Developing a Model-in-the-Loop Testbench for Battery Management Systems: Advancing Test Methodologies for State-of-X Estimation

I. Sanz-Gorrachategui, A. Barrutia, A. Martín, X. Arraztoa-Lazkanotegi, D. Marcos, P. Onaindia

Developing a Model-in-the-Loop Testbench for Battery Management Systems: Advancing Test Methodologies for State-of-X Estimation

I. Sanz-Gorrachategui, A. Barrutia, A. Martín, X. Arraztoa-Lazkanotegi, D. Marcos, P. Onaindia
IKERLAN
José María Arizmendiarrieta Pasealekua, 2, 20500
Arrasate-Mondragón, Spain
943 71 24 00
{isanz, abarrutia, amartin, xarraztoa, dmarcos, ponaindia}@ikerlan.es

Acknowledgements

The research presented throughout this paper has received funding from the European Community under the FASTEST project (grant agreement N° 101103755)

Keywords

«Battery Management Systems (BMS)»,
«Hardware-in-the-Loop (HIL)»,
«State of Charge»,
«Virtual prototyping»,
«Real-time simulation».

Abstract

Lithium-ion battery systems require robust BMS for safety and reliability. Extensive BMS testing presents challenges like safety risks, long testing times, and variability. To address these issues and expedite development, a testing methodology based on X-in-the-Loop (XiL) paradigms is proposed. This methodology ensures comprehensive validation and cost reduction. Using MathWorks Simulink and Simscape Batteries, a virtual testbench is developed to model both the BMS and the battery pack. The BMS includes virtualized master and slave units, emulating sensor behavior and SoC estimation algorithms. A virtual battery cycler enables specific current profile testing. As case study, this setup benchmarks SoC estimation algorithms at MiL and SiL levels, with potential HiL expansion for real BMS device validation, enhancing testing capabilities and automating procedures for safer, faster BMS development.

Introduction

Lithium-ion battery systems have become ubiquitous in modern life, with applications ranging from small-scale electronic devices to mid-scale automotive systems and large-scale stationary applications. To ensure the safe operation of these battery packs, Battery Management Systems (BMS) are essential. The primary functions of BMS include cell monitoring (current, voltage, and temperature), cell balancing, State-of-X estimation (SoX), charge and discharge control, and alarm triggering [1].

Due to their crucial role in the safety and reliability of battery packs, extensive testing of BMS is required to prevent operational failures. Existing methods for assessing BMS functionality significantly contribute to the time-to-market, given the cost of testing infrastructure, the number of required tests, and the duration of each test [2]. Furthermore, directly testing BMS on real battery packs presents issues such as safety concerns—since the BMS is still a prototype and may malfunction—long testing times—due to the need for charging and discharging real batteries—and repeatability issues, as the cell state may vary from test to test. The V-model design methodology (Fig. 1) offers a systematic approach to overcome these challenges. This methodology integrates testing throughout the development process, associating each phase of design with a corresponding phase of testing.

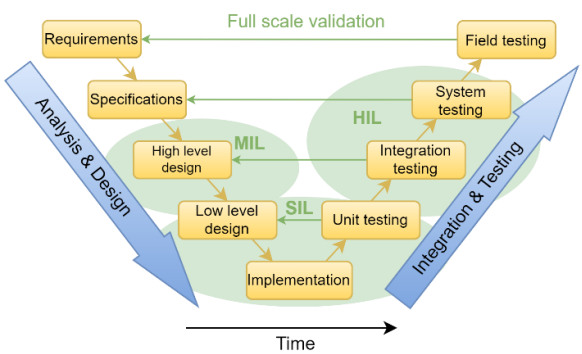

Fig. 1. V model design methodology.

By following the V-model, all requirements are verified and validated at each stage, ensuring thorough validation of the design and reducing the cost and impact of potential changes during the hardware validation phase. This methodology can be effectively applied to BMS development [3 – 4]. After the requirements and specifications, a high level, fully virtual model containing the BMS and the battery pack (Model-in-the-Loop or MiL) is developed. MiL testing allows for the creation and simulation of virtual models representing different system components, such as the whole battery pack, external inputs, and the BMS main algorithms. By simulating the whole system at this stage, it is possible to address early flaws in the design before building hardware prototypes, reducing the risk of expensive errors and delays further into the project.

From this starting point, different virtual blocks can be progressively replaced with real software implementations [5] (Software-in-the-Loop or SiL). This allows for more detailed testing of software algorithms and control strategies in a controlled environment. SiL testing enables developers to check the system behavior under multiple but controlled conditions, allowing for algorithm fine tuning and ensuring their performance before deploying them into hardware prototypes.

Finally, validation plans can be executed using the real BMS alongside a virtual battery pack in a Hardware-in-the-Loop (HiL) setup. This approach combines virtual models with physical components (e.g. testing the actual BMS but emulating the battery and current sensor in the HiL system). This approach creates a realistic and repeatable testing environment where the actual BMS interacts with virtual batteries in real time. HiL testing allows developers to evaluate system performance under consistent operating conditions, including factors such as electrical noise or thermal effects. By validating system behavior in a HiL setup, developers can ensure the BMS operation in real-world applications [6 – 7]. Achieving this requires the design and implementation of dedicated HiL systems [8 – 10], capable of replicating the behavior of virtual cell models and generating the electrical signals for the BMS. Alternatively, off-the-shelf solutions from manufacturers provide seamless integration with simulation environments such as Simulink and can be employed to streamline this process.

This step-by-step validation approach offers capabilities beyond physical testbenches, such as integrating auto-generated code into the BMS or performing test coverage analysis of the virtual model. Coverage is critical for functions such as alarm triggering, where a malfunction could make the system hazardous. For SoX estimation algorithms, it supports comparative analysis among different algorithm families. Furthermore, when coupled with code-generation tools, the development cycle of such elements is accelerated.

As new applications and chemistries emerge, BMS must be designed with shorter development times to reduce time-to-market. This increases the need for automating testing procedures in a fast but reliable manner. To this end, the FASTEST project aims to accelerate battery system R&D and enable more reliable, safer, and longer-lasting battery system designs by developing a fast-track testing platform. This platform uses a strategy based on Design of Experiments (DoE), combining physical and virtual testing. Virtual testing is crucial for ensuring that safety and reliability requirements are met according to various standards (such as ISO 26262 and IEC 61508). Even when standards require physical tests, virtualizing these tests allows for prior Digital Twin (DT) validation, significantly reducing time and material costs and minimizing potential hazards during physical testing.

This paper presents a method for developing a BMS testing testbench using tools such as MathWorks Simulink and Simscape Batteries. As a case study, the testbench is used for benchmarking State-of-Charge (SoC) estimation algorithms at MiL and SiL levels of implementation. The simulation testbench can be easily expanded towards HiL stages via integration with Speedgoat HiL, allowing for the validation of real BMS devices under desired test conditions.

Testbench Description

Testbench Structure

The virtual testbench comprises of two main components: the BMS and the battery pack. The BMS has been virtualized following the architecture described in the subsequent section. It includes the BMS slaves, connected to the battery pack, and the BMS master that coordinates them. On the other hand, the battery pack is modeled after the real battery pack required for the application and remains consistent throughout the MiL – SiL – HiL.

Battery Management System

General BMS architecture

A BMS typically consists of a central unit known as the master and multiple distributed units referred to as slaves (see Fig. 2) [2]. The slaves are directly connected to individual cells and are responsible for tasks such as measuring cell voltage and temperature, as well as balancing the cells. The number of slaves required varies based on the scale of the battery pack, as each slave is connected to a limited number of cells, typically fewer than 20. Conversely, the master unit maintains continuous communication with the slaves, receiving measurements (usually via an isolated daisy-chain protocol) monitoring battery pack current, conducting SoX state estimations, checking cell balance, and triggering warning and alarm systems as necessary.

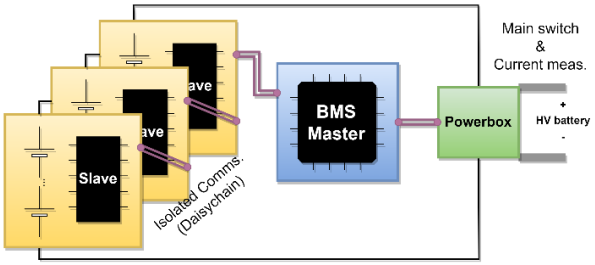

Fig. 2. BMS architecture.

Slave implementation

Each BMS slave monitors cell voltage and temperature. Additionally, the current is shared among all the cells and is provided to the BMS master by an external sensor in the powerbox. The implementation of the BMS Slave is shown in Fig. 3. To emulate the behavior of physical sensors, noise and bias has been added to these measurements, as shown in Table I. At this stage, noise and bias are the only non-ideal factors considered. Future phases of the validation process such as HiL emulation could include communication or measurement errors to emulate other fault conditions.

Table I. Non-ideal sensor magnitudes

Sensor	Bias [A]	AWGN [W]
Current	0.5	3e-7
Voltage	-	3e-3
Temperature	-	1e-2

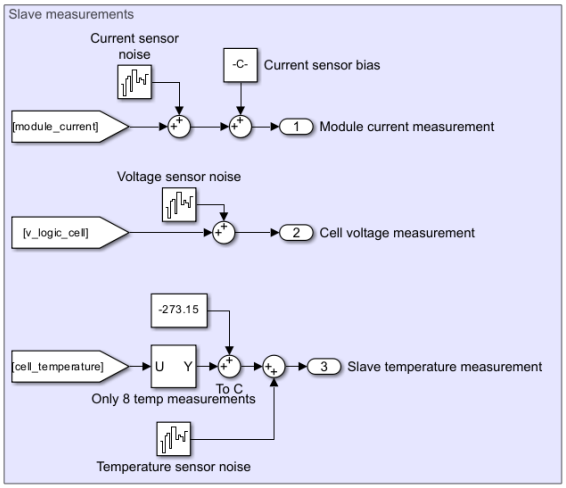

Fig. 3. BMS Slave implementation.

Master implementation

For the purposes of this testbench, a very simplified version of a BMS master is implemented. Its main functions are to receive the measurements from the slaves and run different SoC estimation algorithms, to measure their performance and benchmark them. The master implementation is shown in Fig. 4. The SoC estimation algorithms will be explained in detail in subsequent sections.

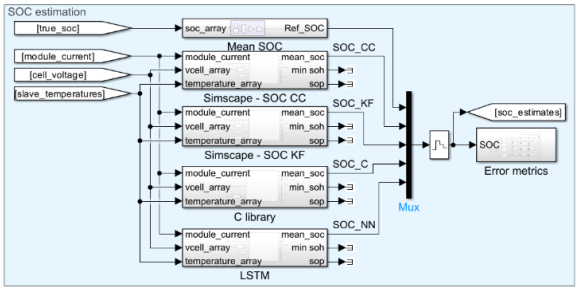

Fig. 4. Master implementation (SoC algorithms).

Battery Pack

Cell electrical model

To generate a model of the battery pack for MiL simulation purposes, the first step is to model cells and group them into a virtual battery. One common way for modelling cells is using Equivalent Circuit Models (ECM) [11]. These models are Thevenin models, consisting of a controlled voltage source and an output impedance model. The voltage of the source, also known as Open Circuit Voltage (OCV), depends on the SoC, SoH and temperature of the cell. The output impedance typically includes a series resistance to model conductivity losses along with a RC network to model diffusion effects,

known as One-Time-Constant (OTC). This basic model can be enhanced by adding more RC networks, or by considering asymmetrical impedances for charge and discharge, which is considered in this study and is shown in Fig. 5. The values of these impedance models usually depend on temperature and the SoC of the cell and may be obtained via manufacturer data or specific laboratory procedures.

Depending on the validation plans, more complex models incorporating features such as aging or fault mechanisms may be considered. At this stage, however, our focus is on developing a functional model that captures the key dynamics necessary for SoC comparison. As such, the simpler model has been deemed sufficient.

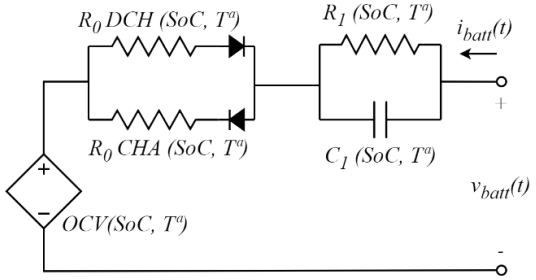

Fig. 5. OTC model with asymmetrical R_0.

Cell under test

The battery pack modeled in this application is based on an LTO cell, characterized using laboratory procedures. It has been modeled using the asymmetrical R_0 OTC model in Fig. 5 (b). The characterized OCV curves for the cell under test are shown in Fig. 6, and the impedance and time constant values for different operation ranges are shown in Fig. 7.

Cell assembly

To ease the modeling and integration process, Simscape Batteries [12] has been used. The tool is integrated in Matlab / Simulink, and allows for physical (electrical, mechanical and thermal) modelling of battery packs, from single cells up to battery modules with hundreds of cells.

The cell previously described can be parametrized with the cell builder app. With this cell as core, the battery pack can be modeled. In this case, the pack under tests consists of two sub-modules connected in series, each with an architecture 11s2p. This means that pairs of two cells are connected in parallel to form a parallel assembly, or "logic cell", and then 11 of these assemblies are connected in series to form a submodule. Each of them is connected to a single BMS slave, which will monitor the logic cells separately (the individual cells connected in parallel within a logic cell cannot be monitored independently). Consequently, the complete battery pack has an architecture of 22s2p, with of two BMS slaves connected to a BMS master.

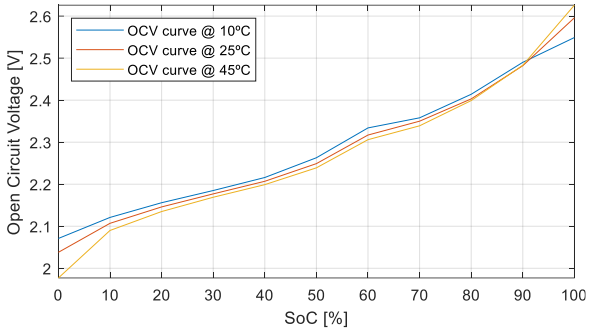

Fig. 6. Open Circuit Voltage for different temperatures and SoC.

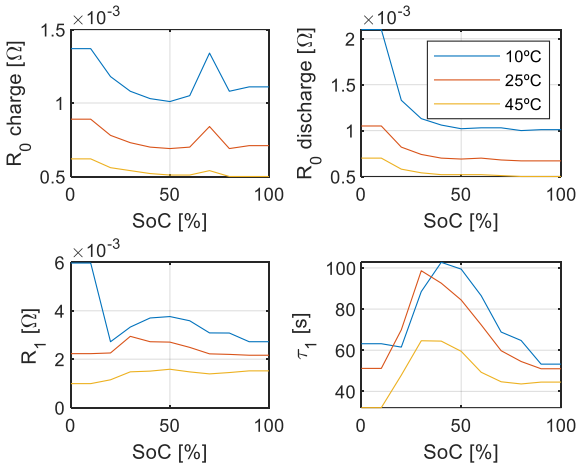

Fig. 7. Impedance model for different temperatures and SoC.

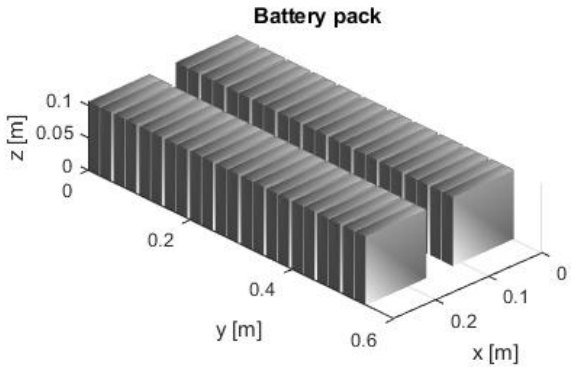

Fig. 8. Module assembly.

Battery Cycler

To test the battery pack with specific current profiles for different validation programs, a virtual battery cycler has been implemented. This virtual cycler is based on two main blocks:

- The battery cycler block integrated in Simscape Batteries, which generates physical magnitudes (voltage / current).
- A state machine that manages the cycler. It reads the test program that implements the test protocol to be followed (which varies according to the validation plan), and controls the inputs to the battery cycler block, which synthesizes the current and voltage towards the battery pack.

Algorithm Benchmarking

State of Charge algorithms

The described simulation platform has been designed as testbench to compare SoX algorithms embedded in the BMS master. As a case study, SoC estimation algorithms are going to be compared at different levels of implementation (MiL or SiL). To this end, the following SoC algorithms have been implemented:

- Coulomb Counting (MiL)
- Coulomb Counting with OCV recalibration (SiL)
- Extended Kalman Filter (MiL)
- Long Short-Term Memory network (MiL)

The basics of the selected algorithms are described in the subsequent subsections.

Coulomb Counting

The Coulomb Counting (CC) method is a straightforward approach to SoC estimation and is widely used due to its simplicity. It relies on integrating the cell's current over time and comparing it to the nominal capacity of the cell. The SoC at any given time (t) is calculated using the following formula:

$$SoC(t) = SoC(0) + \frac{1}{C_{nom}} \int_0^t i_{batt} \cdot dt \qquad (1)$$

Where $SoC(0)$ represents the initial State-of-Charge, i_{batt} denotes the battery current, and C_{nom} is the nominal capacity of the cell.

In ideal conditions where the initial SoC estimation is known and cell capacity remains constant, the CC estimation would be perfect. However, real-world scenarios introduce complexities such as variations in capacity over time or due to temperature, or drift in current measurements, leading to increased errors in the estimation.

In the testbench, this algorithm has been implemented at a MiL level, from the Simscape Batteries libraries.

Coulomb Counting with OCV recalibration

To address the challenges of simple CC-based SoC estimation, modifications over the conventional CC algorithm may be proposed. A first step towards improving the CC algorithm is adding a recalibration process. This algorithm functions as the conventional CC algorithm when there is current flowing through the battery. In resting periods (when the current is zero or close to zero for a long time, depending on the chemistry), cell voltage is assumed to be equal or close to the OCV, and $SoC(t)$ is adjusted accordingly.

This keeps the estimation from diverging due to current drifts or capacity changes but is dependent on the application having sufficiently long pauses, which may never be the case for certain applications such as grid regulation systems.

In the testbench, this algorithm has been implemented at a SiL level, via a Simulink S-Function calling a precompiled C library.

Extended Kalman Filter

Kalman filters are a family of general-purpose estimation algorithms commonly applied to SoC estimation [11]. These filters operate by utilizing a cell model, such as an ECM, as their core.

During each iteration, the Kalman filter performs two key steps. Firstly, it predicts the SoC based on the current state, the cell model, and the input parameters. This prediction provides an initial estimate of the SoC.

Secondly, the Kalman filter corrects this prediction using real-time measurements, typically voltage inputs, to update the SoC estimate. It also adjusts the uncertainty associated with both the model and the input parameters. This correction mechanism ensures that the estimate remains accurate and prevents divergence or the occurrence of unreasonable values.

One of the main challenges of KF-based estimation is developing an accurate model of the cell, as well as dealing with its non-linearities. For this reason, algorithms such as Extended Kalman Filter (EKF) or Unscented Kalman Filter (UKF) are proposed.

In the testbench, this option has been implemented at a MiL level, by using the EKF module provided by the Simscape Batteries toolbox.

Long Short-Term Memory network

Long Short-Term Memory (LSTM) networks represent a class of recurrent neural networks widely utilized for sequence prediction tasks [13], including SoC estimation in BMS applications [14]. LSTM networks can capture long-term dependencies and temporal patterns in time-series data. In the context of SoC estimation, LSTM networks leverage historical voltage, current, and temperature data to predict future SoC values.
During training, LSTM networks (see Fig. 9) adjust their internal layers to adaptively update their internal states based on input sequences, enabling them to effectively model the dynamic behavior of battery systems. This adaptability allows LSTM networks to account for complex nonlinearities and uncertainties inherent in battery operation. By exploiting the temporal dynamics of battery behavior, LSTM networks have shown accurate and robust SoC estimates [14].

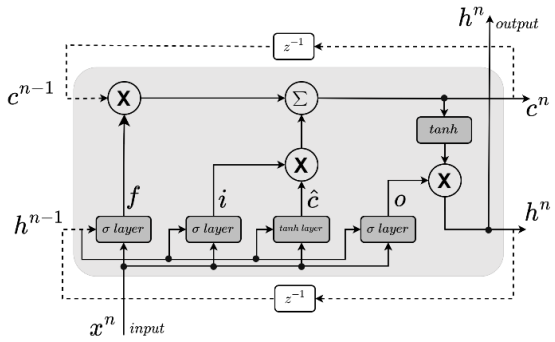

Fig. 9. LSTM cell scheme.

This algorithm has been trained outside the Simulink environment with a small dataset from laboratory tests and imported to in the testbench via an Open Neural Network eXchange (ONNX) model and the Deep-Learning toolbox.

Validation plan

To compare algorithm performance, different current waveforms and ambient temperatures are applied to the virtual batteries to generate voltage and temperature responses. The current, voltage and temperature samples input the multiple SoC estimation algorithms, and their outputs are logged, so they can be compared.

Validation profiles

The input profiles applied to the virtual battery pack feature variable conditions in SoC, Depth-of-Discharge (DoD), ambient temperature, duration, and current dynamics, allowing the algorithms to be tested under a wide span of operation conditions. While the complete test suite may include dozens of test cases, for the purposes of this paper, six validation profiles have been selected to cover SoC estimation under various conditions:

- Static current, full charge-discharge processes under constant temperature.
- Static current, partial charges and discharges under constant temperature
- Dynamic current in mid-high SoC range with constant temperatures.
- Dynamic current in low SoC range with constant temperatures.
- Dynamic current in mid-high SoC range with constant temperatures and pauses.
- Dynamic current in mid-high SoC range with temperature variations (15ºC – 45ºC).

The dynamic current profiles featured in the previous test cases should be selected according to the target application. For the purposes of this study, a Worldwide harmonized Light vehicles Test Cycles (WLTC) current profile has been selected, shown in Fig. 10. This profile has been applied repeatedly to generate a DoD of a 50% in the SoC range specified for each test case.

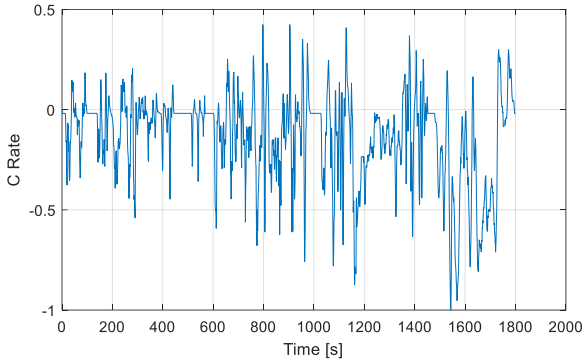

Fig. 10. WLTC-type current profile.

Figures of merit

To compare algorithm performance, different figures of merit will be used. The most common indicators are Root Mean Square Error ($RMSE$), Mean Absolute Error (MAE), and Maximum Absolute Error ($MaxAE$), defined as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}(x_n - \hat{x}_n)^2} \qquad (2)$$

$$MAE = \frac{1}{N}\sum_{n=0}^{N-1}|x_n - \hat{x}_n| \qquad (3)$$

$$MaxAE = \max_n |x_n - \hat{x}_n| \quad (4)$$

These error metrics will be averaged across multiple test cases *k* to obtain global error metrics. The final comparison metrics will be the average RMSE ($\overline{RMSE}$), the average MAE ($\overline{MAE}$), and the average MaxAE ($\overline{MaxAE}$), defined as:

$$\overline{RMSE} = \frac{1}{K}\sum_{k=0}^{K-1} RMSE_k \quad (5)$$

$$\overline{MAE} = \frac{1}{K}\sum_{k=0}^{K-1} MAE_k \quad (6)$$

$$\overline{MaxAE} = \frac{1}{K}\sum_{k=0}^{K-1} MaxAE_k \quad (7)$$

To measure the efficiency in implementation and performance of the algorithms, their execution times will be recorded and compared. In each test case *k*, the execution time of the slowest algorithm, T_{max}^k is used as a reference to normalize the times of each other algorithm in said procedure T_i^k, yielding the relative execution time R_i^k for algorithm *i*, calculated as shown in the following equation:

$$R_i^k = T_i^k / T_{max}^k \quad (8)$$

The average relative execution time $\overline{R_\iota}$ across all *K* test cases is given by:

$$\overline{R_\iota} = \frac{1}{K}\sum_{k=0}^{K-1} R_i^k \quad (9)$$

Finally, the performance indicator P_i for algorithm *i* is defined as the inverse of the average relative execution time, as shown in the following equation. This magnitude provides a quantitative measure of the speed of each algorithm across multiple test case.

$$P_i = \frac{1}{\overline{R_\iota}} = \frac{K}{\sum_{k=0}^{K-1} \frac{T_i^k}{T_{max}^k}} \quad (10)$$

Results

The average error metrics obtained with each algorithm across the six test cases are summarized in Table II. It can be observed that the EKF algorithm performs best, ranking first in all error metrics. The CC with OCV recalibration algorithm ranks second. In contrast, the CC and LSTM algorithms perform significantly worse, with the LSTM exhibiting noisier output (higher RMSE) and the CC showing the highest maximum error.

Table II. Error results

Algorithm	$\overline{RMSE}$	$\overline{MAE}$	$\overline{MaxAE}$
CC	13.06	5.70	6.89
CC with OCV recalibration	6.20	2.32	2.94
EKF	4.41	1.97	2.25
LSTM	22.22	4.83	6.51

Comparing the performance indicator of the algorithms (Table III), it can be observed that the LSTM has an average performance of 1, indicating it is the slowest algorithm in all test cases and serves as the reference for the relative metrics of the other algorithms. The EKF and CC algorithms, as implemented by the Simscape Batteries library, are on average 7.4 and 8.9 times faster than the LSTM algorithm, respectively. The CC algorithm with OCV recalibration, implemented at a SiL level in a precompiled C library, performs the fastest, being 462.6 times faster than the LSTM algorithm on average. This implies it is 62.5 times faster than the KF algorithm, which is the best performing in terms of error. This is expected, since it is implemented at a much lower level than the other algorithms.

Table III. Performance indicator results

Algorithm	P_i
CC (Sims. Batteries)	8.9
CC with OCV recalibration (Precompiled)	462.6
EKF (Sims. Batteries)	7.4
LSTM (Deep-Learning Toolbox)	1

Although the error outcomes in a real-world application might vary due to the simplicity of the models considered, the general conclusions drawn from the algorithm comparison are expected to remain consistent.

Conclusion

This paper presents a methodology for testing BMS functionalities in a virtual environment, utilizing tools such as Simscape Batteries in Matlab/Simulink. A case study comparing various SoC estimation algorithms at different implementation levels is presented.

Performance has been evaluated via error metrics, (RMSE, MAE, and MaxAE) and computational efficiency, using a custom performance indicator. Although the primary objective of the paper is not to conclude on algorithm performance, several insights have emerged from the comparison. Among the tested algorithms, the EKF algorithm consistently outperformed the others, achieving the lowest error metrics across all test cases, followed by CC with OCV recalibration. In contrast, the basic CC and LSTM algorithms exhibited significantly higher error rates, with the LSTM producing noisier outputs and the CC having the highest maximum error.

Regarding computational efficiency, the LSTM algorithm was found to be the slowest, while CC with OCV recalibration, implemented at a SiL level in a precompiled C library, demonstrated exceptional speed (462.6 times faster than the LSTM and 62.5 times faster than the EKF).

The next steps in this research involve expanding the analysis to SiL and HiL stages. Initially, SiL level versions of the remaining algorithms should be implemented to enable fair comparisons in terms of temporal performance.

For HiL testing, the algorithm or algorithms can be deployed onto the chosen embedded system. Utilizing a HiL system such as Speedgoat, the modeled cells can be emulated. These HiL systems typically provide libraries to interface their emulation devices with Simulink for real-time simulation. Consequently, all three versions (model-based, C-based running in Simulink, and C-based implemented in the actual device) can operate concurrently, receiving identical inputs generated both virtually and physically. This setup facilitates comprehensive performance comparisons following the established validation plans.

Additionally, this testbench currently covers only SoC validation. Future work could expand the testbench to validate other estimation algorithms or other functions of the BMS master, such as SoH, SoP, balancing algorithms or warning and alarm triggering.

References

[1]. R. R. Kumar et al. "Advances in Batteries, Battery Modeling, Battery Management System, Battery Thermal Management, SOC, SOH, and Charge/Discharge Characteristics in EV Applications," in *IEEE Access*, vol. 11, pp. 105761-105809, 2023.

[2]. D. Marcos et al. "Verification of an Automotive ASIL C Battery Management System Slave Unit," PCIM Europe, Germany, 2020, pp. 1-8.

[3]. P. Messier et al. "Multi-Cell Emulation for Battery Management System Validation," VPPC 2018 Chicago, IL, USA, 2018, pp. 1-6.

[4]. C. Fleischer et al. "Development of software and strategies for Battery Management System testing on HIL simulator," EVER 2016, Monte Carlo, Monaco, 2016, pp. 1-12.

[5]. A. Kalk et al. “Hardware-in-the-Loop Test Rig for Rapid Prototyping of Battery Management System Algorithms”, ITEC 2022, Anaheim, CA, USA, 2022.

[6]. T. M. N. Bui et al. "An Advanced Hardware-in-the-Loop Battery Simulation Platform for the Experimental Testing of Battery Management System," ICMT 2019, Salerno, Italy, 2019, pp. 1-6.

[7]. C. D. Tschritter et al. "Battery Management System (BMS) Test Stand Utilizing a Hardware-in-the-Loop (HIL) Emulated Battery," ESTS 2021, Arlington, VA, USA, 2021, pp. 1-8.

[8]. Sun B. et al. Virtual Battery Pack-Based Battery Management System Testing Framework. *Energies*. 2023; 16(2):680.

[9]. Di Rienzo R. Modular Battery Emulator for Development and Functional Testing of Battery Management Systems: The Cell Emulator. *Electronics*. 2022; 11(8):1215.

[10]. Verani A. Modular Battery Emulator for Development and Functional Testing of Battery Management Systems: Hardware Design and Characterization. *Electronics*. 2023; 12(5):1232.

[11]. Gregory L. Plett, Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs: Part 2. Modeling and identification, Journal of Power Sources, Volume 134, Issue 2, 2004, Pages 262-276, ISSN 0378-7753.

[12]. https://es.mathworks.com/products/simscape-battery.html, accessed on Aug. 15, 2024.

[13]. S. Hochreiter and J. Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (November 15, 1997), 1735–1780.

[14]. Azkue, M. et al. "Creating a Robust SoC Estimation Algorithm Based on LSTM Units and Trained with Synthetic Data" World Electric Vehicle Journal 14, no. 7: 197, 2023.