UNIVERSITY OF TARTU

FACULTY OF SCIENCE AND TECHNOLOGY

Institute of Technology

Robotics and Computer Engineering

Kristaps Dreija

Online Battery Cell State of Charge Estimation for use in Electric Vehicle Battery Management Systems

Master's Thesis (30 ECTS)

Supervisors: Prof. Gholamreza Anbarjafari

Egils Avots, MSc

Tartu 2018

Online Battery Cell State of Charge Estimation for use in Electric Vehicle Battery Management Systems

Abstract

The electric vehicle (EV) is a complex, safety-critical system, which must ensure the safety of the operator and the reliability and longevity of the device. The battery management system (BMS) of an EV is an embedded system, whose main responsibility is the protection and safety of the high-voltage battery pack. The BMS must ensure that the requirements for health, status and deliverable power are met by maintaining the battery pack within the defined operational conditions for the defined lifetime of the battery. The state of charge (SOC) of a cell describes the ratio of its current capacity (amount of charge stored) to the nominal capacity as defined by the manufacturer. SOC estimation is a crucial, but not trivial BMS task as it can not be measured directly, but must be estimated from known and measured parameters, such as the cell terminal voltage, current, temperature, and the static and dynamic behavior of the cell in different conditions. Many different SOC estimation methods exist, out of which (currently) the most practical methods for implementing on a real-time embedded system are *adaptive* methods, which adapt to different internal and external conditions. This thesis proposes the use of the sigma point Kalman filter (SPKF) for non-linear systems as an equivalent-circuit model-based state estimator to be used in one of the current series production EV projects developed by Rimac Automobili. The estimator has been implemented and validated to yield better results than the currently used SOC estimation method, and will be deployed on the BMS of a high-performance hybrid-electric vehicle.

CERCS codes: Automation, robotics, control engineering (T125)
Keywords: State of charge estimator, Central difference sigma point Kalman filter, Battery management system, Battery cell modelling, Electric powertrain vehicle.

Meetod elektrisõiduki aku laetuse taseme täpsemaks hindamiseks

Lühikokkuvõte

Elektrisõiduk (ES) on keeruline, ohutuse seisukohalt kriitiline süsteem, mis peab tagama operaatori turvalisuse ning seadme töökindluse ja pikaealisuse. ES-i aku juhtimissüsteem (BMS) on sardsüsteem, mille peamine ülesanne on kõrgepingeakude ohutuse tagamine ja kaitse. BMS peab tagama, et aku tervist, olekut ja väljaantavat võimsust puudutavad nõuded on täidetud ajal, mil aku töötab kindlaksmääratud ekspluatatsioonitingimustes ja seda kogu tootja poolt määratletud eluea jooksul. Laetuse tase (SOC) kirjeldab hetkemahtuvuse (mahtuvus, ingl. k "capacity") (salvestatud laengu suuruse) suhet tootja määratud nominaalmahtuvusega. SOC pole mõõdetav, mistõttu on selle täpne hindamine lähtuvalt mõõdetavatest parameetritest nagu näiteks elemendi klemmipinge, vool, temperatuur, elemendi staatiline ja dünaamiline käitumine erinevates tingimustes BMSi üks olulisimatest ülesannetest. Eksisteerib palju erinevaid SOC-i hindamise meetodeid, milledest efektiivsemad meetodid sardsüsteemis reaalajas rakendamiseks on nn adaptiivsed meetodid, mis on kohandatavad erinevatele sise- ja välistingimustele. Käesolev uurimus soovitab kasutada sigma punkti Kalmani filtrit (SPKF) mittelineaarsete süsteemide jaoks kui ekvivalentskeemide mudelipõhise oleku hindajat, mida kasutatakse praegustes Rimac Automobili poolt välja töötatud seeriatoodangu ES projektides. Pakutud algoritm SOC hindamiseks on juba rakendust leidnud suure võimsusega hübriidelektrisõiduki BMS süsteemides ja on rakenduspõhiselt tõestatud pakkuma täpsemaid tulemusi võrreldes seni kasutusel olnud hindamismeetoditega.

CERCSi koodid: automaatika, robootika, juhtimistehnika (T125)
Märksõnad: Laengu eindaja, Keskmine erinevus sigma-punkt Kalmani filter, Aku juhtimissüsteem, Aku modelleerimine, Elektrisõiduk.

Contents

List of Figures

List of Tables

Abbreviations

BMS - Battery Management System

C - General purpose low level structural programming language

C++ - General purpose low level structural programming language

CAN - Controller area network

CDKF - Central difference Kalman filter

CPU - Central processing unit

DOD - Depth of discharge

E-REV - Extended-range electric vehicle

ECU - Electronic control unit

EEPROM - Electrically erasable programmable read-only memory

EKF - Extended Kalman filter

EPTV - Electric powertrain vehicle (PHEV, HEV, E-REV or EV)

ECM - Equivalent circuit model

EV - Electric vehicle

HEV - Hybrid-electric vehicle

HV - High voltage

KF - Kalman filter

KL15 - Vehicle ignition switch position 2 (on)

LUT - Look-up table

LV - Low voltage

MCU - Micro-controller

NEDC - New European driving cycle

OCV - Open circuit voltage

PCB - Printed circuit board

PF - Particle filter

PHEV - Plug-in hybrid electric vehicle

RA - Rimac Automobili

RA-HPPC - Rimac Automobili hybrid pulse power characterization test profile

RMSE - Root mean square error

RT - Reference temperature

SIMD - Single instruction, multiple data

SOC - State of charge

SOH - State of health

SOP - State of power

SPKF - Sigma-point Kalman filter

TT - Testing temperature

UKF - Unscented Kalman filter

UUT - Unit under test

1 Introduction

Recent developments of commercially available PHEVs, HEVs, E-REVs and EVs have raised the need for more efficient, affordable and safer methods for HV battery pack control and management, where the cost of failure is exceptionally high.

A precise, real-time estimation of the different parameters of the battery in this case is crucial, as it allows for a more aggressive and efficient use of the battery and its DOD, optimizing the available power relative to the health and the state of the battery, which in turn allows for a lighter and physically smaller battery design, reducing the cost of the end-product.

Since most of the modern EPTVs nowadays use Lithium-ion (and similar Lithium polymer) batteries, the focus of this Master's thesis is on these cell chemistries. Further use of the term *battery* assumes a Lithium-ion (and similar Lithium polymer) battery pack, and the term *cell* assumes an individual Lithium-ion (and similar Lithium polymer) cell. However, the algorithms used and described in this thesis are general and can be applied to battery packs with other cell chemistries.

1.1 Problem overview

Generally, we can categorize the functional requirements of any arbitrary HV BMS in four categories:

1. **Monitoring and HV control:** The system must be able to measure all of the critical parameters of the battery pack, such as the individual cell voltage, cell or cell module temperatures and the overall battery pack current and voltage. Additionally, the BMS must be able to open, close and monitor the HV contactors, measure isolation and act accordingly (although, this can also be the responsibility of the power distribution unit).

2. **Protection:** The system must be able to protect the individual cells from over-charge, over-discharge, over-current, short circuits and temperatures outside of the operational range.

3. **Communication:** The system must communicate with all of its consumers, informing about all of the (depending on the application) necessary parameters, such as the available current and power, SOC, SOH, isolation status and other important parameters.

4. **Performance and Health management:** The system must be able to estimate the SOC, SOH, calculate the available charge and discharge energy and power limits, and balance the individual cells.

Categories 1 to 3 are fairly easily implemented in embedded software, which is run on the BMS ECU, however, accurate battery performance and health estimation algorithms are computationally heavy for real-time online estimation on embedded hardware. Because of that, most of the low-voltage non safety-critical applications, such as consumer electronics, often use very simple (if any) methods to fulfill these tasks.

In the case of the EV, which is a complex safety-critical system, best practices must be

used in order to ensure the safety and longevity of the device. However, the limited resources of the BMS ECU make the estimation of these parameters challenging, hence the motivation for this thesis.

Figure 1.1: Two Tesla Model S 85kWh battery modules [1]

Figure 1.1 depicts two battery modules of the Tesla Model S 85kWh battery. The complete pack contains a total of 16 battery modules, each of which in a 6S (series) 74P (parallel) cell configuration. Each module (slave) communicates to the BMS coordinator (master) over the CAN bus (private CAN), which in turn communicates with other ECUs over a different CAN bus.

Even though the progress of technology is immense, the automotive industry is hesitant to adopt these technologies, as it requires certain standards to be fulfilled, hence the most powerful state-of-the-art computational units are mostly not used in the industry for series production vehicles, as it takes years for the silicon vendors to certify their

products according to the automotive standards (safety, integrity, non-volatility, redundancy, etc). That's why the majority of the embedded automotive applications are still written in either *C* or *C++* (with critical parts still being written in *Assembly*), or Auto-generated to *C* or *C++* from a high level Model-Based programming language, such as *Matlab Simulink*.

The SOC estimation has been selected for this thesis because it is the first step in the development of a complete performance and health management subsystem for the BMS of the HV battery of an EV. Without an accurate SOC calculation, all other relevant parameter values within the scope of this subsystem are up to a subject of drifting (relative to the inaccurate estimation of the SOC).

However, a successful and precise SOC estimation can as a byproduct bring out additional information (parameters) about the battery, and hence can be further used in SOH estimation, optimum DOD calculation, and energy and power limit calculation.

1.2 Motivation

The main motivation to perform the research and development of an accurate online SOC estimator has arisen from the fact that the author of this thesis currently works as a BMS Application Engineer at Rimac Automobili d.o.o. (RA) in Croatia, developing embedded solutions for new, and improving the existing solutions for EPTV BMSs.

Developing a robust and accurate SOC estimation algorithm will result in more reliable and safer battery packs, reducing the overall cost of the end-product and after-sale cost of support and product recalls due to malfunction. Because of the fact that the battery packs are one of the main selling products developed and sold by RA, the interest in developing such algorithms is considerable. The algorithms need to be easily adaptable to different cell chemistries, as the same BMS hardware (with minimal embedded software adaptations) is being used across different projects for different clients.

1.3 Goals

The main goal of this thesis is to develop a SOC estimation algorithm, with the computational complexity sufficient to be run as an embedded software module on one of the available automotive grade MCUs.

The first part of the thesis consists of an overview of the working principles of Lithium chemistry based battery cells and the BMS. The SOC is defined and explained, a discussion of the importance of the SOC estimation is conducted, and an overview of the different SOC estimation approaches is presented.

In the next section of the thesis, different cell modelling methods are looked at, and the (currently) most effective, practical cell model is chosen for this thesis and described in more detail. After that, an overview of different SOC estimation algorithms is presented, and the one that best fits the constraints is selected for this thesis and described in more detail. Afterwards, the cell model data acquisition procedure is defined, and the appropriate experiments are defined, performed and described.

Next, the state estimation algorithm with the selected cell model is implemented in Embedded-C for use as an embedded software module running on automotive embedded hardware (MCU). The programming code structure is briefly explained, providing some crucial code snippets.

After the estimator has been implemented, the validation test plan is defined and the tests performed with different cell load profiles in a controlled environment. Finally, the results are presented, conclusions are made, and possible future work and improvements to the system are discussed.

2 Background

2.1 Battery cells

A battery cell is the smallest physical electro-chemical energy storage unit. The [8] definition of a cell is: *The basic electro-chemical unit, characterized by an anode and a cathode, used to receive, store and deliver electrical energy*.

A cell is called a *primary cell* if it can only be discharged once (the chemical reactions done to produce current are **irreversible**). A cell is called a *secondary cell*, if it is rechargable (the chemical reactions done to produce current are **reversible**). For this thesis, only secondary cells are considered. A battery comprises of two or more electrically connected individual cells. The main characteristic of a cell is its ability to store and deliver electrical charge to a load circuit. The amount of charge that can be stored in a cell (nominal cell capacity) is quantified in ampere-hours (Ah), and it describes how many hours can a cell be discharged at a discharge current of 1 ampere (A).

The C rate of a cell is a widely used unit of describing different cells with different characteristics and chemistries, and it simply quantifies the battery discharge rate relative to its capacity. So a rate of $1C$ quantifies the amount of current it takes to fully charge or fully discharge the battery (or cell) in an hour. Alternatively, a C rate of $10C$ will charge or discharge the cell in $\frac{1}{10}h$, but at the rate of $C/2$, the battery (or cell) will be fully charged or discharged in 2 hours.

Adding cells in series will increase the total battery voltage, but will keep the same capacity as the individual cell (as per Kichoff's law of voltage). Connecting cells in parallel will yield the same voltage, but accumulate the capacities of the individual cells of the battery, as the load current will be divided between the cells (Kirchoffs law of current).

By designing different series-parallel (*SP*) configurations, the designer can fulfill different requirements for voltage, power, current and capacity. Simply,- if the design asks for a large battery capacity but does not require high voltage, multiple cells can be connected in parallel (for example, four 3Ah cells in parallel (1S4P), yielding a total of 12Ah nominal battery capacity). The same goes for high voltage and high voltage-high capacity configurations. In short, individual cells are used as building blocks to fit different battery solutions.

2.1.1 Lithium chemistry based cells

Basic Lithium chemistry based battery cells are built from five components:

1. **A negative and a positive electrode:** On discharge, the negative electrode releases electrons through external load, oxidizing the electrode (oxidation). On charge, the negative electrode accepts electrons coming from external circuitry, reducing the electrode (reduction). The same process in reverse applies to the positive electrode.

2. **An electrolyte material:** The electrolyte is an electronic insulator (does not allow electrons to pass) and an ionic conductor (allows for the flow of ions), and its main function is to transport the ions from one electrode to the other (and back).

3. **A separator:** The separator isolates the electrodes. As electrolyte, it also prevents electron-flow, but allows the flow of ions. The main purpose of the separator is to avoid a short-circuit between electrodes.

4. **A current collector on each electrode:** Current collectors' main purpose is to reduce the ohmic resistance of an electrode, and also provide a better medium for connecting the electrodes to the terminals of the cell (points to which external circuitry is connected).

Different cells and different manufacturers will produce cells that include other components, however, that is out of the scope of this thesis. Figure 2.1 depicts the schematic diagram of a Lithium-ion cell.

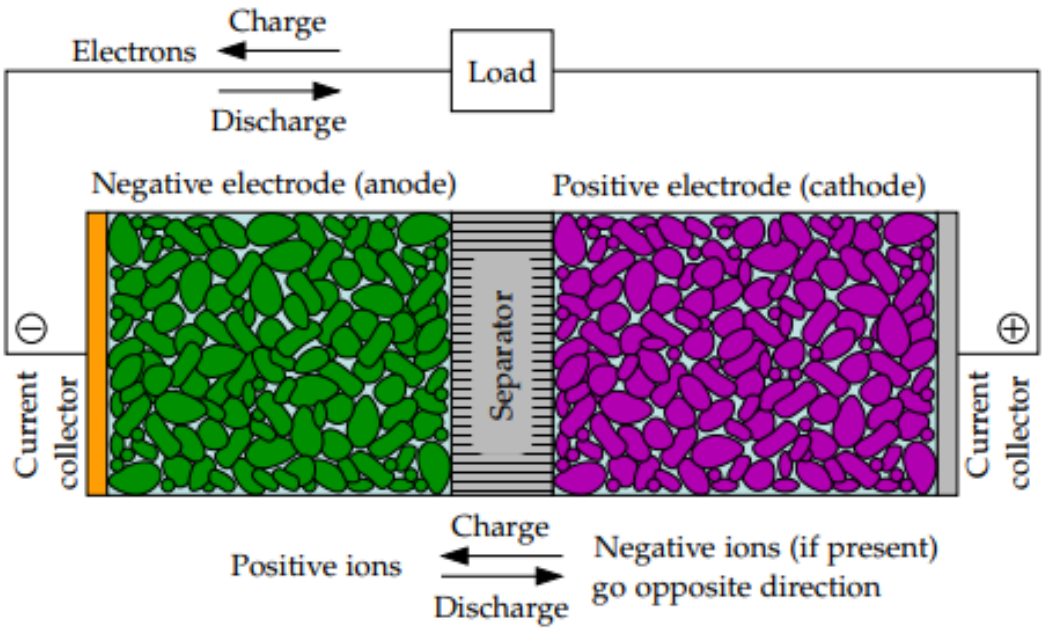


Figure 2.1: Lithium-ion cell [2]

Advantages

The use of Lithium chemistry based cells has many important advantages over other cell chemistries. Compared to other secondary cells, Lithium based chemistry cells have a higher nominal voltage, typically at 3.7V (1.2V for NiCad). Energy density is also much [9] greater than with other chemistries. Because of these facts, less cells can be used to achieve the same battery voltage and energy density, resulting in lighter, more

compact battery packs. Also, the overall design of the battery gets improved with this as there are less nodes that need to be monitored and controlled by the BMS.

Another important property of Lithium chemistry based cells is low self-discharge, which means that the cell will retain its charge for a much longer time, compared to other cell chemistries [2]. Other important advantages (compared to other cell chemistries) include [9]: high power density, variety in sizes and shapes, low weight, the possibility to optimize for capacity or discharge rate, large capacity, the fact that the cells can be discharged at 40C or more, fast charging, large *usable* DOD, no memory effect - no need for reconditioning high coulombic efficiency (95% and more), batteries from these cells can be adapted to almost any voltage, power and energy requirements, and a fast response to charge and discharge calls.

Disadvantages

The main disadvantage of Lithium chemistry based cells, compared to other cell chemistries, is the cost (although, the cost has been falling with the increase of products using Lithium chemistry based cells). Not only the cost of the cell itself, but also the added cost of needing to incorporate a BMS, as the cells must be carefully kept within their designed temperature and voltage ranges (overcharge and overdischarge can lead to catastrophic failures, including fire and explosion hazard [10]).

Other important disadvantages (compared to other cell chemistries) include [9]: internal impedance higher than equivalent NiCad cells, chemical instability - although improved over time (with the likes of LiPo cells with solid electrolyte), Lithium is still chemically very reactive, thus increasing the risk of catastrophic failures, relatively strict shipping regulations for products containing Lithium-ion cells, significant degradation in high temperatures, venting and a possible thermal runaway when physically damaged, and the fact that the SOC estimation is more complex than with other cell chemistries, because of the flat OCV-SOC curve in the most of the operational SOC.

2.2 Battery management system

A BMS [1] [11] is an embedded system (electronics and embedded software) whose main concerns are the protection and safety of the operator of the system, the protection of the battery pack, the prolonging of the life of the battery pack, and the maintenance of the battery pack in order to fulfill the design requirements.

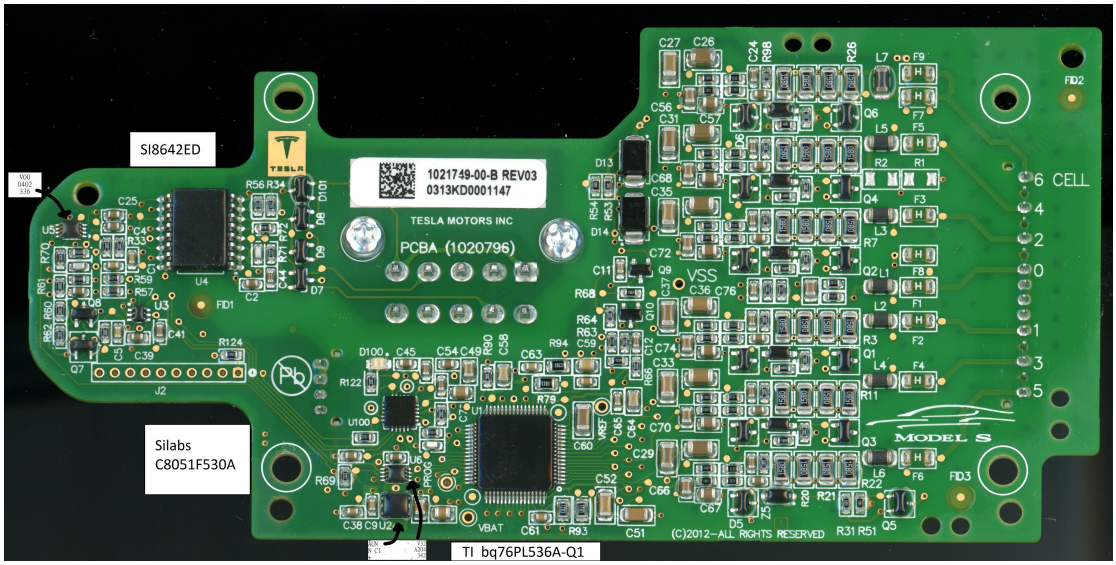


Figure 2.2: Tesla Model S 85kWh BMS PCB [1]

2.2.1 Battery management system workflow

A simplified typical automotive BMS workflow can be described as Algorithm 1:

[1] IEEE Standard 1491 definition of the BMS (*battery monitoring system*: "A permanently installed system for measuring, storing, and reporting battery operating parameters")

Algorithm 1 Typical automotive BMS application [2]

```
while KL15 OFF do
    sleep
end
initialization
while KL15 ON do
    individual cell voltage and temperature measurement
    battery pack current measurement
    SOC estimation
    SOH estimation
    cell balancing
    power and Energy calculation
    CAN Communication
    HV contactor management
    data storage
end
opening contactors
data maintenance and storage
```

The BMS is in *sleep* mode until KL15 has been turned on. On system wake-up, all of the MCU peripherals are initialized, followed by reading of the backed up data from the on-board non-volatile memory, such as EEPROM. After that, all of the BMS algorithms are initialized with initial values, and system checks are performed, flagging any unexpected values and/or errors. While the vehicle is kept in the *on state* (KL15 ON), the main control loop is executed and all of the algorithms are performed. Firstly, all of the necessary measurements are made (cell voltages, cell and PCB temperatures and battery pack current, interlock, other digital and analog inputs) and any inconsistencies with the system and safety rules are reported and immediately acted on. Then, the SOC and SOH are being estimated, updating the according battery parameters. Cell balancing is performed in case any of the cells in the packs and sub-packs are disbalanced (with non-equal state of charge). The available power and energy limits are then calculated, and all of the relevant information is sent over the CAN bus to higher-level governing/control

[2] Algorithm 1 is a simplified representation of what really happens in the system. Most often, many of the processes are software-parallelized, meaning that they are executed by timer and other interrupts (for example, CAN communication can be triggered by a 1ms timer interrupt, sending different CAN messages with different periods. Also, it is assumed, that the BMS embedded software does not use an operating system.

ECUs. Depending on the incoming requests over CAN, the HV contactors are managed (opened/closed), their feedback read and any errors (stuck open/welded contactors) are reported and acted on. Finally, the loop finishes its execution by housekeeping the relevant data by storing it on non-volatile memory. The cycle is then repeated. Once KL15 is turned off, the HV contactors are disconnected, and all data housekeeping and storage is performed, and the device enters sleep mode, preserving LV power.

2.3 State of charge

Electrochemically, the SOC of a cell is related to the average concentration of Lithium in the negative-electrode solid particles to the negative-electrode solid particles. Thus, we can define the SOC of any Lithium based cell as the ratio of the average Lithium concentration to the maximum possible Lithium concentration in the electrode materials. Based on the design limits of the cell, as defined by the cell manufacturers, the SOC is intended to be a fixed value between 0% and 100% [6], and can be explained in terms of electric charge (current $\times$ time (Ah)) as the ratio of its current capacity $Q(t)$ to the nominal capacity Q_n, which is defined by the manufacturer and defines the maximum amount of charge that can be stored in the cell. Thus, we can define the SOC in the following way:

$$SOC(t) = \frac{Q(t)}{Q_n} \tag{2.1}$$

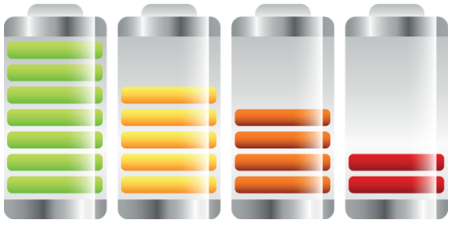


Figure 2.3: Visual representation of the SOC [3]

2.3.1 The problem with state of charge estimation

One of the main reasons why accurate estimation of SOC is not trivial is the fact that (currently) it can not be measured directly with some sort of sensor, but it must be estimated indirectly from known and measured variables, such as cell OCV curves in different temperatures, overall battery current, cell terminal voltages and cell temperatures.

2.3.2 State of charge algorithm complexity

The selection of the SOC estimation algorithm is a direct trade-off between the desired performance and safety of the system, and the cost of embedded software development (and testing and validation) and hardware complexity (more complex and precise algorithms require more powerful computational units and more precise sensors). Nevertheless, precise SOC estimation provides many advantages over simpler methods, such as [6]:

1. **Battery longevity:** Keeping the cells within 0% to 100% SOC (not allowing over or under-charging) .
2. **Battery performance:** Having an accurate SOC estimation algorithm allows for more aggressive and efficient battery utilization, as we can trust that it will not be over or under-discharged.
3. **System reliability and re-usability:** Simple and unreliable SOC estimation algorithms will behave differently with different battery utilization profiles, whereas accurate SOC estimation algorithms are consistent between different usage profiles.
4. **Power density:** Since we can trust an accurate SOC estimation algorithm, the DOD can be increased, thus allowing for higher power density, resulting in smaller, lighter battery pack designs for the same output capacity.

5. **Economy:** As a result of the aforementioned points, the costs of the battery pack can be greatly reduced,- smaller systems cost less and increase the performance of the vehicle (less weight); reliable systems reduce warranty repair/recall costs.

2.3.3 State of charge estimation approaches

Many different SOC estimation approaches exist, and most of them can be used in different applications. However, because of the fact that the HV batteries used in EPTVs are a safety-critical system, poor and imprecise methods must be disregarded, and more complex options must be considered.

Here we consider some SOC estimation approaches which use a combination of measurements and a cell model. However, cell models are limited and there are parametric uncertainties, not allowing for a design of an absolutely perfect algorithm [12]. The classification of the SOC estimation methods varies from literature to literature, but generally they are divide in the following categories [13] [12]:

1. **Direct measurement methods:** Measuring of the physical cell properties.
2. **Book-keeping methods:** Integration of the current flowing out of the cell.
3. **Adaptive methods:** Automatic SOC adaptation depending on different current/temperature profiles.
4. **Hybrid methods:** A combination of the previously stated methods.

In the next sections of this chapter different methods are briefly described, compared and their potential use in EPTVs is pointed out.

Direct measurement

1. **Open-circuit voltage method:** Since the cell terminal voltage is a function of the SOC (Equation 3.3), if we ignore the the hysteresis and slow changes are

disregarded (cell is at rest), we can approximate the cell terminal voltage to be its OCV:

$$v_k \approx OCV(z_k) \tag{2.2}$$

This results in a very simple SOC estimation algorithm which can be implemented by simple LUTs, where each LUT represents the OCV of the cell for different SOCs at a specific temperature. The main problem with this approach is that $v_k \approx OCV(z_k)$ only when the cell is at rest. Also, the character of the Lithium based cell OCV-SOC curve (non-linear relationship) results in very poor resolution within a range of the curve (even as small as 10mV cell voltage change can yield an SOC change of tens of percent) as presented in Figure 2.4.

This approach is often used for SOC re-calibration after an extensive rest period (up to hours), and also in simple safety non-critical BMS systems. However, this method alone is not really applicable for use in EPTVs.

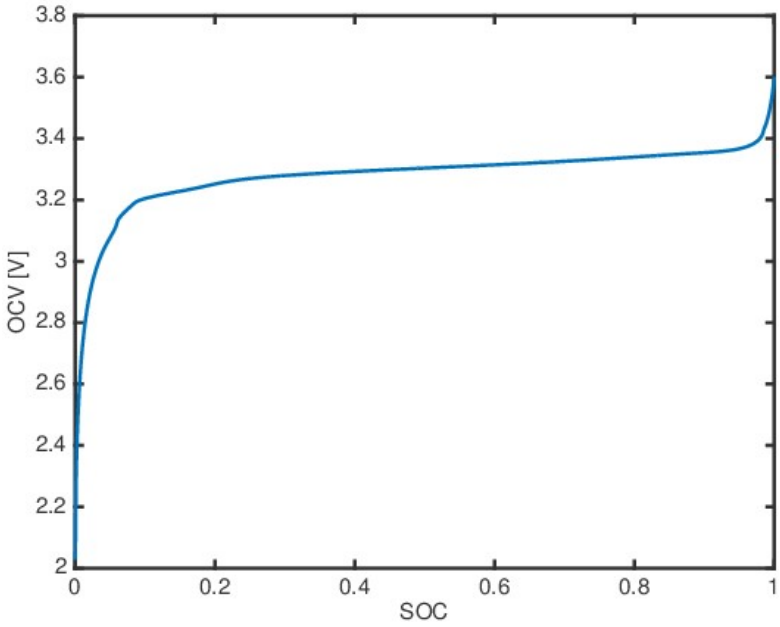


Figure 2.4: OCV-SOC curve for Lithium Iron Phosphate (LFP) cell [4]

Book-keeping

1. **Coulomb counting method - current integration:** Since one of the crucial measurements that a *practical* BMS needs to take is the battery current, and the cell manufacturer always defines the nominal (usable) cell capacity in Ah, we can integrate the current flowing in and out of the battery to estimate the SOC. The SOC in this case can mathematically be described as Equation 2.3:

$$z_k = z_0 - \frac{\Delta t}{Q} \sum_{j=0}^{k-1} \eta_j i_j, \tag{2.3}$$

where

z_k - SOC at discrete time index k,

z_0 - initial SOC,

Δt - total integration time,

Q - total cell capacity (A*s),

η_j - cell coulombic efficiency,

i_j - cell current.

The problem with this method is that the measured current is not precisely equal to the actual cell current, as it also employs measurement noise, leakage currents, cell self-discharge currents, etc. This, together with the fact that the *actual* cell capacity and the coulombic efficiency η are only approximated values, adds to the SOC estimation error, which is integrated along with the actual current. Generally, this method is more accurate than simple OCV-SOC LUT method alone, because of the non-linearities of the OCV-SOC curve, but for more effective use, the integrator needs to be reset and re-calibrated at frequent intervals in order to minimize accumulated error. This can be done with the OCV method when the cell is *at rest*, but is not the most optimal way on how to implement a robust and safe SOC estimation algorithm. Also, in this case the cell cycle life, magnitude of the current, temperature and history are not considered, adding to the error in estimation.

Adaptive

With the recent developments in pattern recognition and artificial intelligence, new, adaptive systems have become a popular research topic, resulting in the making of new, robust mechanisms for SOC estimation. Such methods include various types of Neural Networks, Support-Vector Machines and some variations of the Kalman Filter. Since the parameters of these systems can adapt over time and circumstances, the practical BMS can really benefit from these methods. However, the added computational complexity and time of development very often are the biggest trade-offs when choosing which method to employ in a commercial product, such as the EPTV.

Hybrid

Hybrid methods are also gaining popularity, as they allow for combining the positive aspects of different methods that used alone would be insufficient, such as in an example above, where the Coulomb-counting method was combined with the OCV method to re-calibrate the integrator at rest-times.

2.3.4 Choosing the state of charge estimation method

Looking at the different SOC estimation methods, it is easy to define the best method for use in EPTV's BMS. Open circuit voltage method is the simplest of all methods, as it only includes the use of a LUT or multiple LUTs (temperature dependent). However, this method is very inaccurate, as the terminal voltage approaches the OCV only after long rest periods, hence not applicable for use in EPTVs.

Coulomb-counting consists of a simple current integrator, and, while a little more complex than the OCV method, is still trivial. Provides better results than the OCV method, but is still inferior to more advanced methods due to unknown measurement errors, which will accumulate together with the integrator.

Adaptive and hybrid methods really are the obvious choice for use in EPTVs, but an educated choice has to be made on which methods can be employed for use on simple, automotive MCUs. Neural Networks and Support-Vector Machines are a new trend, but not currently widely used in industry because of their relatively high computational complexity.

The Kalman filter (and it's non-linear versions) has been used since 1960s [14] [15], where it was first applied to navigation systems for the Apollo Project. The computer used in Apollo missions used only 2kB magnetic core RAM; the CPU clock speed was under 100 kHz [16], and the Kalman filter was simple enough computationally to put on such minuscule hardware for today's standards. The computational simplicity relative to the achievable performance is the main reason why the KF is still extensively used today, and has been chosen for this thesis as the cell state and SOC estimation algorithm.

3 Related Work

3.1 Model-based estimation

Two fundamental cell models currently exist [2], namely the empirical cell modelling, and the physics-based cell modelling. Physics-based cell modelling is based on the internal mechanisms of the cell, as opposed to the empirical cell modelling, which employs the approximation of the cell's voltage response to different inputs (current, temperatures) via electrical circuits.

Currently, due to the relative simplicity and availability of different algorithms and methods, empirical cell modelling is used almost exclusively in the industry [2], however, reduced-order physics-based cell modelling and its application in practical solutions is being researched extensively [17]. A model-based state estimator block diagram is depicted in Figure 3.1:

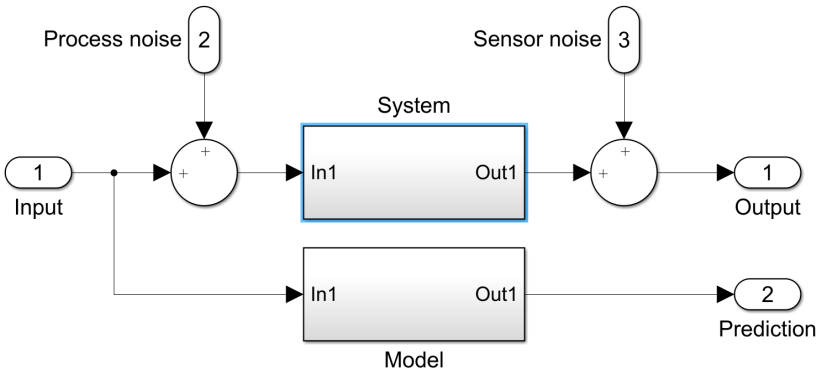


Figure 3.1: Model-based estimator

The downside of empirical cell modelling methods is that if the cell is being used in different scenarios than those in which the cell model had been made (if the dynamic behavior of the cell is not properly acquired), the model cannot be fully trusted. However, the simplicity of the calculations performed to use these methods (currently) outperform the physics based modelling methods.

Because of that, and also the fact that most embedded engineers who develop these algorithms have more experience with circuit design than electro-chemistry [6], empirical cell modelling using equivalent circuit methods is chosen for this thesis, as the main goal is the implementation of the SOC estimation algorithm on embedded hardware for real-time operation.

3.1.1 Equivalent circuit models

The best methods for accurate SOC estimation must use computationally simple sets of mathematical equations that describe the battery cell dynamics. Equivalent circuit model's representation of the cell is an electrical circuit. The models are empirically built by performing experiments that show the dynamic cell behavior under different circumstances (temperatures, currents). Equivalent-circuit model-based estimation can combine all of the known variables (cell temperatures, terminal voltages and battery current), and not only estimate the SOC, but also give information about other internal states of the model.

The inputs (battery current and cell temperatures) are propagated through the model, predicting the output (cell terminal voltage). The estimated output is compared to the actual measured terminal voltage, and the difference (error) is used as a feedback to update the estimation, effectively reducing the error over time. Additionally, the different immeasurable noises can be incorporated in the model to improve the estimator.

3.1.2 First-order equivalent-circuit cell model

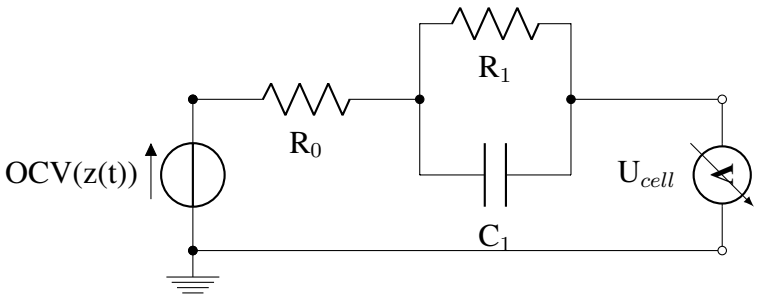


Figure 3.2: First-order ECM

Figure 3.2 depicts the first-order ECM, where:

- $OCV(z(t))$ is the OCV represented as a voltage source;
- R_0 is the internal resistance of the cell;
- R_1 and C_1 models the cell diffusion voltages.

More RC nodes can be added to improve the model, however, it adds to the computational complexity and is a trade-off when designing the algorithms. Since R_0, R_1 and C_1 can not be measured directly, they need to be optimized to experimentally acquired cell data to best represent the behavior of the cell. [18] Once the algorithm is proven to behave as expected, optimization methods must be performed in order to find the optimal number of RC nodes, relative to the desired performance and the added computational complexity.

The ECM can be described by means of mathematical equations in discrete-time, as that's how it will be implemented in embedded software. The derivation of the following equations is explained in detail in [2]. Hysteresis model needs be added in case the cell [19] (for example $LiFePO_4$) exhibits it. However, the cell selected for this project (Section 4.1) has minimal hysteresis and is assumed to be negligible.

State of charge

$$z_k + 1 = z_k - \frac{\eta_k i_k \Delta t}{Q}, \tag{3.1}$$

where

z_k - SOC at discrete-time index k,

η_k - *cell coulombic efficiency* ($\frac{\textit{discharge capcity}}{\textit{charge capacity}}$),

i_k - cell current,

Δt - sample time (s),

Q - total cell capacity (A*s).

Diffusion current

$$i_{R_1,k+1} = exp\left(\frac{-\Delta t}{R_1 C_1}\right) i_{R_1,k} + \left(1 - exp\left(\frac{-\Delta t}{R_1 C_1}\right)\right) i_k, \tag{3.2}$$

where

$i_{R_1,k+1}$ - current flowing through R_1. This term of the model describes the slow changes of the diffusion processes which happen within the cell.

ECM state equation

We can define the state equation, which describes all of the dynamic effects of the cell:

$$v_k = OCV(z_k, T_k) - R_0 i_k - \sum_i R_i i_{R_i,k}, \tag{3.3}$$

where

v_k - cell terminal voltage,

R_0 - internal cell resistance,

i_k - current flowing through the circuit.

3.2 The discrete Kalman filter

The Kalman filter (also: **linear** quadratic estimator) is a recursive state estimation algorithm that uses (noisy and inaccurate) sensor measurements to estimate the internal state of a dynamic system. A good online source of knowledge about Kalman filters, the fundamental KF mathematics, and practical applications is the University of Colorado Colorado Springs (UCCS) course *Applied Kalman Filtering* (ECE5550). All of the lecture notes and videos are available on [20].

The algorithm can be described as a two step process:

- **Prediction:** Predict the internal state variables and output, and their uncertainties.
- **Update:** Update the prediction based on the new observations (measurements).

The prediction step looks at the state estimation from the previous time-step to create the current estimate. It is called (*a priori*) (further (in equations) denoted with the superscript $^-$) state estimate as it does not take into account the observation (measurement) at the current time-step. The update step combines the *a priori* state produced in the prediction step with the measurement to improve the estimate. At this point, the state estimate is called *a posteriori* (further denoted with a superscript $^+$). The two steps are typically recursively alternating between each other, but, for example, if the measurements are taken less frequently, the update steps can be skipped until a new measurement has been taken.

To make the concept clear within the context of cell state estimation, we can consider a BMS that can measure the terminal voltage of the cell, the cell temperature and the current. From Section 3.1, it is known that to estimate the SOC, we have to take into account the directly immeasurable (virtual) internal components of the cell, as illustrated in Figure 3.2. The values of R_0, R_1 and C_1 are the *hidden variables*. The state estimation in this case involves *predicting* the state of the system based on the current and

temperature (the inputs of the system), and updating the prediction based on the measured terminal voltage. The difference between the estimated output (terminal voltage) and the measured output acts as a feedback to the estimator, which is used to correct the estimate as a weighted average between the prediction and the measurement (basically deciding which to trust more: the measurement or the model prediction). The two main Kalman Filter state equations are illustrated in Algorithms 3.4 and 3.5 [21]:

$$x_k = F_k X_{k-1} + B_k u_k + w_k, \tag{3.4}$$

where

x_k is estimated new state,

F_k is the state transition model applied to the previous state x_{k-1},

B_k is the control (input) model applied to the control vector u_k,

w_k is the process noise (could model the current sensor inaccuracy in our context, assumed to be zero-mean Gaussian).

$$z_k = H_k x_k + v_k, \tag{3.5}$$

where

z_k is the current observation of the system,

H_k is the observation model, mapping the true state space into the observed space (measurement),

v_k is the sensor noise (observation noise, assumed to be zero-mean Gaussian noise, could model the output voltage measurement inaccuracy in our context).

The Kalman filter is a great optimal state estimator for linear systems, however, it is easy to notice the non-linear behavior of our ECM, as is is easily visible in the example OCV-SOC curve illustrated in Figure 2.4, and in the Equation 3.3. The linear Kalman filter assumes a Gaussian distribution. If the state transition function is linear, then the resulting distribution after performing a linear transformation will also be with

Gaussian properties. However, it will not be the case if the state transition function is non-linear. In this case, the filter will not converge, providing very poor estimates. Globally linearizing the curve to work with the KF would also yield very poor results, hence non-linear versions of the Kalman filter must be considered instead.

3.2.1 Non-linear Kalman filters

There are multiple available non-linear extensions to the classic linear Kalman filter, the most popular being [7]:

1. **Extended Kalman filter (EKF):** Still very popular in practice, however, it yields poor results if the system is very non-linear, as it analytically linearizes the model (the non-linear function) around the mean of the current state estimate (local linearization). It is more computationally complex than linear KF, as it involves Taylor series expansion to linearize the system equations [22].

2. **Sigma-point Kalman filter (SPKF):** Similar computational complexity as EKF. Linearizes the model at each point in time by using statistical/empirical linearization (educated sampling), but, in contrast, provides much better results [6] than EKF, especially if the function is very non-linear.

3. **Particle filter (PF):** Many orders of magnitude more computationally complex than EKF and SPKF, and are thus very impractical in real-life applications. However, it is the most precise method of the three. The main working principle of PFs is to utilize Monte-Carlo integration methods to directly approximate the system dynamics (sampling a lot of *particles*, whereas in SPKF, only a handful of *particles* are **chosen**).

After analyzing the pros and cons of the three popular non-linear KF filtering methods, it is clear that the best choice for a practical cell state and SOC estimation is to implement the SPKF.

3.2.2 Sigma-point Kalman filter

Compared to the EKF, the SPKF approximates the probability distribution of the estimate, instead of approximating the non-linear function. A visual representation of the linearization methods (sampling, EKF, SPKF) is depicted in Figure 3.3:

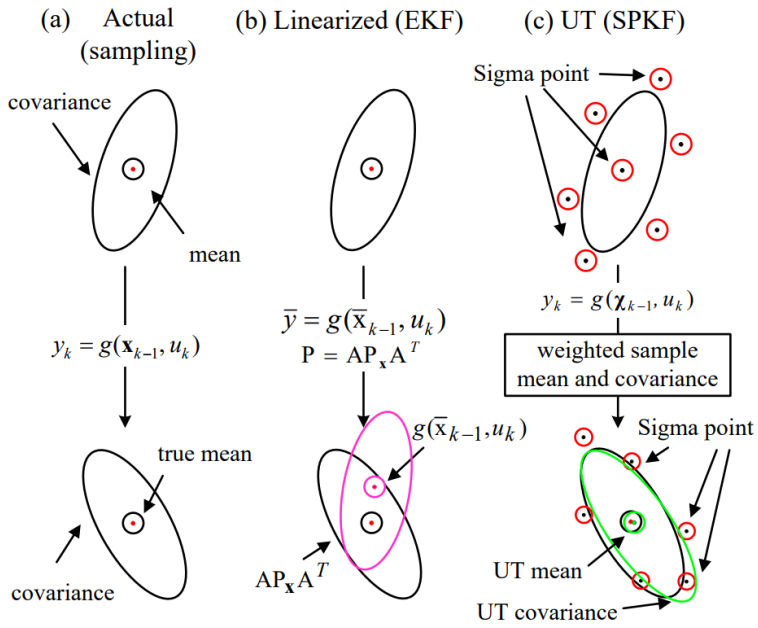


Figure 3.3: SPKF mean and covariance propogation, as compared with actual (sampling) and EKF method [5]

The non-linear state-space representation of the model is a set of two equations:

$$\begin{cases} x_k = f_{k-1}(x_{k-1}, u_{k-1}, w_{k-1}) \\ z_k = h_k(x_k, u_k, v_k), \end{cases} \tag{3.6}$$

where

v_k, w_k - independent, Gaussian (as in Equations 3.4 and 3.5).

The SPKF algorithm has been explained well in [6], [10] and [5], and can be described in the following steps:

1. Choose a set of (sigma) points $\mathcal{X}$ (symmetrically distributed around the mean), so that the mean and the covariance of the sigma points matches the mean ($\bar{x}$) and covariance ($\Sigma_{\tilde{x}}$) of the modelled a priori variable.

   The sigma points are calculated deterministically. Given the input variable x has L dimensions, $p+1 = 2L+1$ sigma points will be calculated as in Equation 3.7:

$$\mathcal{X}_{k-1}^{a,+} = \{\hat{x}_{k-1}^{a,+}, \hat{x}_{k-1}^{a,+} + \gamma\sqrt{\Sigma_{\widetilde{x},k-1}^{a,+}}, \hat{x}_{k-1}^{a,+} - \gamma\sqrt{\Sigma_{\widetilde{x},k-1}^{a,+}}\}, \tag{3.7}$$

   where

   $\mathcal{X}$ is indexed from 0 to p,

   $\sqrt{\Sigma}$ is calculated using the *Cholesky decomposition* [1].

   γ is a tuning parameter (constant), mathematically explained in Table 3.1.

   The weighted mean is calculated as in Equation 3.8:

$$\hat{x}_k^- = \sum_{i=0}^{p} \alpha_i^{(m)} \mathcal{X}_{k,i}^{x,-}, \tag{3.8}$$

   where

   $\alpha_i^{(m)}$ is the weight factor for the mean value, $\mathcal{X}_i$ is the ith member of $\mathcal{X}$.

   The weighted error covariance is calculated as in Equation 3.9:

$$\Sigma_{\widetilde{x},k}^- = \sum_{i=0}^{p} \alpha_i^{(c)} (\mathcal{X}_{k,i}^{x,-} - \hat{x}_k^-)(\mathcal{X}_{k,i}^{x,-} - \hat{x}_k^-)^T, \tag{3.9}$$

   where

   $\alpha_i^{(c)}$ is the weight factor for the covariance.

   Different methods for choosing $\alpha_i^{(m)}$, $\alpha_i^{(c)}$ and γ exist, two most popular being the UKF and CDKF, and have been summarized in Table 3.1:

   Because of the fact that **CDKF** only has one filter tuning parameter, providing a

[1] The definition and implementation in various programming languages can be found in [23].

Table 3.1: SPKF parameter calculation methods [6] [7]

Method	**UKF**	**CDKF**
γ	$\sqrt{L+\lambda}$	h
$\alpha_0^{(m)}$	$\frac{\lambda}{L+\lambda}$	$\frac{h^2-L}{h^2}$
$\alpha_k^{(m)}$	$\frac{1}{2(L+\lambda)}$	$\frac{1}{2h^2}$
$\alpha_0^{(c)}$	$\frac{\lambda}{L+\lambda}+(1-\alpha^2+\beta)$	$\frac{h^2-L}{h^2}$
$\alpha_k^{(c)}$	$\frac{1}{2(L+\lambda)}$	$\frac{1}{2h^2}$

simpler implementation, it has been chosen as the **UKF** method for this thesis.

2. Propagate the chosen sigma points through the non-linear model equation (function), resulting in a new set of points ($\mathcal{Z}$):

$$\mathcal{Z}_{k,i} = h_k(\mathcal{X}_{k,i}^{x,-}, u_k, \mathcal{X}_{k-1,i}^{v,+}) \tag{3.10}$$

3. Approximate the a posteriori mean with the propagated set of points ($\mathcal{Z}$):

$$\hat{z}_k = \sum_{i=0}^{p} \alpha_i^{(m)} \mathcal{Z}_{k,i} \tag{3.11}$$

4. Calculate the a posteriori gain covariance matrices:

$$\Sigma_{\tilde{z},k} = \sum_{i=0}^{p} \alpha_i^{(c)} (\mathcal{Z}_{k,i} - \hat{z}_k)(\mathcal{Z}_{k,i} - \hat{z}_k)^T \tag{3.12}$$

$$\Sigma_{\tilde{x}\tilde{z},k}^{-} = \sum_{i=0}^{p} \alpha_i^{(c)} (\mathcal{X}_{k,i}^{x,-} - \hat{x}_k^{-})(\mathcal{Z}_{k,i} - \hat{z}_k)^T, \tag{3.13}$$

where

$\mathcal{X}_k^{x,-}$ is calculated in Equation 3.5

$\mathcal{Z}_k$ is calculated in Equation 3.10

$\hat{z}_k$ is calculated in Equation 3.11

$\hat{x}_k^{-}$ is calculated in Equation 3.8

5. Calculate the Kalman gain:

$$L_k = \Sigma^{-}_{\hat{x}\hat{z},k}\Sigma^{-1}_{\hat{z},k} \tag{3.14}$$

6. The new state estimate (updated with the new measurements) is then calculated as:

$$\hat{x}^{+}_k = \hat{x}^{-}_k + L_k(z_k - \hat{z}_k) \tag{3.15}$$

7. The new error covariance matrix can then be calculated as:

$$\Sigma^{+}_{\hat{x},k} = \Sigma^{-}_{\hat{x},k} - L_k\Sigma_{\hat{z},k}L^T_k \tag{3.16}$$

4 The Proposed Methodology

4.1 Selected cell specification

The cell selected for modelling is a high-performance cylindrical cell in the 26700 package developed by *A123 Systems*. Unfortunately, the exact model and internal chemical specification of the cell is confidential and not disclosed. This cell has been selected, because it is the cell that's going to be used for one of the current projects RA is developing. The developed model and algorithms will thus be deployed for this project.

The cell is designed for high-performance automotive applications, therefore it has a very high operating temperature range, low internal resistance, and a very high maximum discharge current. The cell specification provided by the manufacturer is depicted in Table 4.1:

Table 4.1: A123 26700 cell specification

Parameter	**Value**
Cell Dimensions (mm)	26700
Cell Capacity (nominal, mAh)	3200
Internal Resistance (DC SOC:50%, T:23°C, Typ., $m\Omega$)	2.8
Nominal Voltage (V)	3.65
Max Pulse Voltage (V,max)	4.2
Min Pulse Voltage (V,min)	2.5
Max Disharge Current (A, 10sec)	375
Cycle life (300A 75C Race profile Cycles)	1000 cycles
Operating Temperature (T°C)	-30 to +85
Specific Power (nominal, W/kg)	12400
Power density (nominal, W/L)	29200

4.2 Model data acquisition

In order to discover the ECM parameters, empirical experiments need to be performed on the selected battery cell. The parameters can be classified as *static* and *dynamic*. Thus, two separate experiments need to be performed, where the only static parameter is the OCV-SOC curve at different temperatures, and the rest of the parameters are dynamic.

The static and dynamic cell testing procedures used for this thesis have been designed by Dr. Gregory L. Plett, a professor and researcher at University of Colorado at Colorado Springs (UCCS), and are described in the graduate course *Modeling, Simulation, and Identification of Battery Dynamics* at UCCS. Lecture notes [18] and lectures themselves [24] from this course are available and free of charge on the UCCS web-page. The reason for choosing these particular testing procedures is that they are the current practice at RA. For the cell test equipment, three main components are used:

1. **Battery tester for cell (dis)charging:** Arbin *LBT21084-05̃V-60/5/0.5/0.02A-8CH-220V1P*. Arbin instruments are the industry standard battery tester (cell, module and pack level), used by companies and organizations, such as GE, Tesla, NASA, Ford, Boeing, DELL, MIT, etc. [25]

2. **Temperature and climatic test chamber:** Weisstechnik *VCL 4006*, allowing -70°C to +180°C temperature range. A more detailed specification can be found on the product website [26].

3. **Test computer:** Controlling and monitoring the battery tester and the temperature and climatic test chamber, as well as logging the data.

The thermal chamber is programmatically controlled by the battery tester with communication over *RS232* standard. The battery tester is connected to the test-computer running the testing program and acquiring the data. The test setup is depicted in Figure 4.1.

Figure 4.1: Test setup. Left: thermal chamber. Middle: battery tester. Right: test computer.

The Figure 4.2 depicts the UUT (A123 26700 cell) in the thermal chamber.

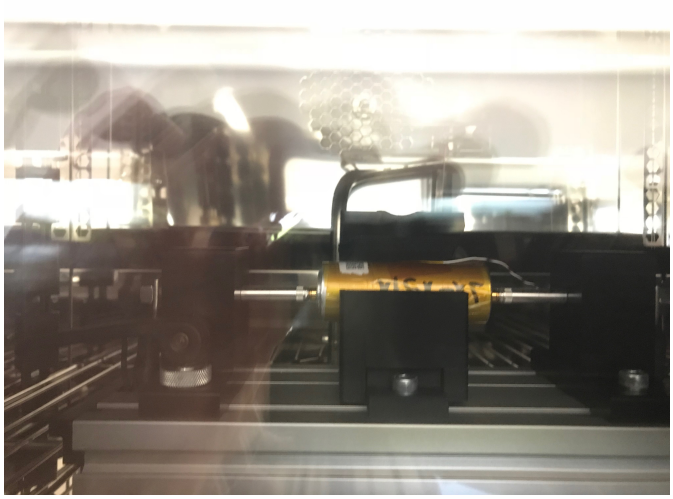


Figure 4.2: Inside the thermal chamber. UUT

4.2.1 Static cell parameter acquisition

Test procedure

The OCV-SOC relation at different operating temperatures is performed by discharging and charging the cell at a small constant current (defined below), and acquiring the

accumulated current (Ah) and cell terminal voltage at discrete time-steps. The test is repeated multiple times at different temperatures. The test procedure is described in Algorithm 2, where reference temperature (RT) is +25°C [1]:

Algorithm 2 Static cell parameter acquisition procedure

(1) Charge the UUT until the maximum defined terminal voltage has been reached.
for *All testing temperatures (TTs)* **do**
(2) Let the UUT temperature reach the TT and soak for at least 2 hours.
(3) Discharge the UUT with a constant-current of *C/30*, until the minimum voltage has been reached.
(4) Let the UUT temperature reach the RT and soak for at least 2 hours.
(5) **if** $V_{cell} < V_{cell_{min}}$ **then**
Charge the UUT at C/30 rate until $V_{cell} == V_{cell_{min}}$.
end
(6) **if** $V_{cell} > V_{cell_{min}}$ **then**
Disharge the UUT at C/30 rate until $V_{cell} == V_{cell_{min}}$.
end
(7) Let the UUT temperature reach the TT and soak for at least 2 hours.
(8) Charge the UUT at C/30 rate until $V_{cell} == V_{cell_{max}}$.
(9) Let the UUT temperature reach the RT and soak for at least 2 hours.
(10) **if** $V_{cell} < V_{cell_{max}}$ **then**
Charge the UUT at C/30 rate until $V_{cell} == V_{cell_{max}}$.
end
(11) **if** $V_{cell} > V_{cell_{max}}$ **then**
Disharge the UUT at C/30 rate until $V_{cell} == V_{cell_{max}}$.
end
end

Coulombic efficiency calculation from acquired data

Since $V_{cell_{min}}$ and $V_{cell_{max}}$ in these tests are calibrated to RT (+25°C) (and SOC is temperature dependant), the integrated current (ampere hours) at other temperatures will differ from the ampere hours at the calibrated temperature. Coulombic efficiency is temperature dependant, and it needs to be calculated for each of the operating temperatures.

The calculation of the Coulombic efficiency at RT is straightforward, because all steps

[1] Cell minimum and maximum voltage is defined in the cell description in the cell specification 4.1.

of Algorithm 2 are performed at the same temperature. The calculation is presented in Equation 4.1:

$$\eta(RT) = \frac{\sum I_{discharged}(RT)}{\sum I_{charged}(RT)} \tag{4.1}$$

However, since the SOC is calibrated to RT, and at temperatures other than RT will be different, we can not be sure that we are using exactly 100% of the SOC. Hence, in the Algorithm 2 we are performing steps (5),(6),(10) and (11) to make sure we use 100% of the SOC. Consequently, the equation for calculating the Coulombic efficiency needs to be adjusted, as described in Equation 4.2:

$$\eta(T) = \frac{\sum I_{discharged}(T) + \sum I_{discharged}(RT)}{\sum I_{charged}(T)} - \eta(RT)\frac{\sum I_{charged}(RT)}{\sum I_{charged}(T)}, \tag{4.2}$$

where

η - cell Coulombic efficiency at a particular temperature,

RT - reference temperature (+25°C),

T - test temperature,

I - current at each time-step.

The calculated $\eta = f(T(^{\circ}C))$ for the UUT is depicted in Table 4.2 below:

Table 4.2: Coulombic efficiency at different temperatures

+10°C	**+25°C**	**+45°C**	**+55°C**	**+65°C**
0.98591798	0.99131714	0.98712135	0.99020027	0.98796274

OCV-SOC calculation from acquired data

Since the SOC is temperature dependant, the output of the static tests will be a group of OCV-SOC look-up tables. At each time-step (*t* is used and it is assumed to increase in discrete steps, because *T* is already used for describing temperature), the DOD (and

consequently the SOC) can be calculated using equations 4.3 and 4.4:

$$\begin{aligned} DOD(t,T) = &\left(\sum_{n=0}^{t} I_{n(discharged)}(T))\right) \\ &- \left(\eta(RT)\sum_{n=0}^{t} I_{n(charged)}(RT)\right) \\ &- \left(\eta(T)\sum_{n=0}^{t} I_{n(charged)}(T))\right) \end{aligned} \tag{4.3}$$

$$SOC(t,T) = 1 - \frac{DOD(t,T)}{Q(T)}, \tag{4.4}$$

where

$Q(T)$ - total cell capacity (Ah) at the test temperature, measured during performing Algorithm 2.

The calculated $SOC = f(OCV, T)$ for the UUT for different temperatures is depicted in Figures 4.3 and 4.4 below:

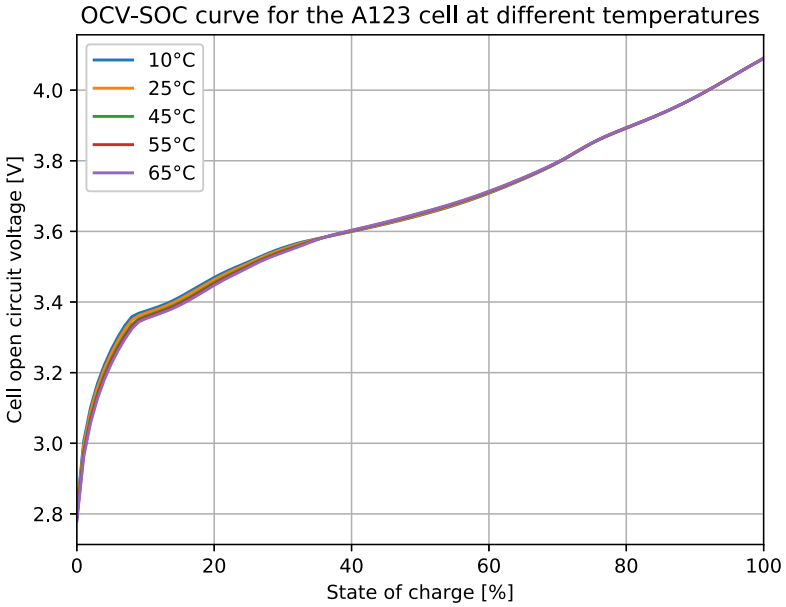


Figure 4.3: UUT OCV-SOC relation at different temperatures

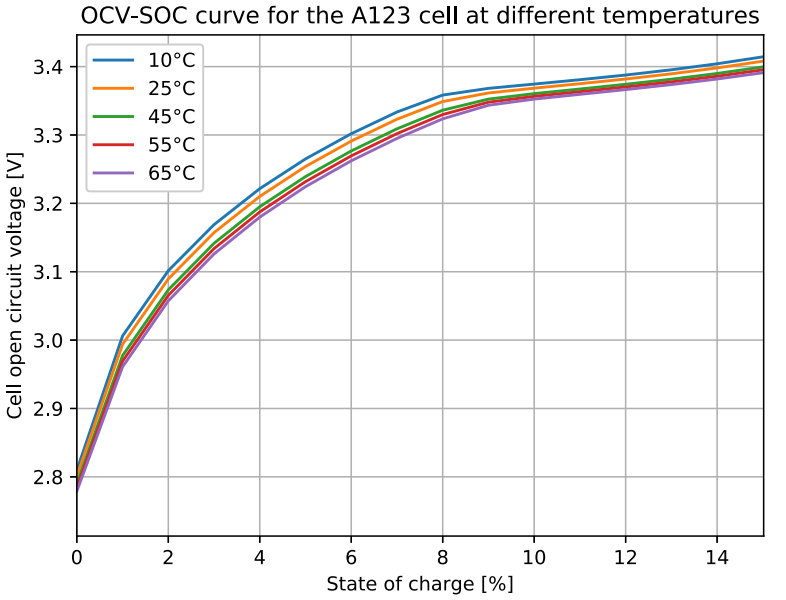


Figure 4.4: UUT OCV-SOC relation at different temperatures: 0% to 15% SOC

It is noteworthy to point out how linear the OCV-SOC curve is in the SOC range from 10% to 100% for this cell.

4.2.2 Dynamic cell parameter acquisition

To acquire the dynamic cell behavior, a requested power profile similar to that of the real-world application must be applied over the entire SOC range and all operating temperatures. The terminal voltage response will determine the cell's behavior in these conditions.

The power profile for use in EPTVs can be acquired by either simulating the particular vehicle, which the battery is going to power, or experimentally acquiring the data in conditions that best describe the operation of the vehicle. For automotive applications, many different *driving cycles* are available. The driving cycle is defined as a series of data points describing the vehicle's speed over time.

In Europe, the most commonly used driving cycle standard is the NEDC, which represents the typical usage of the vehicle with both - urban driving and extra-urban driving. Currently it is used for CO_2, fuel consumption and electric range (in EPTVs) measurements for vehicle regulations.

However, since the project where the cell is going to be used is a high-performance racing vehicle, using *only* NEDC to acquire the dynamic cell behavior would be inappropriate. The drive cycle has been provided by the project owner, and it describes the power profile in racing conditions of an actual test vehicle, making the data very appropriate for acquiring the cell dynamics.

It is also a common practice to extract the cell parameters for different drive cycles, and using different parameters for different *modes* of the car (ECO-mode, Dynamic-mode, Drag-race, Track, etc.). However, in this case only the driving cycle provided by the project owner is used, as incorporating new configurations in the future is relatively trivial once the estimation algorithms are proved to work properly. An example of an acquired *track-mode* drive-cycle is illustrated in Figure 4.5 below:

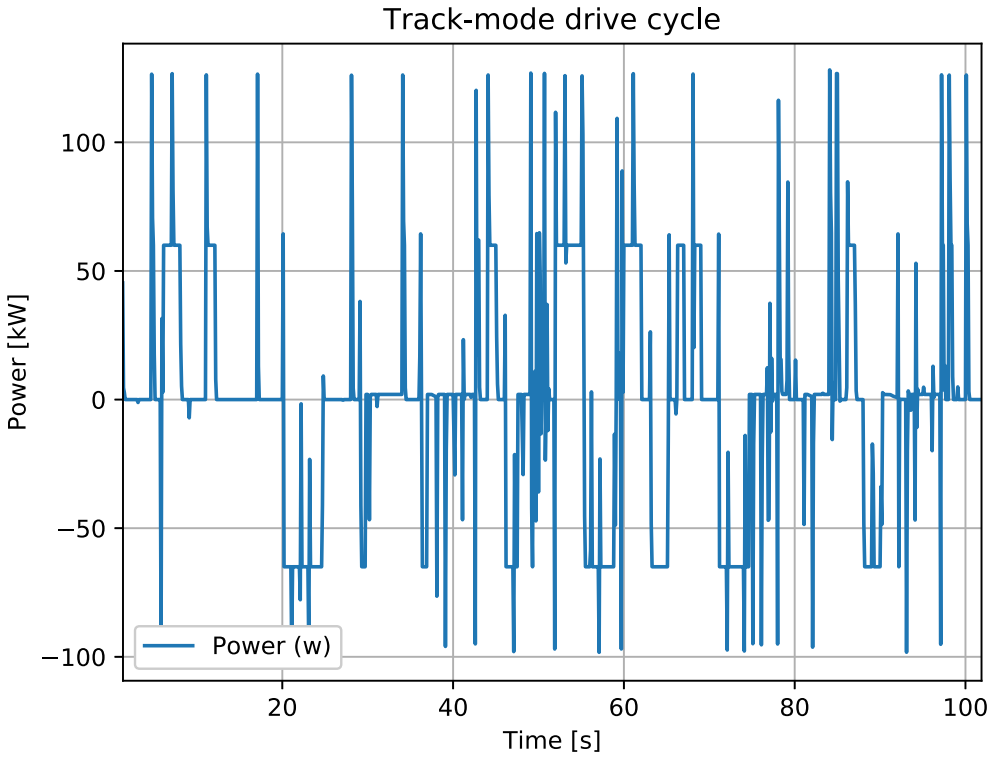


Figure 4.5: Track mode drive cycle: power per cell, first 100 seconds

Test procedure

The dynamic cell test is performed by first (slowly) discharging the cell to 90% SOC to make sure that the cell will not be over-charged in case of dynamic charging (regenerative braking, recuperation), followed by discharging the cell with the drive-cycle profile (RA-HPPC) (repeating the cycle until the SOC reaches 10% to avoid over-discharging the cell), followed by charging the cell back to 100%, acquiring the accumulated current (Ah), cell temperature and cell terminal voltage at discrete time-steps. The test is repeated multiple times at different temperatures. The test procedure is described in Algorithm 3, where reference temperature (RT) is +25°C [2]:

[2]Cell minimum and maximum voltage is defined in the cell description in the cell specification 4.1.

Algorithm 3 Static cell parameter acquisition procedure

(12) Charge the UUT until the maximum defined terminal voltage has been reached.
for *All testing temperatures (TTs)* **do**
(13) Let the UUT temperature reach the TT and soak for at least 2 hours.
(14) Discharge the UUT with a constant-current of *C/1*, until 90% SOC has been reached.
(15) Perform the dynamic power profile on the cell until 10% of SOC has been reached.
(16) Let the UUT temperature reach the RT and soak for at least 2 hours.
(17) **if** $V_{cell} < V_{cell_{min}}$ **then**
Charge the UUT at C/30 rate until $V_{cell} == V_{cell_{min}}$.
end
(18) **if** $V_{cell} > V_{cell_{min}}$ **then**
Disharge the UUT at C/30 rate until $V_{cell} == V_{cell_{min}}$.
end
(19) Charge the UUT at C/1 rate until $V_{cell} == V_{cell_{max}}$.
(20) Charge the UUT with constant-voltage until the current drops below C/30.
end

Now that all of the necessary data has been acquired, it is possible to calculate the rest of the model parameters necessary for solving the ECM state equation (Equations 3.3 and 3.1): R_0, η_k, Q, γ, $R1$ and $C1$.

4.3 State of charge and battery parameter estimator implementation in C programming language

The estimator has been implemented for the latest RA BMS version, which supports management of up to 18 series connected battery cells. The BMS is using MPC5744P [27] 32-bit MCU (running at 200MHz clock frequency) as the main processing and management device, which is based on the Power Architecture® developed by NXP. This micro-controller has been specifically developed with the Automotive industry in mind, suitable for **ISO26262** [28] **ASIL-D** (a risk classification scheme defined by the ISO26262) applications.

The estimator has been implemented as an embedded software module for the existing

embedded software architecture, which does not utilize an operating system, but has a simple, linear 100 task scheduler running at a 100Hz frequency, allowing each task to take up to $100\mu s$.

The embedded software has been hand-written in C, by using the S32DS-PA development environment - *S32 Design Studio IDE for Power Architecture based MCUs*, which is based on Eclipse and internally uses the open-source GNU Compiler Collection (GCC) and GNU Debugger (GDB).

The module has two main public functions - the state estimator initialization function, and the iteration (step) function. The initialization function is depicted in Listing 1, and the functionality is explained in the code below. This function is called by the *BMS_SLAVE_initModules()* function, which initializes all of the modules and attaches the necessary handles to the main RA BMS handle, which holds pointers to all of the main module handles (pointers to structures).

Listing 1 State estimator initialization function C code

```
1 void MDL_SE_init(MDL_SE_handle_S *handle, MDL_CAN_handle_S
  ↪  *canHandle,
2         BatMon_handle_S *batmonHandle, MDL_SE_cell_model_LUT_S
          ↪  *cellModelLUTs, MDL_SE_error_E *err) {
3     //prechecks
4 #ifdef MDL_SE_FUNCT_PRECHECKS_ENABLED
5     //check parameters
6     if(handle == NULL || batmonHandle == NULL || cellModelLUTs ==
       ↪  NULL) {
7         if(err != NULL) {
8             *err = MDL_SE_ERROR_NULL_PARAMENTER;
9         }
10        return;
11    }
12 #endif // #ifdef MDL_SE_FUNCT_PRECHECKS_ENABLED
13
14    if(err != NULL) {
15        *err = MDL_SE_ERROR_NONE;
16    }
17    // assign and initialize the handles and variables
18    handle->batmonHandle = batmonHandle;
19    handle->cellModelLUTs = cellModelLUTs;
20    handle->soc = 0.0f;
21    handle->coulomb_soc = 0.0f;
22    handle->soc_estimation_err_3SD = 0.0f;
23    handle->isSPKFCellInitialized = FALSE;
24    handle->state = MDL_SE_SPKF_GET_PARAMS;
25    handle->canHandle = canHandle;
26    // clear (init) the algorithm-variable structure
27    memset(&handle->spkfAlgVars, 0, sizeof(MDL_SE_SPKF_alg_var_S));
28    // initialize the common SPKF parameters - common for
29    // all cells
30    init_data_SPKF_common(&data_SPKF_com);
31    // initialize current averaging
32    init_rolling_avg(&handle->rollingAvgCurrent);
33 }
```

The main iteration function (step function called in predetermined time intervals) is called by the scheduler every $10ms$ (the period of the scheduler), and each call (in the current scheduler setting) can take up to $100\mu s$, which means that without modifying the scheduler, the iteration function must be split up in multiple parts, or has to use multiple (consecutive) slots in the scheduler (or both). The iteration function is depicted in Listings 2 - 6.

Listing 2 State estimator step function C code. Step 1.

```
void MDL_SE_iter_SPKF(MDL_SE_handle_S *handle, MDL_SE_error_E *outErr) {
    // Extract the pointers for more-readable code.
    MDL_SE_SPKF_handle_S *cell_state = &handle->cell_state;
    MDL_SE_cell_model_LUT_S *cellModelLUTs = handle->cellModelLUTs;
    MDL_SE_SPKF_alg_var_S *spkfAlgVars = &handle->spkfAlgVars;
    MDL_SE_cell_model_S * dynamic_cell_data = &spkfAlgVars->dynamic_cell_data;
    // Enter the state machine
    switch(handle->state) {
        case MDL_SE_SPKF_STEP_1:
            // Test temperature set to 25 degrees Centigrade
            spkfAlgVars->cellTemp = SE_TEST_DEBUG_TEMPERATURE;
            // Get the cell voltage from the battery
            // monitor module. Cell 1 used [index 0]
            spkfAlgVars->cellVoltage = handle->batmonHandle->cellVoltages[0] / 1000.0f; // converted from mV to V
            if(!handle->isSPKFCellInitialized) {
                fp32_t initialSOC = get_soc_from_ocv_lut(spkfAlgVars->cellVoltage, spkfAlgVars->cellTemp, cellModelLUTs);
                init_cell_state(&data_SPKF_com, cell_state, initialSOC);
                // For coulomb counting (to compare)
                spkfAlgVars->accumulatedAh =  initialSOC * cellModelLUTs->QParam[DEBUG_TEMP_LUT_IDX]; // initial estimated Ah from OCV
                handle->isSPKFCellInitialized = TRUE;
            }
            populate_dynamic_cell_data(dynamic_cell_data, cellModelLUTs, spkfAlgVars->cellTemp);
            if(spkfAlgVars->cellCurrent < 0) {
                spkfAlgVars->cellCurrent = spkfAlgVars->cellCurrent * dynamic_cell_data->etaParam;
            }
            if(fabsf(spkfAlgVars->cellCurrent) > dynamic_cell_data->QParam / 100) {
                cell_state->signIk = signum(spkfAlgVars->cellCurrent);
            }
            chol(cell_state->SigmaX, spkfAlgVars->SigmaXa_temp, data_SPKF_com.Nx);

            populate_est_error_cov_matrix(spkfAlgVars->SigmaXa_temp, spkfAlgVars->SigmaXa);
            populate_state_vector(cell_state, spkfAlgVars->xhata);
            handle->state++;
            break;
```

Step 1 of the iteration function sets the main variables for the SPKF: cell temperature, cell voltage, dynamic (temperature dependant) cell model data and the cell current (not in the code here, as it is updated using a CAN callback function *outside*). It also makes the initial state prediction, based on the OCV-SOC LUT, if the filter is being run for the first time, and performs the Cholesky decomposition, as explained in equation 3.5, and prepares the data for the next step (populates the necessary arrays).

Listing 3 State estimator step function C code. Step 2.

```
case MDL_SE_SPKF_STEP_2:
    calculate_sigma_points(spkfAlgVars->xhata,
      ↪ spkfAlgVars->SigmaXa, spkfAlgVars->Xa, L, L);
    sigma_point_state_fn(spkfAlgVars->Xa, cell_state->priorI,
      ↪ dynamic_cell_data,
          spkfAlgVars->Xa_new);
    state_prediction_time_update(spkfAlgVars->Xa_new,
      ↪ spkfAlgVars->xhat_new);
    calculate_mean_deviation(spkfAlgVars->Xa_new,
      ↪ spkfAlgVars->xhat_new, spkfAlgVars->Xs,
    NR_OF_STATES);
    error_cov_time_update(spkfAlgVars->Xs,
      ↪ spkfAlgVars->SigmaX_new);
    handle->state++;
    break;
```

Step 2 calculates the sigma points (equation 3.5), and propagates them through the SPKF state equations, resulting in a new set of points. After that, the state prediction time update is calculated as the weighted average of the sigma points. In linear algebra, this is done with a simple matrix multiplication, given that the weight matrix is diagonal. Following, the error covariance time update is calculated by performing matrix multiplication of the mean deviation of the output sigma points with the diagonal covariance weight matrix, and the result of this operation is then used to perform another matrix multiplication with the transposed mean deviation matrix.

Listing 4 State estimator step function C code. Step 3.

```
case MDL_SE_SPKF_STEP_3:
    system_output_prediction_fn(cell_state, cellModelLUTs,
     ↪ spkfAlgVars->Xa_new,
        spkfAlgVars->cellCurrent, &spkfAlgVars->Xa[(1 + 2 * L)
         ↪ * (NR_OF_STATES + 1)],
        spkfAlgVars->cellTemp, dynamic_cell_data,
         ↪ spkfAlgVars->Y_outEst);
    handle->state++;
    break;
```

Step 3 propagates the new sigma points (after passing them through the state function) through the system output function to predict the new system output. This is the most computationally heavy function in the whole SPKF algorithm.

Listing 5 State estimator step function C code. Step 4.

```
case MDL_SE_SPKF_STEP_4:
    output_prediction_time_update(spkfAlgVars->Y_outEst,
     ↪ &spkfAlgVars->yhat);
    calculate_mean_deviation(spkfAlgVars->Y_outEst,
     ↪ &spkfAlgVars->yhat, spkfAlgVars->Ys, 1);
    cov_XY(spkfAlgVars->Xs, spkfAlgVars->Ys, spkfAlgVars->SigmaXY);
    cov_Y(spkfAlgVars->Ys, &spkfAlgVars->SigmaY);
    calculate_kalman_gain(spkfAlgVars->SigmaXY,
     ↪ spkfAlgVars->SigmaY, spkfAlgVars->L_gain);
    handle->state++;
    break;
```

Step 4 performs the output prediction time update, which predicts the new system output (the cell terminal voltage) by calculating the weighted mean of the input points. After computing the necessary covariance matrices, the estimator gain matrix (Kalman gain) is also calculated in this step.

Listing 6 State estimator step function C code. Step 5.

```
        case MDL_SE_SPKF_STEP_5:
            state_estimate_measurement_update(
            spkfAlgVars->xhat_new,
            spkfAlgVars->L_gain,
            (spkfAlgVars->cellVoltage - spkfAlgVars->yhat));
            error_cov_measurement_update(spkfAlgVars->SigmaX_new,
              ↪  spkfAlgVars->L_gain,
                    spkfAlgVars->SigmaY);
            //  Save data in spkfData structure for next time
            cell_state->priorI = spkfAlgVars->cellCurrent;
            uint32_t i;
            for(i = 0; i < (NR_OF_STATES * NR_OF_STATES); i++) {
                cell_state->SigmaX[i] = spkfAlgVars->SigmaX_new[i];
            }
            for(i = 0; i < (NR_OF_STATES); i++) {
                cell_state->xhat[i] = spkfAlgVars->xhat_new[i];
            }
            handle->soc = MDL_SE_get_SOC(cell_state);
            handle->soc_estimation_err_3SD =
              ↪  MDL_SE_get_estimation_error_3SD(cell_state);
            canTxDBGStatusPXI.B.stateOfCharge = (uint32_t)
              ↪  (handle->soc * 10000);
            canTxDBGStatusPXI.B.socEstimationErr3SD = (uint32_t)
              ↪  (handle->soc_estimation_err_3SD
                    * 10000);
            coulomb_counting_soc(handle, outErr);
            canTxDBGStatusCoulomb.B.stateOfCharge = (int32_t)
              ↪  (handle->coulomb_soc * 10000);
            canTxDBGStatusCoulomb.B.cellCurrent_mA = (int32_t)
              ↪  ((spkfAlgVars->cellCurrent * 1000.0f));
            handle->state = MDL_SE_SPKF_STEP_1;
            break;
        default:
            break;
    }
}
```

Step 5 performs the state estimate measurement update. The new state vector is calculated by adding the measurement error (cell terminal voltage measurement minus estimate) multiplied by the Kalman gain to the state vector estimate from the previous SPKF iteration. After that, the error covariance matrix measurement update is also performed. This step also overrides the previous state estimate and error covariance matrices with the new values, and sets the new SOC, and populated the CAN interface data with the new values. Now that the last step has been executed, the iteration func-

tion goes back to step 1, and this procedure is repeated indefinitely.

In the first, non-optimized version of the code, it has been calculated that running the SPKF for one cell with the First-order equivalent-circuit cell model once per scheduler cycle ($10ms$) takes up **21** scheduler slots. Adding the second **RC** node and hysteresis would increase this number further. However, to validate that the algorithm works and performs as expected, this is sufficient, as we can run the filter at $50ms$ period by splitting the function into a linear state-machine with five states and run it only for one cell.

However, the final version of the code must be optimized for speed, and the SPKF must be tuned to best fit the application, which can be done offline by running the filter with actual cell data (temperature, current and terminal voltage) acquired from a real driving test in a high-level programming language such as, for example, Matlab or Python, and running an optimization method to find the best fit of the model for the particular application. Thus, by fitting the parameters for different drive cycles, different driving modes (like *ECO/City, Sport, Track*) can be developed, changing the SOC estimator behavior depending on driving style.

These and other steps for improving the algorithm are further discussed in the Section 6.2, in which an overview on the future work and possible improvements is conducted.

4.4 State of charge estimator validation test

4.4.1 Introduction

The purpose of this test is to validate the SPKF SOC estimation algorithm on the RA BMS for a single A123 cell (Section 4.1) at a constant temperature. Because of how long the static and dynamic tests take to execute (for example, the static tests take 70h of testing and data acquisition for one temperature, which yields a total of 700h of testing time for 10 temperatures to get a decent amount of sample points for interpo-

lation), currently only $+25°$C temperature (controlled in the thermal chamber) is used for validation of the algorithm, which, naturally will be extended to other temperatures when all data is acquired and the operation of the algorithm at $+25°$C determined to be satisfactory.

4.4.2 Test equipment

The equipment necessary to perform the validation tests can be summarized:

1. **Data acquisition computer** - used for logging CAN data from the RA BMS (SOC, estimation error, current, cell voltage);
2. **TENMA 72-10480 Digital-Control DC power supply 0-30V 3A** - set to +12V and 1A, used for powering the RA BMS and the IVT-S current sensor;
3. **RA BMS** - used for cell voltage and temperature measuring, and running the SPKF algorithm. Sending the estimated SOC and error bounds (three standard deviations) over CAN to the data acquisition computer;
4. **A123 cell** - test subject;
5. **ARBIN cell tester** - used for providing the load profile to the cell, as well as measuring the current, temperature and terminal voltage of the cell;
6. **Temperature and climatic test chamber:** - used for controlling and monitoring the cell temperature.
7. **IVT-S current sensor** [29] - the actual current sensor used in the final application of the RA BMS. Used for providing the cell current to the RA BMS over CAN.
8. **Vector VN1610 CAN interface** [30] - used for receiving CAN data from the RA BMS for logging and data analysis.

9. **Miscellaneous cables, connectors and tools** - cell holder, high-precision voltage measurement needles (used for feeding back the cell terminal voltage to the Arbin cell tester), wires.

4.4.3 Test setup

The test setup block diagram is visualized in Figure 4.6:

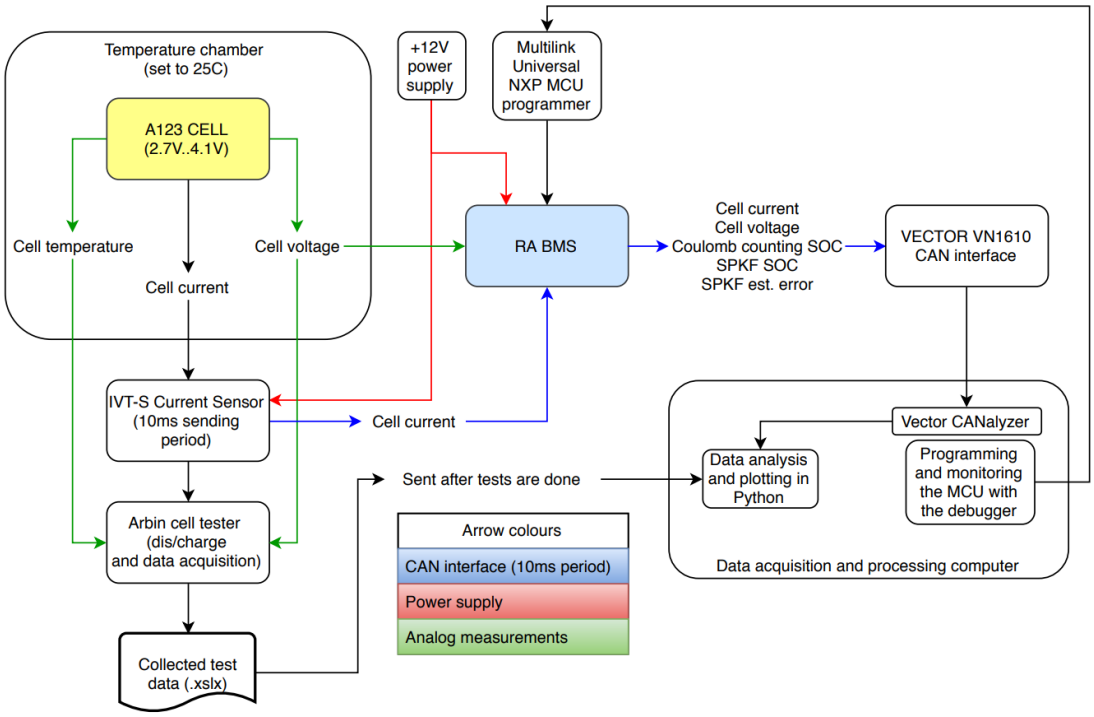


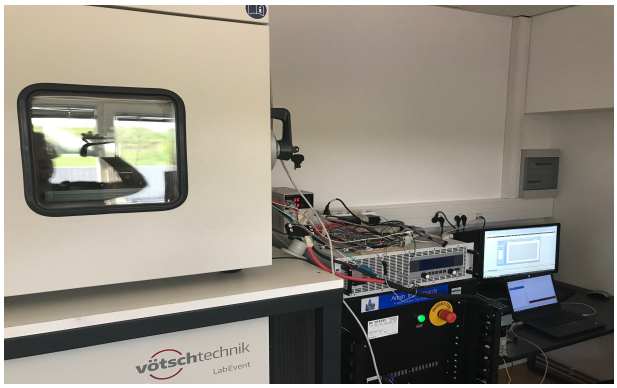
Figure 4.6: Validation test setup block diagram

The cell is placed in a custom cylindrical cell holder (as shown in Figure 4.2) with special spring-pins that make a very good solder-less connection with the cell's terminals for precise voltage measurements. A temperature sensor is also connected to the cell for feedback. The cell in the holder is then placed in the temperature and climatic test chamber, with the temperature set to +25°C. The cell terminal voltage measurement pins are connected to channel 0 of RA BMS, as well as to the Arbin cell tester. The IVT-S current sensor is connected in series with the load (positive side) - the Arbin cell

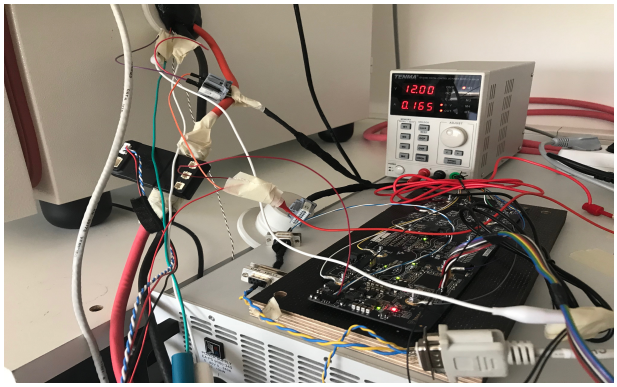
tester; it is also connected to the +12V power supply and the RA BMS CAN interface.

The RA BMS is powered by the same +12V power supply as the IVT-S current sensor. The RA BMS CAN connector (DB9) is connected to the Vector VN1610 CAN interface, which sends the CAN data over USB 2.0 to the data acquisition and processing computer, where all of the CAN messages are logged with the Vector CANalyzer software. The BMS is sending three (relevant to this test) CAN messages with a period of 10ms. CAN message ID 0x111 contains the cell terminal voltage, message 0x26 contains the coulomb-counting SOC and the IVT-S current (resolution: 1mA), and the message 0x27 contains the SPKF SOC and the SOC estimate error bounds. The RA BMS is also connected to the data acquisition and processing computer with the Multilink Universal NXP MCU programmer for real-time debugging of the embedded software.

The Arbin cell tester is running the pre-programmed cell load profile and logging the temperature, cell terminal voltage and current (internal, high precision and accuracy current sensor, not IVT-S). The data is stored in a SQL Database, which after the test is exported to MS Excel (.xslx) and sent to the data acquisition and processing computer for analysis and plotting. The physical test setup is illustrated in Figure 4.7:



(a) The whole test setup



(b) RA BMS and current sensor

Figure 4.7: Validation test setup

4.4.4 Test parameters

Tests with multiple different load profiles are performed within the same testing framework in order to see the behavior of the state estimator in different conditions, and can be summarized in a list:

1. **RA-HPPC-60-1:** A hybrid pulse power characterization test profile designed at RA. Initial pulse is set to 60A, where each pulse lasts 0.5s, and each next pulse is with a half of the amplitude of the previous pulse. After the pulses have been performed, there is a 60s rest period, after which a constant 1A current discharge is performed for 10min, after which, the cell is at rest (0A current) for 15 minutes and the cycle is repeated until the cell terminal voltage reaches 2.7V. Depicted in Figure 4.8 and Figure 4.9 (zoomed for one pulse cycle).

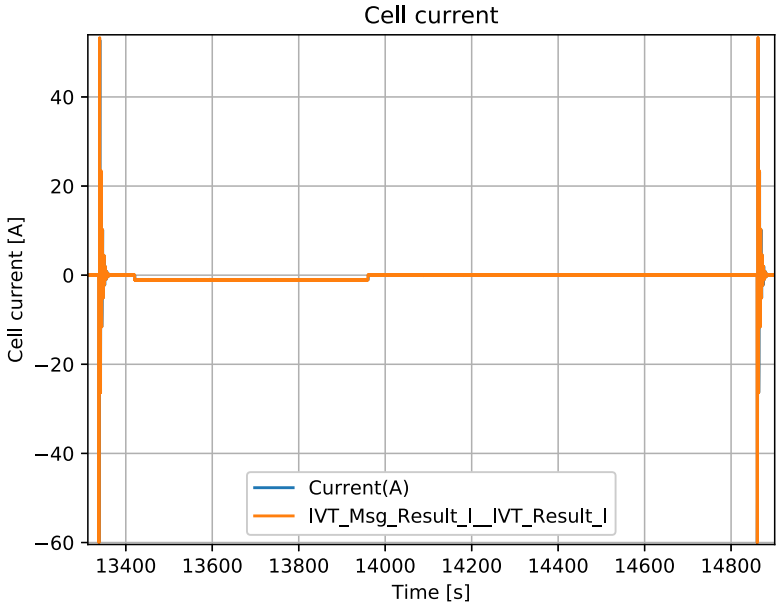


Figure 4.8: RA-HPPC load profile

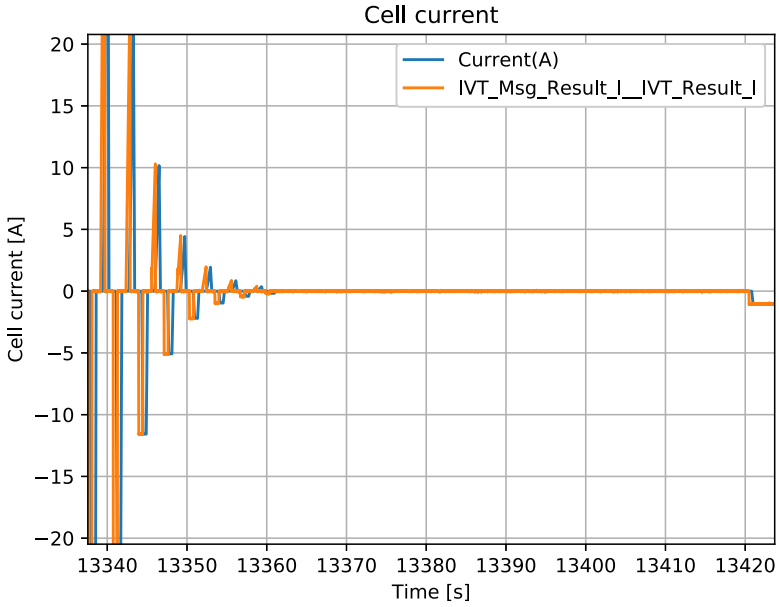


Figure 4.9: RA-HPPC load profile one pulse cycle.

2. **Constant 60A discharge from 100% SOC to 2.7V cell terminal voltage.**

4.4.5 Acquired data formatting

Since the data acquired from the tests comes from two different, time non-synchronized sources, some data processing needs to be performed in order to align the time-base, plot the results and evaluate the performance. The CAN data acquired with the Vector VN1610 CAN interface on the data acquisition computer through the Vector CANalyzer software has been exported as a .mat Matlab formatted data file, with the CAN data synchronized in time with a sample rate of 10ms. The data acquired by the battery tester is by default exported as an .xslx MS Excel file with event-triggered time-base. A Python script has been written that uses the flexible, open-source *pandas* [31] data analysis and manipulation package. The script is depicted in Listing 7 and 8, and the code is documented and explained within the listings.

Listing 7 Data analysis Python script (part 1)

```
# Import all packages
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
import pandas as pd
from sklearn.metrics import mean_squared_error

FULL_CAPACITY_25 = 2.8910473 # Ah
CURRENT_SPIKE_LIM = 10
def mat_to_pandas_df(filename, time_key, exclude_list):
    mat = loadmat(filename)  # load mat-file
    # Return a Pandas dataframe with the selected index
    return pd.DataFrame({key:mat[key].flatten() for key in mat if
      ↪  key not in exclude_list},index=mat[time_key].flatten())
# Get the Cell tester data
arbin_df = pd.read_excel(r"tester_data.xlsx",
  ↪  sheetname='Channel_1_1')
arbin_df = arbin_df.dropna(axis=1, how='all') # drop all columns
  ↪  with NaN's
arbin_df.index=arbin_df['Test_Time(s)'].round(2) # round to two
  ↪  decimal places (10ms resolution)
arbin_df_last_idx = arbin_df.index[-1]
arbin_data_fields = arbin_df.columns.tolist()
# Get the CAN data
filename = r"can_data.mat"
exclude_list = ["Time"]
can_df = mat_to_pandas_df(filename, "Time", exclude_list)
can_data_fields = can_df.columns.tolist()
```

Listing 8 Data analysis Python script (part 2)

```
# Merge the two dataframes, interpolate the empty spaces from the cell tester data (NaNs)
df3=can_df.merge(arbin_df, how='outer', right_index=True, left_index=True).interpolate(method='linear')
# Scale the current from mA to A for the CAN data. Correct the current sign
df3['DBG_Cell_State_Coulomb__DBG_Current_mA_IVT_S'] = df3['DBG_Cell_State_Coulomb__DBG_Current_mA_IVT_S']/-1000
df3['IVT_Msg_Result_I__IVT_Result_I'] = df3['IVT_Msg_Result_I__IVT_Result_I']/-1000
# Resync time to the best of our abilities
# Find the first occurence of a current spike
first_spike_idx_diff = np.argmax(df3['Current(A)'].values<-CURRENT_SPIKE_LIM) - np.argmax(df3['DBG_Cell_State_Coulomb__DBG_Current_mA_IVT_S']
.values<-CURRENT_SPIKE_LIM)
# Shift the cell tester data in time
for column in df3:
    if column in arbin_data_fields:
        df3[column] = df3[column].shift(periods=-1*(first_spike_idx_diff))
# Remove all rows after the last row for which we have the Cell tester data (as we shifted it up in time)
df3 = df3.drop(df3.index[np.argmax(df3.index==arbin_df_last_idx - first_spike_idx_diff):])
# Calculate the SOC and error bounds, and put it into the dataframe
df3['SOC'] = 100 * (1 - (df3['Discharge_Capacity(Ah)'] - (df3['Charge_Capacity(Ah)']) / FULL_CAPACITY_25))
df3['err3sd_pos'] = df3['DBG_Cell_State_PXI__DBG_StateOfCharge'] + df3['DBG_Cell_State_PXI__DBG_SOCEstimationError3SD']
df3['err3sd_neg'] = df3['DBG_Cell_State_PXI__DBG_StateOfCharge'] - df3['DBG_Cell_State_PXI__DBG_SOCEstimationError3SD']
rms_err = np.sqrt(mean_squared_error(df3['SOC'], df3['DBG_Cell_State_PXI__DBG_StateOfCharge']))
# Plot everything (for example, absolute error)
plt.figure()
(df3['SOC'] - df3['DBG_Cell_State_PXI__DBG_StateOfCharge']).abs().plot()
plt.ylabel('Absolute error', size=14)
plt.title('Absolute SOC error over time', size=14)
plt.xlabel('Time [s]', size=14)
plt.locator_params(nbins=20, axis='x')
plt.locator_params(nbins=20, axis='y')
plt.grid()
```

5 Results

5.1 Validation test 1

First of all, to make sure that the voltage sensor works properly, and the time-base of the two different data sources is aligned, the cell terminal voltage measured by the battery tester has to be visually compared with the voltage measured by the RA BMS. In this case the evaluation of the sensor precision and accuracy relative to the battery tester is not important, as sensor noise is accounted for by the state estimator. The only concern here is whether it follows the same trend within reasonable bounds, so that it can be said that the sensor can be *trusted*, in case the state estimator doesn't converge with reality and starts drifting. Figure 5.1 depicts the cell terminal voltage over the whole test time, and Figure 5.2 zooms in on one current spike cycle.

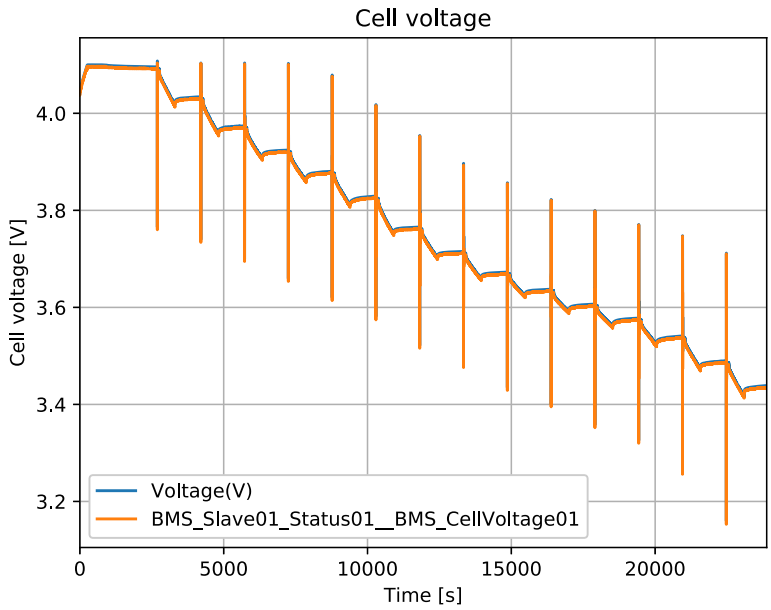


Figure 5.1: Cell terminal voltage. Blue line - voltage measured by the cell tester, orange line - voltage measured by the RA BMS

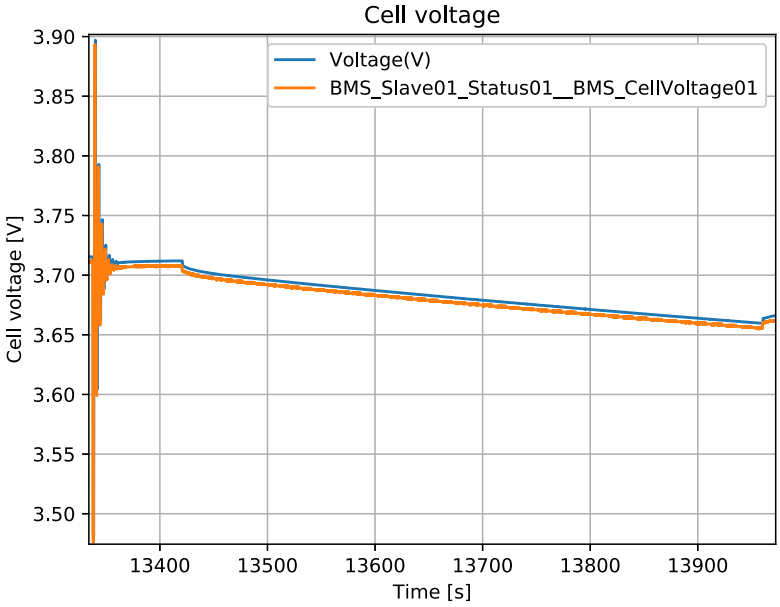


Figure 5.2: Cell terminal voltage zoomed in for one spike cycle. Blue line - cell tester, orange line - RA BMS

By means of visual inspection, it can be concluded that the voltage sensor works as

expected, and even though there is a small constant voltage difference (less than 5mV) in amplitude measured by the two systems (which can be thought as a DC bias sensor error), improving the sensor accuracy and precision is out of the scope of this thesis, and the voltage sensor provides *reasonable* measurements. Figure 5.3 depicts the estimated SOC over the whole test time, and Figure 5.4 zooms in on one RA-HPPC cycle. Figure 5.5 depicts the absolute error over time, and Figure 5.6 depicts percent error over time.

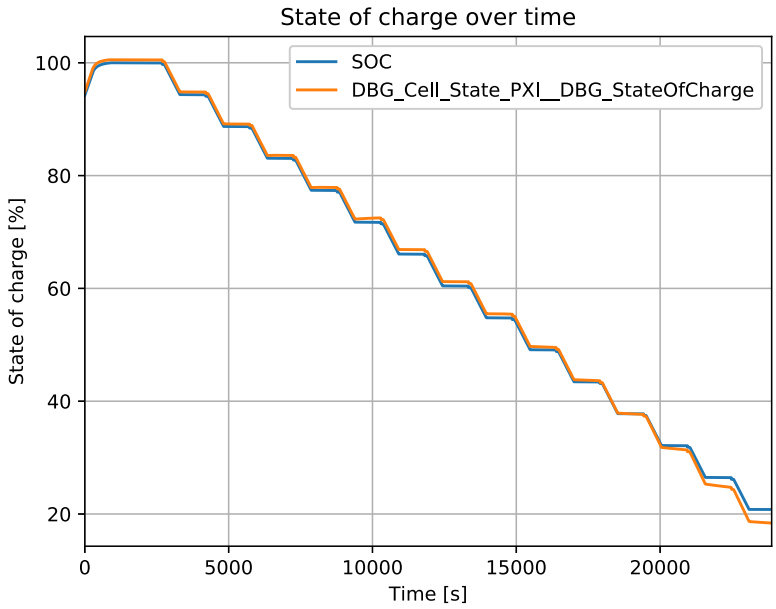


Figure 5.3: Cell SOC. Blue line - reference SOC, orange line - estimated SOC

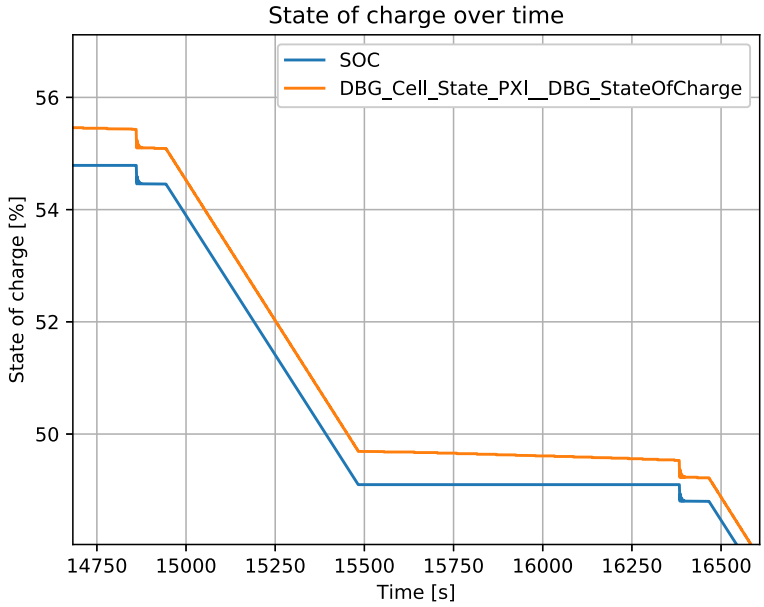


Figure 5.4: Cell SOC zoomed in for one spike cycle. Blue line - reference SOC, orange line - estimated SOC

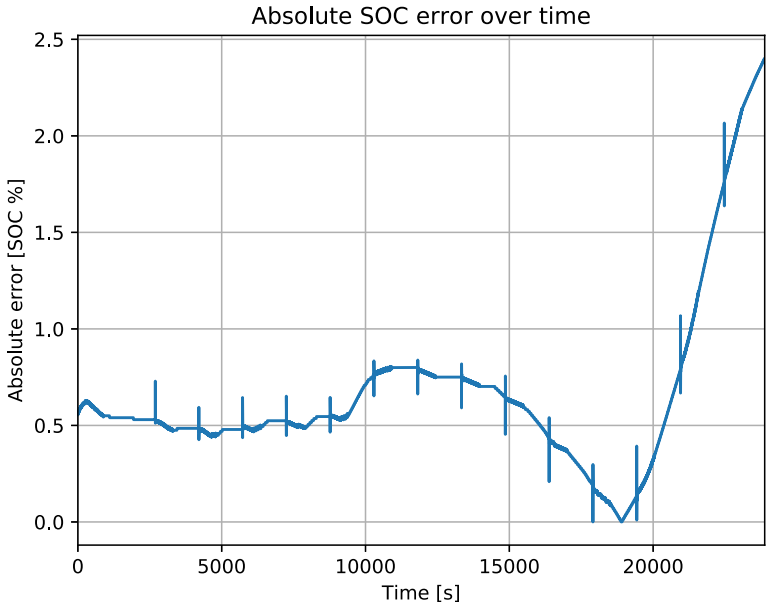


Figure 5.5: Absolute SOC estimation error

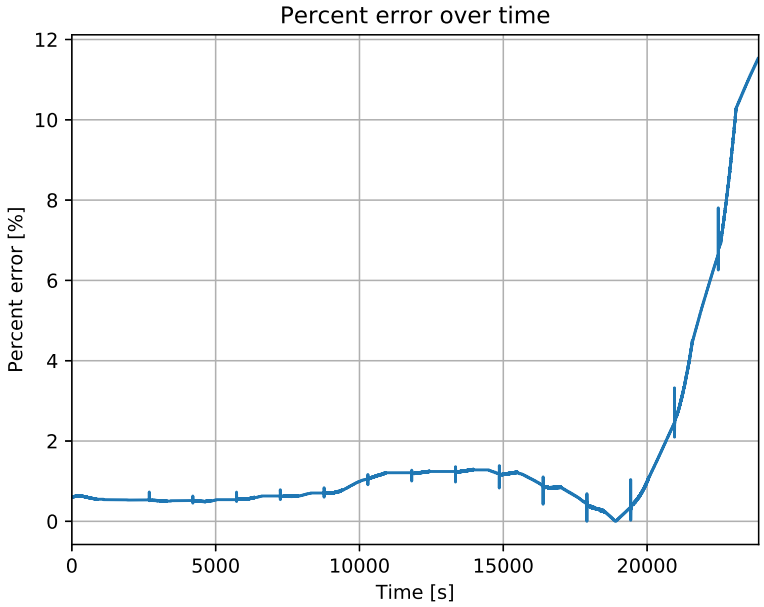


Figure 5.6: SOC estimation percent error

From these figures, it can be concluded that the SOC estimation works as expected, and, even though, the SOC estimation error increases, as the SOC decreases, that is to be expected, as the model tuning parameters have initial *reasonable* values and have not yet been tuned for the particular setup. The main point here is,- the estimator performs well and does not *unreasonably* diverge from reality.

5.2 Validation test 2

The second test involves discharging the cell from 100% SOC to $2.7V$ cell terminal voltage at a constant current of $60A$ at $+25°C$ [1]. The acquired data has been formatted in a similar matter as described in Section 4.4.5. The results are illustrated in the figures below.

[1]The cell will heat up to a higher temperature at this current, however, since the dynamic data for all selected temperatures has not yet been acquired, the filter will assume $+25°C$, which will yield in worse results than if the filter had all the data.

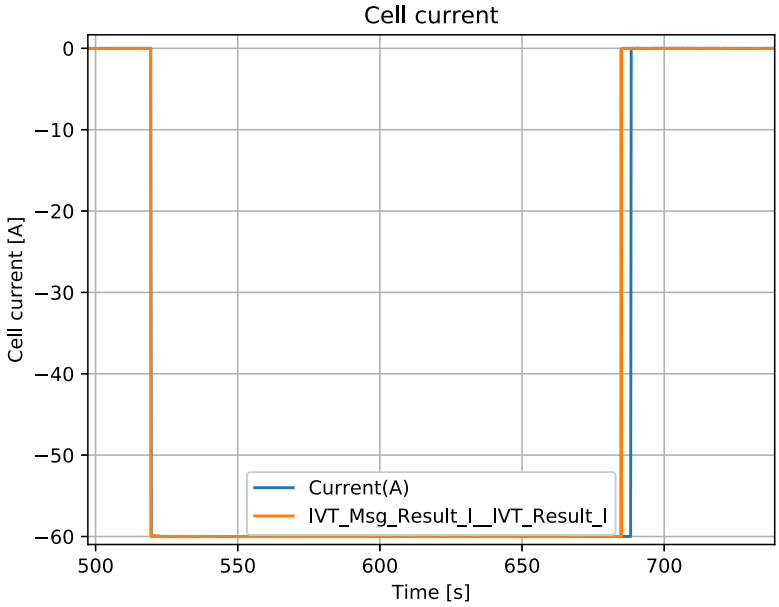


Figure 5.7: Cell current over time. Blue line - cell tester, orange line - BMS

As in validation test 1, the two data sets are aligned with respect to the first current spike. The reason why the current reported by the cell tester is at 60A for a few seconds longer is the way the data is sampled (lower sample rate than the BMS measurement).

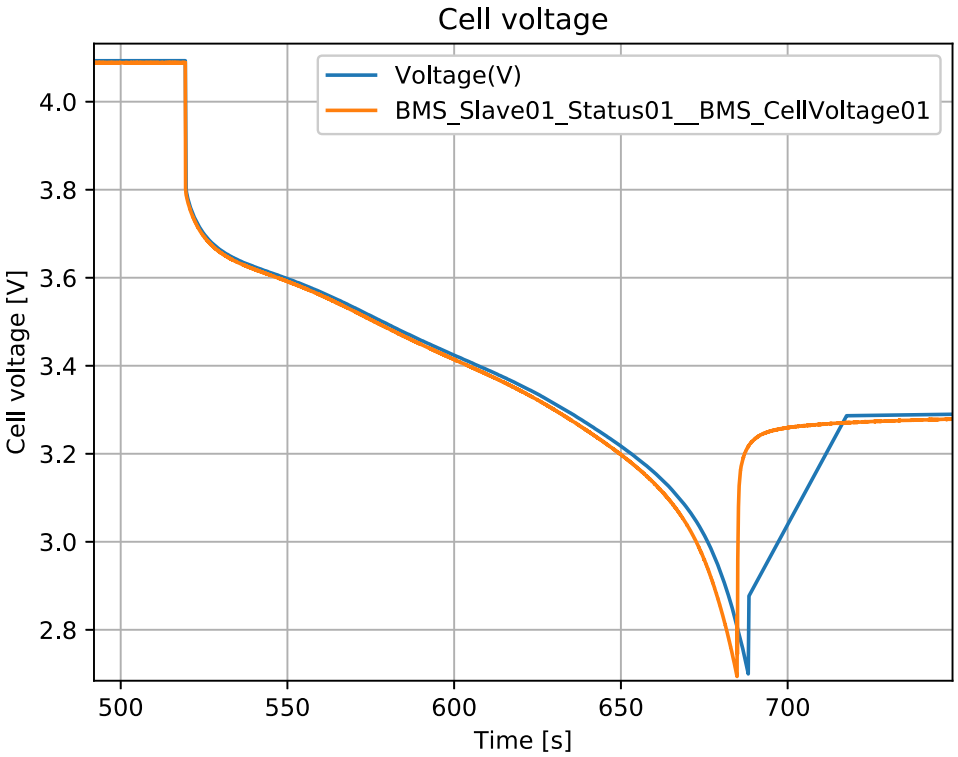


Figure 5.8: Cell terminal voltage over time. Blue line - cell tester, orange line - BMS

Same behavior can be noticed in the cell terminal voltage graph - the voltage measured by the cell tester sharply changes from 2.85V to 3.3V because of the lower sample rate, whereas the BMS sends the voltage measurement over CAN every 10ms. However, both devices provide *close enough* results to trust the BMS measurement.

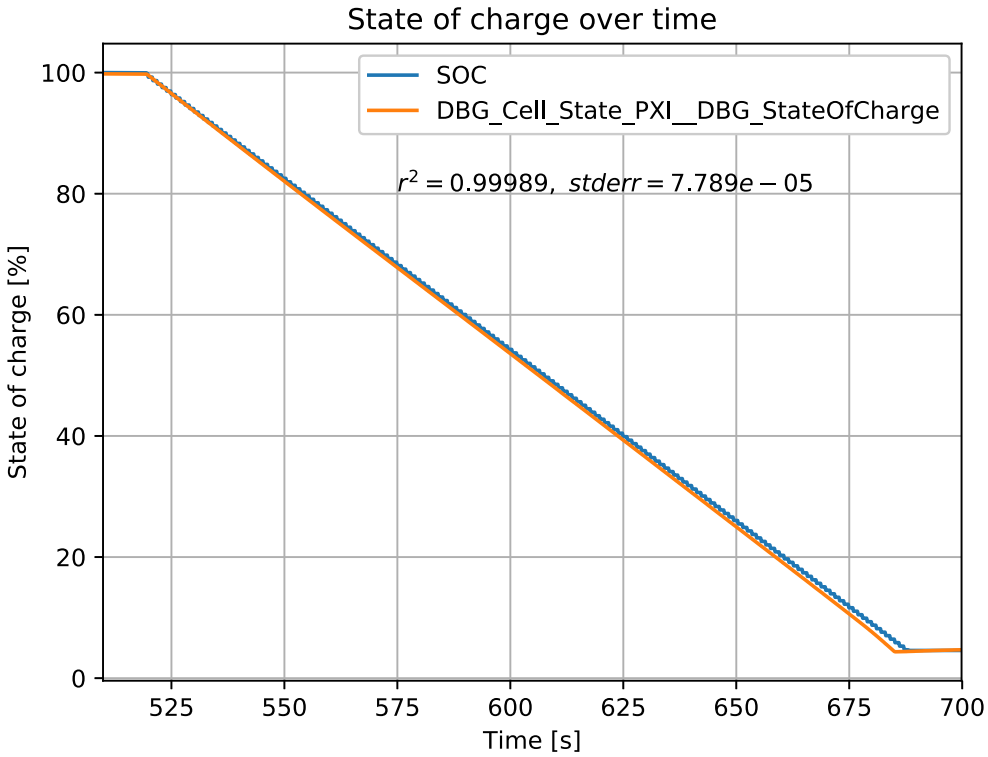


Figure 5.9: Cell state of charge. Blue line - cell tester, orange line - BMS

The state of charge over time graph indicates that the SOC estimation algorithm works very well. Calculating cross-correlation between the two data sets yields in 0.99386, which is a very good result. Calculating the RMSE for our estimate yields in 0.73886, whereas the RMSE for the SOC estimate with Coulomb counting yields in 1.598, which is quite a considerable difference in favour of the SPKF approach.

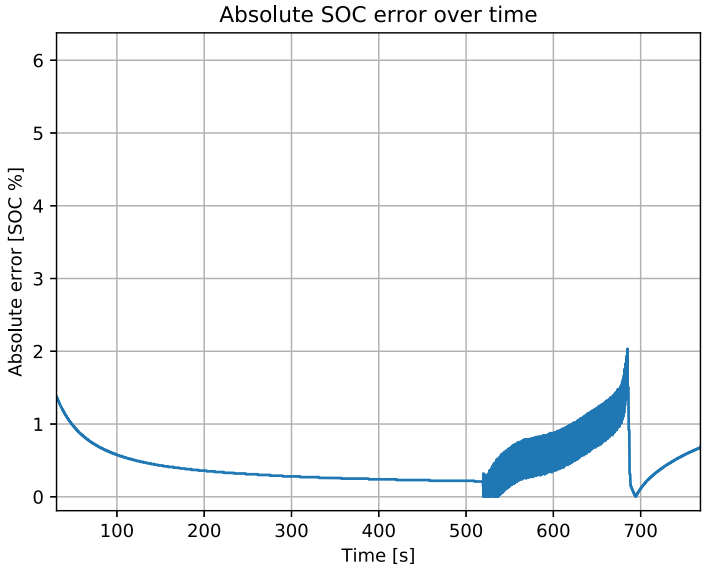


Figure 5.10: SOC estimation absolute error

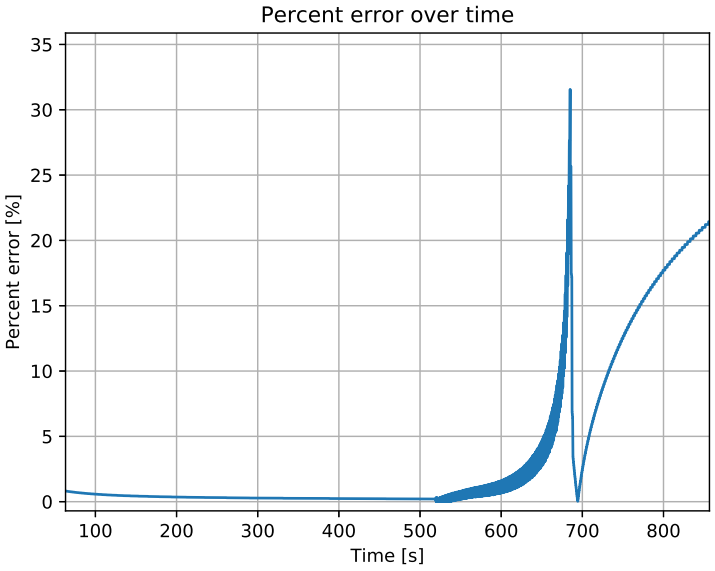


Figure 5.11: SOC estimation percent error

The error in estimate still increases as the SOC decreases, as expected. The percent error in test 2 is considerably larger than in test 1, because test 2 actually approaches 0% SOC.

6 Conclusion and Future Work

6.1 Conclusion

The main goal of this thesis, which was to develop a flexible and precise real-time battery cell state of charge estimation algorithm that can be executed as a software module on the Rimac Automobili battery management system and performs better than the existing state of charge estimation methods at Rimac Automobili, has been reached successfully. In the tested conditions, the selected non-linear central difference sigma-point Kalman filter state estimation algorithm, in conjunction with the equivalent-circuit battery cell model, provides better results than the mostly predominant Coulomb counting method, which diverges over time and has to be re-calibrated often to provide relatively stable results, and gives no information on the uncertainty of the estimate.

To reach this goal, an overview of the important terms, components, concepts and theories has been made, and the optimal SOC estimation algorithm for the defined use case has been discussed and selected. Experiments to acquire the battery cell behavior under static and dynamic conditions for the selected cell have been conducted, and the equivalent-circuit cell model parameters needed for the SOC estimation algorithm have been calculated. The embedded programming code, which integrates into the existing battery management system's software architecture, has been implemented, and the system has been operating continuously for more than two weeks without showing signs of divergence from truth or instability.

After acquiring the static and dynamic battery cell parameters for all other temperatures within the operating range of the system, tuning the sigma-point Kalman filter parameters, extending/optimizing the embedded software module to work with all 18 (series) cells for the particular project (while at the same time performing all of the other necessary functions) and performing extended testing in real-life conditions, the developed state of charge and battery cell parameter estimation algorithm will be deployed on a real-world application, which is a series production high-performance hybrid-electric vehicle.

6.2 Future Work

6.2.1 Static and dynamic cell parameter acquisition

In order to validate the cell model, the static and dynamic cell parameters need to be acquired for multiple cells in the exact same conditions, running in parallel at the same time. This is necessary to eliminate the fact that the cell, from which the parameters are acquired, could potentially be faulty or in any way divergent from the *average* cell within the batch, yielding completely inaccurate results.

6.2.2 Embedded programming code optimization

The main BMS MCU is capable of executing SIMD instructions, which allows the processor to perform the same instruction on multiple data points at the same time This means that the computationally time-consuming matrix (array) operations (Cholesky decomposition, matrix inversion, matrix multiplication etc.) can be parallelized, reducing the computational units required to run the estimator. This would mean re-writing the mathematical operation functions using intrinsic *Assembly* code instructions within the C code (because of the *closed-source* nature of the automotive software components, there are no available libraries currently provided by NXP (MCU manufacturer)).

6.2.3 Cell model optimization

Each battery cell behaves differently, hence each application will have different filter settings, namely, the number of RC nodes in the equivalent series circuit and hysteresis. The same goes for different driving modes. In race-track driving conditions the cell will behave differently to stimuli, as compared to calm city-driving. The optimal settings for a particular application can be acquired by measuring the current, temperature and cell terminal voltages in these conditions, and building the model from there for each mode (the mode can often be selected in high-performance cars either on the steering-wheel or the dashboard). Additionally, the optimal number of RC nodes and whether or not modeling hysteresis is necessary can be estimated by running an optimization algorithm on the state estimator in various conditions and various settings, and measuring performance and computational complexity, and then deciding on the best settings based on the desired computational cost versus performance. This does not need to be executed on the real-time embedded system, in fact, that would be unpractical. Instead, the state estimator must be implemented in a high-level programming language (like, for example, *Python* or *Matlab*), validated to work exactly as the estimator running on the embedded system, and optimized from there.

6.2.4 Testing

In order to deploy the software module on the real-world application, extensive testing both, in a laboratory (simulated drive-cycles), and in field (real drive-cycles), has to be conducted and compared against the existing state of charge estimation algorithm.

Acknowledgements

I would like to thank my supervisors Gholamreza Anbarjafari and Egils Avots for advice and guidance during the writing of this thesis.

/ Kristaps Dreija on 20.05.2018 /

References

[1] "Model s bms hacking," 2017, https://hackaday.io/project/10098-model-s-bms-hacking.

[2] G. L. Plett, *Battery Management Systems Volume I. Battery Modelling*. Artech House, 2015, vol. I.

[3] "Visual representation of soc," 2016, https://www.linkedin.com/pulse/accutronics-adopt-30-state-charge-shipping-li-ion-1st-michele-windsor/.

[4] "Ocv-soc curve for lifepo4 cell," 2015, https://www.researchgate.net/figure/SOC-versus-OCV-Curve-for-an-18650-Lithium-Iron-Phosphate-LFP-Battery-Cell_fig1_304655337.

[5] R. van der Merwe; Eric Wan, "Sigma-point kalman filters for probabilistic inference in dynamic state-space models," pp. 4–8, 2004, http://www.gatsby.ucl.ac.uk/~byron/nlds/merwe2003a.pdf.

[6] G. L. Plett, *Battery Management Systems Volume II. Equivalent-Circuit Methods*. Artech House, 2016, vol. II.

[7] D. G. L. Plett, "Nonlinear kalman filters," pp. 1–29, 2016, http://mocha-java.uccs.edu/ECE5550/ECE5550-Notes06.pdf.

[8] N. Committee, "The definition of a battery cell," *National Electric Code*, p. 379, 2014, https://www.nfpa.org/Assets/files/AboutTheCodes/70/70-A2013-ROPDraft.pdf.

[9] "Battery and energy technologies: Rechargable lithium batteries," 2018, http://www.mpoweruk.com/lithiumS.htm.

[10] D. G. L. Plett, "Extended kalman filtering for battery management systems of lipb-based hev battery packs part 1. background," *Journal of Power Sources 134*, pp. 252–261, 2004.

[11] W. B. M. W. Group, "Ieee std 1491-2012 (revision of ieee std 1491-2005)," in *IEEE Guide for Selection and Use of Battery Monitoring Equipment in Stationary Applications*. IEEE, 2012.

[12] A. M. N. Watrin, B. Blunier, in *Review of adaptive systems for lithium batteries state-of-charge and state-of-health estimation*. IEEE, 2012, pp. 1–6.

[13] E. J. W. V. Prajapati, H. Hess, in *A literature review of state of-charge estimation techniques applicable to lithium poly-carbon mono ouride (LI/CFx) battery*. IICPE, 2011, pp. 1–8.

[14] "Mit tutorial: The kalman filter," 2017, http://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf.

[15] A. P. A. Mohinder S. Grewal, "Historical perspectives," *IEEE Control Systems Magazine*, pp. 69–78, 2010, http://ieeecss.org/CSM/library/2010/june10/11-HistoricalPerspectives.pdf.

[16] "An interview with jack crenshaw," http://www.trs-80.org/interview-jack-crenshaw/.

[17] "University of colorado colorado springs research publications," 2018, http://mocha-java.uccs.edu/dossier/research.html.

[18] D. G. L. Plett, "Equivalent-circuit cell models," p. 14G, 2016, http://mocha-java.uccs.edu/ECE5710/ECE5710-Notes02.pdf.

[19] D. S. J.B. Gerschler, in *Investigation of open-circuit-voltage behaviour of lithium-ion batteries with various cathode materials under special consideration of voltage equalisation phenomena.* European Association for Battery, Hybrid and Fuel Cell Electric Vehicles, 2009.

[20] "Lecture notes and recordings for ece5550: Applied kalman filtering," 2014, http://mocha-java.uccs.edu/ECE5550/index.html.

[21] "Kalman filter," 2018, https://en.wikipedia.org/wiki/Kalman_filter.

[22] C. D. Ghilani, *Nonlinear equations and Taylor's theorem*. John Wiley & Sons, Inc., 2010, vol. V.

[23] "Cholesky decomposition," 2018, https://rosettacode.org/wiki/Cholesky_decomposition.

[24] "Lecture notes and recordings for ece4710/5710: Modeling, simulation, and identification of battery dynamics," 2015, http://mocha-java.uccs.edu/ECE5710/index.html.

[25] "Arbin customers," 2018, http://www.arbin.com/careers/.

[26] "Benchtop temperature and climatic test chamber technical data," 2018, http://weiss-uk.com/products/temperature-and-climatic-testing/temperature-and-climatic-test-chambers/benchtop-temperature-climatic-test-chambers.

[27] "Mpc5744p mcu reference manual," 2018, http://www.bdtic.com/download/nxp/mpc5744prm.pdf.

[28] "Iso26262 functional safety standards," 2018, https://www.iso.org/standard/43464.html.

[29] "Ivt-s high precision current measurement device with can interface datasheet," 2018, https://www.isabellenhuette.de/fileadmin/Daten/Praezisionsmesstechnik/Datasheet_IVT-S.pdf.

[30] “Vector vn1600 can interface user manual,” 2018, https://vector.com/portal/medien/cmc/manuals/VN1600_Interface_Family_Manual_EN.pdf.

[31] “Pandas - open-source python package for fast, flexible, and expressive data analysis and manipulation,” 2018, https://pandas.pydata.org/pandas-docs/stable/index.html.

Non-exclusive licence to reproduce thesis and make thesis public

I, Kristaps Dreija (date of birth: 17th of February 1993),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

ONLINE BATTERY CELL STATE OF CHARGE ESTIMATION FOR USE IN ELECTRIC VEHICLE BATTERY MANAGEMENT SYSTEMS

supervised by Assoc. Prof. Gholamreza Anbarjafari and Ms. Egils Avots

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu 20.05.2018