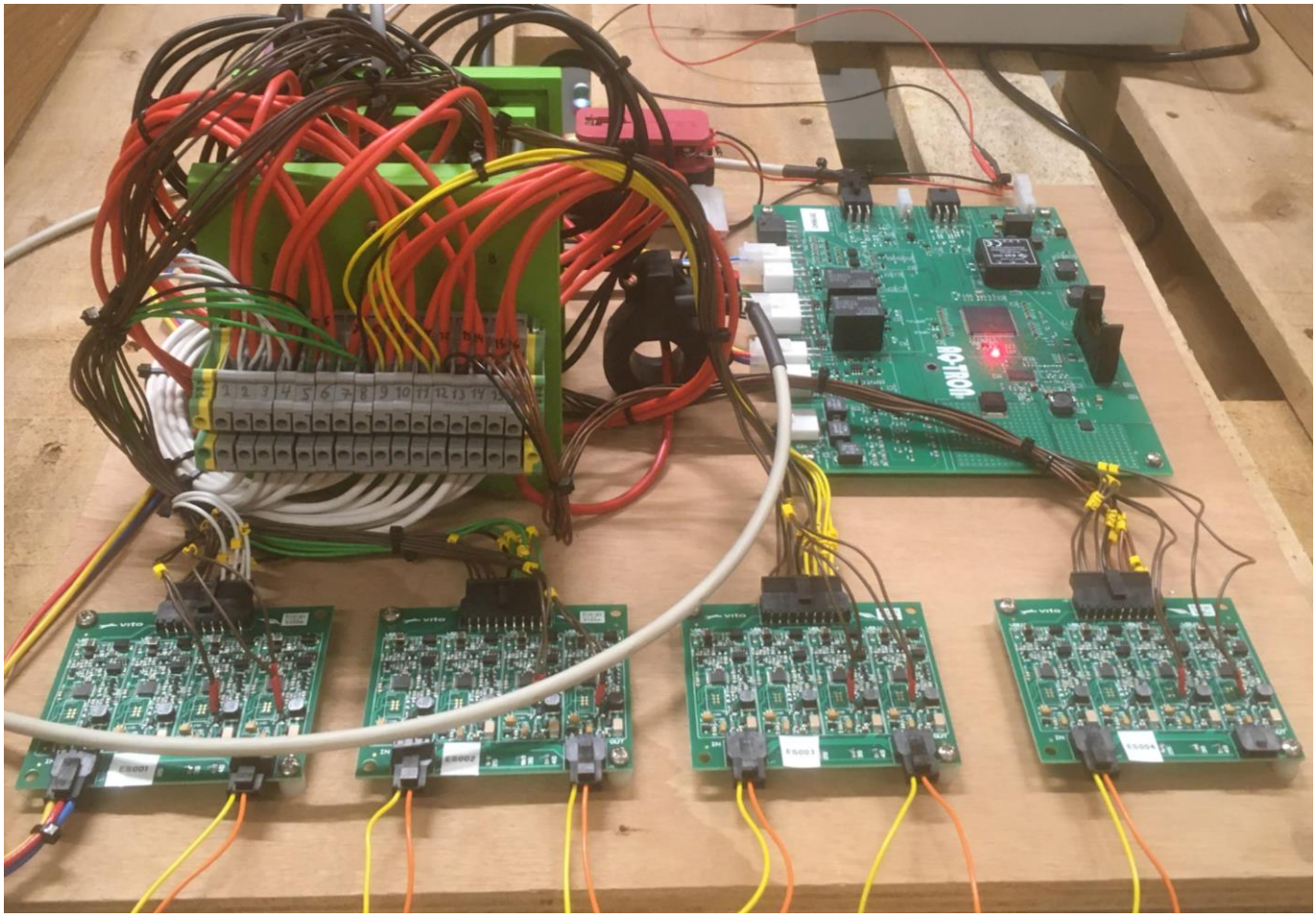

Figure 2: Battery module (16S1P) setup (BMS, temperature sensors & individual cell connections)

The applied test profiles can be divided into four categories: (1) characterisation tests, (2) calendar ageing tests, (3) cyclic ageing tests and (4) driving profiles.

For the purpose of training and evaluating our structured recurrent neural networks, data from categories (1-3) (collected from Task 2.4) is mainly used to train the network, while category (4) represents realistic patterns to be observed in the field, and thus forms a good test set. Figure 4 to Figure 7 show examples of the tests used to train our models, visualising voltage, current, and temperature.

Characterisation tests were performed to determine the cell parameters, containing a charge/discharge cycle at C/20 and current pulses to determine internal resistance at different SOC levels. Tests were performed at four different temperatures (4°C, 25°C, 35°C, and 45°C). Unlike the majority of the methods in section 1.1, we do not fit a parametrised curve through the data gathered from our offline OCV tests. Because we have a rather dense set of SOC/OCV data points, with a 1% SOC resolution, we have instead opted to use bilinear interpolation. The resulting curves from these tests are illustrated in Figure 3.

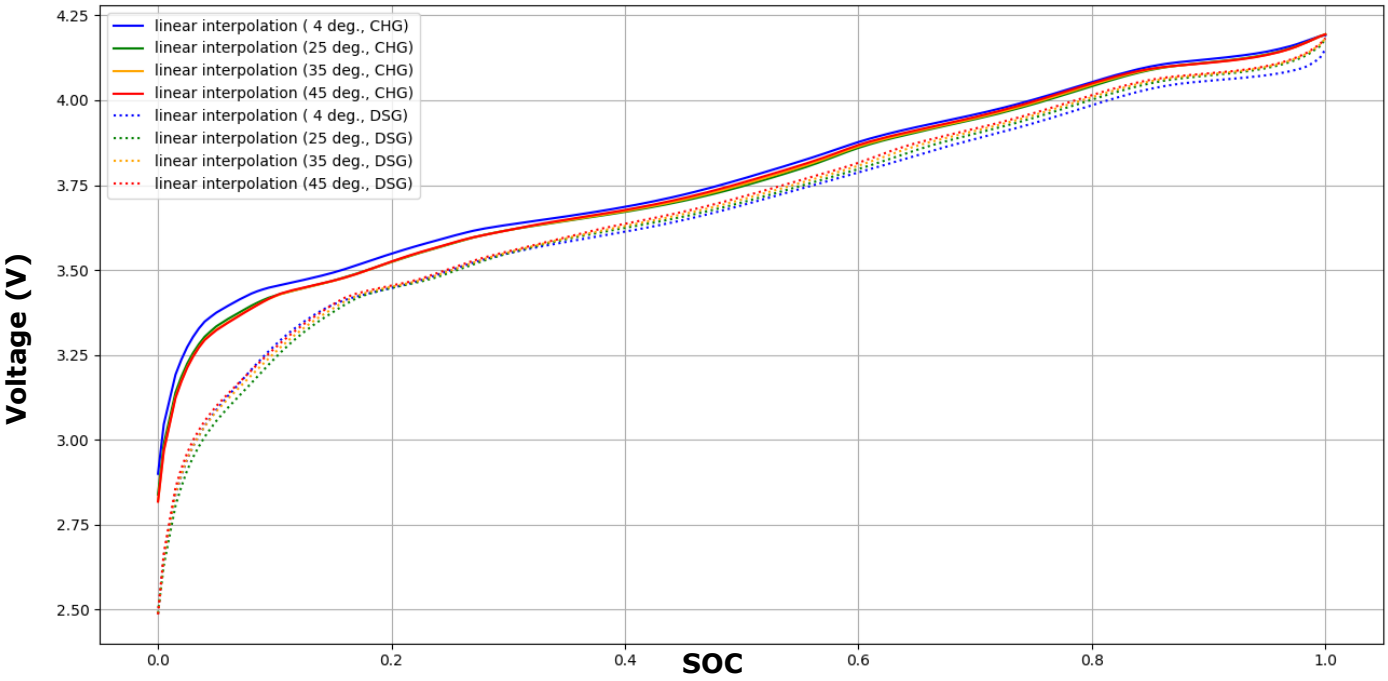

Figure 3: OCV curves at different temperatures and after charging and discharging at C25

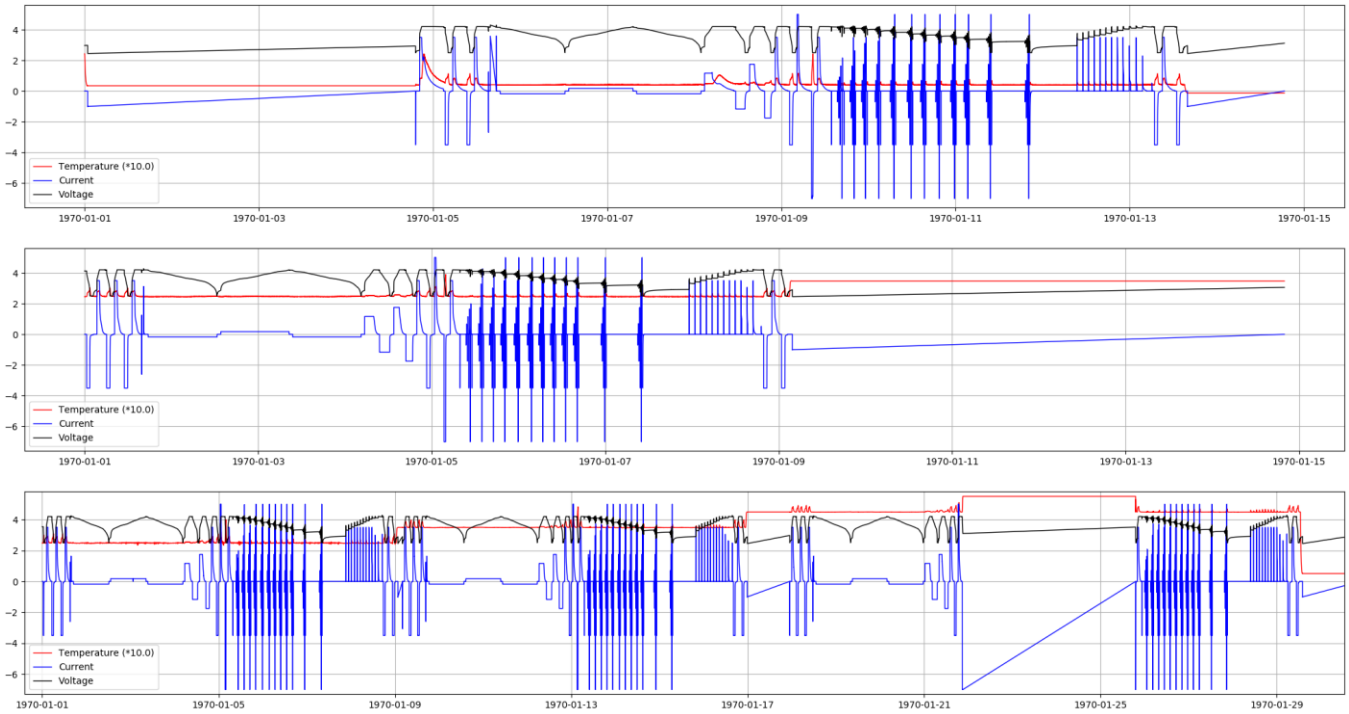

Figure 4: Examples of characterisation tests (time relative to 1970-1-1)

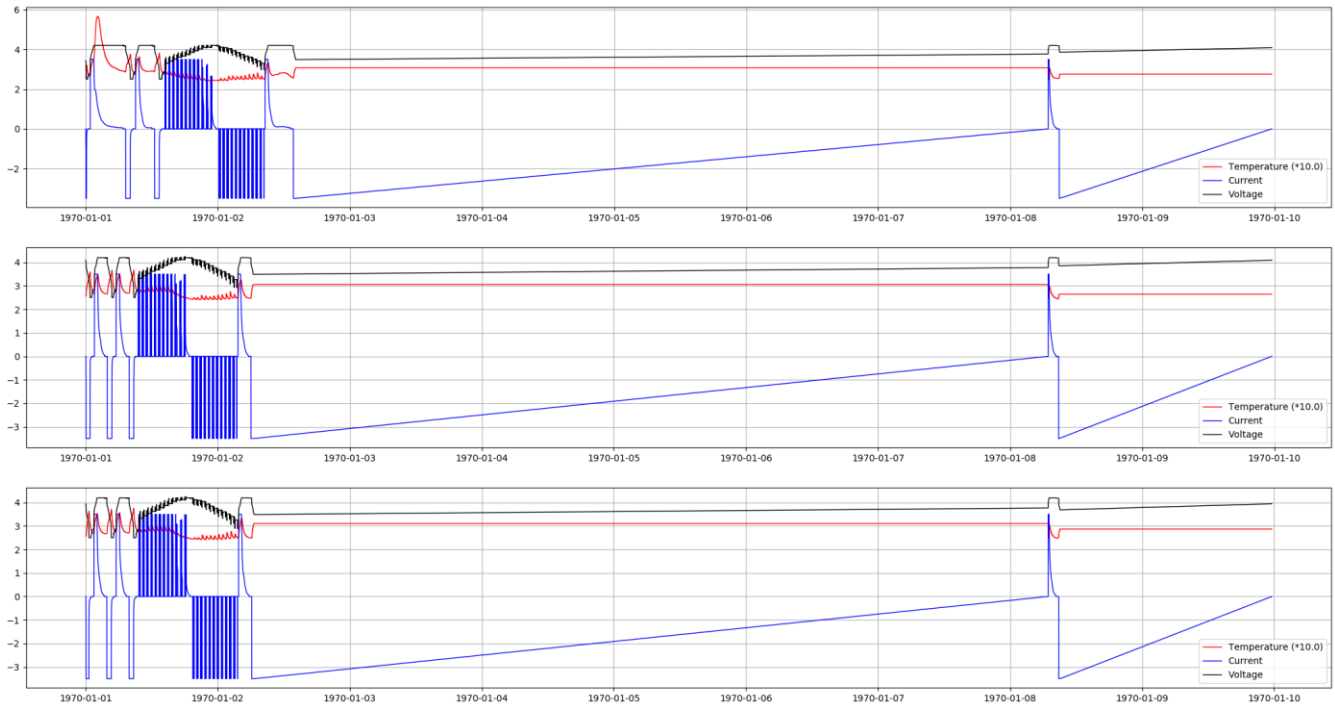

Figure 5: Examples of calendar ageing tests

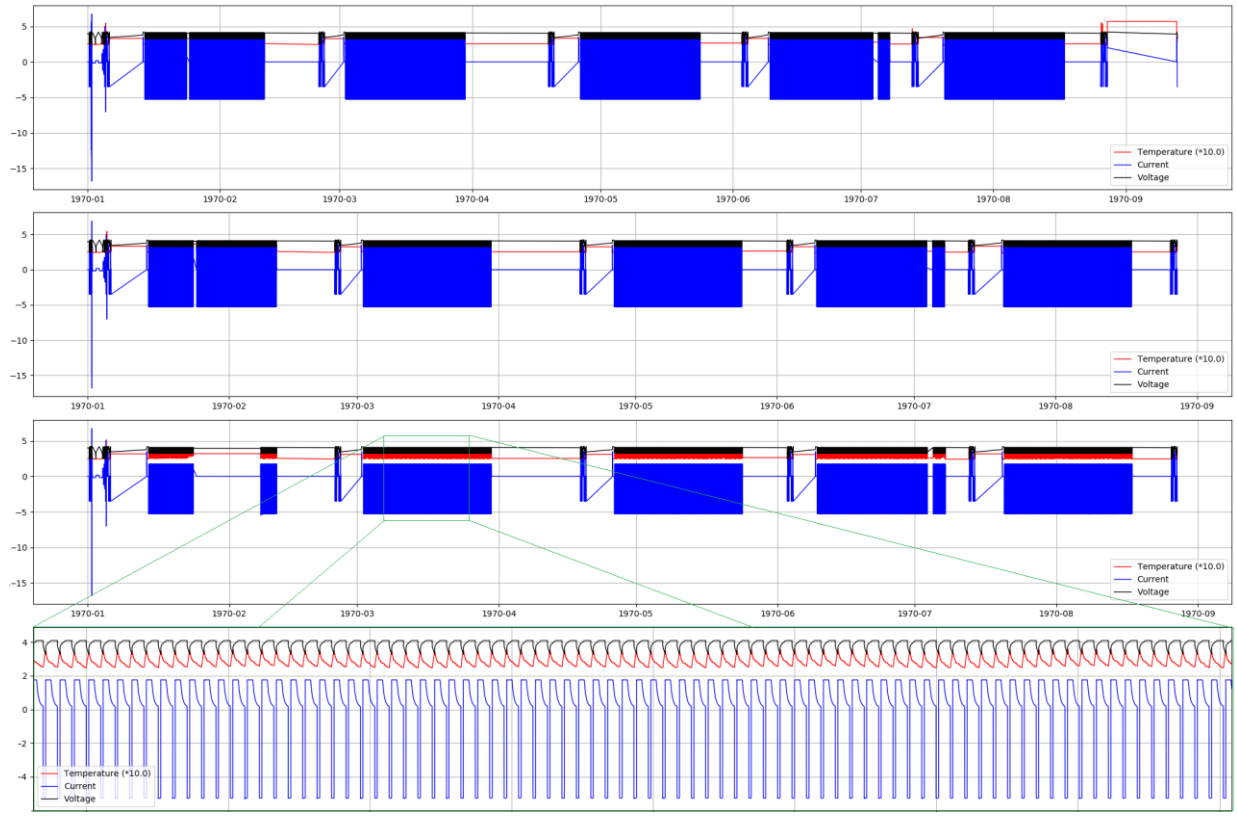

Figure 6: Examples of cyclic ageing tests

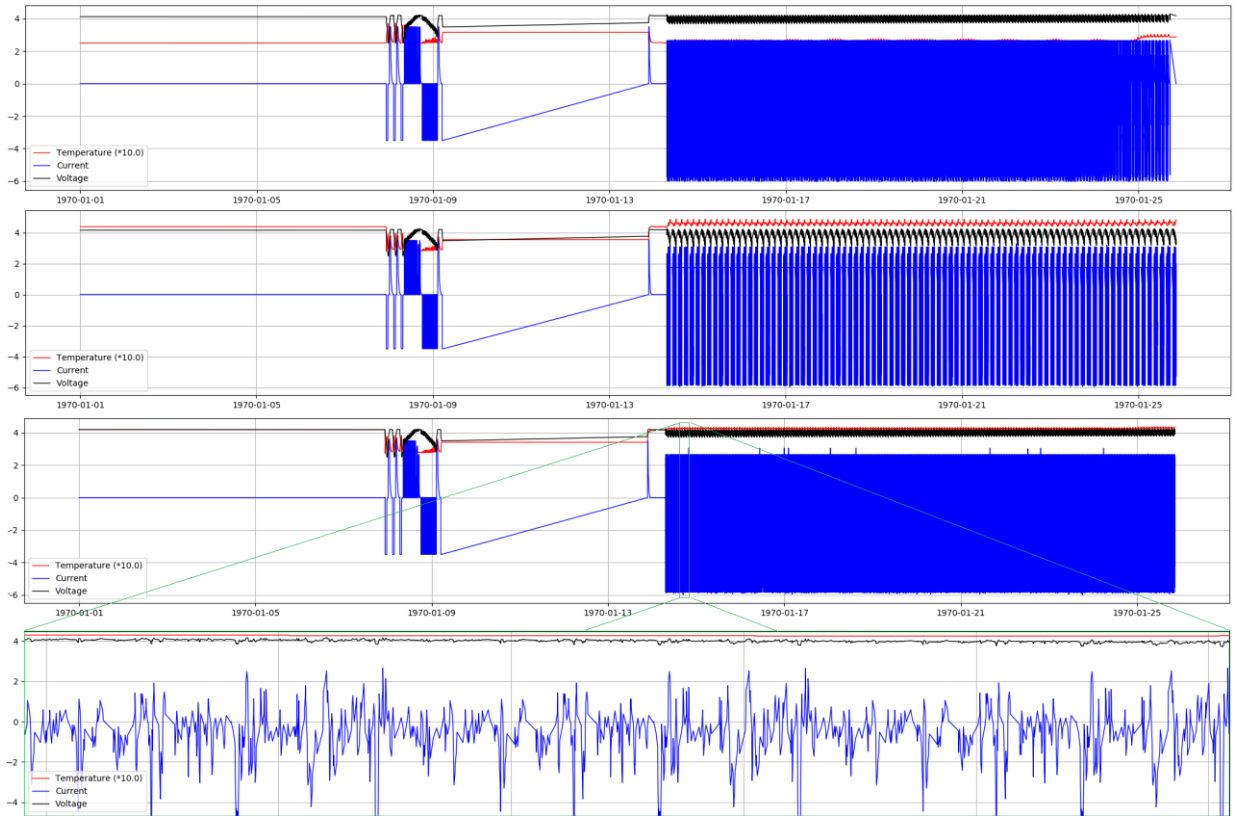

Figure 7: Examples of driving profiles

3 THE BASE MODEL

3.1 THE KALMAN FILTER

As introduced in the section 1.2.2, the Kalman Filter and its nonlinear variants (Extended Kalman Filter and Unscent Kalman Filter) are recursive estimators which combine information from a linear/nonlinear models and measurement sensors assuming both information affected by Gaussian noise.

In its most general form, the dynamic system can thus be described by the following set of equations, where $\mathbf{x}_k$ and $\mathbf{z}_k$ respectively represent the hidden state and the data measured by the sensors at timestep *k*. In addition, $\mathbf{u}_k$ represents the employed control actions, *f* is the transition function to go from one state to the next, and *h* is the observation function that translates the hidden state into the corresponding sensor data that we expect to see, based on the control actions taken by the user. The terms $\mathbf{w}_k$ and $\mathbf{v}_k$ are Gaussian noise terms.

$$x_k = f(x_{k-1}, u_k) + w_k \quad (1)$$
$$z_k = h(x_k, u_k) + v_k \quad (2)$$

For such a system, the iterative Kalman cycle of predicting and correcting works as follows:

- Given the initial state $\mathbf{x}_{k-1}$ of the system at timestep *k-1* and the control actions $\mathbf{u}_k$, we simply apply the transition function *f* to predict the next state $\hat{x}_k = f(x_{k-1}, u_k)$.
- Similarly, we predict the updated state covariance matrix $\mathbf{P}_k$ using the Jacobian $\mathbf{J}_f$ and the process covariance/noise matrix $\mathbf{Q}_k$:

$$\boldsymbol{P}_k = \boldsymbol{J}_f \boldsymbol{P}_{k-1} \boldsymbol{J}_f^T + \boldsymbol{Q}_k \quad (3)$$

- We then move on to correcting these predictions using the so-called *Kalman Gain* **K**, which is based on the state covariance matrix $\mathbf{P}_k$, the sensor covariance/noise matrix $\mathbf{R}_k$, and the Jacobian $\mathbf{J}_h$ of the observation function *h*:

$$\boldsymbol{K} = \boldsymbol{P}_k \boldsymbol{J}_h^T (\boldsymbol{J}_h \boldsymbol{P}_k \boldsymbol{J}_h^T + \boldsymbol{R})^{-1} \quad (4)$$

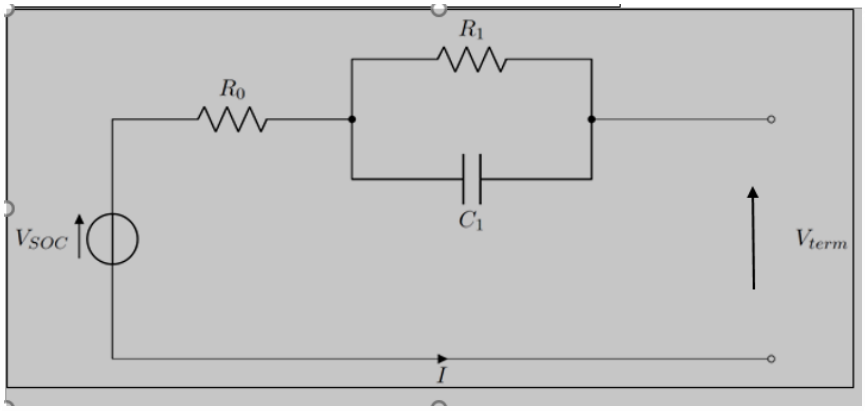

Figure 8: Implemented battery model, a 1st order RC circuit

- Using this matrix, we correct the predicted $\hat{x}_k$, based on the difference between the predicted observations $h(\hat{x}_k, u_k)$ and the observations z_k made by the sensors.

$$x'_k = x_k + \boldsymbol{K}\big(z_k - h(x_k, u_k)\big) \tag{5}$$

- Finally, we correct the state covariance matrix $\boldsymbol{P}_k$ using $\boldsymbol{K}$ and the Jacobian $\boldsymbol{J}_h$:

$$\boldsymbol{P}'_k = \boldsymbol{P}_k - \boldsymbol{K}\boldsymbol{J}_h\boldsymbol{P}_k \tag{6}$$

All Jacobians are computed concerning the current state vector $\hat{x}_k$. For a more detailed explanation, we refer to the literature [20][23]. We have visualised the computational flow described above in Figure 9.

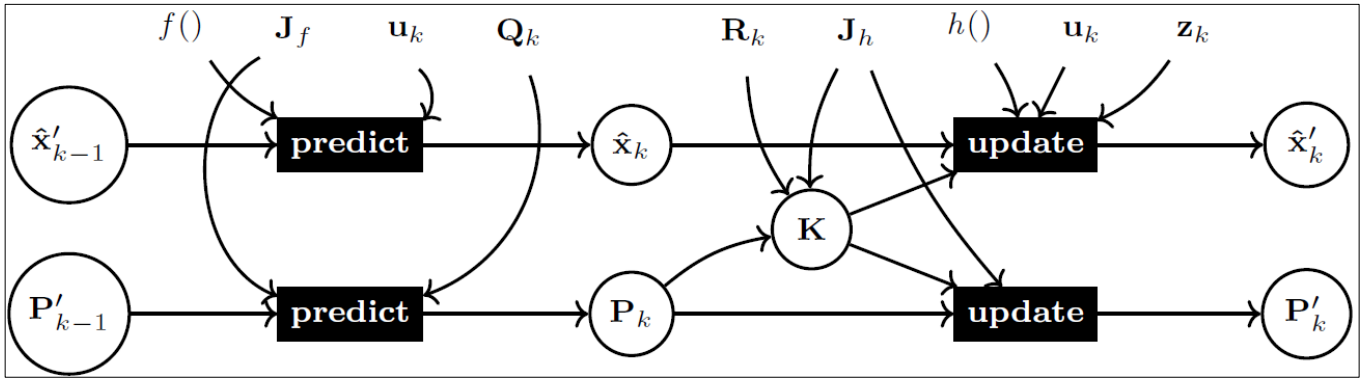

Figure 9: The computational flow of the (Extended) Kalman Filter

3.2 The Battery Model

Like many others in our work, we use an equivalent circuit model (ECM) to represent the relationships between the internal state, the current, and the terminals' voltage. In this ECM, illustrated in Figure 8, the behaviour of the cell is modelled using a variable voltage source V_{SOC} (depending on the actual SOC via the OCV function h_s), an internal resistance R_0, and a single RC branch representing the polarisation effects, using the elements C_1 and R_1.

The voltage at the terminal can thus be written as:

$$V_{term} = h_s(SOC) + I \cdot R_0 + V_1 \tag{7}$$

in which the voltage over the parallel branch V_1 can be expressed as a function of its previous state as well:

$$V_{1,k} = \left(1 - \frac{\Delta t}{R_1 C_1}\right) \cdot V_{1,k-1} + \frac{\Delta t}{C_1} \cdot I_k \tag{8}$$

and the state of charge can be computed by simple Coulomb counting:

$$SOC_k = SOC_{k-1} + \frac{\Delta t}{C_n} \cdot I_k \tag{9}$$

Altogether, our dynamic system can be summarised as follows:

$$x_k = \begin{bmatrix} SOC_k \\ V_{1,k} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{\Delta t}{R_1 C_1} \end{bmatrix} x_{k-1} + \begin{bmatrix} \frac{\Delta t}{C_n} \\ \frac{\Delta t}{C_1} \end{bmatrix} u_k \tag{10}$$

$$u_k = [I_k] \tag{11}$$

$$z_k = h(x_k, u_k) = V_{term} = h(SOC_k) + V_{1,k} + I_k R_0 \tag{12}$$

Additionally, to employ the extended Kalman filter, we need to linearise the observation function h around (x_k, u_k) at each timestep k:

$$J_{h,k} = \begin{bmatrix} \frac{\partial h_s(SOC)}{\partial SOC_k} & 0 \\ 0 & 1 \end{bmatrix} \tag{13}$$

The Jacobian for the transition function J_f will be static throughout the time series, but the Jacobian of the observational function J_h is always dependent on the SOC at timestep k. However, for the brevity of notation, we will drop the k for each Jacobian as there will never be any ambiguity as to which timestep the Jacobian belongs to.

4 EXTENDING THE MODEL WITH NONLINEAR BEHAVIOUR

In this section, we will detail how we extended the general EKF algorithm with a nonlinear state transition error correction term. In order to do this, we have made several assumptions that may or may not hold depending on the application:

- The employed model in the vanilla EKF does not capture all the nonlinear behaviour, but it can at least be described in terms of known state and control vectors.
- The observation function is known and correct, up to a Gaussian noise term. So we know how the state vector translates into a corresponding sensor measurement and that any changes in the predicted sensor readings are due to changes in the transition function alone.

So mathematically, this means that we know the dimensions of vectors x and u, and have a defined observation function h that does not change over time.

For the remainder of this section, the equations for the uncorrected base model will be referred to with a* superscript:

$$x_k^* = f^*(x_{k-1}, u_k) \quad (14)$$
$$z_k^* = h(x_k^*, u_k) \quad (15)$$

where the transition function f^* refers to the transition function of the base model. After correction, the new dynamics can thus be described as:

$$\hat{x}_k = f(x_{k-1}, u_k) = f^*(x_{k-1}, u_k) + f^{\varepsilon}(x_{k-1}, u_k) \quad (16)$$
$$\hat{z}_k = h(\hat{x}_k, u_k) = h(x_k^*, u_k) + e(x_k^*, u_k) \quad (17)$$

where transition function f^{ε} refers to the correction of transition function that we are trying to estimate, and $e(x_k^*, u_k)$ is the difference between the predicted sensor measurements by the base model $h(x_k^*, u_k)$ and the corrected model $h(\hat{x}_k, u_k)$.

During our research, we started by trying to estimate the transition error correction term $f^{\varepsilon}(x_{k-1}, u_k)$ directly from the observed time series. We tried first two slightly different approaches, which we will discuss in the following sections 4.1 and 4.2. Then, we explain the pitfalls of both methodologies in 4.3 and our final solution based on structured recurrent neural network in 4.4.

4.1 METHOD 1: STRUCTURE IN THE MEASUREMENT ERROR

The first version of our method focused on the measurement error made by the base model, trying to learn f^{ε} from sensory information alone. The underlying assumption here was that this measurement error contains the inherent structure and thus is not simply the product of Gaussian noise. If there is a structure, it can be learned and translated into a corresponding transition error correction term.

Procedure Overview

In order to learn the error corrected state $\hat{x}_k$ from the observed error $e(x_k^*, u_k)$, we followed the following procedure:

- Learn the structure in the measurement error $e(x_k^*, u_k)$, as x_k^* is known from the training set, thus minimising the distance $\|z_k - \hat{z}_k\|$between the measurements z_k and the error corrected predictions $\hat{z}_k$.

- Based on these predictions, find the function $f^{\varepsilon}(x_{k-1}, u_k)$ that allows us to locate the error corrected state $\hat{x}_k$.
- Compute the Jacobian of $f^{\varepsilon}(x_{k-1}, u_k)$ in order to update the Jacobian of $f(x_{k-1}, u_k)$.

Step 1: Learning the Structured Measurement Error

Learning the function $e(x_k^*, u_k)$ can be done by applying any regressor of choice, as long as the function it learns is differentiable. We explored the usage of linear regression, ensembles of extreme learning machines (ELMs), feedforward neural networks (MLPs), and convolutional neural nets (CNNs).

Step 2: Determining the Correction for the Hidden State

We recall the definition of the corrected predicted output states:

$$\hat{z}_k = h(\hat{x}_k, u_k) = h(x_k^*, u_k) + e(x_k^*, u_k) \tag{18}$$

If $h : (x, u) \to z$ is injective or there exists a surjective $h^+ : (z, \theta) \to x$, where θ contains all information required to uniquely map z back to x, this implies:

$$\begin{cases} (\hat{x}_k, u_k) = h^{-1}(\hat{z}_k) & \text{if } h \text{ is injective} \\ \hat{x}_k = h^+(\hat{z}_k, \theta) & \text{if } \exists h^+ : (z, \theta) \to x \text{ where } h^+ \text{is surjective} \end{cases} \tag{19}$$

Step 3: Computing the Jacobian

Updating the Kalman filter requires us to be able to compute the Jacobian of $f = f^* + f^{\epsilon}$. As Jacobian of a sum of functions is equal to the sum of the Jacobians of those functions, $J_f = J_* + J_{\epsilon}$, it suffices to compute the Jacobian of f^{ϵ} w.r.t. the input vector x.

4.2 Method 2: Structure in the Transition Error

In the previous trial method, we focused on estimating the structure in measurement error $e(x_k^*, u_k)$ in order to compute $\hat{x}_k$. The most ambiguous component in this procedure was the specification of the function h^+, which maps an observation z back onto a unique state vector x. To be more precise: in the previous method, we were using h^+ to map an updated output *estimate* $\hat{z}$ back to the corresponding corrected hidden state $\hat{x}$.

Alternatively, as this approach already required us to specify this h^+, we also could choose to learn the transition function correction $f^{\epsilon}(x_{k-1}, u_k)$ directly instead. The general approach remains the same as in the first method, except it becomes slightly easier because we estimate the thing we are interested in *directly*.

Procedure Overview

To learn the error corrected state $\hat{x}$ from the structural transition error $f^{\epsilon}(x_{k-1}, u_k)$, we followed the following procedure:

- Use the inverse observation function h^+ to map each observation z_k to a unique hidden state x_k.
- Train a differentiable estimator $f^{\epsilon}(x_{k-1}, u_k)$, minimising the distance between these hidden states and the error corrected predictions: $\|x_k - \hat{x}_k\|$.
- Compute the Jacobian of $f^{\epsilon}(x_{k-1}, u_k)$ in order to update the Jacobian of $f(x_{k-1}, u_k)$.

This is a cleaner procedure, as h^+might contain nonlinearities that would result in poorer performance of the initial method.

Step 1: Learning the Structured Transition Error

As in the previous method, we require h^+ to be surjective: each observation z must be mapped onto a single corresponding hidden state x. If dim(x) ≥ dim(z), h is certainly not bijective, and we have an entire manifold of potential solutions for x. One way to deal with this is by picking the point x on this manifold that minimises the distance to the estimated hidden state by the incomplete model: $\|x - x^*\|$.

Step 2: Determining the Correction for the Hidden State

Like with the structured measurement error, learning the function $f^\epsilon(x_{k-1}, u_k)$ can be done by applying any estimator of choice, as long as the computed function is differentiable.

Step 3: Computing the Jacobian

Once again, $J_f = J_* + J_\epsilon$, so it suffices to compute the Jacobian of f^ϵ w.r.t. the input vector x.

4.3 The Problems of Method 1 and Method 2

There is one very important caveat with the two methods above: as the function f describes the transition from the previous state to the next, this means that if x_{k-1} is an estimate, the error on x_k and every subsequent estimated state $x_{k+i}, \forall\, i \geq 1$ will accumulate with every additional timestep.

As a result, we cannot simply first run our simulation with (or without) a Kalman filter, use the uncorrected model to generate an initial set of x_k^* and z_k^* values, and subsequently use these values to estimate the function f^ϵ using the methods described above.

There are possible solutions to this problem, as we already mentioned in report D2.4. Among them, we briefly introduce our proposed approach in the following section.

4.4 Proposed Solution: Structured Recurrent Neural Network

The idea of our approach is using the estimates made by the faulty base model in order to estimate f^ϵ, we incorporate the base model itself into the optimisation procedure. This means we will need to estimate the complete transition function using an estimator that can directly incorporate the previous model. In addition, the estimation will have to be based solely on the observation data. We propose to use a Structured Recurrent Neural Network (SRNN) to build such an estimator which is based on observation data, but it also can incorporate the model structure described through the equations (14)-(17). We can describe such SRNN as a regular feedforward neural network coupled with an additional layer representing the model *f* or *h* in the eqs. (14)-(17).

The main problem with the methods described in sections 4.2-4.3 stems from the fact that we are trying to estimate the transition error correction term f^ϵ based on consecutive points from a *fixed/static* trace $\boldsymbol{X}^*$ that has deviated too much from the true trace due to accumulated error. So if we want to improve upon our base model's performance, we must allow for this trace to be updated dynamically during the optimisation procedure. As such, the base model itself must be incorporated in the optimisation as well.

Choosing this backpropagation approach provides us with a general framework to improve the initial model in several additional ways:

- So far, we have only talked about extending the transition function f by adding an extra f^{ϵ} term. Nothing is stopping us at this point from also adding an extension to the observation function: $h = h^{*} + h^{\epsilon}$. This could introduce small corrections to the OCV curve, structured biases due to non-Gaussian measurement noise, or other imperfections.
- Because we are dealing with a structured neural network, we can use this architecture to perform parameter estimation for the initial RC model (e.g. refining the initial ohmic resistance R_0 and total capacity C_n), or correcting the provided initial state x_0 at the beginning of a provided time series.

4.4.1 General Outline of our Algorithms

Data Processing

As we have mentioned in section 2, our data consists of four types of tests: (1) characterisation tests, (2) calendar ageing tests, (3) cyclic ageing tests and (4) driving profiles. For the purpose of training and evaluating our structured recurrent neural networks, data from categories (1-3) will be used to train and validate the network, while category (4) represent realistic charge patterns to be observed in the field and thus form a good test set.

All segments consist of multiple segments with 0.5 s resolution, with the exception of longer breaks when determining ageing effects or pauses to let the battery reach a stable state again. We cut all sequences at these longer breaks and resample each subsequence at the same resolution of 0.5 s. For computational efficiency reasons, we once more cut longer subsequences into equal parts.

Estimating the initial state of charge x_0

Most training sequences start with a known state of charge x_0, e.g. the battery either being completely empty or full. After running the simple RC model on the sequence (employing the EKF or UKF in the process), we also have initial state estimates for all other timesteps in the sequence.

However, these estimates are approximate at best, and the usage of a Kalman filter cannot solve them all. Therefore, as a first step, we make use of the first version of our structured recurrent neural network: no transition function extension f^{ϵ}, no observation function extension h^{ϵ}, only the simple RC model with the base function f^{*}, h^{*}, and a single parameter vector: Δx_0, initialised as a zero vector. Using backpropagation, we iteratively adjust the initial state and adjust the value of Δx_0 in order to minimise the error residuals $\|\hat{z}_i - z_i\|$ between the estimated voltage and the measured voltage.

The computation of the initial state of charge is dependent on a single sequence so that it can be performed for all subsequences in each dataset separately.

Estimating the Ohmic Resistance R_0, Total Capacity C_n

After we have estimated the initial state of change x_0, we perform another parameter estimation step. Even though we have a theoretical estimate for the ohmic resistance and the total battery capacity, these values change over time as the battery gets depleted. These values are directly linked to the State of Health (SOH) of a battery, and a common way to compute this SOH is to opt for a joint or dual version of the Kalman filter that is used with your model. These parameters are then given appropriate covariance matrices to ensure that they are only allowed to change slightly over time.

The Kalman filter can only make estimates of these parameter values based on the change of the state over a single timestep. As an alternative, you could train an SRNN over a short recent history to compute estimates for these values, making gradual adjustments to these values similar to how you would make adjustments in a normal Kalman filter. This effectively turns our $SRNN_{R_0,C_n}$ into a SOH estimator.

Estimating the Structural Error Terms f^{ϵ} and h^{ϵ}

Now that we have performed parameter estimation for the base model and have properly initialised all training sequences, we can use their collective datasets to train the parameters of the functions f^{ϵ} and h^{ϵ}. Depending on the complexities of the individual battery we are dealing with, we have to experiment with a proper architecture for these functions. We have ended up using two 2-layer feedforward neural networks, with dimensions [20,10] for both f^{ϵ} and for h^{ϵ}, using Mish as our activation function of choice.

Whereas we have experimentally validated our choice, theoretically, it is also possible to turn this into a set of tunable hyperparameters for Bayesian optimisation. Whether or not this approach would be computationally tractable remains to be seen, however.

4.4.2 RESULTS AND DISCUSSION

In this section, we report the results from two sets of experiments. First, we detail the impact of our proposed $SRRN_{x_0}$ and $SRNN_{R_0,C_n}$ on the initial state of charge estimation and on the parameters $\{R_0, C_n\}$ respectively, for a single-cell battery element. Then, we evaluate the overall performances of the proposed approach on a more general case of a multi-cells battery system.

Case study 1: Single-cell battery system

Considering a single-cell battery system, we use Figure 10 to illustrate the impact of proper initialisation of our starting state x_0 on the sequence.

The dotted lines refer to the usage of the base model (f^*, h^*), while full lines refer to the base model and proper initialisation of x_0. Red lines refer to the pure application of the model. Blue lines are results produced by combining the model with an extended Kalman filter. Green lines represent the results produced by combining it with the unscented Kalman filter. The 1st sequence visualises the measured/predicted terminal voltage. The 2nd sequence shows the current pattern sent to the battery. The 3rd sequence visualises the estimated state of charge.

The first thing we can see is that both the extended and unscented Kalman filters are able to converge to approximately correct values over time. It should be noted, however, that this is not the case for all sequences in the training set. However, applying our $SRRN_{x_0}$ enables the system to compute a better fit from the beginning of the sequence.

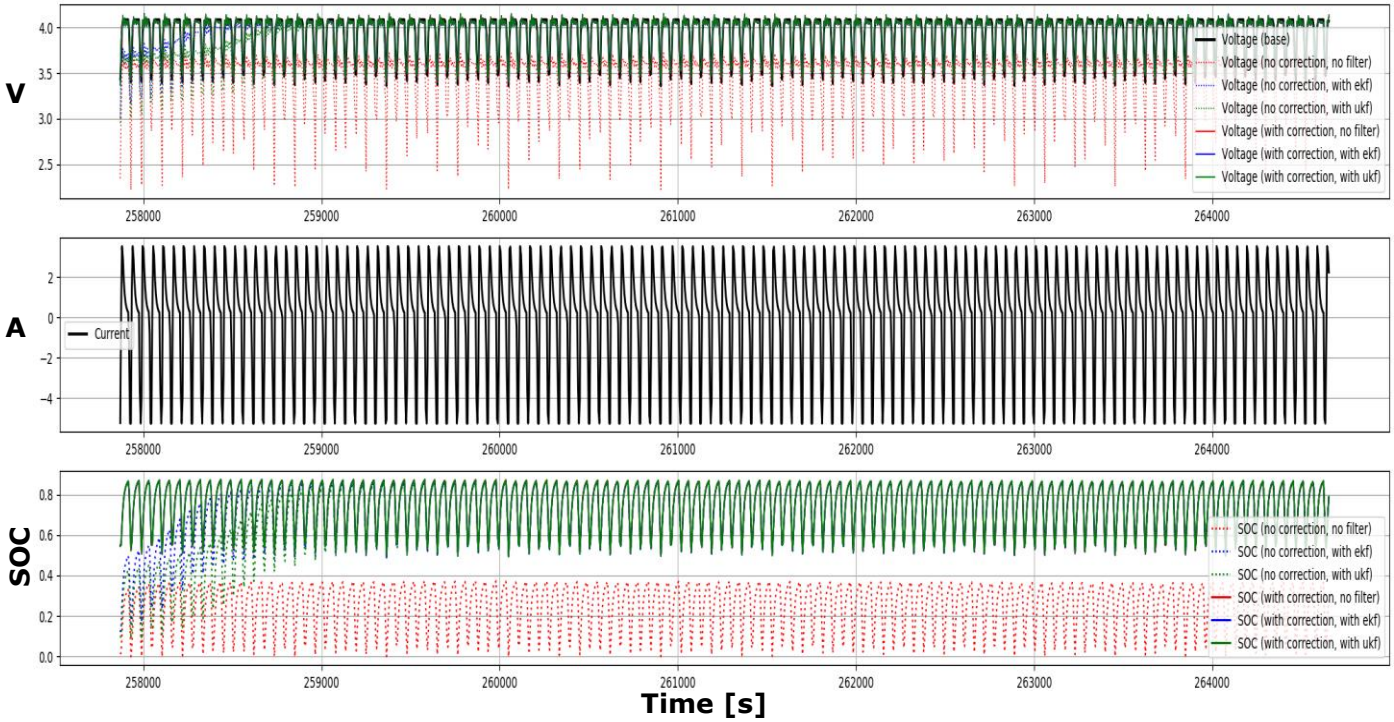

Figure 10: The influence of adjusting the Initial State

After we have estimated the initial state of charge, the next step consists of estimating the resistances and capacities of each subsequence. The ideal reference for this is cyclic ageing tests, where we know there is going to be a steady decline in capacity and a rise in resistance.

In Figure 11, we show the plot of a subsequence of a cyclic ageing test. It can be observed that the initial provided estimate of the SOC is far off (at 5%, instead of the true value is around 70%). The red dotted line shows the evolution by the model without the use of the Kalman filter. The green dotted line shows that the Kalman filter manages to correct the offset in time. Correcting the initial state with the default resistance/capacity already significantly reduces the error made by the model, with and without applying the UKF. Additionally, freeing up the resistance/capacity does so even further. To quantify this example, we refer to Table 1.

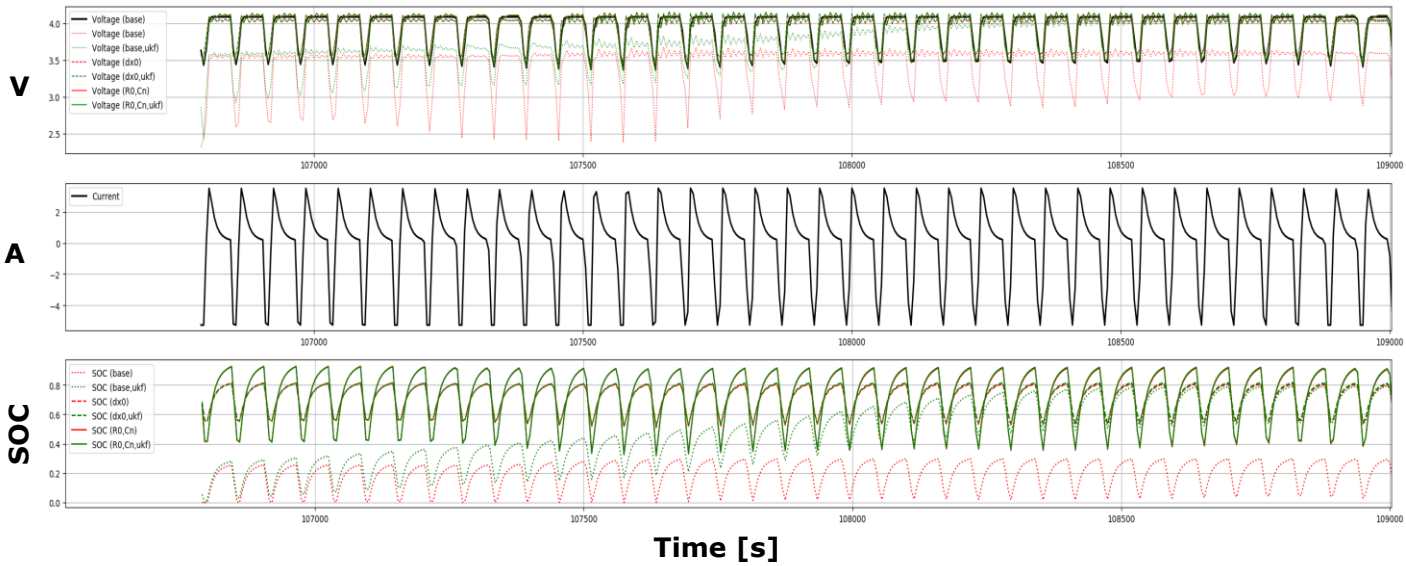

Figure 11: The influence of adjusting the Ohmic Resistance and Total Capacity

Sequence ID	base	dx_0	R_0,C_n	base,ukf	dx_0,ukf	R_0,C_n,ukf
0	0.26488	0.00725	0.00291	0.04253	0.00713	0.00282

Table 1: Parameter estimations for a single-cell battery

The previous subsequence shows only a small subset from a larger cyclic ageing test. We display the evolution of the computed total capacity for the entire test in Figure 12 and Figure 13. It is easy to see a declining trend to be perceived in the estimated capacity and a rising trend in the resistance. Before we go on to the next phase in the estimation process, we fit a simple linear model[4] through this sequence to bring capacities and resistances in sync over a larger dataset. This is a good way to handle this for a training sequence. However, the recorded outliers are once more indicative of just how hard it is to perform a stable, consistent State of Health estimation. Thus, in a live system, it is probably a good idea to combine this technique with a stabilising algorithm to avoid high variance in our model parameters.

The computed error residuals for all subsequences in the cyclic ageing test are listed in Table 2. Even though there are a few subsequence exceptions where the model without capacity corrections performs better without the Kalman filter, adding the UKF back to the simulation clearly shows improvements over the entire line. When applying the Kalman filter, we see a halving in error on average when applying the correction to the initial state (0.01655 to 0.00864) and another reduction by more than half when correcting the capacity (0.00864 to 0.00334).

[4] To be more precise, we use RANSAC [32] to reduce influence from outliers.

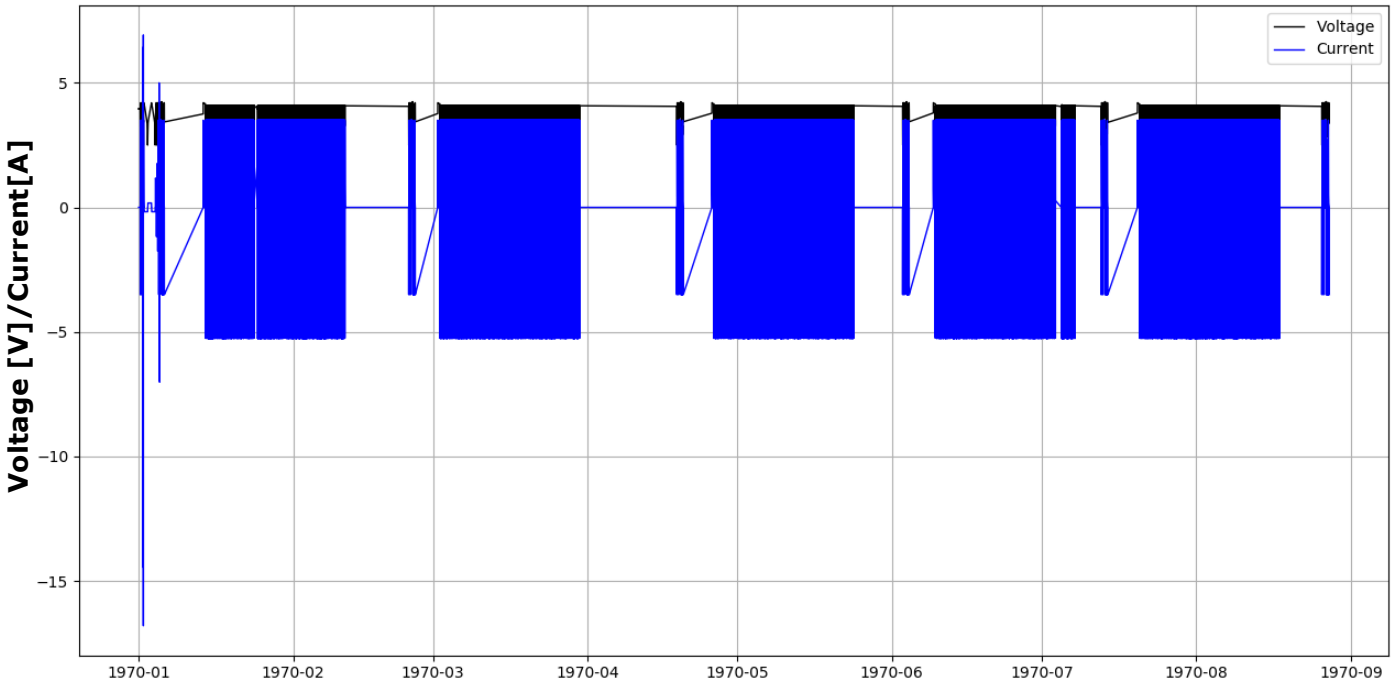

Figure 12: Raw Cycling Ageing Test (single-cell battery)

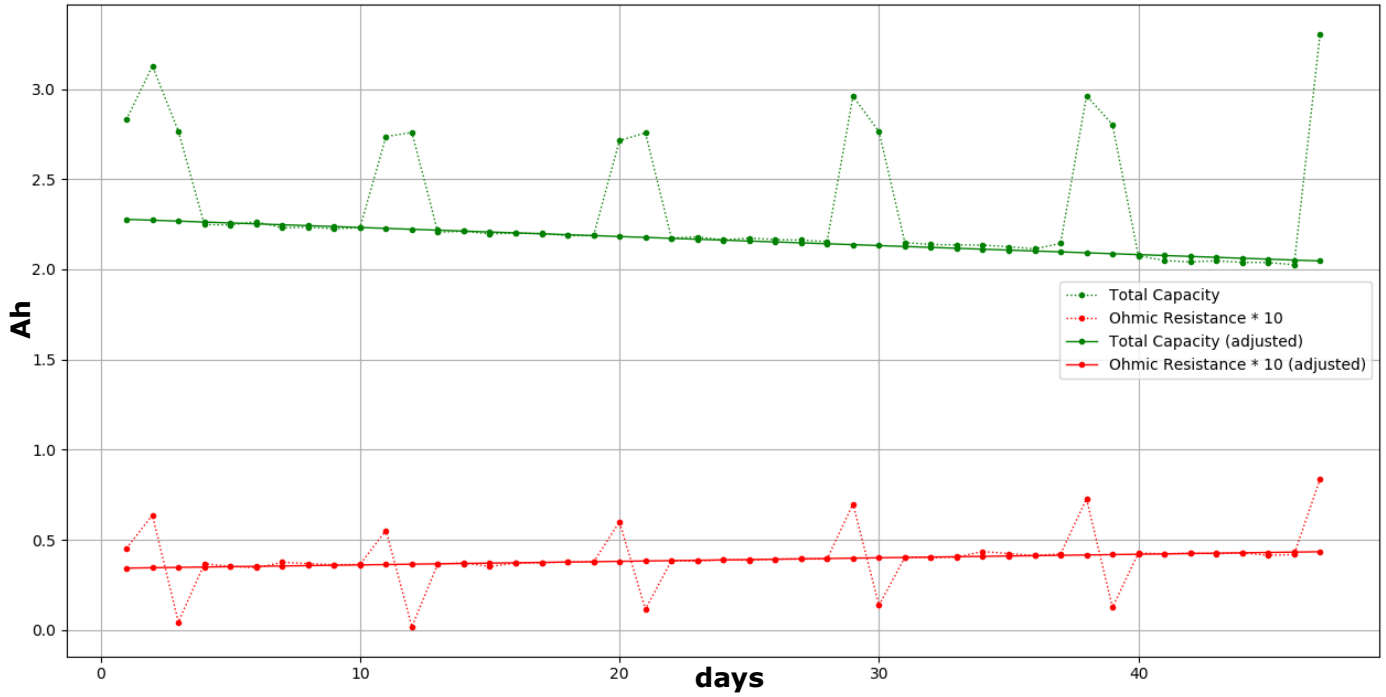

Figure 13: Individual estimated (& adjusted) capacities & resistances for test (single-cell battery)

SubSequence	base	dx_0	R_0,C_n	base,ukf	dx_0,ukf	R_0,C_n,ukf
0	0.02934	0.02062	0.04843	0.00873	0.00823	0.00519
1	0.02629	0.04527	0.06854	0.00806	0.00816	0.00717
2	0.00601	0.00600	0.01336	0.00543	0.00543	0.00295
3	0.08869	0.00637	0.00252	0.00958	0.00631	0.00249
4	0.23951	0.00644	0.00256	0.02609	0.00638	0.00253
5	0.01463	0.00654	0.00262	0.00679	0.00647	0.00257
6	0.23575	0.00664	0.00265	0.02298	0.00658	0.00262
7	0.23795	0.00681	0.00278	0.02935	0.00675	0.00274
8	0.10680	0.00682	0.00274	0.01048	0.00675	0.00270
9	0.15377	0.00686	0.00278	0.01406	0.00680	0.00273
10	0.03632	0.03124	0.03663	0.01623	0.01667	0.00479
11	0.00664	0.00663	0.01442	0.00599	0.00598	0.00326
12	0.01166	0.00701	0.00284	0.00712	0.00692	0.00274
13	0.00817	0.00708	0.00290	0.00703	0.00699	0.00279
14	0.00707	0.00679	0.00252	0.00673	0.00672	0.00248
15	0.26488	0.00725	0.00291	0.04253	0.00713	0.00282
16	0.26223	0.00707	0.00274	0.03482	0.00700	0.00266
17	0.24437	0.00708	0.00273	0.02048	0.00701	0.00266
18	0.02978	0.00708	0.00268	0.00795	0.00699	0.00259
19	0.06182	0.03793	0.04817	0.01712	0.01672	0.00604
20	0.00703	0.00692	0.01479	0.00623	0.00615	0.00308
21	0.01945	0.00697	0.00253	0.00740	0.00689	0.00246
22	0.15153	0.00693	0.00244	0.01347	0.00684	0.00238
23	0.27120	0.00695	0.00240	0.02858	0.00685	0.00231
24	0.06454	0.00693	0.00245	0.00952	0.00684	0.00228
25	0.00949	0.00694	0.00232	0.00695	0.00687	0.00228
26	0.27237	0.00698	0.00237	0.02816	0.00688	0.00226
27	0.10923	0.00701	0.00232	0.01148	0.00694	0.00227
28	0.06872	0.04030	0.05149	0.01949	0.01869	0.00642
29	0.01060	0.00719	0.01457	0.00665	0.00623	0.00295
30	0.28662	0.00715	0.00237	0.03282	0.00705	0.00227
31	0.02869	0.00723	0.00248	0.00792	0.00712	0.00231
32	0.15247	0.00735	0.00243	0.01470	0.00726	0.00235
33	0.07939	0.00747	0.00260	0.01073	0.00733	0.00239
34	0.29242	0.00759	0.00264	0.02648	0.00747	0.00248
35	0.05969	0.00776	0.00284	0.00992	0.00763	0.00259
36	0.01157	0.00759	0.00262	0.00767	0.00749	0.00250
37	0.04316	0.04262	0.05175	0.01997	0.01981	0.00676
38	0.01439	0.00824	0.01468	0.00776	0.00714	0.00339
39	0.32349	0.00856	0.00348	0.02996	0.00843	0.00328
40	0.10358	0.00896	0.00391	0.01354	0.00879	0.00365
41	0.32903	0.00910	0.00403	0.04557	0.00890	0.00368
42	0.18339	0.00932	0.00417	0.01848	0.00913	0.00387
43	0.33057	0.00955	0.00436	0.03473	0.00939	0.00414
44	0.00957	0.00957	0.00456	0.00944	0.00944	0.00422
45	0.04019	0.00968	0.00458	0.01102	0.00957	0.00448
46	0.10385	0.06300	0.04505	0.03154	0.02888	0.00724
Average	**0.11591**	**0.01229**	**0.01114**	**0.01655**	**0.00864**	**0.00334**

Table 2: Error residuals (MSE) associated with a full cyclic ageing test (single-cell battery)

Case study 2: Multi-cells battery system

This section details the experiments carried out on 9 datasets from a 16S1P battery (~60V battery module) described in section 2. We splitted all the sequences included in such a dataset into training sets and test sets with a partition of approximatively 95% and 5%, respectively. We carried out tests for different sample times, namely: 60 seconds, 30 seconds, 10 seconds, 5 seconds, 3 seconds, 1.5 seconds and 0.5 seconds, to study the impact that the dynamics of the variables involved (i.e. current, voltage and SOC) make on the accuracy of the proposed methodologies.

We compared a total of eight methods, as mentioned in Table 3.

	Method name	Description
1	base	Vanilla RC model
2	base_ukf	Vanilla RC model with Unscent Kalman Filter
3	XNN0	Vanilla RC model with initial SOC estimation using $SRNN_{x_0}$
4	XNN0_ ukf	XNN0 method with Unscent Kalman Filter
5	RCNN	XNN0 method with RC model correction using $SRNN_{R_0,C_n}$
6	RCNN_ ukf	RCNN method with Unscent Kalman Filter
7	εNN	RCNN method with f^ϵ and h^ϵ corrections using $SRNN_{x_0}$ and $SRNN_{R_0,C_n}$
8	εNN_ ukf	εNN ith Unscent Kalman Filter

Table 3: List of the compared methods

The methods include two vanilla RC methods (method 1 and 2), i.e. 'base' and "base_ukf". The difference is only in an additional Kalman Filter, the Unscent Kalman Filter. The reason why we use the UKF among others (e.g. Extended Kalman Filter, etc.) reside in the better overall performance of the UKF for complex dynamics. In the confidential D2.4, a more detailed analysis of this choice has been described. Similarly, we have our three proposed methods, namely XNNO, RCNN and εNN with and without an additional UKF.

The goal of our experiments is to compare the proposed methods (listed at the rows 3-8 in Table 3) against the standard methods 'base' and 'base_ukf'.

Implementation Details

All neural networks were implemented in Python, using the PyTorch framework.

Unlike classical ANNs or CNNS, RNNs do not actually gain a speed boost by performing the computations on GPU due to a lack of parallelisation. The main computational bottleneck thus becomes single-core CPU speed.

For the optimisation of all our SRNNs, we use the Rectified Adam optimiser: a drop-in alternative for the de facto industry standard Adam optimiser [31]. All optimiser related parameters mentioned in Table 4 thus refer to RAdam parameters. In addition, we have made use of a stepwise learning rate scheduler to gradually decrease the learning rate as time progresses.

Parameters	lr	betas	stepsize	gamma	valsplit	patience	clipnorm
x_0	0.1	(0.9, 0.999)	50	0.99	N/A	250	0.1
R_0, C_n	0.1	(0.9,0.999)	50	0.99	N/A	250	1.0
f^ϵ, h^ϵ	1e-5	(0.9,0.999)	200	0.99	0.20	2000	1.0

Table 4: Hyperparameter values used by our different SRNNs

Furthermore, as we use a validation set for the estimation of functions f^ϵ and h^ϵ, the validation split is only defined for these parameters. It should be noted that we use complete (sub)sequences as

validation sets and not cut each subsequence into a training and validation part. Unlike the latter scenario, this way, the results produced by the validation set are completely independent of the results produced by the training set.

The parameter *patience* in table 4 refers to the number of training epochs that need to have passed without improvements before we accept the solution. Additionally, we provide the gradient clipping norm that we have employed, as per the suggestion of Pascanu et al. [26].

Results

We use the Mean Squared Error (MSE) as a performance indicator throughout the experiments.

As mentioned in the previous section, the experiments differ among them for using a different sample time. Each test set for each experiment consists of a number of driving sequences. Furthermore, each of those sequences have the same number of samples.

Tables 6-8 show some examples of results from experiments with 10, 5, and 1.5 seconds respectively. Notice that in the experiment related to the sample time of 1.5 s, we have more sequences because, for computational reasons, we have limited the maximum number of samples for a single sequence. This results in a higher number of total sequences for the lower sample time experiments. For space reasons, we omit to report all the tables relative to each experiment and each test set. However, in order to provide a general overview of the results, we have considered all the experiments with the same sample time and averaged the results that emerged from each sequence.

Table 5 reports this global outcome. In such a table, we report for each methodology the average MSE resulting from all test sequences with the same sample time. The performance results for each method (in columns) are reported against the sample time used throughout the entire set of experiments.

Sample Time [s]	base	base_ukf	XNN0	XNN0_ukf	RCNN	RCNN_ukf	εNN	εNN_ukf
60	0,03584	0,00394	0,03094	0,00304	0,02624	0,00216	0,01220	**0,00098**
30	0,03295	0,00235	0,00277	**0,00124**	0,00445	0,00167	0,00368	0,00148
10	0,14602	0,00920	0,00251	**0,00096**	0,02817	0,01651	0,02132	0,05526
5	0,22021	0,01551	0,00857	**0,00040**	0,00708	0,00144	0,00325	0,00159
3	0,11541	0,00858	0,00346	**0,00040**	0,00095	0,00041	0,00094	**0,00040**
1,5	0,12450	0,00989	0,00064	**0,00043**	0,00646	0,00167	0,00604	0,00160
0,5	0,04222	0,00349	0,00118	**0,00010**	0,00388	0,00305	0,01998	0,00882

Table 5: Average MSE results from all test sequences (best results in bold)

A first macroscopic result is that the Kalman filter version of each method improves the corresponding method's performance without the filter in almost all the cases.

Furthermore, comparing all the Kalman filter version methodologies among them, the proposed approaches, namely *XNN0_ukf, RCNN_ukf* and the *εNN_ukf* improve the performance of the vanilla method aimed with the Kalman filter in the vast majority of the cases. The best results overall are achieved by the RC model with the SOC initial estimation using the proposed $SRNN_{x_0}$ (i.e. the XNN0_ukf approach). For instance, on the 60 seconds sample time experiment set, the proposed XNN0_ukf improves the vanilla aided Kalman filter method (i.e. base_ukf) by a 22% considering the MSE performance indicator. This improvement raises to a 47% when using 30 seconds sample time, 89% when using 10 seconds sample time up to a maximum of 97% for the 0.5 seconds sample time.

The other two proposed methods, the RCNN_ukf and εNN_ukf, while achieving a generally worse performance compared to the XNN0_ukf, they still improve the base vanilla method with the Kalman

filter. In particular, regarding the RCNN_ukf, it improves the base_ukf method by a 45% for the 60 seconds sample time experiments up to a maximum of 90% in the 5 seconds sample time experiments, with a slight performance degradation for shorter sample time experiments, where the improvement is only of about 12% for the 0.5 seconds case study. Similarly, the third proposed method, the εNN_ukf which includes the correction of the structural terms using both the $SRNN_{x_0}$ and $SRNN_{R_0,C_n}$ described in the section 4.4, improves the vanilla method with Kalman filter by a 75% for the 60 seconds up to a 95% for the 3 seconds sample time experiment set (same results as for XNN0). Then, there is a performance degradation for shorter sample times, where for the 0.5 seconds sample time case, it worsens the vanilla method with the Kalman filter by a factor of 52%.

	base	base_ukf	XNN0	XNN0_ukf	RCNN	RCNN_ukf	εNN	εNN_ukf
0	0,16968	0,01275	0,00511	0,00211	0,00085	0,00064	0,00050	**0,00045**
1	0,01404	0,00087	0,00077	0,00038	0,00114	0,00037	0,00039	**0,00025**
2	0,02314	0,00335	0,00065	0,00053	0,00125	0,00089	0,00061	**0,00051**
3	**0,00165**	0,00166	0,00403	0,00165	0,03596	0,00672	0,02113	0,00458
4	0,00116	0,00034	0,00049	0,00032	0,00048	**0,00022**	0,00101	0,00030
5	0,00146	0,00119	0,00163	0,00113	0,00122	0,00042	0,00014	**0,00013**
6	0,02389	0,00206	0,00206	0,00202	0,00173	0,00152	0,00158	**0,00109**
7	0,05802	0,00160	0,00594	0,00097	0,00307	0,00087	0,00086	**0,00081**
8	0,00605	0,00103	0,00551	0,00102	0,00152	**0,00065**	0,00507	0,00084
9	0,02340	0,00142	0,00166	**0,00141**	0,00230	0,00230	0,00272	0,00225
10	0,03130	0,00194	0,00199	**0,00189**	0,00228	0,00224	0,00401	0,00277
11	0,00096	0,00079	0,00083	**0,00079**	0,00263	0,00228	0,00180	0,00184
12	0,01336	0,00233	0,00697	**0,00230**	0,00600	0,00259	0,01023	0,00377
13	0,10262	0,00302	0,00311	**0,00139**	0,00406	0,00166	0,00446	0,00205
14	0,02357	0,00093	0,00076	0,00070	0,00226	0,00175	0,00073	**0,00058**

Table 6: MSE resulting from test sequences with 30 seconds sample time (best results in bold)

	base	base_ukf	XNN0	XNN0_ukf	RCNN	RCNN_ukf	εNN	εNN_ukf
0	0,02813	0,00357	0,00034	0,00033	0,00018	**0,00017**	0,00096	0,00069
1	0,01107	0,00073	0,00056	0,00045	0,00018	**0,00010**	0,00073	0,62370
2	0,00092	0,00044	0,00039	0,00042	0,00017	**0,00014**	0,00044	0,00015
3	0,20591	0,02817	0,00400	**0,00095**	0,00816	0,00306	0,00292	0,00187
4	0,23037	0,00386	0,00084	**0,00027**	0,00428	0,00089	0,00096	0,00029
5	0,00478	0,00230	0,00361	0,00230	0,01892	0,00132	0,00130	**0,00122**
6	0,07487	0,00843	0,00307	0,00067	0,01751	0,00234	0,00073	**0,00019**
7	0,00845	0,00114	0,00116	0,00048	0,00031	0,00014	0,00011	**0,00007**
8	0,00617	0,00048	0,00046	0,00034	0,00065	0,00069	0,00030	**0,00026**
9	0,04127	0,00260	0,00327	0,00253	0,00145	**0,00134**	0,00402	0,00320
10	0,33039	0,01109	0,01208	**0,00152**	0,07157	0,00168	0,00281	0,00329
11	0,00374	0,00150	0,00169	0,00145	**0,00110**	0,00121	0,01950	0,01619
12	0,23009	0,01274	0,00527	**0,00134**	0,29848	0,22670	0,28232	0,21665
13	0,01843	0,00156	**0,00127**	0,00133	0,02728	0,02414	0,01634	0,01483
14	1,12531	0,06694	0,00169	0,00075	0,00043	0,00025	0,00031	**0,00007**
15	0,01646	0,00170	0,00046	0,00027	0,00009	**0,00007**	0,00743	0,00141
16	0,02813	0,00357	0,00034	0,00033	0,00018	**0,00017**	0,00096	0,00069

Table 7: MSE resulting from test sequences with 10 seconds sample time (best results in bold)

	base	base_ukf	XNN0	XNN0_ukf	RCNN	RCNN_ukf	εNN	εNN_ukf
0	0,00118	0,00104	0,28283	0,02233	0,00005	**0,00003**	0,00039	0,00011
1	0,00127	0,00116	0,10220	0,00941	0,00317	**0,00014**	0,00160	0,00016
2	0,00135	0,00147	0,13144	0,01292	0,00381	**0,00013**	0,00146	0,00015
3	0,00171	0,00195	0,05743	0,00432	0,00015	**0,00005**	0,00162	0,00016
4	0,00204	0,00259	0,05308	0,00424	0,00267	**0,00013**	0,00298	0,00022
5	0,00296	0,00363	0,06712	0,00698	0,00374	**0,00016**	0,00237	0,00018
6	0,00367	0,00492	0,05082	0,00520	0,00028	**0,00005**	0,00298	0,00020
7	0,00497	0,00626	0,04080	0,00432	0,00193	**0,00010**	0,00307	0,00020
8	0,00855	0,00753	0,02357	0,00230	0,00192	**0,00009**	0,00128	0,00012
9	0,01027	0,00845	0,02181	0,00258	0,00016	**0,00003**	0,00118	0,00019
10	0,01361	0,00935	0,02237	0,00314	0,00043	**0,00005**	0,00115	0,00028
11	0,02325	0,01143	0,01380	0,00204	0,00068	**0,00006**	0,00071	0,00021
12	0,00255	0,00158	0,00067	0,00013	0,00011	**0,00004**	0,01671	0,01072
13	0,00145	0,00131	0,00313	0,00041	0,00139	**0,00022**	0,02125	0,01315
14	0,00135	0,00123	0,00749	0,00061	0,00135	**0,00027**	0,02431	0,01440
15	0,00150	0,00123	0,00360	0,00030	0,00007	**0,00003**	0,02568	0,01454
16	0,00124	0,00119	0,00390	0,00026	0,00189	**0,00016**	0,03156	0,01681
17	0,00153	0,00126	0,02792	0,00083	0,00209	**0,00032**	0,03221	0,01580
18	0,00141	0,00118	0,00736	0,00032	0,00010	**0,00003**	0,03622	0,01731
19	0,00128	0,00104	0,00832	0,00029	0,00071	**0,00006**	0,04316	0,01962
20	0,00170	0,00082	0,00831	0,00022	0,00091	**0,00008**	0,04483	0,02061
21	0,00111	0,00064	0,01708	0,00020	0,00012	**0,00003**	0,05329	0,01767
22	0,00111	0,00073	0,02508	0,00017	0,00022	**0,00004**	0,06319	0,02006
23	0,00202	0,00124	0,03325	0,00017	0,00038	**0,00004**	0,06628	0,02887
24	0,00118	0,00104	0,28283	0,02233	0,00005	**0,00003**	0,00039	0,00011

Table 8: MSE resulting from test sequences with 10 seconds sample time (best results in bold)

Limitations & Future Work

The results we have shown in this section prove that the proposed methodologies achieve better performances in the state of charge estimation if compared with the state-of-art methodologies.

However, despite the good results achieved, there is a deterioration in the performance of the structured error correction in the case of shorter sample times (below 1.5 seconds). We think that the reason behind this behaviour is related to the associated python implementation. Indeed, the estimation of $SRRN_{x_0}$ and $SRRN_{C_n,R_0}$ is currently parallelised in the Python implementation and will run efficiently even for very large amounts of datasets. But a truly scalable estimation of $SRRN_{f^\epsilon,h^\epsilon}$ requires efficient multi-threading instead of multi-processing, as we need shared memory for its operations.

This is currently not possible in Python due to the Global Interpreter Lock (GIL), and even multi-processing already carries a significant overhead with it. Technically, a C++ implementation should be able to resolve these issues, though it requires significant development efforts that were beyond the scope of this research.

Another interesting direction to investigate consists of blending the Kalman filter with the SRNNs. We could do this by internalising the Kalman filter into the actual optimisation process (when performing backpropagation), or the other way around: replace the Kalman parameter (not state) update step by an estimate made by our SRNN with a limited backward horizon.

CONCLUSIONS

In this work, we have provided two novel contributions to the field of SOC/SOH research:

(1) We have introduced the notion of Structured Recurrent Neural Networks (SRNN), in which we leverage the idea of a structured neural network into the realm of indirect SOC/SOH estimation techniques. We have shown that they can be used as a suitable alternative (and/or complement) to methods such as dual Kalman filters commonly used in literature to estimate the time-varying battery parameters in SOH estimation. Unlike the traditional approaches, which only use the current state to update the SOH parameters, we use the entire recent history to make this estimate.

(2) We introduce a very general, universally applicable method to augment any base model – such as an ECM, in which each parameter has a physical meaning and is interpretable – with a black-box extension that is able to capture some of the structured leftover unexplained behaviour that was not captured by the physical model.

Both contributions have been explained theoretically and verified experimentally, based on datasets acquired in the lab.

REFERENCES

[1] C. Lin, H. Mu, R. Xiong, and J. Cao. Multi-model probabilities based state fusion estimation method of lithium-ion battery for electric vehicles: State-of-energy. *Applied Energy,* 194:560-568, 2017.
[2] Y. Zheng, M. Ouyang, L. Lu, and J. Li. Understanding ageing mechanisms in lithium-ion battery packs: From cell capacity loss to pack capacity evolution. *Journal of Power Sources*, 278:287-295, 2015.
[3] G.L. Plett. Extended kalman filtering for battery management systems of lipb-based hev battery packs: Part 2. modeling and identification. *Journal of Power Sources,* 134(2):262-276, 2004.
[4] C.P. Zhang, J.C. Jiang, and L. Zhang. A generalized soc-ocv model for lithium-ion batteries and the soc estimation for lnmco battery. *Energies*, 9, 2016.
[5] M. Chen and G. on Mora. Accurate electrical battery model capable of predicting runtime and i-v performance. *IEEE Transactions on Energy Conversion - IEEE TRANS ENERGY CONVERS*, 21:504-511, 06 2006.
[6] Y. Hu, S. Yurkovich, Y. Guezennec, and B.J. Yurkovich. Electro-thermal battery model identification for automotive applications. *Journal of Power Sources*, 196(1):449-457, 2011.
[7] C.Weng, J. Sun, and H. Peng. A unified open-circuit-voltage model of lithium-ion batteries for State of Charge estimation and State of Health monitoring. *Journal of Power Sources*, 258:228-237, 2014.
[8] R. Zhang, B. Xia, B. Li, L.Cao, Y. Lai, W. Zheng, H. Wang, W. Wang, and M. Wang. A study on the open circuit voltage and state of charge characterization of high capacity lithium-ion battery under different temperature. *Energies*, 11:2408, 09 2018.
[9] G. Dong, J. Wei, C. Zhang, and Z. Chen. Online state of charge estimation and open circuit voltage hysteresis modeling of lifepo4 battery using invariant imbedding method. *Applied Energy*, 162:163-171, 2016.
[10] R. Xiong, F. Sun, Z. Chen, and H. He. A data-driven multi-scale extended kalman filtering based parameter and state estimation approach of lithium-ion polymer battery in electric vehicles. *Applied Energy*, 113:463-476, 2014.
[11] R. Xiong, F. Sun, H. He, and T.D. Nguyen. A data-driven adaptive state of charge and power capability joint estimator of lithium-ion polymer battery used in electric vehicles. *Energy,* 63:295-308, 2013.
[12] Z. Liu, X. Dang, and H.Sun. Online state of charge estimation for lithium-ion battery by combining incremental autoregressive and moving average modeling with adaptive h-infinity filter. *Mathematical Problems in Engineering*, 2018:1-16, 07 2018.
[13] J.C.A. Anton, P.J.G. Nieto, C. Blanco, and J. Vilan. Support vector machines used to estimate the battery state of charge. *Power Electronics, IEEE Transactions on*, 28:5919-5926, 12 2013.
[14] J.C.A. Anton, P.J.G. Nieto, F.J. de Cos Juez, F.S. Lasheras, M.G. Vega, and M.N.R. Gutierrez. Battery State of Charge estimator using the svm technique. *Applied Mathematical Modelling*, 37(9):6244 - 6253, 2013.
[15] T. Hansen and C. Wang. Support vector based battery state of charge estimator. *Journal of Power Sources*, 141(2):351-358, 2005.
[16] C. Chen, R. Xiong,R. Yang, W. Shen, and F. Sun. State of Charge estimation of lithium-ion battery using an improved neural network model and extended kalman filter. *Journal of Cleaner Production*, 234:1153-1164, 2019.
[17] W. He, N. Williard, C. Chen, and M. Pecht. State of charge estimation for li-ion batteries using neural network modeling and unscented kalman filter-based error cancellation. *International Journal of Electrical Power & Energy Systems*, 62:783-791, 2014.
[18] D. Andre, A. Nuhic, T. Soczka-Guth, and D. Sauer. Comparative study of a structured neural network and an extended kalman Filter for state of health determination of lithium-ion

batteries in hybrid electric vehicles. *Engineering Applications of Artificial Intelligence*, 26:951-961, 03 2013.
[19] K.S. Ng, C.S. Moo, Y.P. Chen, and Y.C. Hsieh. Enhanced coulomb counting method for estimating State of Charge and State of Health of lithium-ion batteries. *Applied Energy,* 86(9):1506-1511, 2009.
[20] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35-45, 03 1960.
[21] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the kalman filter to nonlinear systems. pages 182-193, 1997.
[22] Eric A. Wan and Alex T. Nelson. Dual kalman filtering methods for nonlinear prediction, smoothing, and estimation. In *Advances in Neural Information Processing Systems 9*, 1997.
[23] G. Welch and G. Bishop. An introduction to the kalman filter, 1995.
[24] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *In A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.
[25] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*. 9 (8): 1735–1780, 1997
[26] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *30th International Conference on Machine Learning, ICML 2013*, 11, 2012.
[27] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.
[28] V. Nair and G. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of ICML*. 27. 807-814, 2010
[29] P. Ramachandran, B. Zoph, and Q.V. Le. Searching for activation functions. 2017
[30] Diganta Misra. Mish: A self regularized non-monotonic neural activation function, 2019.
[31] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. On the variance of the adaptive learning rate and beyond, 2019.
[32] M.A. Fischler and R.C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Comm. ACM.* 24 (6): 381–395, 6, 1981