POLITECNICO DI TORINO

Electronic Engineering College

Master's Degree
in Electronic Engineering

Master's Degree Thesis

Study of lead acid battery life cycle for automotive

Relators

Prof. Pasero Eros Gian Alessandro

PhD. Randazzo Vincenzo

Candidates

di Simone Giuseppe

Radici Christian

March 2019

Contents

Chapter 1.

Abstract

In modern vehicles the number of electronic control units (ECUs) keeps growing up; *in 2009 the number of microprocessor on a single vehicle ranged from 70 to 100. Even low-end cars now have 30 to 50 ECUs embedded in total* [1]. An efficient and stable power supply system must be used to provide the required energy to all the electronics, especially for those systems that deal with stability and braking but also comfort and infotainment. One of the key sections of this system is related to the automotive battery, of which the most available on market is the 12V lead acid type.

Among the operation to perform, the first task that a battery must accomplish is also the most problematic one: the cranking of the engine. A good monitoring and prediction of the dynamics of the battery can be an additional requirement to a more efficient and reliable system.

Knowing the power system is effective and under control, the useful battery life, the fuel economy and the vehicle maintainability, safety and availability are improved. In short: a reliable and accurate knowledge of the battery state is essential to sustain the ordinary operations of a vehicle, starting from turning it on.

In this context, the market is already full of automotive devices for battery management. Among the commercialized low-cost products, the solutions usually have a low efficacy and, by contrast, the high efficiency ones are too expensive and complex. An innovative approach could be the development of a solution in between.

Two main aspects of a battery are the State of Health (SoH) and the State of Charge (SoC). The SoH is a percentage indication of the capability of a battery to operate without damaging itself: higher the SoH, longer the operating life. On the other hand, the SoC is a percentage value for the remaining usable charge capacity with respect to the nominal one.

$$SoC_{\%} = \frac{available\ charge\ capacity\ [Ah]}{nominal\ charge\ capacity\ [Ah]} \cdot 100$$

Several studies have been conducted about monitoring SoH by using a simplified model in which the battery is treated as a voltage generator in series with a very low resistance (see 2.3 - Equivalent models); even though this approach could be too simplistic to be related to real life behavior, it is good enough to detect some derives from the normal usage.

A second analysis using a more accurate model was performed consecutively so that the battery is substituted by a voltage generator in series with the parallel of a very low resistance and a capacitance: the resistance guarantees that a very high current can be provided to the load; the capacitance explains why the current profile has exponential rising and falling curves.

Several acquisition campaigns applied to different samples of car battery were executed to build an algorithm efficient but still simple. The first measurement bench was based about a spectrograph that injects sinusoidal stimuli with variable frequency to the battery and then elaborate the frequency responses. After notice that method was time consuming and difficultly scalable down, a charge and discharge approach was adopted where the charging is not controlled if not from a charger for car battery and the discharging is controlled by a custom current sink device and monitored by a Data Acquisition instrumentation device (DAQ). Last but not least, a cranking simulator system was built to perform rapid discharge acquisitions up to 40A.

From the data acquired and analyzed, an algorithm will be extracted in the future and suited for an ARM microcontroller placed in a custom board that acquire the voltage of the battery and communicate via Bluetooth to an app that informs the final user about the status of the battery. Beside the acquisitions, this document will focus on the development of the custom board that embeds the ARM microcontroller, but not the algorithm neither the app.

The project was developed alongside with two Italian companies, Brain-technologies and SIVE.

Chapter 2.

State of the Art

This chapter outlines some of the newest and more promising studies in the field of SoH prediction and analysis. The subject is described in terms of literature review (techniques and equivalent battery models) and products offered on the market.

2.1 How the SoH describes the battery life cycle

Many parameters change with the aging of a battery and can be exploited to produce an estimation of its state of health. For instance, the most popular and effective techniques can make use, amongst others, of internal impedance (resistance, capacitance and other parameters depending on the equivalent model considered), state of charge and true capacitance, charge/discharge cycles, activation time (charge or discharge response) or temperature increase during operation.

As the state of health decreases with age, the charge the battery can provide during discharge decreases. The inertia towards charges movement, i.e. the internal resistance, rises. The more it increases the more the battery heats up, due to the power dissipated inside it, leading to poor performances during cranking and subsequent heat failure of the unit itself. Besides, during the activation time of the battery (in dynamic conditions) this same phenomenon takes place and can be analyzed in frequency through the internal impedance.

Over-charge and under-discharge are shown to affect heavily the life cycle of a battery. The former causes the water in the acid to undergo electrolysis; gas hydrogen builds up, and, in more extreme cases, thermal runaway occurs that, especially in a sealed battery, can easily lead to its explosion. The latter, instead, results in sulfation, a stratification of lead sulfate that obstructs the successive motion of charges.

In this sense, a correct monitor and estimation of the state of charge of a battery can bring to an improvement of its state of health.

2.2 List of methods designed to estimate the SoH

Nowadays the trend in the SoH estimation field is leaning towards implementing machine-learning algorithms, more generally algorithms that can be trained to perfect the estimation as the data collection increase.

The resulting datasets can be divided into three major groups depending on their complexity: Scalar uses load tests in terms of on/off current; Vector adds the AC component to the scalar one by using multiple frequencies; Electrochemical Impedance Spectroscopy (EIS) uses pure sinusoidal signals at different frequencies to extract the spectrum (see pag.11 of Battery Diagnostics and Monitoring). The first two (Scalar and Vector) induce a charge or discharge event in the battery, while the last one is not so invasive; in general, the more information the dataset can provide the more complex it gets to extract.

Examples of techniques that exploit scalar and vector datasets:

- *SOH estimation of lead acid battery by artificial deterioration* [1]: batteries are deteriorated by discharge and subsequently charge, so that parameters of the equivalent model can be extracted and then used to produce an on-line evaluation of the SoH. It is defined as "on-line" because it does not require the supply to the battery to be shutdown (improvement of off-line evaluations that involves the measure of the charging capacity, but the impossibility of exploiting the test during normal operation as reported in the article *A deterioration estimating system for 200-Ah sealed lead-acid batteries* [2]). The equivalent model used in this case is the Thevenin one, two different circuit are used to describe the battery respectively during charge and discharge. The variation of the internal battery resistance is shown to have a more direct link to the SoH than capacity; in particular, it increases proportionally to the decrease of the SoH.

- *Parity relation-based method* [3]: implemented to characterize the behavior of a good battery during cranking. This technique shows that, besides the internal resistance, also the voltage loss during cranking can provide useful information of the battery SoH. The algorithm estimates the battery voltage at zero current (intercept voltage) and through its difference with the open circuit voltage (before cranking) it infers the SoH value, conducting a linear regression. This difference is shown to increase as the battery ages, just like its internal ohmic resistance. Therefore, the characterization of the two parameters can lead to a better estimation of the SoH.

- *Particle Swarm Optimization for State Vector Machine (PSO-SVM) in SoH estimation* [4]: here the accuracy of the estimation is enhanced with the implementation of a large dataset (split between training and testing) and an automated learning approach. SVM are used as training algorithms and are shown to be very effective in handling non-linear classifications of samples, therefore ideal to be applied to datasets concerning varying battery parameters.

The PSO, instead, is used as a stochastic optimization technique that searches for the near optimal value of a parameter. The fast-dynamic behavior is found to be affected not only by the internal impedance and charge cycles but also by the activation time of the battery, the time it takes for it to react to the current gradient.

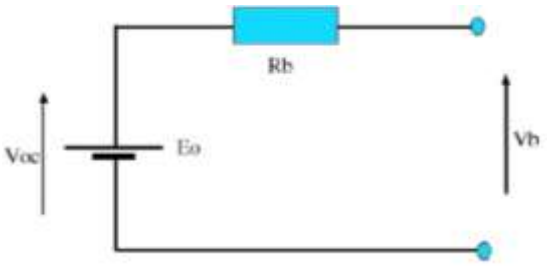
Examples of techniques that exploit EIS datasets:

- *Electrochemical Impedance Spectroscopy (or AC impedance spectroscopy)* [5]: very useful to characterize slow dynamics of the battery compared to other techniques. EIS allows, using the Ohm's law, to extract the internal impedance of the battery, it involves the injection of a sinusoidal signal (current or voltage) and the analysis of the output signal response (current or voltage) being propagated through the battery. The signal generated is characterized through its phase displacement with respect to the input one while the frequency is swept through a bandwidth of interest, and generally displayed using a Nyquist plot. The variation of the impedance is then used to extract useful parameters for SoH prediction (with a Complex Non-linear Least Square, a.k.a. CNLS, method) in an intelligent charge battery that monitor automatically and periodically the battery.

2.3 Equivalent models

The urge to simplify the complex behavior of a battery requires considering an equivalent electric model of it. Besides, depending on what kind of parameters the applied estimation technique demands, the electric components composing the model can vary a lot.

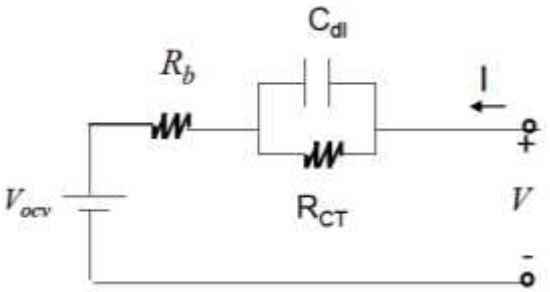
Ohmic model: see pag.3 of *Dynamic model of a lead acid battery for use in a domestic fuel cell system* [6].



Improved Ohmic model: see pag.3 of *Dynamic model of a lead acid battery for use in a domestic fuel cell system* [6].

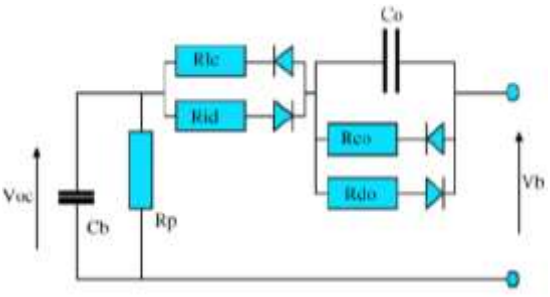
The internal battery resistance now depends on the SoC.

Thevenin (or Randles cell) model: see pag.2 of *Parity relation-based method* [3] (during cranking), and pag.1 of *SOH estimation of lead acid battery by artificial deterioration* [1] (during charge the values of the parameters change).



The capacitance represents the true capacitance of the battery, the parallel resistance is related to the contact between the plates and the electrolyte while the series one is the internal resistance of the battery.

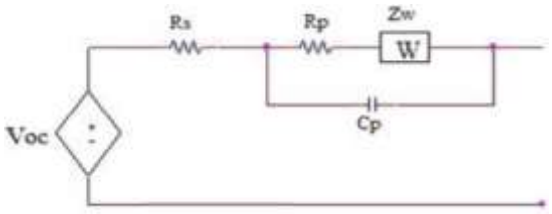
Non-Linear Dynamic model: see pag.3 of *Dynamic model of a lead acid battery for use in a domestic fuel cell system* [6].



It improves the Thevenin model considering also the non-linear parameters. This model, along with the Thevenin one, address more realistically the state of charge of the battery.

Finite Warburg Impedance model: see pag.3 of *Estimation of Battery Internal Parameters* [7].

It introduces a heavily frequency dependent impedance into the Thevenin model which mimics the diffusion rate of the charges into the battery (higher at low frequency).



2.4 Products on the market

The devices used alongside lead-acid batteries in the automotive field are here classified in terms of market target and capabilities. The first factor is ruled out, mainly, from price and vendors while the second one from datasheets.

An arbitrarily list composed of three categorizes is proposed and for each one of them an example, in terms of sold product, is featured.

- Corporate for monitoring: *Electronic Battery Sensor by Bosch*. This product can be associated to a class of automotive systems more broadly called IBS (Intelligent Battery System). It integrates a sensor and a status-tracking algorithm (BZE) that calculates the SoC of the battery and predict its SoH. Then, Bosch own EEM (electrical-power management) control unit can use this information and adjust the electronic controls of the vehicle accordingly, in order to reduce the battery load and improve its life. The data sampled by the sensor are battery voltage and current, since it integrates a shunt resistor attached to the negative terminal of the battery, through which it is able to flow in its entirety. This kind of device is designed to be always connected to the battery and to work in synergy with the electronic control unit (EEC) of the car. The connection is established via LIN protocol, which can only be configured by or for auto designer companies. This approach can lead to more efficient and accurate results since the full battery history is known, but it needs an entire system to be connected to and, more importantly, be robust and reliable enough so that the high current demands of a car electrical system are safely fulfilled. Since the device needs to operate under the hood of the car, it is rated IP6K9K (dustproof and waterproof) and for $-40°C$ to $+105°C$ operating temperature.

Figure 1. Electronic Battery Sensor by Bosch

- Professional for diagnosis: *Spectro CA-12 Battery Rapid Tester by Cadex*. This portable device checks the battery impedance and returns to the user an indication about its SoH and Cold Cranking Amp (CCA) rating. The former is shown to be connected to the residual capacity of the battery while the latter, a parameter linked to the capacity of a battery to start the engine of a vehicle at cold temperatures, to its internal resistance. In particular, the device extracts the impedance using a patented algorithm while collecting the data through electrochemical impedance spectroscopy using sinusoidal signals with frequency that goes from 20Hz to 2kHz. This approach, despite being more complex and expensive, leads to less invasive and faster tests, but its efficiency and accuracy depend for the most part on the goodness of the algorithm itself. The device capability is extended via PC using serial communication (RS232 standard) and Bluetooth. Besides EMI and EMC, the only major regulation to which it complies to is the UL3101 for electrical equipment for laboratory (safety) use.

Figure 2. Spectro CA 12 Battery Rapid Tester by Cadex

- Retail for analysis: *E-Motion kit by Midac*. This portable device, designed in a very compact form factor, is intended to verify SoC, SoH and operation of the alternator through the simple measure of the battery voltage. It connects to the battery poles with two cables and can be attached to it with double-sided adhesive tape. As soon as the device is powered up, it connects via Bluetooth with the smartphone and transfers all the data on the app where it can be stored onto cloud, for future reviews. This kit also comes integrated directly onto a 12V and 72Ah battery, thus eliminating the need of connecting two cables to the poles. In this latest form factor, it is tested for temperatures down to -18°C and certified ISO/TS 16949, for quality standards in automotive industry.

Figure 3. E Motion kit by Midac

Chapter 3.

Theory and technical review

A lead acid battery is a portable power supply unit that, thanks to chemical reactions, provides current demanded by the connected load, at a nominal voltage of 12V. Unlike other types of batteries, in the automotive field the current they must provide is usually higher, up to 800A, but mostly during the cranking of the engine. Knowing a priori if the engine of a car will start is a task that not all the control units do, but all the car owners would like to.

3.1 Battery models

Several electric models have been designed to simplify the real behavior of a battery. It is not to be surprised, considering how difficult and even destructive can be to open a sealed battery for checking and monitoring, both because of the thermo-isolated plastic case and the corrosive lead acid stored inside. Therefore, to facilitate the analysis, the battery is compared to simpler models that behave similarly, at least in some preset conditions, and are described by well-known electronic elements.

The simplest electronic model is the Ohmic one. It is used to characterize several parameters such as internal resistance, SoC, open circuit voltage, and others. The easiness to setup a test based on this model allows to have enough degrees of freedom to replicate or try out lots of different experiments. The improved Ohmic model takes into account the SoC percentage for each measurement acquired.

Another, more common model used, is the Thevenin one. In addition to the parameters found in the Ohmic model, it can describe also the charging and discharging rising and falling times, by including a series capacitor. More broadly, it is used to characterize the cranking time and internal impedance and, by decomposing it into internal resistances and capacitance, the frequency response of the battery. To simplify the test bench and the extraction of the parameters, the internal impedance can be treated as a resistance that varies in frequency putting the reactive phenomena into the background, at least initially.

Looking at graphs of current and voltage of a real battery installed on a car in several phases (power off, turn on of the dashboard, cranking, charging from the alternator), the effects of the series capacitance are immediately noticeable in the quantities' non-linear changes. As expected, the Thevenin model is more reliable than the Ohmic one.

3.2 Units under test

Nine batteries were used as units under test. Three models of batteries from different makers, each of them in three possible initial conditions:

- Used: used for normal operations for more than a year and still working fine;
- New: almost brand-new, just some burn-in and execution tests applied;
- Pristine: as if it comes straight from the manufacturer.

The pristine set of battery were tested later, after the first two campaigns, when new batteries start to become used after a dozen of discharge tests were performed.

The models are listed below together with two main parameters: nominal capacity and nominal Cold Cranking Ampere (CCA).

Index	Brand	Model	Nominal Capacity [Ah]	Nominal CCA [A]
1	BOSCH	S3000 (S3001)	40	360
2	ENERGECO	CLAUSUM-S562.059.057	62	570
3	FIAMM	L150P (L150+)	50	460

3.3 Measurement Instruments

From a design point-of-view, some instruments are essential, such as a conditioning circuit, an acquiring system and a power supply. Sometimes it may happen that some of these elements are combined into one; this is the case with the NI PXIe-6363, a multifunctional Data Acquisition system (DAQ) from National Instruments. The module is inserted in one of the five slots of the NI PXIe-1073 chassis and connected to the shielded I/O connector block SCB-68A to expand and contain all the external connections. The device is then connected to a PC via PCIe-8361, a PCI board that manages the entire chassis, and controlled through its software platform, LabVIEW 2017 SP1. Amongst its features there are easy to use internal and external and automatic and manual calibration, programmable gain amplifier to accept a wide range of signals, automatic conversion of the input to a selection of unit measurements and an easy-to-use block diagram structure for the design. However, the best use of this ready-to-use system is to build an Automatic Test Equipment (ATE).

Figure 4. NI PXIe 6363, a multifunctional Data Acquisition system

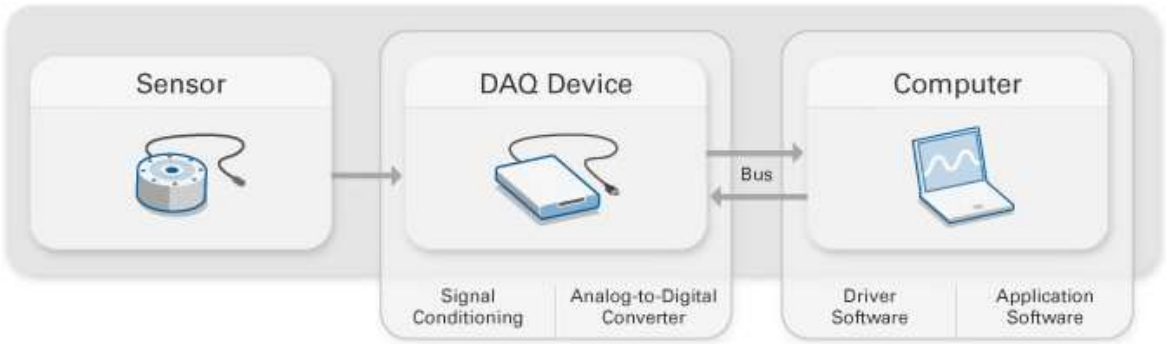


Figure 5. Generic structure of an acquisition system

3.3.1 Signal conditioning circuitry

Depending on the nature of the signal to be measured, the sensor employed and the converter used, it is often necessary to make use of a conditioning chain with which transduce, filter and adapt the signal to the data acquisition system.

For many types of applications, a voltage level shifter followed by a low pass filter compose the most basic signal conditioning chain. When the signal is not adapted to the input of the sampler, the two dynamics need to be adjusted without losing any information in the process. In this sense, what is usually done consists of separating the impedances seen by the sensor and the converter, so that the original signal is not contaminated. Another issue associated with the sampling technique is aliasing. Aliasing is a phenomenon caused by the overlap in the frequency domain of two or more spectra, and lead to a distortion of the signal to be measured. This phenomena can be linked to a noisy environment, a disturb in the acquisition chain or simply to the limitless bandwidth of the original signal. In this case, to reduce any possible artifact, a low pass filter suitably designed is usually employed that takes into account requirements from both the converter of the acquiring system and the output of the sensor.

3.3.2 Acquisition systems

Alongside the DAQ, several instruments were employed to measure and monitor different kind of analog signals. A list of the devices used is given below.

- Digital Multimeter RIGOL DM3051 with 5 ¾ digits precision;
- Portable True RMS Multimeter Tester FLUKE 179;
- AC/DC Clamp Meter Lafayette PA-37 with Hall effect sensor;
- Development board STM32 Nucleo-F303K8 with STM32F303K8 micro-controller unit;
- Mixed Domain Oscilloscope Tektronix MDO3104

3.3.3 Application software

LabVIEW 2017 SP1 was used together with the DAQ from National Instruments as the main software platform. The greatest advantage of using an automated assisted tool for instrumental measurement resides in the interface. The user that wants to start a test can use the front panel of the virtual instrument (VI) without having to know how it works in details. The designer, in turn, can exploit the user interface to let the user set the parameters before starting the test, or even during run time.

To manage big amounts of data, several scripts on MATLAB 2015b were implemented to convert, adapt, save, plot and acquire too. It is a largely used tool that allows easily management of data in any format, to access and save files and folders, establish serial communications with the any external apparatus, create stand-alone executable application for Windows PC and many other features. In short, it is a versatile and powerful tool.

3.4 Power supplies

The power supplies were employed to charge batteries under test and power other equipment. The power supplies used were:

- Automotive battery charger CTEK;
- DC power supplies GW Instek GPS-4303 with 4 channels at 0-30V and 3A.

In order to handle more safely the life cycle of the batteries and considering that the tests were all performed in a lab without using an actual motor, it was decided to use a specialized tool to charge the batteries. In this sense, the CTEK charger performs a

7-steps sequence of operations. The first, and most important, is the desulfation process, followed by a pattern of constant and variable current and voltage.

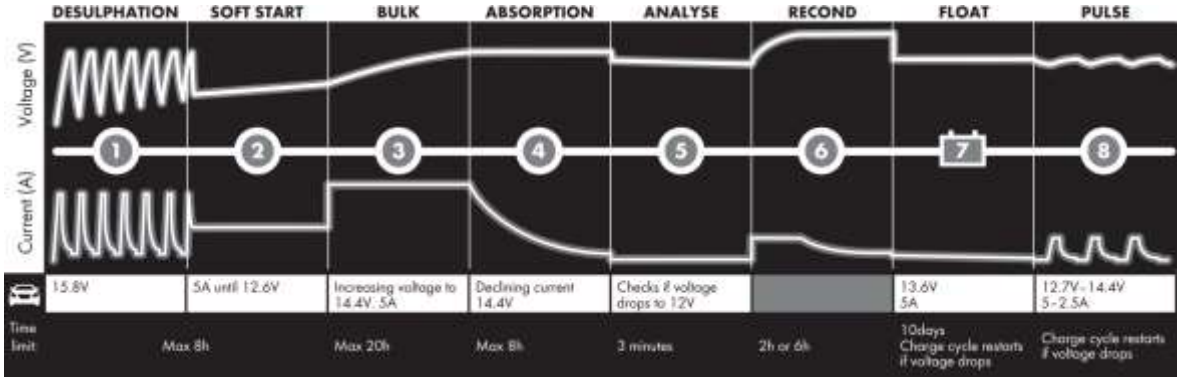


Figure 6. Charging profile of CTEK automotive battery charger

To minimize the waste time during acquisitions, a second identical CTEK charger was used as it was expected that the acquisitions would be performed on two batteries at the same time.

As a precaution, for improving the lifetime of the batteries, after every charge cycle, they were left aside for at least 24 hours to stabilize any undergoing chemical reaction and obtain a more accurate voltage, as suggested in the manual. Furthermore, the battery charger was capable to apply a specific pulsed charge designed to remove small signs of sulfation, i.e. the crystallization of the lead-sulfate under the electrodes before any charge.

3.5 Environmental conditions

The temperature and relative humidity in the laboratory, where the experiments were performed, were kept substantially constant around (22 ± 1)°C and (35 ± 5) %, respectively.

To study the behavior of a lead-acid battery outside the conditions of the lab, a climatic chamber was configured to simulate hot and cold environments. The climatic chamber used in those experiments was the Angelantoni Challenge 250.

Chapter 4.

Design of Experiments

This chapter outlines the development of the designs of data acquisition system and several experimentation phases of the project, preceding the actual data collection.

The subject is presented in terms of general concepts of the campaigns, its software and hardware design choices.

4.1 First approaches

As a first approach, it was decided to characterize the battery by measuring its internal impedance, applying sinusoidal signal injection. However, due to the complexity of implementing a system able to directly measure phase and module of the impedance, the attention was shifted towards another parameter close to the impedance, the battery internal resistance.

However, it was clear from the beginning that this task was not so straightforward, considering that the resistance value is known to vary around 15mΩ for batteries in good conditions. At these values of resistance, any direct measure can be influenced potentially by factors which are usually negligible, such as the resistance of the connections or the wire itself. For instance, an isolated wire of 24AWG (standard measure scale for the thickness of an isolated wire that stands for *American Wire Gauge* [8]) and 4cm long has a resistance of about 42mΩ, which is non-negligible since almost three times greater than the internal resistance of a brand-new lead-acid battery.

The equivalent model adopted to characterize the internal behavior of the battery is the Thevenin model. The impedance is treated as a frequency dependent resistance that varies with the voltage level of the battery. This choice was taken in order to reduce the complexity of the test that leads to an easier and immediate representation.

4.1.1 Software

LabVIEW is used as software platform through which automate the test, acquire values and save data. The user interface is the main graphic component used to set and monitor all the useful information related to the test. At the top of the front panel are positioned labels, such as brand, model and condition, which can be set before starting the test, while all the other parameters can still be modified during run-time.

Initially both virtual and external scopes, together with the speakers of the computer host, for a quicker response, were used to debug the generation of the sinusoidal signals. The iteration of each frequency was tested using a pattern of musical notes (D6 (1175Hz), C#6 (1109Hz), C6 (1047Hz) and C#6 again) played until a certain condition was reached. The frequencies were chosen arbitrarily between 900Hz and 1.5kHz, range shown to give better results in distinguishing the response of any kind of model of lead-acid battery (see 2.4 - Products on the market).

4.1.1.1 Structure of the Virtual Instrument

In this first version of the VI, three major blocks can be distinguished: the first is for the collection of the basic information about the unit under test and the filename generation, the second is for the sinusoidal signal generation towards the output and the third is for the acquisition of the signals, converted to data adapted to the correct unit measurement, and the accumulation of those data into text files.

Using the front panel of the VI, known parameters such as Brand, Model, Value and Status, are selected and the filename is composed. Fundamental indicators are also shown in the front panel as well as a start button (the Batman logo) and the slider for the amplitude of the output signal.

Beside the filename, a folder containing the Brand name selected is created and a folder named as the selected Model is created inside, if they don't exist already.

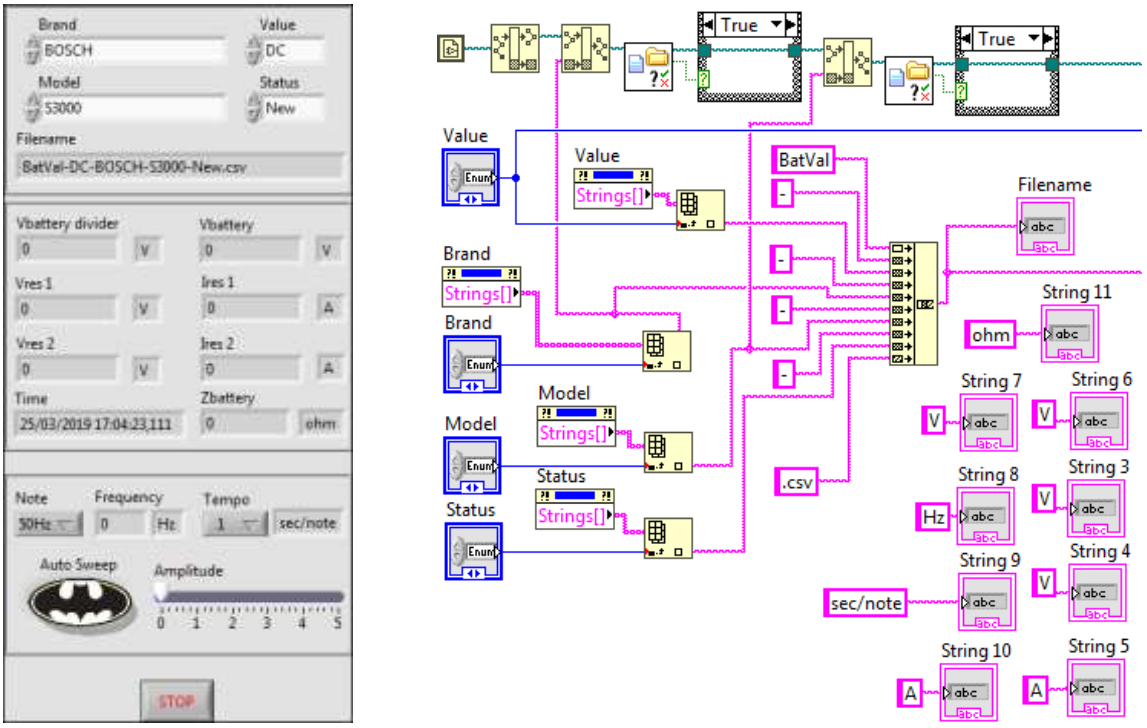


Figure 7. Front panel and section of block diagram for saving path and filename structure on first approaches VI

The array of values encapsulated by blue rectangular shapes represents the frequencies of the selected four notes. The digital signal is then enveloped, filtered and finally placed in the predefined output pin, as well as played by the speakers of the host PC.

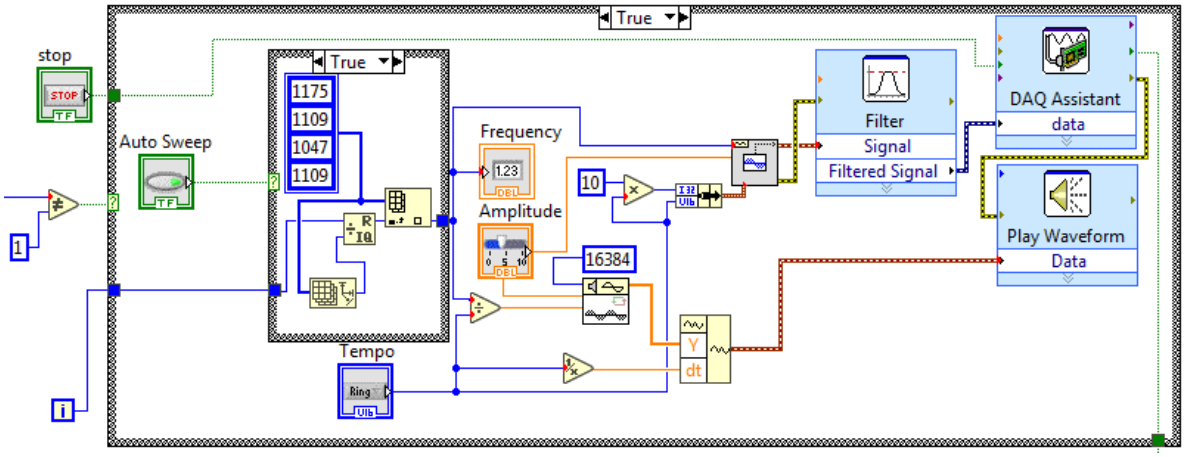


Figure 8. Section of block diagram to generate sinusoidal wave output signal on first approaches VI

A check on the existence of the text file named as the pre-composed filename is done and, if it not exists, it is created. The text file is opened in append mode; that means the string line containing the data to save are put at the end of the file. The string line follows the pattern text inside the pink rectangular shape so that the line contains in order a timestamp, the voltage of the battery, the voltage of the power supply, the computed voltage on the resistance, the computed internal impedance and the frequency.

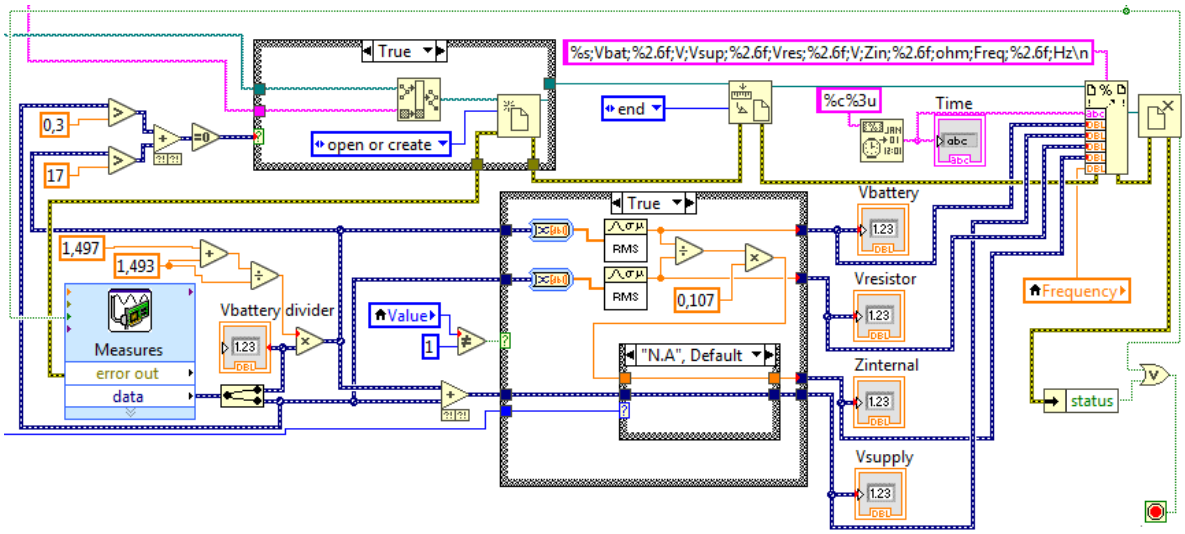


Figure 9. Section of block diagram to acquire and save data on first approaches VI

4.1.2 Hardware

The voltage of the battery was acquired through a resistive divider, designed to adapt the battery voltage, which can reach 13.5V in 12V batteries, to the input dynamic of the DAQ, with full-scale range set at ±10V. Two 330kΩ resistors were chosen for the divider, cutting in half the battery output dynamic.

The output signal from the connector block of the DAQ was applied to the gate of a power MOSFET through a very high resistor, so that a discharge event could be induced in the battery whenever the voltage level of the square wave was high enough to open the transistor. Between the drain of the MOS and the anode of the battery, a power resistor with integrated finned heat sink of 0.1Ω was connected in series acting as a load. The MOSFET and the resistor were chosen considering their maximum power ratings.

[INSERIRE IMMAGINE DELLO SCHEMATICO]

Figure 10

4.1.3 Design efficacy

During the first functional tests, the case of the MOSFET and resistive load reached a temperature of about 70°C in just a minute, while the battery resulted barely discharged.

Unfortunately, given the low precision in setting the wanted current, which in this circuit depends directly on the load and transconductance of the transistor, the design was discarded.

Due to the complex procedure involving a sinusoidal signal injection, this test was quickly discarded. Instead, a frequency-controlled discharge system took its place: the structure of the DAQ in LabVIEW remained almost the same but the signal generator changed from sinusoidal to square wave.

4.2 “Spectrum” design

The first test applied to the batteries was a series of discharge cycles induced by means of a current profile having a square wave shape with a constant mean value of 4A (with peaks up to 8A) and variable frequency.

The stop condition of the test was set when the voltage of the battery reached 10.8V, threshold below which the battery is considered mostly discharged and not able to ignite a vehicle, if not at high risk of irreversible damages.

The intention was to extract the internal resistance of the battery as the ratio between the measured voltage drop and the known variation of the current (Ohm's law).

4.2.1 Software

The block diagram behavior in LabVIEW remained almost the same as before.

Since two identical test circuits were implemented for this campaign, the output generation signal and analog input reading were duplicated.

The control voltage frequency was chosen from a list whenever the automatic start test button (Batman symbol) was pressed. Before performing this experiment for all the test batteries, some frequencies (and notes) were set during acquisition to define the best time domain in which the period was long enough to capture the battery response entirely. A good range of values was found to be from 0Hz (static field) to 1kHz.

4.2.1.1 Structure of the Virtual Instrument

Beside the doubling of management for two batteries at the same time, and so the initial parameters and indicators, some improvements and modifications were added starting from the generation of an internal folder named as the date and the time in which the test started.

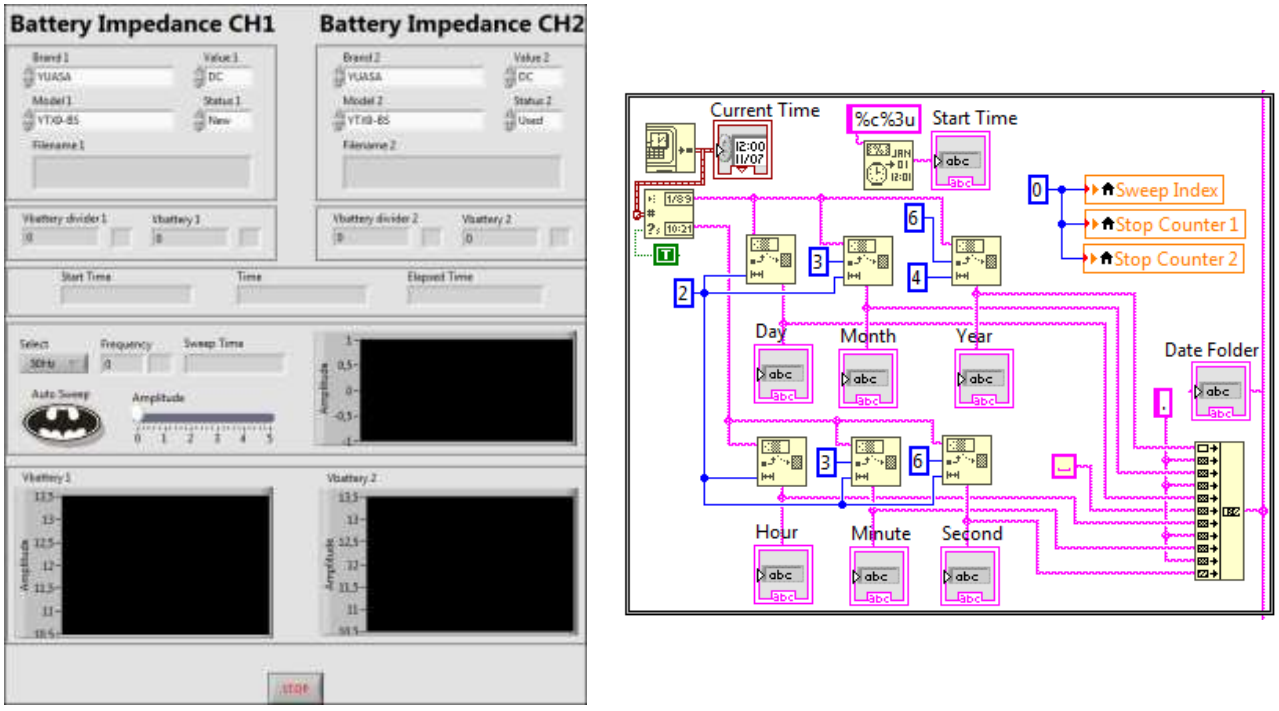


Figure 11 Front panel and section of block diagram for saving path and filename structure on "Spectrum" VI

The array of values of frequencies represented are two, as explained in the contextual Design efficacy, but only one is used, the one that is linked. The signal to generate changed to a square wave already configured as an analog output.

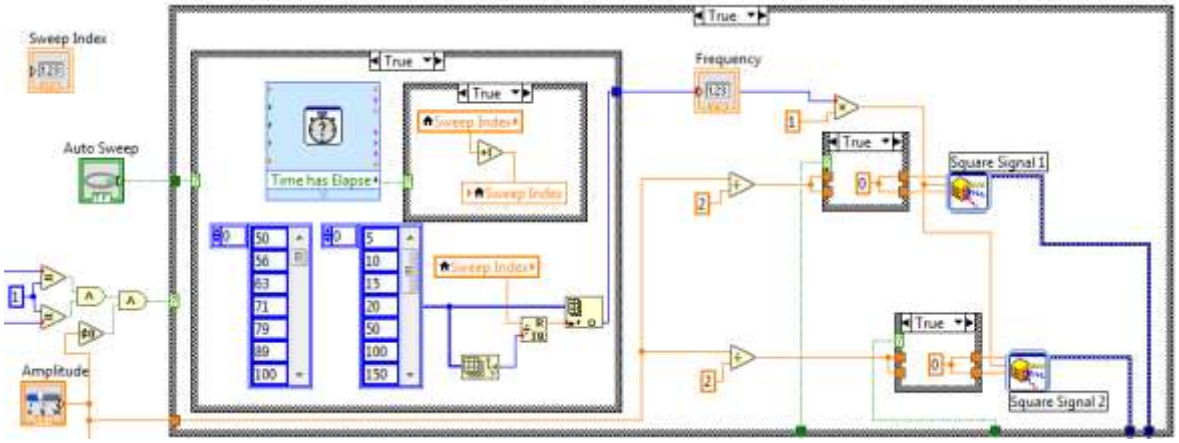


Figure 12. Section of block diagram to generate square wave output signal on "Spectrum" VI

The input voltages pass through a voltage divider of 1/3; the precise value of the resistor used are specified in the block diagram to back-compute the values of voltages of the batteries.

The save file is not a text file anymore; a LabVIEW Measures LVM file is used to obtain better performance and features, such as a header and a comment section.

In the rectangle on the bottom a real time and an elapsed time indicator are placed to see during the test how much time passed on current test.

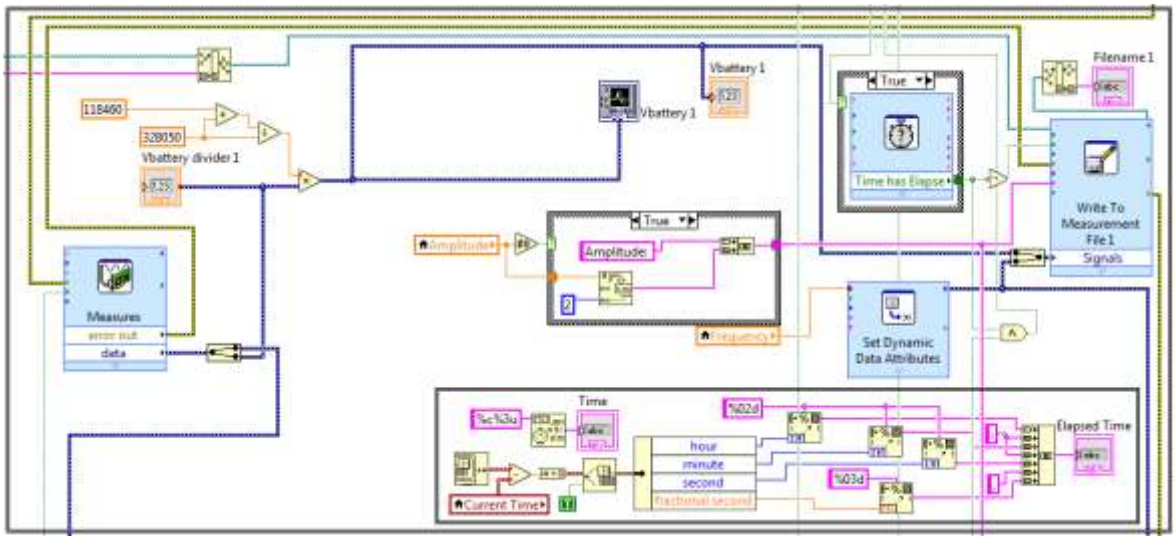


Figure 13. Section of block diagram to acquire and save data on "Spectrum" VI

To stop the execution of the test, when it finishes, a voltage condition is added: thoroughly, the voltage of the battery must be greater than 10.8V; if, for 10 times, the battery voltage read is below that threshold, the output signal for that battery branch is disabled as well as the saving of the data is interrupted. If both stop checks are verified, the test stops completely. Some variables are also reset to their defaults.

Another major check that stops the entire test is on errors: when an error occurs, in any form, the test execution stops abruptly, resetting also all the variables to their default values.

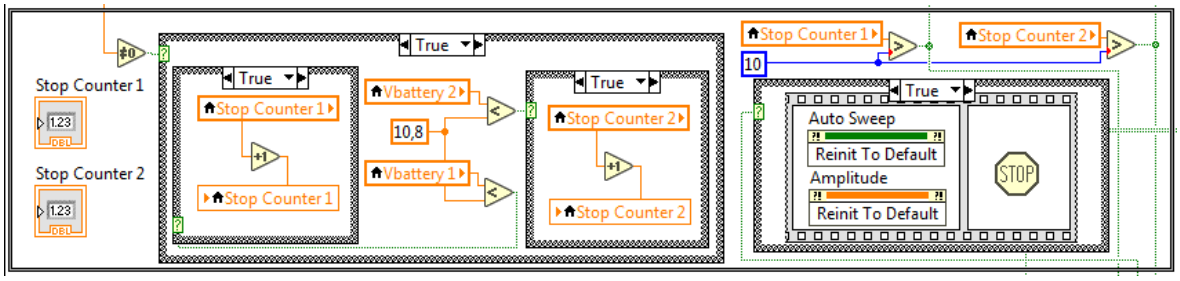


Figure 14. Section of block diagram to check conditions and stop the execution on "Spectrum" VI

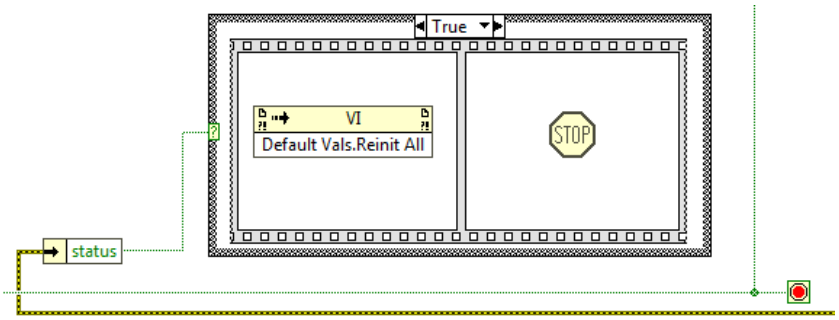


Figure 15. Section of block diagram to check errors and stop the execution on "Spectrum" VI

4.2.2 Hardware

Two main blocks compose the circuit implemented for this experiment task: a current sink able to discharge the battery at a certain current level and a voltage sensing circuit, which senses and adapts the signal to the input dynamic of the DAQ analog-to-digital converter.

Starting with an analysis of the system, the hardware design is divided into general notions about the circuit requirements, components choice, design analysis and simulation results.

The whole circuit was mounted on perfboard and placed into a metallic box, previously used as PC power supply unit, with integrated 12V fan of 15cm in diameter.

The DC power supply unit provides the 12V supply for the circuit. Besides, the negative pole of the battery is used, alongside the negative terminal of the +12V supply, as reference voltage (star point connected to 0V for all the components) and chassis connection.

[INSERIRE IMMAGINE DEI DUE SCATOLOTTI o solo di 1?]

4.2.2.1 Current sink: design analysis

In the current sink, the discharge battery current is set through a control voltage (V_{CTRL}), generated by the digital-to-analog converter of the DAQ, across a setting resistor (R_S), and it can change in terms of shape (a square wave for all the experiments), amplitude and frequency following the relations of the equations in (1).

$$\begin{cases} I_{BATT,peak} = 2 \cdot \dfrac{V_{CTRL,peak}}{R_S} = 4 \cdot V_{CTRL,peak} \\ f_{I_{BATT}} = f_{V_{CTRL}} \end{cases} \quad (1)$$

For this experiment, the circuit was designed aiming to a maximum current of 10A, corresponding to a maximum control voltage of 2.5V, with margin in case a higher current was needed.

The main concern with this type of circuit was power dissipation. A passive heatsink by Fischer Elektronik and the integrated fan are used, for this reason, in order to improve the heat transfer. The passive heatsink is mounted so that its fins are directed to the fan, so that the heated air can be removed and sucked out by the active cooling system.

The cables, battery alligator clip terminals and power connectors were all sized according to the maximum current to be handled.

[INSERIRE DIAGRAMMA BLOCCHI FUNZIONAMENTO? with only one branch]

4.2.2.2 Current sink: components choice

For this application, the N-channel MOSFET has to work only in cut-off and saturation. Here the transistor behaves like a voltage controlled current generator. This current generator is constant since the MOSFET transconductance (g_m in the small signal equivalent model) is independent from the drain-source voltage (V_{DS}) in saturation, and the gate-source voltage (V_{GS}) is set by the op amp depending on V_{CTRL}.

For this reason, power consumption issues result to be the main characteristics influencing the components choice. Although in this kind of application it should not be, in theory, of any use, the value of the drain-source on-resistance ($R_{DS(on)}$) was chosen, alongside the maximum drain-source saturation voltage (V_{DSS}), in order to not have a high Zero Temperature Coefficient (ZTC) point. Despite not risking any thermal runaway using the MOSFET in saturation region, it was decided to prefer higher values of $R_{DS(on)}$ and V_{DSS} ($> 20V$) because shown to lead to a lower ZTC.

The transistor used is the *IRFP250M HEXFET power transistor by International Rectifier* [8]. It comes in a TO-247 package, has a maximum V_{DSS} of 200V, $R_{DS(on)}$ of 75mΩ and a maximum continuous drain current of 30A at a case temperature of 25°C and gate-source voltage of 10V. The big package allows to increase thermal dissipation, leading to low values of junction-to-case thermal resistance ($R_{TH(jc),max} = 0.7°C/W$) and case-to-sink thermal resistance ($R_{TH(cs)} = 0.24°C/W$).

The op amp choice was driven mainly by three requirements: extended input common-mode and output voltage ranges, for accommodating low values of V_{CTRL} and increased MOSFET drive capability; low input offset voltage, for less DC error on the current; good thermal characteristics (low dependence) due to its proximity to heat sources.

The op amp used is the *LMC6482AIN general purpose dual op amp by Texas Instruments* [9]. The LMC6482 is a CMOS dual rail-to-rail (CMOS RRIO), ultralow input current op amp that can be supplied with voltages between $[3 \div 15.5]$ V. The output short circuit current of about 20mA, the relatively low input offset voltage (110μV typ. at 25°C) and the extended input common-mode and output swing ranges increase the effectiveness of its application as a capacitive load buffer driver. Besides, the linearity of its input offset voltage, with respect to the common-mode input voltage, allows for less or no crossover distortion (problematic in CMOS op amps) (see fig.22 of [9]).

Its main AC characteristics are 1V/μs of slew rate and 1.5MHz of unity-gain bandwidth, which are suitable for this kind of application.

The setting resistor is the *MP930 by Caddock Electronics* [11]. The value of the resistance, of $0.5Ω \pm 1\%$, was chosen considering mainly power dissipation constraints. Its rated power is 30W, which is a lot higher than the one experienced at maximum dissipation, due to the better thermal performances. Besides a low inductance resistor is preferable in order to reduce the effect of this kind of parasitic in the circuit.

In its simplest form, the current sink circuit configuration is derived from a known topology and consists of a power MOSFET, a setting resistor and one operational

amplifier. The last one serves two purposes: drive the gate of the transistor into saturation and efficiently mirror V_{CTRL}. The first is guaranteed by the op amp output voltage swing ($> V_{CTRL,max} + V_{GS,max}$) and short-circuit output current. The second one ensures that V_{CTRL}, fed at its non-inverting input, is forced onto a setting resistor (R_S) so that a corresponding current can flow out of the battery back into the circuit, according to the equation (2).

$$I_D = \frac{V_{CTRL}}{R_S} \qquad (2)$$

At this stage, the circuit was not protected from reverse battery voltage polarity, overcurrent or over-temperature. The first one was omitted in order to avoid any influence on the behavior of the battery. The other two, instead, were introduced in the later version of the circuit. The overvoltage protection is integrated into the voltage sensing circuit, in order to protect it from any spike created by the parasitic inductances and capacitances reacting to the sudden current gradient.

For troubleshooting and monitoring purposes, the voltage across the setting resistor is sampled by the DAQ so that the current can always be observed.

The first analysis conducted was the thermal design. Considering that the amount of power to be dissipated can be as high as 130W, it was decided to duplicate the current sink circuit and provide each with a passive heatsink, to which mount one setting resistor and one MOSFET.

The major part of the power is dissipated by the MOSFET. The worst case (maximum current and voltage) correspond to a current of 5A and a voltage drop of 11V ($V_{DS,max} = V_{BATT,max} - V_{CTRL,max}$).

With these values and looking at the graph showing the safe operating area (SOA), the MOSFET can be safely operated at maximum requirements. However, since the control voltage is not a single pulse, the temperature of the transistor junction must be estimated in order to ensure that is does not exceed the maximum rating of 175°C.

Due to the lack of a thermal model for the IRFP250, the exponential behavior of the junction temperature remains unknown, while only the peak of the junction temperature can be estimated (*Power MOSFET Thermal Design and Attachment of a Thermal Fin* [11]) using the transient thermal impedance (or thermal response Z_{th}), with the equation (3).

$$T_{ch,pk} = P_{max} \cdot \left(D \cdot R_{ja} + D \cdot Z_{th,max}\right) + T_A \qquad (3)$$

The equation is simplified considering that the worst case is when the pulses of V_{CTRL} last 10s (0.1Hz), which gives the maximum Z_{th}, and that the square wave is symmetric thus with duty cycle (D) equal to 50%.

Finally, the term R_{ja} represents the total junction-to-ambient thermal resistance and can be extracted from the thermal model, using equation (4). This equivalent circuit (figure (X)) contains all the thermal resistances linked to the heat generated into the junction and flowing, through the heatsink, into the external ambient.

[INSERIRE SCHEMATICO CIRCUITO TERMICO EQUIVALENTE con legenda annessa]

Figure 16

$$R_{ja} = R_{jc} + \frac{R_{ca} \cdot \left(R_{cs} + R_{tp} + R_{hf}\right)}{R_{ca} + R_{cs} + R_{tp} + R_{hf}} \qquad (4)$$

Since an active mean of cooling is also adopted, the heat sink thermal resistance is scaled down with an adjusting factor linked to the airflow that the fan can generate (*The Effect of Forced Air Cooling on Heat Sink Thermal Ratings* [12]). From the airflow expressed in LFM (ft/min), and adjusting for the backpressure resisting the main airflow, it is derived a derating factor of 0.3. The chosen passive heatsink has a nominal thermal resistance of 2.7°C/W, so the total equivalent thermal resistance is:

$$R_{hf} = K \cdot R_h = 0.3 \cdot 2.7\,{}^{\circ C}/_{W} = 0.81\,{}^{\circ C}/_{W} \qquad (5)$$

With these values, the total junction-to-ambient thermal resistance is 1.77°C/W.

Considering the worst case and using the equation (3), it is calculated a peak junction temperature of 95°C, at room temperature, which is well below the maximum rating.

Unfortunately, also the power dissipated by the setting resistor influences the junction, because attached with the MOSFET to the same heatsink. In order to simplify the analysis, an equivalent model was simulated by adding to the circuit in

figure (X-vedi schem therm) another current generator corresponding to the power dissipated by the resistor and its thermal resistance.

This simulation does not take into account the dynamic behavior of heat transfer, therefore only the peak value was taken into account.

Without the setting resistor, through the simulation the peak junction temperature is 124°C, which is a 30% overestimation of what actually happen. With the 12.5W dissipated by the setting resistor (5A, 2.5V) the junction reaches a peak temperature of 134°C, which is still under the maximum rating.

Considering the equivalent model of figure (X-vedi schem therm) without the power dissipated by the MOSFET, the setting resistor would reach a maximum junction temperature of 90°C, at room temperature, which is below its maximum rating.

At this stage, the circuit contains only one feedback loop that can be used to stabilize the operation of the system and compensate for the tolerances and mismatches of the transistor. Actually, the former can be attained only with a second loop (*Implementation and Applications of Current Sources and Current Receivers* [13]). In fact, the only isolation resistor (R_{ISO}) is not enough to guarantee the stability. The resistor and the parasitic gate-source capacitance of the transistor (C_{GS}) create a pole in the path and thus a zero in the $1/\beta$ curve, which leads to a 40dB/decade rate around its interception with the open loop gain (A_{OL}) curve, therefore creating an unstable loop (phase margin of loop gain T less than 45°). In order to compensate for this zero a pole is added using the components C_F and R_F, so that the $1/\beta$ curve intercept A_{OL} at a rate of 0dB/decade.

The schematic of one branch is shown in figure (X), while the graph in figure (X) displays how the important signals vary during operation.

[INSERIRE SCHEMATICO COMPLETO (senza valori) CON FRECCE PER FB1 E FB2, VOA, VFB]

Figure 17

[INSERIRE PLOT FUNZIONAMENTO CON: Vctrl, Vgs, Ig, Id, Vds, Pdiss,M, Tj?]

Figure 18

4.2.2.3 Current sink: stability, transient and dissipation analysis

At a given frequency, the feedback that dominates the loop is that with the lowest $1/\beta$, it follows that the total $1/\beta$ is found by the superposition of the $1/\beta$ curve of the two paths. In this case, the path FB_1 is active at low frequency while the other,

FB_2, at high frequency. R_F sets a DC path for the current from the inverting input to the setting resistor mirroring the control voltage onto R_S, while C_F engages at high frequency and brings back the op amp into a stable voltage buffer.

The stability analysis is performed considering the worst case in which $1/\beta_1 = (1/\beta_1)_{min}$ (since $T = A_{OL} \cdot 1/\beta$), i.e. using the maximum value of the control voltage, and $V_{BATT} = V_{BATT,min}$, that is to say at minimum V_{DS}.

At DC, the value of $1/\beta_1$ can be computed by inspection. Supposing that with $V_{GS} = 4.5V$ [ID/VGS graph on datasheet] M_1 is able to sink 5A of current with 2.5V of V_{CTRL} mirrored onto R_S, and assuming a threshold voltage $V_{GS,th}$ of 3.6V (derived from simulating the DC characteristic of M_1) the calculated $1/\beta_1$ term is 2.6dB, according to equation (6).

$$\left(\frac{1}{\beta_1}\right)_{DC} = \frac{\Delta V_{OA}}{\Delta V_{FB}} = \frac{V_{GS,max} + V_{CTRL,max} - V_{GS,th}}{V_{CTRL,max}} \tag{6}$$

Since the op amp is manufactured using CMOS technology, its output impedance shows both a capacitive and a resistive nature. For frequencies higher than 100Hz, the output impedance of the op amp is considered simplified to a series resistance R_O. This assumption simplifies the calculations and it is based on the observation that the parallel output capacitance, since it is very large (tens of µF), is more of a short long before other capacitances.

From this assumption and the simplified small signal equivalent circuit of figure (X), the expression of $1/\beta_1$, in the Laplace domain, can be found knowing that $\beta_1 = V_{FB,1}/V_{OA}$. Then the frequency of its zero ($f_{Z,1/\beta 1}$) can be easily extracted thanks to the equations in (7).

$$\begin{cases} \dfrac{1}{\beta_1} = \dfrac{1 + g_m \cdot R_S}{g_m \cdot R_S} \cdot \dfrac{1 + s \cdot \dfrac{C_{GS} \cdot (R_O + R_{ISO} + R_S)}{1 + g_m \cdot R_S}}{1 + s \cdot \dfrac{C_{GS}}{g_m}} \\ f_{Z,1/\beta_1} = \dfrac{1 + g_m \cdot R_S}{2\pi \cdot C_{GS} \cdot (R_O + R_{ISO} + R_S)} \end{cases} \tag{7}$$

The analysis of FB_2 is based on two assumptions. The first is about the value of C_{GS} while the other, derived from the former, concerns the value of C_F. Despite being parasitic parameters and therefore not guaranteed by the manufacturer, on the datasheet, a graph is usually given that shows the variation of parasitic capacitances

with respect to V_{DS}. Thanks to this graph, one can extract an approximate value of C_{GS}, C_{GD} and C_{DS} (see A.1.1 - MOSFET parasitic capacitances for extended analysis). The second assumption is aimed to simplify the computation of the β_2, in fact, since FB_2 is dominant at high frequency, by choosing a value of $C_F < C_{GS}/10$ then the parasitic capacitance C_{GS} will be more of a short long before C_F, thus excluding it from the equivalent small signal model.

Further analysis also brought to neglect the effects of C_{GD}, since in saturation it is not as dominant as C_{GS} and, even if it was, its effects can be seen at very high frequency.

From these assumptions and the simplified small signal equivalent circuit of figure (X), the expression of $1/\beta_2$, in the Laplace domain, can be found knowing that $\beta_2 = V_{FB,2}/V_{OA}$. Then the frequency of its pole ($f_{P,1/\beta 2}$) can be easily extracted thanks to equations (8).

$$\begin{cases} \dfrac{1}{\beta_2} = \dfrac{R_{ISO}+R_O}{R_{ISO}} \cdot \dfrac{1 + s \cdot C_F \cdot \dfrac{(R_O + R_{ISO} + R_S) \cdot {R_O}/{R_{ISO}} + R_F}{1 + {R_O}/{R_{ISO}}}}{s \cdot R_F \cdot C_F} \\ f_{P,1/\beta_2} = \dfrac{1}{2\pi \cdot R_F \cdot C_F} \end{cases} \tag{8}$$

[INSERIRE CIRCUITI EQUIVALENTI DI PICCOLO SEGNALE PER β1 e β2]

Figure 19

The selection of R_{ISO}, R_F and C_F values were made looking at the different effects that they create.

In particular, it was observed that:

- As R_{ISO} increases the closed loop (A_{CL}) bandwidth decreases which means that the speed of the system decreases;
- If R_{ISO} is too small the overall $1/\beta$ (and therefore A_{CL}) is more "distorted";
- If R_{ISO} is too big a pair of complex conjugate poles arises in the $1/\beta$ curve, that leads to a not desirable gain peaking in the A_{CL} (despite having a seemingly stable system) (see *Operational Amplifier Stability Part 10 of 15: Capacitive Load Stability: RISO with Dual Feedback* [11]);

- As R_F increases a DC error caused by the bias and input offset currents of the op amp increases, which, in this case, due to their low values in the LMC6482 is not a problem;
- R_F can be used to minimize the offsets (at a certain temperature) (see A.1.2 - Feedback resistor value: offset minimization for extended analysis).

Alongside these observations, the following requirements were also considered:

- $\varphi_M \geq 45°$, where φ_M is the phase margin;
- $f_{Z,1/\beta 1} < f_t$, where f_t is the frequency at which A_{OL} is 0dB;
- $C_F < C_{GS}/10 \cong 400pF$;
- $f_{3dB,ACL} \geq 4/\pi\tau$, where $f_{3dB,ACL}$ is the cut-off frequency of the closed loop gain (which represents the bandwidth of the system) and τ is the minimum between rise and fall time of V_{CTRL} ($1/\pi\tau$ is the second pole of a square wave, while 4 is an arbitrary constant chosen in order to reduce any changes of the wave shape);
- $f_{Z,1/\beta 1} = 2f_{P,1/\beta 2}$, this relation was chosen arbitrarily after some trials. This constant can be chosen between 2 and 10 depending on the application and components.

Other values considered for the selection were:

- $R_O = 59\Omega$ (from simulation; see A.1.3 - Op amp output impedance simulation for extended analysis);
- $g_m = 10.74S$ (from calculation, as $2I_D/(V_{GS} - V_{TH})$, and simulation);
- $C_{GS} = 3.9nF$, $C_{GD} = 0.9nF$, $C_{DS} = 0.9nF$ (from calculation; see A.1.1 - MOSFET parasitic capacitances for extended analysis).

The adopted step-by-step procedure for the selection of the remaining components value was:

- Choose R_F to minimize the op amp offsets;
- Choose $C_F < 0.1C_{GS}$;
- Calculate $f_{P,1/\beta 2}$ using CF and RF;
- Derive $f_{Z,1/\beta 1}$ using $f_{P,1/\beta 2}$;

- Calculate R_{ISO} using $f_{Z,1/\beta 1}$.

Having designed a stable circuit, the transient response at maximum current discharge shows no peaking or oscillations in any of the circuit signals.

Before running the simulations, the models provided by the manufacturers of the op amp and MOSFET were validated by plotting useful DC and AC characteristics and confronting them with the ones present on the datasheets.

[INSERIRE CIRCUITI SIMULATI (con valori) PER ANALISI IN FREQUENZA]

Figure 20

[INSERIRE BODE PLOT CON FB1 FB2 E TOTALE (AOL, 1/B e ACL)]

Figure 21

The circuit assembly of the current sink followed two main constraints: maintain symmetry between the op amp and the two branches and reduce as much as possible the distance between the MOSFET gate and the output of the op amp, in order to reduce the loop inductance.

4.2.2.4 Voltage sensing: design analysis

In the voltage sensing circuit, a resistive divider, an op amp and an RC low pass filter compose the signal conditioning chain.

The filter is a passive low pass filter that limits the bandwidth of the signal to be sampled and, in doing so, avoid any aliasing and reduce the noise outside the band of interest. This kind of filter was chosen so that its capacitance, other than setting the proper cut-off frequency, can serve also as a charge storage for the ADC input sampling capacitance, C_{ADC}.

[INSERIRE DIAGRAMMA BLOCCHI FUNZIONAMENTO?]

Figure 22

4.2.2.5 Voltage sensing: components choice

The op amp choice was driven mainly by three requirements: low input offset voltage and general high DC precision, to increase the precision of the measurement; enough gain-bandwidth product (GBW) to be suitable for the frequency events in the chain; 12V single supply to reduce the number of supply voltages to one and reduce complexity.

The op amp used is the *LTC1152 by Linear Technology* [10]. The LTC1152 is a rail-to-rail input/output (RRIO) zero-drift op amp with very good DC performances, and can be supplied with voltages between [2.7 ÷ 14] V. Its low input bias current, input offset current and input offset voltage (respectively ±10pA, ±20pA and ±1µV typ. at 25°C), and GBW of 700kHz make it appropriate for this kind of application.

4.2.2.6 Voltage sensing: stability and transient analysis

The divider scales down the battery voltage with a factor of 11/17, using 330kΩ and 180kΩ resistors for low interaction with the battery current, with ±1% of tolerance and 1/4W of rated power dissipation.

This signal, which has a dynamic of [0 ÷ 8.7] V, is fed to the non-inverting input of the op amp through a current limiting resistor of 100kΩ. The input of the op amp has also an additional layer of protection consisting of two clamping diodes connected from the line to 0V or 12V. Differently from the one integrated by the manufacturer inside the amplifier, which are usually aimed at protecting ESD, these diodes have the purpose of reducing any eventual spike caused by the parasitic inductances introduced by the cables connecting the battery to the current sink. Their effectiveness depends on their recovery with respect to the speed of the spikes, linked to the amount of current flowing and the rise and fall times of V_{CTRL}. Simulations have shown that 1N4001 rectifiers are suitable to be used in this configuration.

The op amp acts as a buffer with gain equal to 1 and has three advantages: sense the signal with a high (input) impedance; present to the input of the ADC a low (output) impedance so that the signal can settle before the converter starts sampling; and, finally, reduce the effect of any impedance on the cut-off frequency of the following RC low pass filter.

Lastly, the voltage sampled by the ADC can be expressed as:

$$V_{AI} = V_{Batt} \cdot \frac{R_B}{R_A + R_B} \tag{9}$$

[INSERIRE SCHEMATICO COMPLETO con tensioni]

Figure 23

The cut-off frequency of the filter, and therefore the values for the resistance and capacitance, is linked to the other frequency events. These frequencies are: the maximum bandwidth of the signal to be sampled (f_{SIGNAL}), the op amp gain-bandwidth

frequency (f_{GBW}), cut-off frequency of the anti-aliasing filter (f_{AA}) and the sampling frequency of the ADC (f_S).

Initially, the values were chosen as follows:

- $f_{SIGNAL} =$ 1kHz (maximum frequency of V_{CTRL});
- $f_{AA} = 5f_{SIGNAL} = 5kHz$;
- $f_{GBW} \geq 50f_{AA} = 250kHz$ (see *FilterPro™ User Guide* [13]);
- $f_S = 10f_{SIGNAL} = 10kHz$.

These values were after reduced, maintaining however the same multiplication coefficients, having reduced the maximum V_{CTRL} frequency as well.

The value of the filter capacitance (C_{FLT}) is set to be $10C_{ADC}$, which, for the NI PXIe-6363 is 100pF (see pag.2 on *PXIe-6363 Specifications - National Instruments* [11]). The value of the resistance is calculated with the equation (10), so that the filter network settles within the ADC acquisition time, i.e. its time constant is less than the acquisition time (see *Optimize Your SAR ADC Design* [15]).

$$R_{FLT} \leq \frac{1}{f_S \cdot C_{FLT} \cdot ln(2^{N_{bit}+1})} \qquad (10)$$

Knowing that the NI PXIe-6363 ADC has 16-bit of resolution, the maximum value for R_{FLT} is found to be around 2kΩ. Therefore, the values for R_{FLT} and C_{FLT} are, respectively, 1kΩ and 1nF.

Since, with the set values of R_{FLT} and C_{FLT}, the cut-off frequency of the anti-aliasing filter is not the desired one, an additional capacitor of 220pF was added in parallel to the resistive voltage divider.

Since the op amp is used as a voltage follower, the frequency stability is guaranteed. Therefore, the transient response with maximum current discharge shows no peaking or oscillations in any of the important signal.

Before running the simulations, the model of the op amp used was imported as a macro model in TINA. Then it was validated by plotting useful DC and AC characteristics and confronting them with the ones present on the datasheets.

[INSERIRE PLOT FUNZIONAMENTO CON: Vbatt, Ibatt, Vout]

Figure 24

4.2.3 Design efficacy

The goodness of the design is determined in terms of DC (or static) accuracy on battery discharge current and battery voltage. In turn, the accuracy of the calculated internal resistance of the battery will depend on these two factors.

The uncertainty is calculated using the Root Sum Square (RSS) technique, which gives a better estimate of the propagation with respect to the more pessimistic of the deterministic method. All the sources used in the calculations are considered statistically independent (uncorrelated) and at their maximum (if provided by the manufacturer).

The current sink can generate a current with ±1% of uncertainty, in the range [0 ÷ 8] A.

The main sources of uncertainty are resistor fabrication tolerance and DAC output error. Other causes considered are negligible with respect to these two. They are, for instance, resistor temperature drift; input offset voltage (which is compensated with R_F, but only at 25°C) and its temperature drift; input bias and offset current; DC closed-loop gain error and its temperature drift; effects of the voltage divider.

[INSERIRE SCHEMATICO semplificato con collegamenti al DAQ?!]

Figure 25

From equation (11) the uncertainty on the battery current can be expressed as:

$$\delta I_{BATT} = \sqrt{\left(\left|\frac{\partial I_{BATT}}{\partial V_{CTRL}}\right| \cdot \delta V_{CTRL}\right)^2 + \left(\left|\frac{\partial I_{BATT}}{\partial R_S}\right| \cdot \delta R_S\right)^2} = \sqrt{\left(\frac{2}{R_S} \cdot \delta V_{CTRL}\right)^2 + \left(\frac{2 \cdot V_{CTRL}}{R_S} \cdot \frac{\delta R_S}{R_S}\right)^2} \quad (11)$$

The setting resistors were measured using the Rigol DM3051 digital multimeter, which accuracy is given as ±(0.05% of reading + 0.01% of range).

The term δV_{CTRL} is evaluated with the information provided by the manufacturer of the NI PXIe-6363 in the datasheet, as follows.

$$\delta V_{CTRL} = V_{CTRL} \cdot \varepsilon_{GAIN} + V_{FSR} \cdot \varepsilon_{OFFSET} \quad (12)$$

Where:

$$\begin{cases} \varepsilon_{GAIN} = \varepsilon_{G,res} + T_{C,gain} \cdot \Delta T_{last\ internal\ cal} + T_{C,ref} \cdot \Delta T_{last\ external\ cal} \\ \varepsilon_{OFFSET} = \varepsilon_{OFFSET,res} + T_{C,offset} \cdot \Delta T_{last\ internal\ cal} + \varepsilon_{INL} \end{cases} \quad (13)$$

The following table sums up the uncertainty propagation of the battery current. Referring to the equation (11) the sources of error indicated as "Source A/B" correspond to the ordered products of standard uncertainty and sensitivity coefficients. Note that their Relative Uncertainty is calculated with respect of the battery current uncertainty.

	Value	Absolute Uncertainty	Relative Uncertainty [%]	Unit
R_S	$500 \cdot 10^{-3}$	$\pm 5.00 \cdot 10^{-3}$	$\pm 1.00 \cdot 10^{0}$	Ω
V_{FSR}	$10.0 \cdot 10^{0}$	N.A.	N.A.	V
$\varepsilon_{G,res}$	N.A.	$70.0 \cdot 10^{0}$	N.A.	ppm
$T_{C,gain}$	N.A.	$8.0 \cdot 10^{0}$	N.A.	ppm/°C
$\Delta T_{last\ internal\ cal}$	N.A.	$2.00 \cdot 10^{0}$	N.A.	°C
$T_{C,ref}$	N.A.	$1.00 \cdot 10^{0}$	N.A.	ppm/°C
$\Delta T_{last\ external\ cal}$	N.A.	$8.00 \cdot 10^{0}$	N.A.	°C
$\varepsilon_{OFFSET,res}$	N.A.	$33.0 \cdot 10^{0}$	N.A.	ppm
$T_{C,offset}$	N.A.	$2.00 \cdot 10^{0}$	N.A.	ppm/°C
ε_{INL}	N.A.	$64.0 \cdot 10^{0}$	N.A.	ppm
V_{CTRL}	$2.00 \cdot 10^{0}$	$\pm 0.60 \cdot 10^{-3}$	$\pm 0.03 \cdot 10^{0}$	V
Source A	N.A.	$\pm 2.40 \cdot 10^{-3}$	$\pm 100 \cdot 10^{-3}$	A
Source B	N.A.	$\pm 80.0 \cdot 10^{-3}$	$\pm 99.9 \cdot 10^{0}$	A
$I_{BATT,peak}$	$8.00 \cdot 10^{0}$	$\pm 80.0 \cdot 10^{-3}$	$\pm 1.00 \cdot 10^{0}$	A

The voltage sensing circuit is able to measure a voltage with ±0.11% of uncertainty, in the range [0 ÷ 13.5] V.

The main sources of uncertainty are: resistors tolerance, ADC input error, input offset voltage of the op amp and the voltage drop across the cables when the current is different from zero. Other causes are considered negligible with respect to these four. They are, for instance, input offset voltage drift, input bias and offset current, DC closed-loop gain error and its temperature drift; voltage divider between R_{FLT} (1kΩ) and R_{AIN} (10GΩ), temperature drift of wires resistance, crosstalk between two adjacent inputs of the DAQ.

The cables resistance is derived knowing the diameter and length of the cables connecting the battery to the current sink, using a table that links AWG to resistance

per unit length (13.17mΩ/m for a 16AWG stranded wire). This results in a resistance of 7.9mΩ.

From equation (9) the battery voltage, during the discharge phase (with the circuit closed, CC), is extracted keeping in mind the effects of the resistance of the cables.

$$V_{BATT,CC} = V_{AI} \cdot \frac{R_A + R_B}{R_B} + 2I_{BATT} \cdot R_W \tag{14}$$

Then its uncertainty can be expressed as:

$$\delta V_{BATT,CC} = \\ = \sqrt{\left(\left|\frac{\partial V_{BATT}}{\partial V_{AI}}\right| \cdot \delta V_{AI}\right)^2 + \left(\left|\frac{\partial V_{BATT}}{\partial R_A}\right| \cdot \delta R_A\right)^2 + \left(\left|\frac{\partial V_{BATT}}{\partial R_B}\right| \cdot \delta R_B\right)^2 + {V_{OFF}}^2 + \left(\left|\frac{\partial V_{BATT}}{\partial I_{BATT}}\right| \cdot \delta I_{BATT}\right)^2} \tag{15}$$
$$= \sqrt{\left(\frac{R_A + R_B}{R_B} \cdot \delta V_{AI}\right)^2 + \left(\frac{V_{AI}}{R_B}\right)^2 \cdot \left({\delta R_A}^2 + \left(\frac{R_A \cdot \delta R_B}{R_B}\right)^2\right) + {V_{OFF}}^2 + (2 \cdot R_W \cdot \delta I_{BATT})^2}$$

The term δV_{AI} can be calculated using equation (16) given by the manufacturer in the datasheet of the NI PXIe-6363.

$$\delta V_{AI} = V_{AI} \cdot \varepsilon_{GAIN} + V_{FSR} \cdot \varepsilon_{OFFSET} + u_N \tag{16}$$

Where ε_{GAIN} and ε_{OFFSET} are calculated as before using equation (13), while the last term, which is the noise uncertainty, as:

$$u_N = \frac{3\sigma_N}{\sqrt{n}} \tag{17}$$

The divider resistors were measured using the Rigol DM3051 digital multimeter, which accuracy is given as ±(0.015% of reading + 0.001% of range).

The following table sums up the uncertainty propagation of the battery current. Referring to the equation (15) the sources of error indicated as "Source A/B…" correspond to the ordered products of standard uncertainty and sensitivity

coefficients. Note that their Relative Uncertainty is calculated with respect of the battery current uncertainty.

	Value	Absolute Uncertainty	Relative Uncertainty [%]	Unit
V_{FSR}	$20.0 \cdot 10^{0}$	N.A.	N.A.	V
$\varepsilon_{G,res}$	N.A.	$48.0 \cdot 10^{0}$	N.A.	ppm
$T_{C,gain}$	N.A.	$13.0 \cdot 10^{0}$	N.A.	ppm/°C
$\Delta T_{last\ internal\ cal}$	N.A.	$2.00 \cdot 10^{0}$	N.A.	°C
$T_{C,ref}$	N.A.	$1.00 \cdot 10^{0}$	N.A.	ppm/°C
$\Delta T_{last\ external\ cal}$	N.A.	$8.00 \cdot 10^{0}$	N.A.	°C
$\varepsilon_{OFFSET,res}$	N.A.	$13.0 \cdot 10^{0}$	N.A.	ppm
$T_{C,offset}$	N.A.	$21.0 \cdot 10^{0}$	N.A.	ppm/°C
ε_{INL}	N.A.	$60.0 \cdot 10^{0}$	N.A.	ppm
σ_N	N.A.	$315 \cdot 10^{-6}$	N.A	V_{RMS}
R_A	$118 \cdot 10^{3}$	$\pm 27.8 \cdot 10^{0}$	$\pm 0.02 \cdot 10^{0}$	Ω
R_B	$330 \cdot 10^{3}$	$\pm 59.2 \cdot 10^{0}$	$\pm 0.02 \cdot 10^{0}$	Ω
V_{AI}	$9.83 \cdot 10^{0}$	$\pm 0.01 \cdot 10^{0}$	$\pm 0.11 \cdot 10^{0}$	V
Source A	N.A.	$\pm 15.0 \cdot 10^{-3}$	$\pm 98.8 \cdot 10^{0}$	V
Source B	N.A.	$\pm 0.83 \cdot 10^{-3}$	$\pm 0.30 \cdot 10^{0}$	V
Source C	N.A.	$\pm 0.64 \cdot 10^{-3}$	$\pm 0.18 \cdot 10^{0}$	V
V_{OFF}	N.A.	$\pm 10 \cdot 10^{-6}$	$\pm 0.44 \cdot 10^{-6}$	V
R_W	$7.90 \cdot 10^{-3}$	N.A.	N.A.	A
Source D	N.A.	$\pm 1.30 \cdot 10^{-3}$	$\pm 0.70 \cdot 10^{0}$	V
$I_{BATT,peak}$	$8.00 \cdot 10^{0}$	$\pm 80.0 \cdot 10^{-3}$	$\pm 1.00 \cdot 10^{0}$	A
$V_{BATT,CC}$	$13.4 \cdot 10^{0}$	$\pm 15.1 \cdot 10^{-3}$	$\pm 0.11 \cdot 10^{0}$	V

Finally, the battery internal resistance can be calculated using equation (18) where $V_{BATT,OC}$ represents the battery voltage when not discharging. The uncertainty propagation is performed keeping in mind that the systematic error due to the intrinsic resistance of the cable is present only during the discharge phase. The internal resistance is known with ±17.4% of uncertainty (the value is intended to be an upper limit for the parameter).

$$R_{INT} = \frac{V_{BATT,OC} - V_{BATT,CC}}{I_{BATT,peak}} \tag{18}$$

The propagation of its uncertainty is given by the following RSS.

$$\begin{aligned} &\delta R_{INT} = \\ &= \sqrt{\left(\left|\frac{\partial R_{INT}}{\partial V_{BATT,OC}}\right| \cdot \delta V_{BATT,OC}\right)^2 + \left(\left|\frac{\partial R_{INT}}{\partial V_{BATT,CC}}\right| \cdot \delta V_{BATT,CC}\right)^2 + \left(\left|\frac{\partial R_{INT}}{\partial I_{BATT}}\right| \cdot \delta I_{BATT}\right)^2} \\ &= \sqrt{\left(\frac{\delta V_{BATT,OC}}{I_{BATT}}\right)^2 + \left(\frac{\delta V_{BATT,CC}}{I_{BATT}}\right)^2 + \left(\frac{V_{BATT,OC} - V_{BATT,CC}}{I_{BATT}} \cdot \frac{\delta I_{BATT}}{I_{BATT}}\right)^2} \end{aligned} \tag{19}$$

The following table sums up the uncertainty propagation of the battery current. Referring to the equation (19), the sources of error indicated as “Source A/B…” correspond to the ordered products of standard uncertainty and sensitivity coefficients. Note that their Relative Uncertainty is calculated with respect to the internal resistance uncertainty.

	Value	Absolute Uncertainty	Relative Uncertainty [%]	Unit
$V_{BATT,OC}$	$13.5 \cdot 10^{0}$	$\pm 15.0 \cdot 10^{-3}$	$\pm 0.11 \cdot 10^{0}$	V
$V_{BATT,CC}$	$13.4 \cdot 10^{0}$	$\pm 15.1 \cdot 10^{-3}$	$\pm 0.11 \cdot 10^{0}$	V
I_{BATT}	$8.00 \cdot 10^{0}$	$\pm 80.0 \cdot 10^{-3}$	$\pm 1.00 \cdot 10^{0}$	A
Source A	N.A.	$\pm 1.90 \cdot 10^{-3}$	$\pm 50.3 \cdot 10^{0}$	Ω
Source B	N.A.	$\pm 1.80 \cdot 10^{-3}$	$\pm 49.4 \cdot 10^{0}$	Ω
Source C	N.A.	$\pm 0.15 \cdot 10^{-3}$	$\pm 0.30 \cdot 10^{0}$	Ω
R_{INT}	$15 \cdot 10^{-3}$	$\pm 2.60 \cdot 10^{-3}$	$\pm 17.4 \cdot 10^{0}$	Ω

A first cycle of data acquisition was performed on few test batteries using a set of frequencies that went from 50Hz up to 1000Hz and again down to 50Hz with an increment of 0.05 on logarithmic scale, rounding to the nearest integer number.

The list of frequencies extends from $10^{1.7} = 50$Hz to $10^{3} = 1000$Hz, it is displayed in logarithmic scale in the plot below.

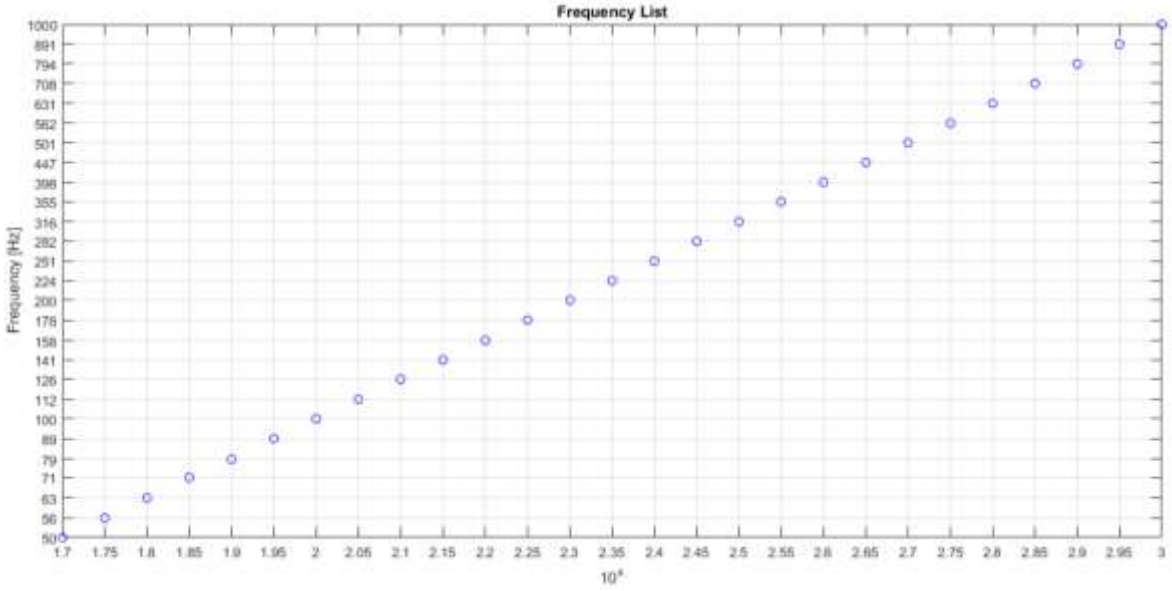


Figure 26. Frequency list initially used

After analyzing the data, it was noticed that over 200Hz the batteries could not reach a stable value of open circuit voltage within a period, making those acquisitions less valuable.

The new list of frequencies was set to contain: [0.1; 10; 20; 50; 100; 150; 200] Hz.

4.3 "SoC" design

The previous test produced very interesting results, amongst which the downsides of having a variable frequency control voltage. As a result, in the second test, it was decided to fix the period of current discharge to 5 seconds, while maintaining all the other control characteristics of wave shape, peak and mean current values (8A and 4A respectively).

The intention was to observe the change in the battery voltage and eventually extract the internal resistance of the battery, like in the previous experiment, but monitoring the variation of SoC.

The SoC can be computed with the following equation:

$$SoC_{\%}(t) = \frac{C_{remaining}(t)}{C_{nominal}} \cdot 100 = \frac{C_{nominal} - \left(N_{period}(t) \cdot C_{period}\right)}{C_{nominal}} \cdot 100 \quad (20)$$

Where:

$$C_{period} = \int_0^T I(t)\, dt = \int_0^{5s} 4A\, dt = 20As = 0.0056\, Ah \quad (21)$$

The discharge profile is set so that after every 10% drop in the SoC, the battery is not discharged for one hour. After that, the pulsed discharge starts again. The test is intended to stop if $V_{BATT} < 10.8V$, threshold at which the battery charge is considered too low for normal cranking operation. If this condition is met, the discharge must be interrupted and the one hour relax starts immediately after.

4.3.1 Software

The LabVIEW virtual instrument was simplified by removing the usage of different frequencies. On the other hand, the real-time computation of the SoC was added using

as initial condition that the SoC of the battery after being fully charged was 100%. Moreover, the condition to split the acquisition file was added to act by the end of the countdown timer of one hour, which follows the 10% decrement of SoC percentage. The one-hour waiting time was added to allow the voltage of the battery to be stable after the discharge phase. The stop condition on the battery voltage remained the same (threshold of 10.8V).

4.3.1.1 Structure of the Virtual Instrument

The only modification on the front panel was to substitute the frequency selector with SoC indicators. The handling of folder and file generations remained the same as the previous.

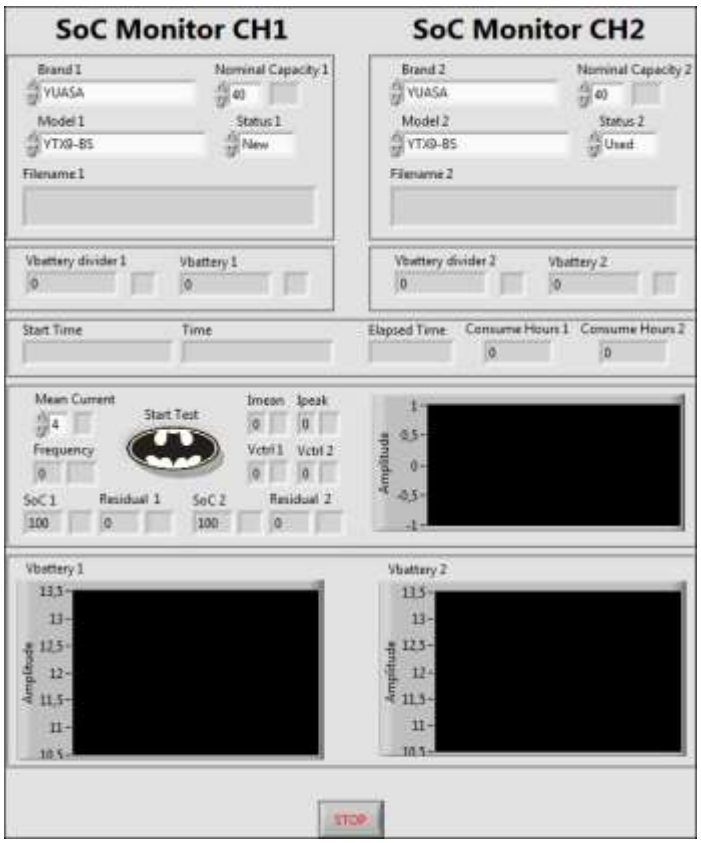


Figure 27. Front panel on "SoC" VI

The generation of the signal become more complex because of the controls and indicators behind its activation; the output signal is enabled when the SoC must diminish by its decremented percentage step that is 10%. Once the SoC decrement reaches the 10%, the output signal is disabled (0V) and a one-hour timer will start. The value commented by Seconds to Wait in SoC Ranges includes 3600, second in one hour, and 30, seconds between a new cycle and the start of the discharge and used as a settling time.

When the system forces the discharge, the indicator for Consume Hours will increment consequently.

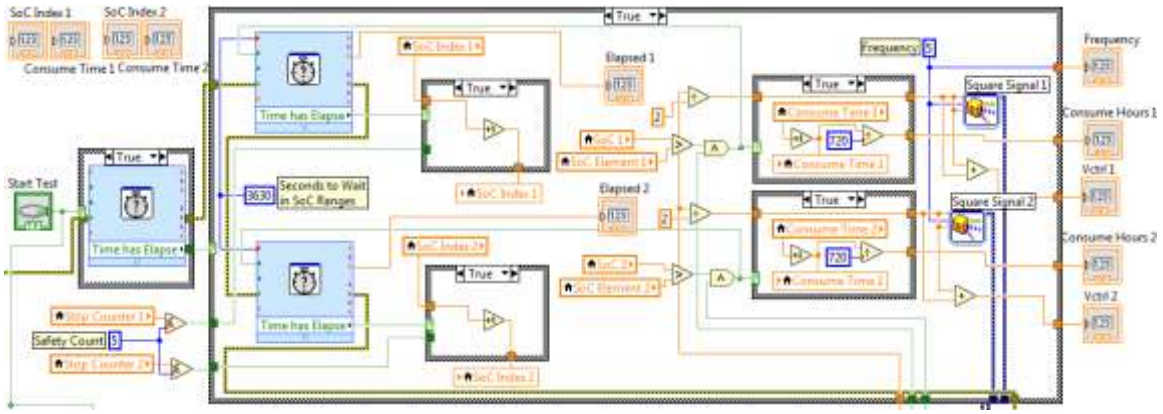


Figure 28. Section of block diagram to acquire and save data on "SoC" VI

The core of this VI resides in the computations of the SoC: starting from the nominal capacity, parameter set before starting the test, the capacity consumed by the battery will be removed from the nominal obtaining the residual capacity. This computation is possible thanks to the constant current in discharge drained from the battery and the continuous update of the time in which the battery is being discharged, obtaining how much capacity has been consumed in real-time.

Instead of putting some random numbers, in order to make the block diagram more readable, several variables with easy-to-recognize names were created and used to compute main values such as the mean current or the resistive load seen by the battery.

In the array (the numbers inside the blue rectangular shape) the SoC percentages goals are listed, going from 90 to 0 at a decrement step of 10.

To organize better the management of the files that contain the acquired measurements, instead of creating a single big file holding data of the entire test (that could last 10 hours), the file is split whenever the one-hour timer reaches one hour, separating each file on the basis of SoC starting percentage.

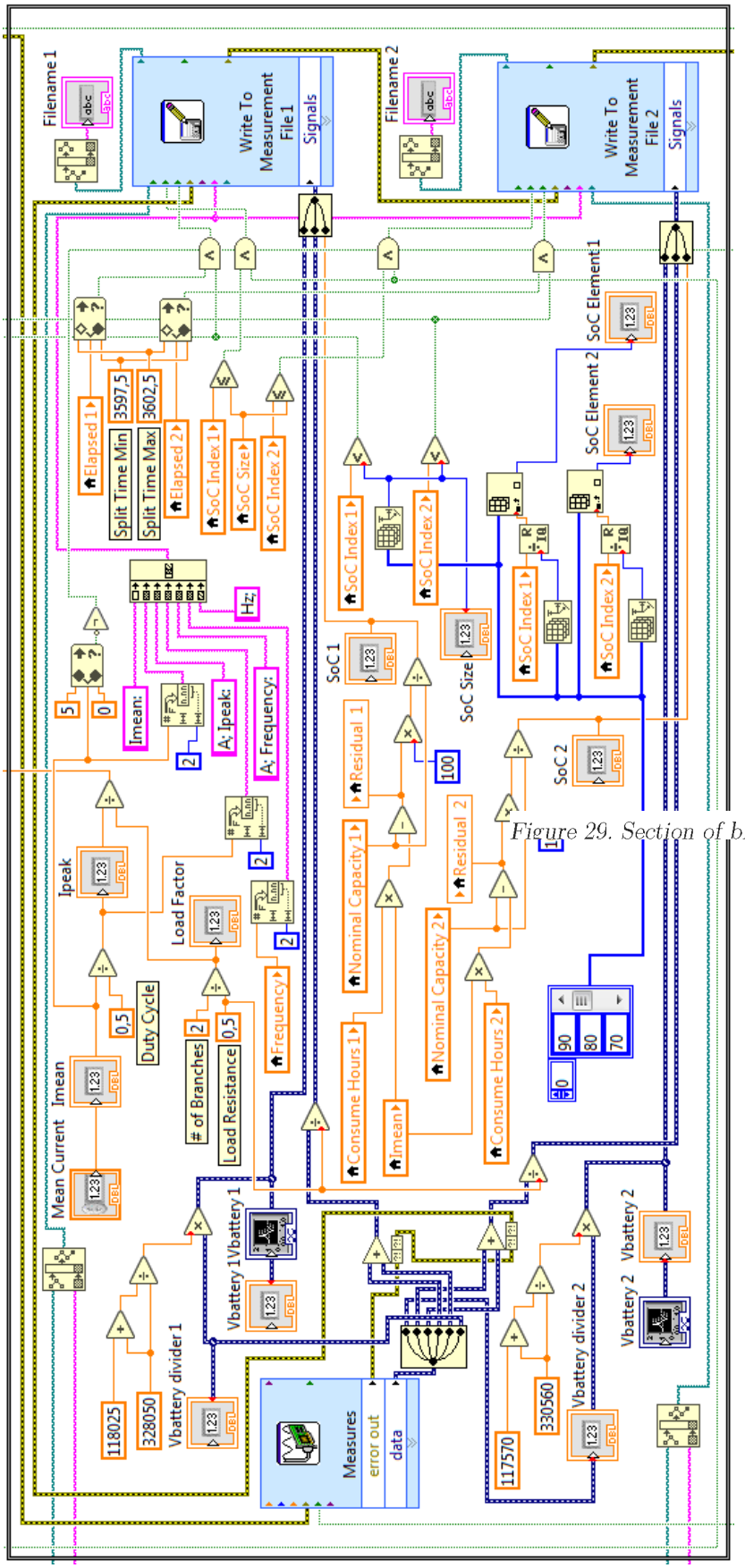


Figure 29. Section of block diagram to acquire and save da

The stop condition preventing the breaking of the battery, threshold of 10.8V in which the battery cannot go, is preserved.

A check on the SoC level being inside the ranges is done and, if it goes outside the SoC list, the test will stop.

The error check is preserved too, even if it was featured with a message box that will pop-up whenever an error occurs.

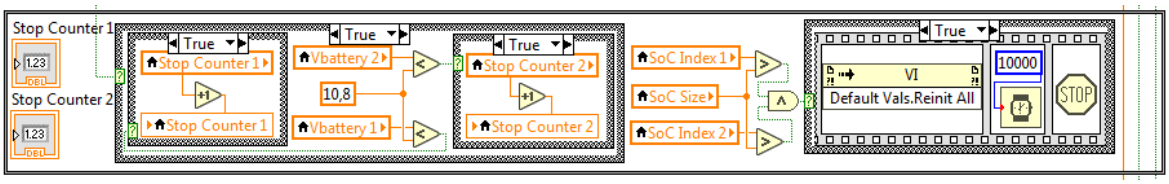


Figure 30. Section of block diagram to check conditions and stop the execution on "SoC" VI

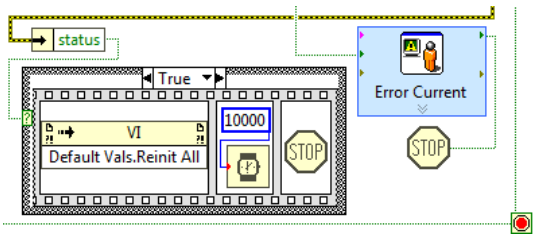


Figure 31. Section of block diagram to check errors and stop the execution on "SoC" VI

4.3.2 Hardware

The hardware platform remained unchanged, since the specifications about maximum power dissipation, current profile and control voltage were almost identical to the ones of the previous experiment.

4.3.3 Design efficacy

The goodness of the design is determined in terms of accuracy on the battery SoC. From equations (20) and (21), the uncertainty linked to the SoC calculations is simply that of the battery current, considering negligible the uncertainty on the passing time. Consequently, the SoC is known with an uncertainty of $\pm 1\%$ (see 4.2.3. - Design efficacy for further explanations).

4.4 "SoCSafe" design

Since the previous Design of Experiment yielded some impressive results, a lower decrement step of SoC was set to increase the resolution. Besides the focus was shifted towards higher values of SoC, range in which the battery shows a more characteristic

behavior. The decrement step on SoC was decreased from 10% to 5% and the cycles were limited to five discharge cycles each corresponding to a certain level of SoC ranging from 100% to 75%. As in the "SoC" design, a one hour of relax for the battery was applied every 5% decrement of SoC.

To avoid a deeper discharge and reduce its unwanted bad effects, the threshold corresponding to the stop condition was raised from 10.8V to 11.8V.

4.4.1 Software

This version of the LabVIEW project remained almost the same as the previous one, with two minor changes. Now each file is created every 5% drop in the SoC, instead of the previous 10%, until the 75% is reached; while the threshold voltage increase from 10.8V to 11.6V.

As before, between the two stop conditions, reaching the threshold voltage constitutes as a stronger exit situation and an occurring of an error is the strongest one.

4.4.2 Hardware

The hardware platform remained unchanged, since the specifications about maximum power dissipation, current profile and control voltage were identical to the ones of the previous experiment.

4.4.3 Design efficacy

The design allows to an increase in the SoC resolution by imposing the battery the one-hour relax period every 5% drop in the SoC. The accuracy of the related parameters remains the same of the previous experiment (see 4.2.3 - Design efficacy for further explanations).

4.5 "Deep" discharge design

During cranking, the battery is subject to a rapid and very high change in the current demand. To simulate in a smaller way this operational phase of the vehicle, the discharge profile was then enhanced. The idea was to apply a three-stage pattern in which the discharge current peak varied from 0A to 40A, at different time rates. These three stages are variable discharge pattern, constant discharge and one-hour relax time.

The variable discharge pattern consists of a sequence of patterns on three-level current profiles set at 10A, 30A and 40A, each lasting a fixed period of 0.5s or 5s. Every

discharge pattern is separated by a 30 second relax time; the entire sequence is reported below.

[INSERIRE IMMAGINE MATLAB PATTERNS]

Figure 32

To conform the global behavior to the previous two designs, a third stage consisting of a constant current discharge set at 4A was added. This stage aims at leveling, for every kind of battery, the SoC drop during the first cycle of test to a 5% decrease. Therefore, after having calculated the total charge in Ah of the variable pattern and adapted it in terms of SoC to the battery under test, the constant pattern period was set in order to discharge the unit of the remaining percentage to reach the 5% drop.

Finally, the usual one-hour countdown timer was set to relax the battery.

As in the "SoCSafe" design, this entire sequence was cycled until the 75% of SoC was reached or the voltage on the battery went below 10.8V.

To remove the dependency of the design from LabVIEW and the expensive instruments from National Instruments, a tiny developer board with an integrated micro controller unit (MCU) was used, making the system easier to employ and much more compact.

The instrument built to manage the entire operation is called Programmable Electronic Load (PEL).

4.5.1 Software

Instead of LabVIEW, MATLAB was employed as the main software platform in charge of gathering the data and administer all the operations of the system. The Nucleo development board was used to sample any useful quantities, transmit the acquired data and generate the controlling signals.

Practically speaking, the DAQ was replaced by the ADCs and DACs on the MCU, while the LabVIEW project by its firmware, for what concerned the signals generation and the data acquisition, and by the script on MATLAB for the data collection, computation activities and file storage.

To send and receive data between the Nucleo board and MATLAB, a UART serial communication was established, via TTL-232R-3V3 cable. The baud rate was set to the maximum applicable for both, that is 921600 baud.

The firmware of the MCU was configured to wait for commands from UART communication, for example a reset is performed whenever the received character is

'R', or the Monitor mode is engaged whenever the received character is 'M', and so on. Before a command execution is sent, a handshake transmission between both parties is performed to reduce any error communication caused by desynchronization.

The main task on the firmware is divided in two main routines. This choice was taken after an issue was noticed during testing, concerning the timers used to cycle through the discharge pattern that leaded to a stall of the DACs output. The issue is caused by the upper limit a timer can reach ($(2^{32} - 1)$ that corresponds to 71.6 minutes) at which the overflow simply goes undetected. The main task division, together with the reset performed between the two routines, ensured that each of them would last less than the limit.

~~During the functionality test, an error on real-time clock and, in general, on timer: using microsecond resolution, the maximum number obtainable is $(2^{32} - 1)$ that corresponds to 71.6 minutes. In worse cases, with higher nominal capacity battery, the task composed by deep current discharge, constant current discharge and one hour relax lasts longer than 71 minutes, almost twice. To avoid any stall hazard due to overflow on timer, the task was divided in half: the first executes the deep current discharge, constant current discharge and ten minutes of the one hour relax; the total time lasts less than 70 minutes. The second executes the remaining 50 minutes of the one hour relax.~~

The ADCs and DACs configurations were performed at low level, using the hardware abstraction layer (HAL) libraries. ADC and DAC sampling frequencies and resolution, calibration, input/output impedance, V_{ref}, single ended or diff, each ADC input is preceded by an RC filter.

The following flow chart clarifies the execution of the firmware.

[INSERIRE IL FLOW CHART DEL FIRMWARE]

Figure 33

On the other hand, the script in MATLAB was developed to send commands to the MCU exploiting the serial communication. A simple and intuitive graphic user interface (GUI) was created to simplify the management of the communication. Instead of performing different tasks manually, both the script and firmware run at the same time under the supervision of the script itself. The script manages the transfer continuously, saving all the data at a rate of 0.05s and, only every second, it updates a window with the last most significant data.

To start the main task, namely the cranking simulation test, the script in MATLAB sends a 'T' after an handshake communication, the firmware catches the character and, if it is 'T', an acknowledge is sent to MATLAB and the function managing the

pattern for cranking simulation is called; meanwhile MATLAB is prepared to receive data continuously and, to prove it, every second a window that visualize significant data updates. When the stage is finished, a 'End' string is sent to MATLAB.

The software was adapted by sending a '1' or a '2' after the 'T' character. Just for task '1', MATLAB sent the index of the list of battery inserted in the code, which needs to set for how long the constant current discharge must be executed to reach the 5% of SoC decrement because it depends on the nominal capacity.

Notice that for the user point-of-view, the characters '1' and '2' are hidden; the user starts the automated test, MATLAB sends every character, one at the time, after the firmware has finished and sent the 'End' string or an error occurred.

Every time a character command is received, after that the function returns, the developer board resets. The board is also reset when the command 'T' is invoked.

4.5.2 Hardware

The hardware design is divided into its different operational blocks. Each block performs a certain operation and communicates the relative data so that the entire system can work in a seamless way.

Each block is described in terms of function and circuit requirements, components choice, design analysis and simulation results.

[INSERIRE SCHEMA BLOCCHI]

Figure 34

As it can be seen in figure (X), the PEL is composed by current sink, voltage and current senses, temperature monitor and Nucleo board related circuitry.

The system is divided between two perfboards: one contains the analog circuit performing the current discharge, that is to say the current sink, along with two tubular heatsinks with integrated active cooling; the other board sits between the two heatsinks, raised from the first one by means of two metal spacers, and contains all the other circuitry alongside the Nucleo board.

The PEL is placed inside the usual metallic box, previously used as PC power supply unit, with integrated 12V fan of 15cm in diameter.

In the front panel of the unit there are: on the left, three LEDs (one yellow and two red) reporting any abnormal activity, above them two banana connectors sockets (red and black) for the +12V supply coming from the DC power supply; at the center the

big cooling fan, and on the right the XT90 power connector for the battery cables, with the custom 3D printed socket.

[INSERIRE IMMAGINI DEL PEL: interno ed esterno]

Figure 35

The negative pole of the battery is used, alongside the negative terminal of the +12V supply, as reference voltage (star point connected to 0V for all the components) and chassis connection.

A 100Ω resistor is placed between the two negative rails in order to eliminate a current injection phenomenon that occurred each time a current higher than 10A was flowing into the current sink. What happened was that some of this current would flow into the reference path of the sensing and digital circuit, inducing oscillations in several signals across the whole system. A simple resistor allowed to stop the injection current while maintaining the same voltage between the two negative rails.

Only components with through hole technology were chosen for the PEL. The packages used in the design are: PDIP-8T for op amps, TO-92 for thermistors and BJTs and TO-247 for MOSFETS.

4.5.2.1 Current sink

In the current sink, the discharge battery current is set through V_{CTRL}, generated by the DACs of the Nucleo board, across R_S, and it can change in terms of shape, amplitude and frequency following the relations of the equations (22).

$$\begin{cases} I_{BATT,peak} = 8 \cdot \dfrac{R_C}{R_C + R_D} \cdot \dfrac{V_{CTRL,peak}}{R_S} \cong 12.33 \cdot V_{CTRL,peak} \\ f_{I_{BATT}} = f_{V_{CTRL}} \end{cases} \qquad (22)$$

Comparing with the last two previous versions, the number of branches employed in the current sink for this test increased to eight, allowing the circuit to manage up to 10A of current. The branches are organized in two channels of four, each having a dedicated heatsink.

Before being fed to the non-inverting input of the op amp, a resistive divider scales down the control voltage by a factor of 6.49, this was meant to reduce the susceptibility of the signals to any nearby source of disturbance. The divider is composed of resistors having both tolerance of ±0.1%, while their value was chosen

to adapt the maximum DAC output voltage to the maximum voltage drop across R_S, which at 40A is 0.5V.

[INSERIRE SCHEMATICO CURRENT SINK]

Figure 36

One of the main challenges here was to organize the connections between battery, drain of the MOSFETs and setting resistors. For this purpose, the power distribution is managed using a 35µm copper clad board properly sized, inspired by the so-called BusBar (or PowerBus). Two identical boards are used one for the distribution of the negative pole of the battery, to which the setting resistors are connected, and the other for the positive pole, for the connections of the MOSFETs' drain. A hole of 5mm in diameter allows for a strong and secure connection by nut and bolt of the boards to the cables, provided with crimp ring terminal.

The assembly of the circuit is organized keeping in mind symmetry and maintaining the same distance between the same nodes in every branch. This strategy allows for an equal share of the current and a simultaneous engagement of each op amp.

[INSERIRE IMMAGINE COPPER CLAD E VISTA INFERIORE]

Figure 37

The remaining part of the circuit, the heart of the current sink, remained the same of the previous version. The only change made was to the setting resistor, now a *15FR100E by Ohmite* [19], is a 5W, 100mΩ ± 1% axial resistor (instead of the 30W, 500mΩ ± 1% of the previous version). The resistor is still low inductive and has good thermal characteristics. Therefore, all the relative design analysis about components choice and frequency stability is valid for this version too (see 4.2.2 - Hardware).

Considering that the total amount of power to be dissipated can be as high as 520W, the thermal design played a very important role. As for the previous version, each branch still has to dissipate around 60W of power, rating at which an isolated MOSFET junction would warm up to about 100°C, in the prior version. In the PEL, the four MOSFETs forming a channel are connected to the same heatsink, the *LAM3K by Fischer Elektronik* [19]. This heatsink, to which the MOSFETs are securely attached by metallic clips, has a tubular shape with internal fins and integrated 12V fan at one end that allows the air to flow through it, removing the heat. This heatsink together with the bigger fan mounted to the metallic enclosure, allows the air to be sucked from the back by its integrated fan and ejected with a higher rate at the front, guaranteeing a more efficient heat removal.

This setup leads to a total equivalent thermal resistance of around 0.3°C/W, as calculated for the previous version of the current sink (see 4.2.2.3 - Current sink: stability, transient and dissipation analysis). In the worst-case scenario (40A for 5s), the simulated equivalent circuit of figure (X) gave a maximum junction temperature of 160°C, at 27°C of ambient temperature. Considering the overestimation of the simulation, the design was approved despite a 15°C headroom on the maximum temperature rating of the MOSFETS' junction. The results are shown in figure (X).

[INSERIRE SCHEMATICO SIMULAZIONE TERMICA]

Figure 38

[INSERIRE GRAFICO RISULTATI SIMULAZIONE TERMICA]

Figure 39

4.5.2.2 Voltage sensing

The voltage sensing circuit was not changed too much from the previous version. The main changes were caused by the input dynamic of the Nucleo board ADCs, set at 3.3V. Consequently, the ratio for the resistive divider was modified to adapt the maximum expected battery voltage of 13.5V to the new range, leading to the choice of a 100kΩ and 324kΩ resistors, both with a ±0.1% tolerance. Across the divider a capacitor of 33pF was placed to act, together with the divider itself, as a low pass filter with cut-off frequency of about 60kHz.

[INSERIRE SCHEMATICO VOLTAGE SENSING]

Figure 40

Series resistor and clamping diodes remained. About the latter, the diode connected to the positive rail can be safely be joined to the +5V coming from the Nucleo board, due to the low current involved in the voltage spikes that might occur.

The newly introduced +5V supply voltage, was used to supply the op amp used as a buffer between the divider and the ADC. This requirement, alongside the ones used for the choice of the LTC1152, leaded to change the op amp for the *OPA350 by Texas Instruments* [19]. The OPA350 is a single rail-to-rail input/output (RRIO), low noise op amp with very good DC performances, and can be supplied with voltages between $[0 \div 7]$ V. Differently from the LTC1152, it has wider bandwidth (38MHz) but higher input offset voltage (±1µV typ. at 25°C).

Also, the values of the components forming the RC filter, preceded by the buffer, were changed. Based on the observations made on the activation time of the battery with

respect of the control voltage the following values, concerning the frequency events in the signal conditioning chain, were chosen:

- f_{SIGNAL} = 6kHz;
- $f_{AA} = 10 \cdot f_{SIGNAL}$ = 60kHz;
- $f_{GBW} \geq 50 \cdot f_{AA}$ = 3MHz (see *FilterPro™ User Guide* [16]);
- $f_S = 45 \cdot f_{SIGNAL}$ = 300kHz.

Using equation (10), already seen in the previous version of the voltage sensing circuit, the values for R_{FLT} and C_{FLT} are, respectively, 470Ω and 150pF.

Since this version of voltage sensing circuit is like the previous one, the transient and frequency stability are valid for this version too (see 4.2.2.6 - Voltage sensing: stability and transient analysis).

The battery voltage can be, once again, expressed as:

$$V_{BATT} = V_{AI} \cdot \frac{R_A + R_B}{R_B} + 2I_{BATT}R_W \qquad (23)$$

4.5.2.3 Current sensing

The current sensing is implemented extending the source of each MOSFET through a series 150kΩ resistor and measuring the mirrored control voltage drop across the setting resistor. The series resistors are used for isolation purposes. For each channel, the four sources, of their respective branch, are fed to a non-inverting adder composed by an op amp with positive gain. An RC low pass filter, which components are chosen following the same reasoning applied for the filter in the voltage sensing, is placed after each adder, before sampling. This solution, dictated by the numbers of ADC inputs equipped by the Nucleo board, allowed to monitor the current flowing in each of the two channels independently, so that this information could be interpreted to alert the user in case of abnormal operations (see 4.5.1 - Software for further info).

Following the nomenclature of the schematic in figure (X), the voltage present at each of the adders' output can be expressed as:

$$V_{I,CH1} = \frac{V_{RS1,CH1} + V_{RS2,CH1} + V_{RS3,CH1} + V_{RS4,CH1}}{4} \cdot \left(1 + \frac{R_{G2}}{R_{G1}}\right) \qquad (24)$$

The gain of the op amp is chosen so that at the output of the adder, the sum of the input voltages is present without being amplified ($G_{ADDER} = 4$).

From equation (25) the battery current can be expressed as:

$$I_{BATT} = \frac{V_{I,CH1} + V_{I,CH2}}{R_S} \tag{25}$$

The op amp used is the *OPA2340, by Texas Instruments* [19]. The OPA2340 is a dual rail-to-rail input/output (RRIO), general-purpose op amp with good DC performances that can be supplied with voltages between [0 ÷ 5.5] V. It has an input offset voltage of 150µV (typ. at 25°C) and a bandwidth of 5.5MHz.

Considering the resistive nature of the feedback and that the capacitive components show their frequency dependent effects at very low (op amp output impedance) or very high (C_{FLT}) frequency, the stability is guaranteed.

Before running the simulations, the model of the op amp used was imported as a macro model in TINA. Then it was validated by plotting useful DC and AC characteristics and confronting them with the ones present on the datasheets.

[INSERIRE SCHEMATICO ADDER]

Figure 41

4.5.2.4 Temperature monitor

The temperature monitor has the double function of protection and troubleshooting. The actual sensing is implemented using two thermistors. The sensors are placed near one of the four MOSFETs of a channel and are mounted in the same manner, using a metal clip. The ADC converters of the Nucleo board then samples the output of the thermistors and reacts shutting down the test if the maximum temperature is exceeded.

The component used is the *MCP9701A by Microchip* [19]. The MCP9701A is a low-power linear active thermistor with wide measurement range ([-40 ÷ +125] °C), and operating voltage range between [3.1 ÷ 5.5] V. The integrated circuit is optimized to be used with an ADC converter without any additional conditioning circuitry, however a series 200Ω resistor and 1µF decoupling capacitor can be added between the supply rails to reduce noise.

The sensor converts a temperature to an analog voltage, following equation (26).

$$T = \frac{V_{OUT} - V_0}{T_C} \tag{26}$$

Where V_0 is an offset of 400mV, corresponding to the output at 0°C, and T_C is the temperature coefficient of 19.5mV/°C.

The maximum accuracy of the sensor can be improved, from ±2°C to ±0.5°C, by performing a calibration at 25°C. Since the thermistor is said to be linear, the calibration will affect the offset voltage. The procedure was performed one time for each thermistor, following these steps:

- a thermo-hygrometer (Testo 735, with ±0.2°C of accuracy in the range of [-100 ÷ +199.9] °C) was placed close to the thermistor, taking note of the temperature when steady enough;
- without discharging the battery (in monitor mode, see 4.5.1 - Software for further info) the temperature indicated by the thermistor was found averaging the samples of a 5 minutes test;
- the new offset, V'_0, was then calculated using the following equation:

$$V'_0 = (T_{thermistor} - T_{t-h}) \cdot T_C + V_0 \tag{27}$$

4.5.2.5 Nucleo board related circuitry

The Nucleo development board is the digital core of the system. It controls the discharge current through V_{CTRL}, samples battery voltage and current, monitors channels' temperature, supplies some of the op amps in the system and turn on and off the BJTs driving the LEDs on the front panel.

The device choice was driven mainly by two requirements: small size, since the enclosure was not customized and there were little space remaining after mounting the heatsinks and the current sink board; available peripherals to keep tract of several signals while send or save them and managing the controls.

The *STM32 NUCLEO-F303K8 development board* [19] mounts an Arm Cortex 32-bit MCU and comes in a 32-pin package measuring only (18 x 50) mm. It is equipped, amongst others, with two ADCs having multiplexed channels, two DACs with un-buffered outputs, several digital I/O ports, a +5V input/output and USART serial

interface. It can be powered via USB or external voltage supply of [7 ÷ 12] V, +5V or +3.3V.

The BJT used is the *BC337-40 by ON Semiconductor* [20]. The BC337-40 is an NPN amplifier transistor with maximum ratings of 45V for collector-emitter voltage, 5V for base-emitter voltage and 800mA for collector current.

The op amp used as buffer for the converters is the OPA2340 by Texas Instruments (see 4.5.2.3 - Current sensing for further info)

The same control voltage for the current sink is generated by two independent DACs, one for each channel, so that they can be controlled independently, if needed. Each DAC is followed by a little capacitor of 6.8pF, aimed at smooth out the start-up or reduce any random fast glitch of the converters, and an op amp (OPA2340) configured as voltage follower, to minimize the interaction between the output impedance of the converter and the rest of the circuit, thus reducing the load seen by the DAC output.

The circuit implemented to drive the LEDs is very simple and it uses only one BJT and one resistor per LED. Three of the Nucleo digital outputs are connected to the base of the BJTs and, when high, activate the transistors. The base and emitter currents are set via the emitter resistor so that through the LEDs can flow a current of 5mA, which, in turn, generates a knee voltage of 1.7V. Besides, using the minimum DC current gain, ensured by the manufacturer in this operating condition, the base current is verified to be lower than the maximum output current of the digital port. The overall drop across the diode and resistor with respect to the +12V, used to supply the LEDs, and the 3.3V of the digital output, must allow some room for the saturated base-emitter and collector-emitter voltages to drop. The value of resistance is then found as:

$$R_E = \frac{V_{OH} - V_D - V_{BE,sat}}{I_E} \tag{28}$$

The board is supplied with the +12V coming from the DC power supply. Two 220µF bulk electrolytic capacitors are placed one on each of the two perfboards, as energy reservoirs and to flatten out the supply voltage from the noise created by the fans.

The +5V of the Nucleo board is used, in turn, to supply: the OPA2340 used as buffer for the DACs, the OPA350 employed in the voltage sense, the adders in the current sense and the thermistors. All these components require a fraction of the maximum 500mA allowed from the manufacturer.

[INSERIRE SCHEMATICO NUCLEO, Cbyp, BJT LED]

Figure 42

4.5.3 Design efficacy

As in the previous version of the circuit containing current sink and voltage sense (see 4.2.3 - Design efficacy for further info) the efficacy of the system is determined in terms of DC (or static) accuracy on battery discharge current and battery voltage.

The uncertainty is calculated using the Root Sum Square (RSS) technique, which gives a better estimate of the propagation with respect to the more pessimistic of the deterministic method. All the sources used in the calculations are considered statistically independent (uncorrelated) and at their maximum (if provided by the manufacturer).

The current sink can generate a current with ±1.14% of uncertainty, in the range between $[0 \div 40]$ A (the value is intended to be an upper limit for the parameter). During the constant discharge phase at 4A, the current seems noisier than the one sampled in the other range of values. This variation in precision could be caused by the number of active branches and low current per branch but is not considered linked to a higher uncertainty of the current itself.

The main sources of uncertainty are the fabrication tolerance of the resistor, DAC and ADC accuracy. Other causes are considered negligible with respect to these two.

From equation (22), the uncertainty on the battery current can be expressed as:

$$\delta I_{BATT} = \\ = \sqrt{\left(\left|\frac{\partial I_{BATT}}{\partial R_C}\right| \cdot \delta R_C\right)^2 + \left(\left|\frac{\partial I_{BATT}}{\partial R_D}\right| \cdot \delta R_D\right)^2 + \left(\left|\frac{\partial I_{BATT}}{\partial R_S}\right| \cdot \delta R_S\right)^2 + \left(\left|\frac{\partial I_{BATT}}{\partial V_{CTRL}}\right| \cdot \delta V_{CTRL}\right)^2} \quad (29) \\ = \sqrt{\left(\frac{8 \cdot V_{CTRL}}{R_s} \cdot \frac{R_C}{R_C + R_D}\right)^2 \cdot \left(\left(\frac{R_D}{R_C} \cdot \frac{\delta R_C}{R_C}\right)^2 + \left(\frac{\delta R_D}{R_C + R_D}\right)^2 + \left(\frac{\delta R_S}{R_S}\right)^2 + \left(\frac{\delta V_{CTRL}}{V_{CTRL}}\right)^2\right)}$$

The term δV_{CTRL} corresponds to the Total Unadjusted Error (TUE) of the DAC and it is evaluated as RSS of its main sources of error.

$$\delta V_{CTRL} = \sqrt{GAIN^2 + OFFSET^2 + DNL^2 + INL^2} \quad (30)$$

Where:

	From Datasheet (max.)	Converted value [mV]
Gain error	±0.5%	±16.5
Offset error	±12LSB	±9.6
DNL error	±2LSB	±1.6
INL error	±4LSB	±3.2
TUE	N.A.	±19.4

The term δV_{RS} corresponds to the Total Unadjusted Error (TUE) of the ADC. It is evaluated in the same way of δV_{CTRL} (via RSS), and provided directly from the manufacturer as ±5LSB (±4.0mV).

The setting resistors were measured using the Rigol DM3051 digital multimeter, which accuracy is given as ±(0.05% of reading + 0.01% of range).

The following table sums up the uncertainty propagation of the battery current. Referring to the equation (29) the sources of error indicated as "Source A/B…" correspond to the ordered products of standard uncertainty and sensitivity coefficients. Note that their Relative Uncertainty is calculated with respect to the battery current uncertainty.

	Value	Absolute Uncertainty	Relative Uncertainty [%]	Unit
R_S	$100 \cdot 10^{-3}$	$\pm 1.00 \cdot 10^{-3}$	$\pm 1.00 \cdot 10^{0}$	Ω
V_{CTRL}	$3.24 \cdot 10^{0}$	$\pm 19.4 \cdot 10^{-3}$	$\pm 0.60 \cdot 10^{0}$	V
R_C	$10.0 \cdot 10^{3}$	$\pm 10.0 \cdot 10^{0}$	$\pm 0.10 \cdot 10^{0}$	Ω
R_D	$54.9 \cdot 10^{3}$	$\pm 54.9 \cdot 10^{0}$	$\pm 0.10 \cdot 10^{0}$	Ω
Source A	N.A.	$\pm 0.22 \cdot 10^{0}$	$\pm 23.0 \cdot 10^{0}$	A
Source B	N.A.	$\pm 0.03 \cdot 10^{0}$	$\pm 0.55 \cdot 10^{0}$	A
Source C	N.A.	$\pm 0.40 \cdot 10^{0}$	$\pm 76.42 \cdot 10^{0}$	A
Source D	N.A.	$\pm 0.74 \cdot 10^{-3}$	$\pm 0.26 \cdot 10^{-3}$	A
I_{BATT}	$40.0 \cdot 10^{0}$	$\pm 0.46 \cdot 10^{0}$	$\pm 1.14 \cdot 10^{0}$	A

The voltage sensing circuit can measure a voltage with ±0.17% of uncertainty, in the range [0 ÷ 13.5] V (the value is intended to be an upper limit for the parameter).

The main sources of uncertainty are: resistors tolerance, ADC input error, input offset voltage of the op amp and the voltage drop across the connections when the current is different from zero. Other causes are considered negligible.

The load linked to the connections is caused by the resistance of the cables and power planes distribution. The former is derived known the diameter and length of the cables connecting the battery to the current sink. The latter using the resistivity of copper and the dimensions of the plates. The values found are, respectively, of 2.61mΩ and 6.03mΩ, which summed give a total resistance of 8.64mΩ.

Using equation (23), the uncertainty of the battery voltage, during the discharge phase (with the circuit closed, CC), can be expressed as:

$$\delta V_{BATT,CC} =$$

$$= \sqrt{\left(\left|\frac{\partial V_{BATT}}{\partial V_{AI}}\right| \cdot \delta V_{AI}\right)^2 + \left(\left|\frac{\partial V_{BATT}}{\partial R_A}\right| \cdot \delta R_A\right)^2 + \left(\left|\frac{\partial V_{BATT}}{\partial R_B}\right| \cdot \delta R_B\right)^2 + {V_{OFF}}^2 + \left(\left|\frac{\partial V_{BATT}}{\partial I_{BATT}}\right| \cdot \delta I_{BATT}\right)^2} \quad (31)$$

$$= \sqrt{\left(\frac{R_A + R_B}{R_B} \cdot \delta V_{AI}\right)^2 + \left(\frac{V_{AI}}{R_B}\right)^2 \cdot \left({\delta R_A}^2 + \left(\frac{R_A \cdot \delta R_B}{R_B}\right)^2\right) + {V_{OFF}}^2 + (2 \cdot R_W \cdot \delta I_{BATT})^2}$$

The term δV_{AI} corresponds to the Total Unadjusted Error (TUE) of the ADC. It is evaluated in the same way of δV_{CTRL} (via RSS), and provided directly from the manufacturer as ±5LSB (±4.0mV).

The following table sums up the uncertainty propagation of the battery voltage. Referring to the equation (31) the sources of error indicated as "Source A/B…" correspond to the various partial derivatives and coefficients. Note that their Relative Uncertainty is calculated with respect of the battery voltage uncertainty.

	Value	Absolute Uncertainty	Relative Uncertainty [%]	Unit
V_{AI}	$3.18 \cdot 10^{0}$	$\pm 4.00 \cdot 10^{-3}$	$\pm 0.12 \cdot 10^{0}$	V
R_A	$324 \cdot 10^{3}$	$\pm 324 \cdot 10^{0}$	$\pm 0.10 \cdot 10^{0}$	Ω
R_B	$100 \cdot 10^{3}$	$\pm 100 \cdot 10^{0}$	$\pm 0.10 \cdot 10^{0}$	Ω
Source A	N.A.	$\pm 0.01 \cdot 10^{0}$	$\pm 48.3 \cdot 10^{0}$	V
Source B	N.A.	$\pm 0.01 \cdot 10^{0}$	$\pm 19.6 \cdot 10^{0}$	V
Source C	N.A.	$\pm 0.01 \cdot 10^{0}$	$\pm 19.6 \cdot 10^{0}$	V
V_{OFF}	N.A.	$\pm 0.50 \cdot 10^{-3}$	$\pm 0.05 \cdot 10^{0}$	V
R_W	$8.64 \cdot 10^{-3}$	N.A.	N.A.	Ω
Source D	N.A.	$\pm 8.00 \cdot 10^{-3}$	$\pm 12.5 \cdot 10^{0}$	V
I_{BATT}	$40.0 \cdot 10^{0}$	$\pm 0.46 \cdot 10^{0}$	$\pm 1.14 \cdot 10^{0}$	A
$V_{BATT,CC}$	$12.9 \cdot 10^{0}$	$\pm 0.02 \cdot 10^{0}$	$\pm 0.17 \cdot 10^{0}$	V

Finally, the battery internal resistance can be calculated using equation (32) where $V_{BATT,OC}$ represents the battery voltage when not discharging. The uncertainty propagation is performed keeping in mind that the systematic error due to the intrinsic resistance of the cable is present only during the discharge phase. The internal resistance is known with ±5.25% of uncertainty (the value is intended to be an upper limit for the parameter).

$$R_{INT} = \frac{V_{BATT,OC} - V_{BATT,CC}}{I_{BATT,peak}} \tag{32}$$

The propagation of its uncertainty is given by the following RSS.

$$\begin{aligned} &\delta R_{INT} = \\ &= \sqrt{\left(\left|\frac{\partial R_{INT}}{\partial V_{BATT,OC}}\right| \cdot \delta V_{BATT,OC}\right)^2 + \left(\left|\frac{\partial R_{INT}}{\partial V_{BATT,CC}}\right| \cdot \delta V_{BATT,CC}\right)^2 + \left(\left|\frac{\partial R_{INT}}{\partial I_{BATT}}\right| \cdot \delta I_{BATT}\right)^2} \\ &= \sqrt{\left(\frac{\delta V_{BATT,OC}}{I_{BATT}}\right)^2 + \left(\frac{\delta V_{BATT,CC}}{I_{BATT}}\right)^2 + \left(\frac{V_{BATT,OC} - V_{BATT,CC}}{I_{BATT}} \cdot \frac{\delta I_{BATT}}{I_{BATT}}\right)^2} \end{aligned} \tag{33}$$

The following table sums up the uncertainty propagation of the battery current. Referring to the equation (33), the sources of error indicated as "Source A/B…" correspond to the ordered products of standard uncertainty and sensitivity coefficients. Note that their Relative Uncertainty is calculated with respect to the internal resistance uncertainty.

	Value	Absolute Uncertainty	Relative Uncertainty [%]	Unit
$V_{BATT,OC}$	$13.5 \cdot 10^{0}$	$\pm 0.02 \cdot 10^{0}$	$\pm 0.16 \cdot 10^{0}$	V
$V_{BATT,CC}$	$12.9 \cdot 10^{0}$	$\pm 0.02 \cdot 10^{0}$	$\pm 0.17 \cdot 10^{0}$	V
I_{BATT}	$40.0 \cdot 10^{0}$	$\pm 0.46 \cdot 10^{0}$	$\pm 1.14 \cdot 10^{0}$	A
Source A	N.A.	$\pm 0.54 \cdot 10^{-3}$	$\pm 46.9 \cdot 10^{0}$	Ω
Source B	N.A.	$\pm 0.55 \cdot 10^{-3}$	$\pm 48.4 \cdot 10^{0}$	Ω
Source C	N.A.	$\pm 0.17 \cdot 10^{-3}$	$\pm 4.70 \cdot 10^{0}$	Ω
R_{INT}	$15 \cdot 10^{-3}$	$\pm 0.79 \cdot 10^{-3}$	$\pm 5.25 \cdot 10^{0}$	Ω

Alongside the analysis of the accuracy of the system, the design efficacy has to consider also the operation of the system protections.

The thermal protection engaged a couple of times during testing due to a bug in the code that made the DACs' output to stall for few minutes at 1.62V, corresponding to a battery current of 20A. The temperature rose up until the 50°C threshold was exceeded, and the system was shut off without being damaged. Considering the thermistors' thermal response of 1.65s and the quicker firmware, the engaging was quite quickly.

During normal operation the measured temperature can be seen increasing and decreasing according to the flow of the battery current, reaching a maximum of about 43°C, while, during constant current discharge, it settles at around 38°C. An example can be seen in the figure below.

[INSERIRE GRAFICO MATLAB CON ESEMPIO DI TEMPERATURA A 40A]

Figure 43

Chapter 5.

Measurements campaigns

To maintain well organized each acquisition, a path sequence and filename patter were chosen during the first elaboration of acquisition bench and they were always applied: starting from the workspace folder, a set of subfolders were created: first for the Brand, second for the Model and third for the Date and Time when the test started:

Workspace folder / “Brand Name” / “Model Name” /
“Date & Time of start” – “Comments” /

Finally, a file containing all useful data, acquired and not, regarding the test was created and split at a specific time interval or amount of space occupied on disk, respecting this pattern:

“TestType”-“BrandName”-“ModelName”-“ConditionOfBattery”_“FileIncrement”

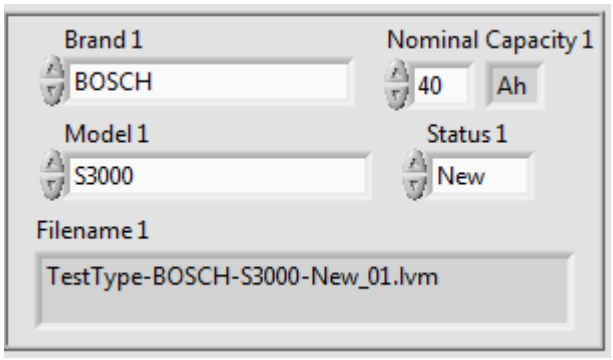


Figure 44

Among the data saved, there are the timestamp, the voltage on the voltage divider of the battery, the computed voltage of the battery, the voltage on the load, the computed current on the load, the computed impedance and the frequency.

5.1 Data Handling

To analyze easily so much measures, MATLAB was needed. The data produced by LabVIEW were essentially text data files with coma separation in between values. MATLAB itself was able to catch those data given the proper configuration on the “import data from text” functionality and save them to the workspace.

After understanding how MATLAB could recognize, transform and adapt those values from that text file, a script and, consecutively, an executable were built to accelerate the elaboration process, giving it a nice aesthetic touch by adding an explorer window for the path location of the LabVIEW files to convert and a progress bar for each file to convert found in that folder.

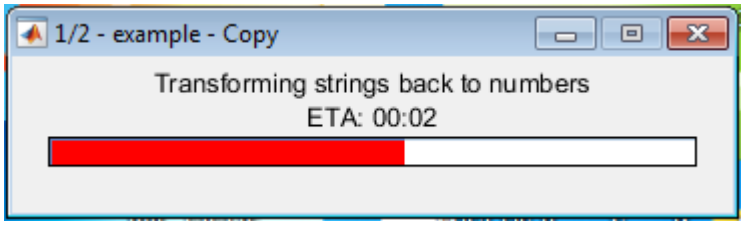


Figure 45. Execution windows of .LVM to .MAT converter

The main returns using such conversion tool were that, given the same pattern of LabVIEW file (and every LabVIEW virtual instruments implemented the same pattern for filename generation and data structure and order since the beginning), the configuration, especially for the pattern recognition, was already implemented once for all and using MATLAB, the data were also compressed in accordance to MATLAB compression system. In this way, the compress ratio was about 80% and the loading time on MATLAB of that big amount of data was minimized.

	Total time [hour]	Original size of data	Compress size of data
Spectrum	1400	218	N.A.
SoC / SoCSafe	410	4.7	0.73
DeepDischarge			

5.2 Recognizer tools

In this paragraph, a series of graphs representing how the data acquired was able to collect only the useful data to further produce comparative graphs.

5.2.1 DeltaV recognizer for “Spectrum” on a single frequency

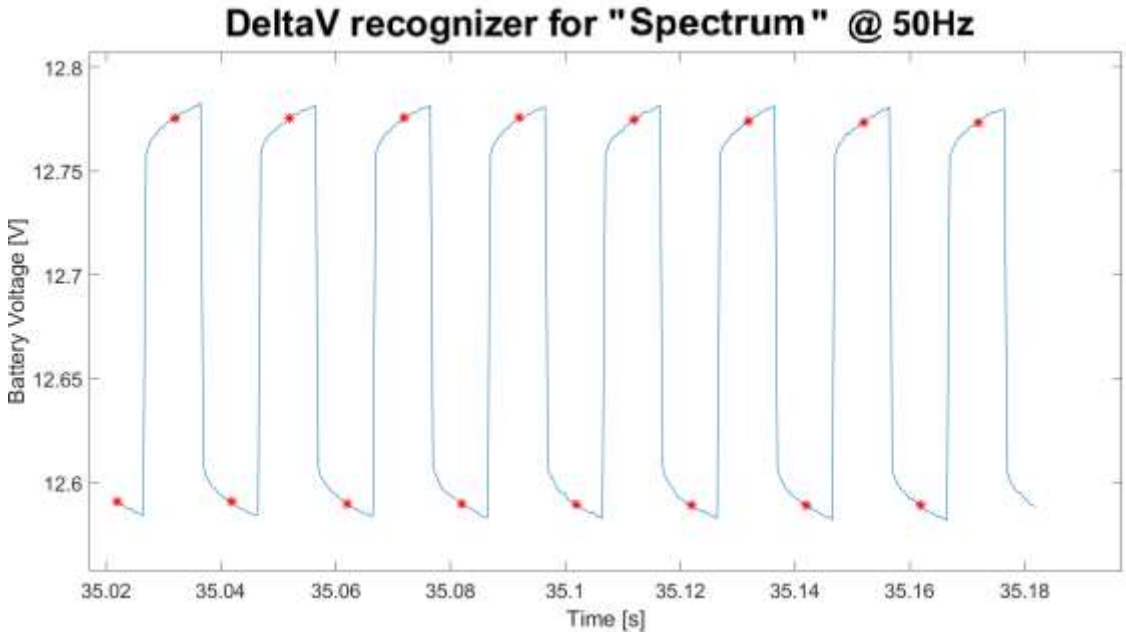


Figure 46. Open circuit and discharging voltage detection for “Spectrum” data

5.2.2 DeltaV recognizer for “SoC” and “SoCSafe”

IMMAGINE

Figure 47. Open circuit and discharging voltage detection for “SoC” and “SoCSafe” data

5.2.3 DeltaV recognizer for “Deep”

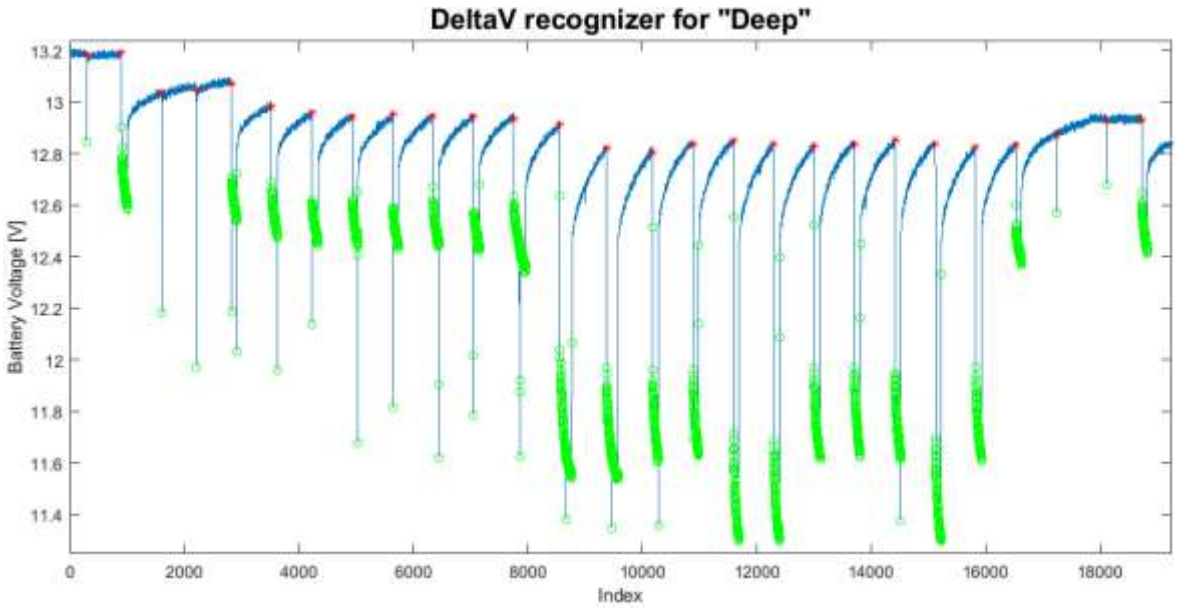


Figure 48. Open circuit and discharging voltage detection for "Deep" data

5.3 "Spectrum" campaign

The experiment was performed 10 times on each battery, gaining a total of 90 test cycles for each SoC level, at least for those the battery has managed to reach.

5.3.1 Concept art

5.3.1.1 Plot of an acquisition cycle

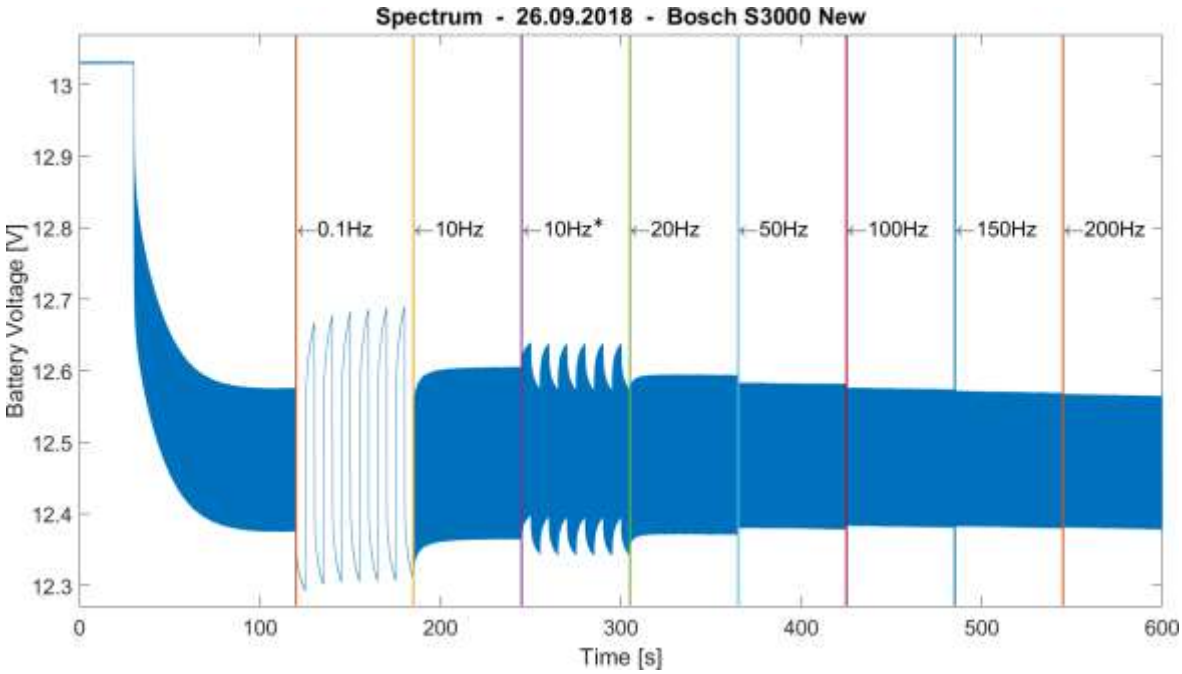


Figure 49. Plot of an acquisition cycle of "Spectrum" data

5.3.1.2 Detail of the plot of an acquisition cycle

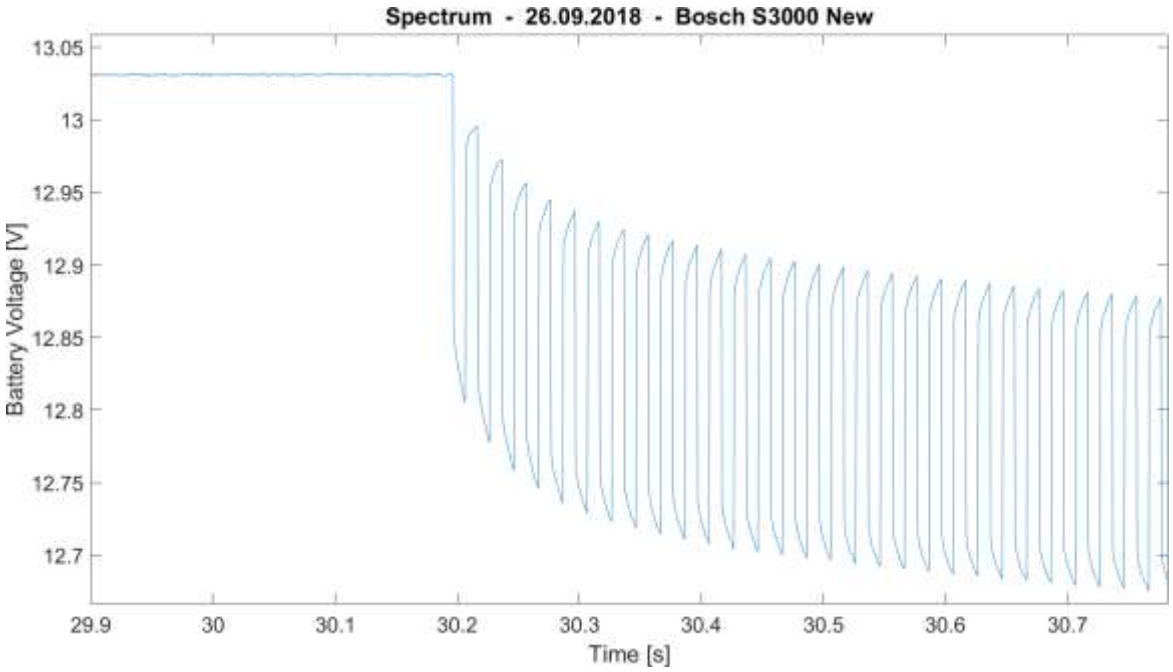


Figure 50. Detail of the plot of an acquisition of "Spectrum" data

5.3.2 Battery analysis

5.3.2.1 Bosch S3000

5.3.2.1.1 New condition

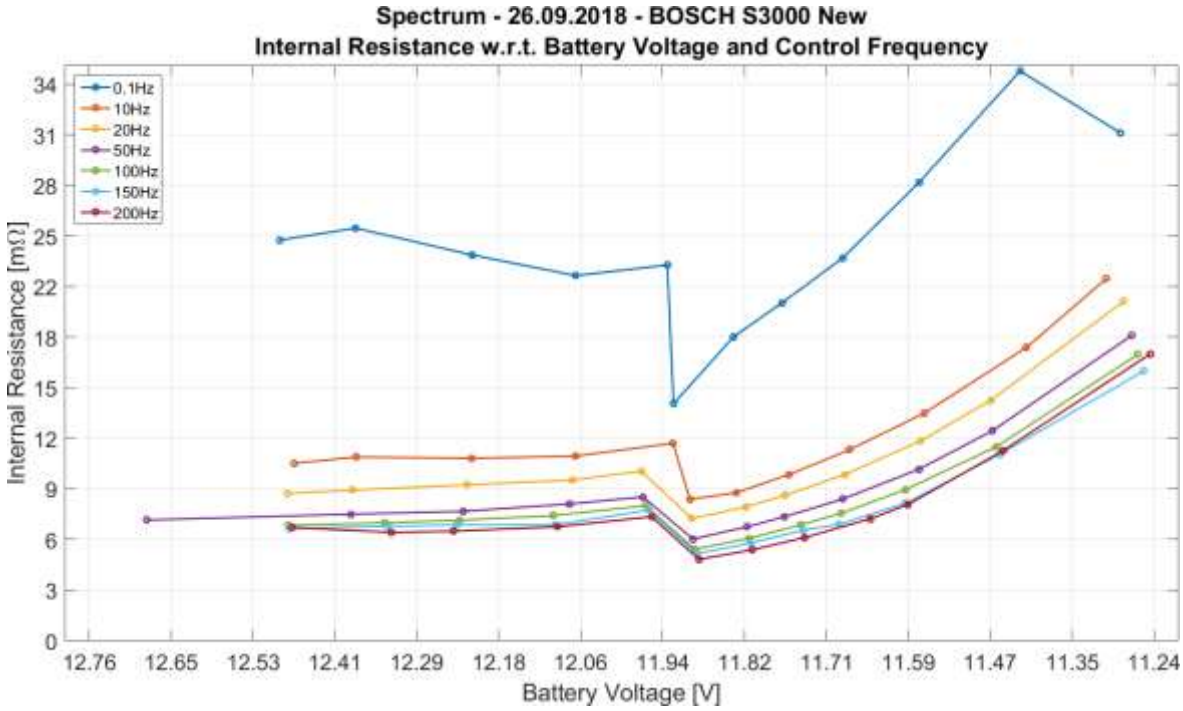


Figure 51. "Spectrum" New Bosch Internal Resistance w.r.t. Battery Voltage and Control Frequency graphs

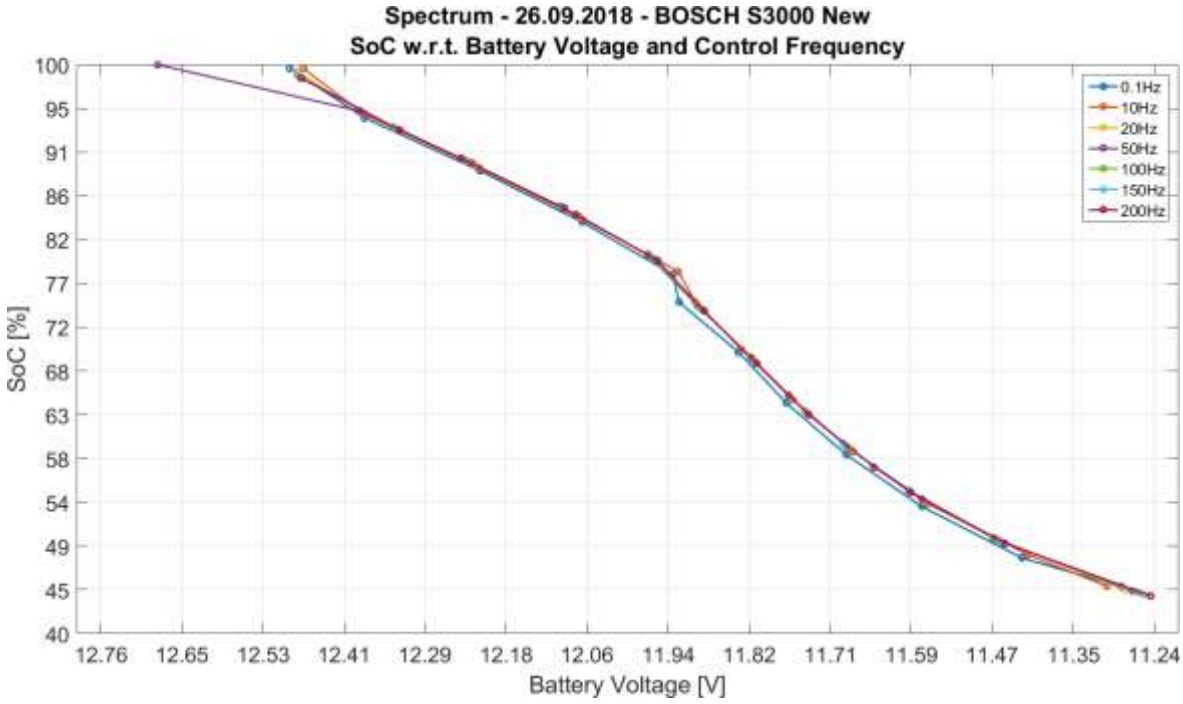


Figure 52. "Spectrum" New Bosch SoC w.r.t. Battery Voltage and Control Frequency graphs

5.3.2.1.2 Used condition

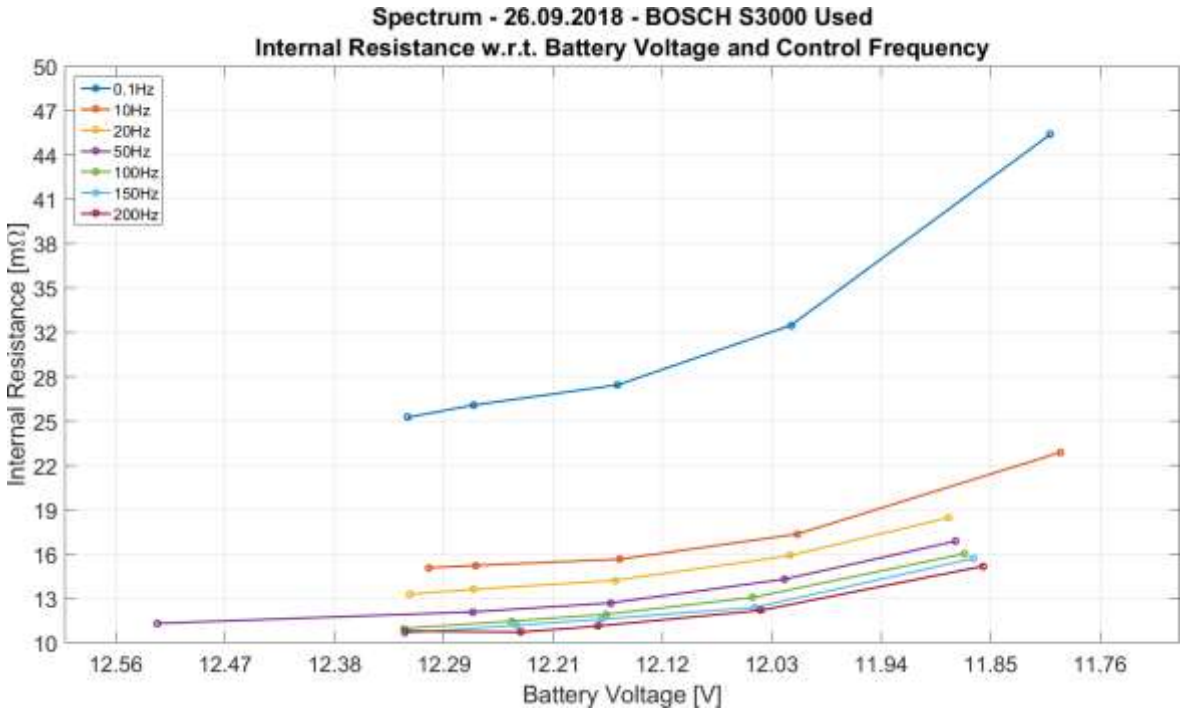


Figure 53. "Spectrum" Used Bosch Internal Resistance w.r.t. Battery Voltage and Control Frequency graphs

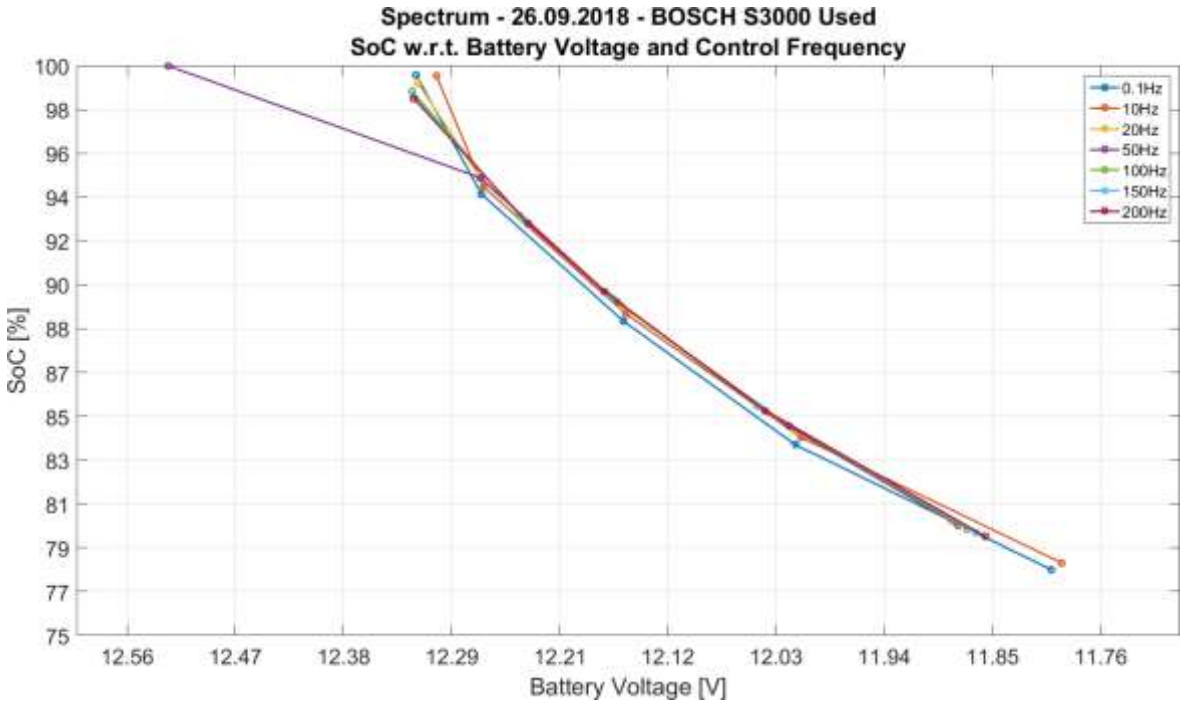


Figure 54. "Spectrum" Used Bosch SoC w.r.t. Battery Voltage and Control Frequency graphs

5.3.2.2 Energeco Clausum-S562.059.057

5.3.2.2.1 New condition

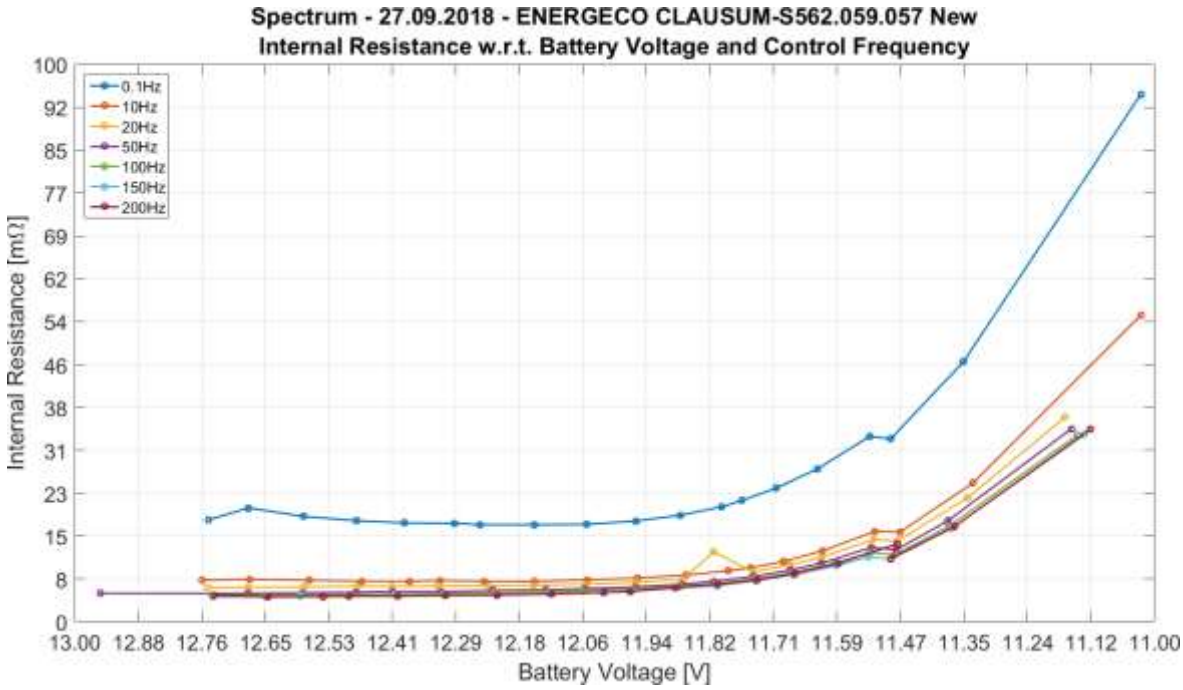


Figure 55. "Spectrum" New Energeco Internal Resistance w.r.t. Battery Voltage and Control Frequency graphs

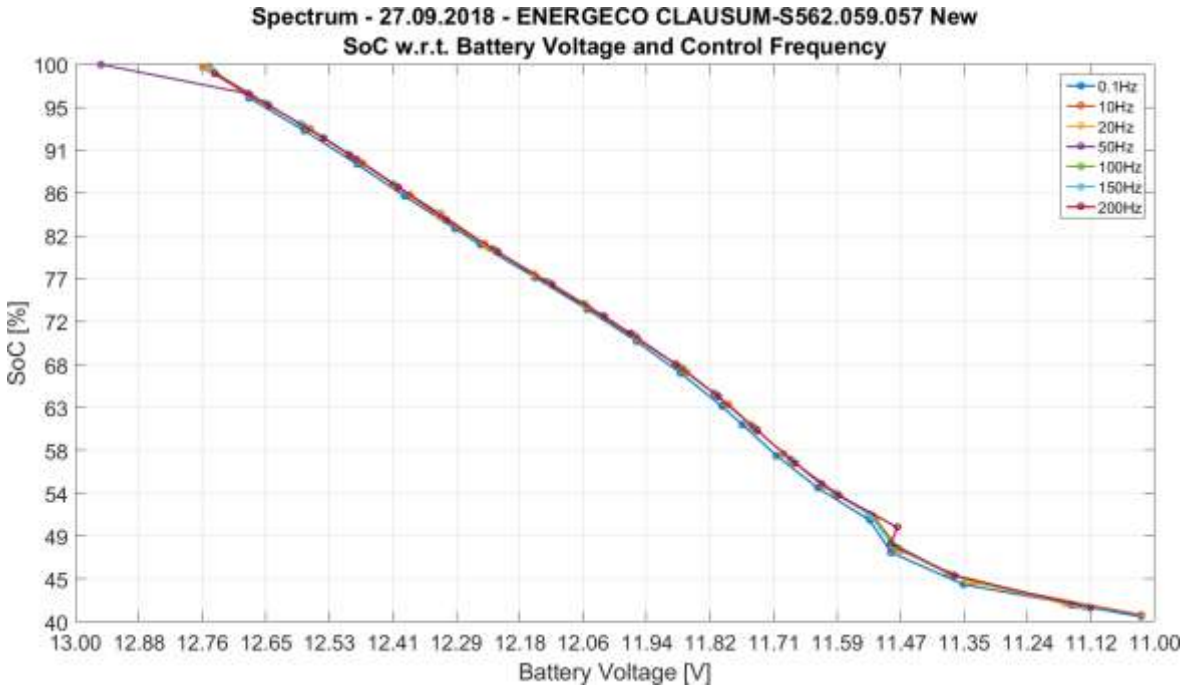


Figure 56. "Spectrum" New Energeco SoC w.r.t. Battery Voltage and Control Frequency graphs

5.3.2.2.2 Used condition

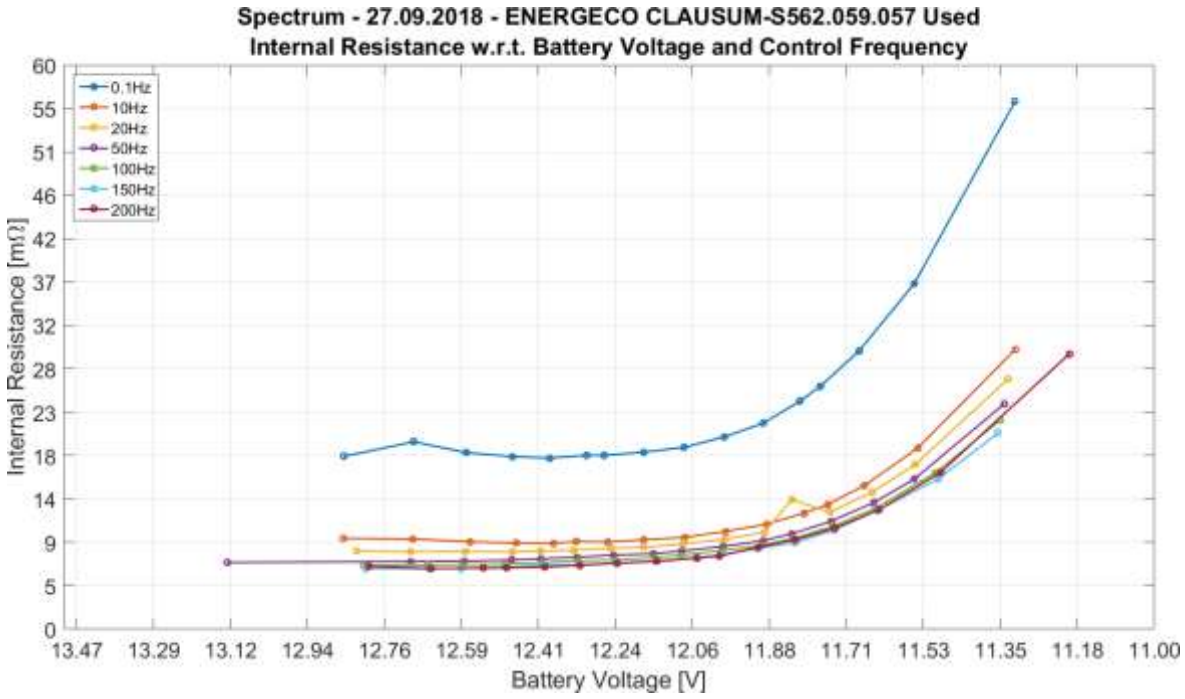


Figure 57. "Spectrum" Used Bosch Internal Resistance w.r.t. Battery Voltage and Control Frequency graphs

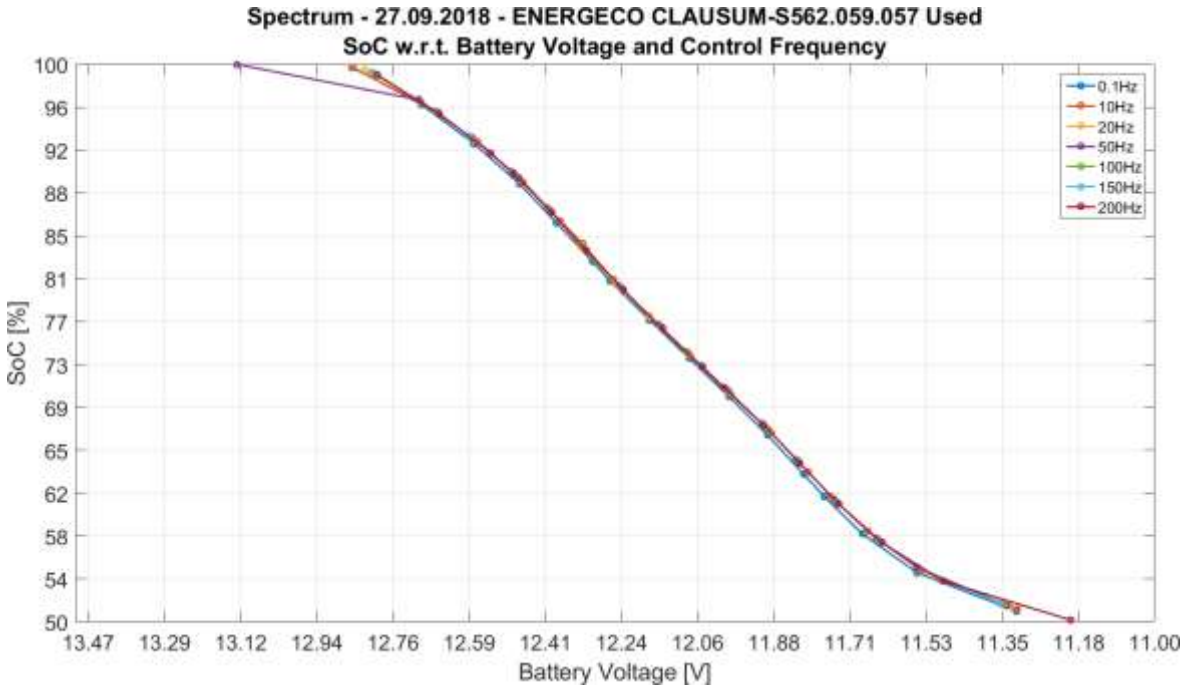


Figure 58. "Spectrum" Used Bosch SoC w.r.t. Battery Voltage and Control Frequency graphs

5.3.2.3 Fiamm L150P

5.3.2.3.1 New condition

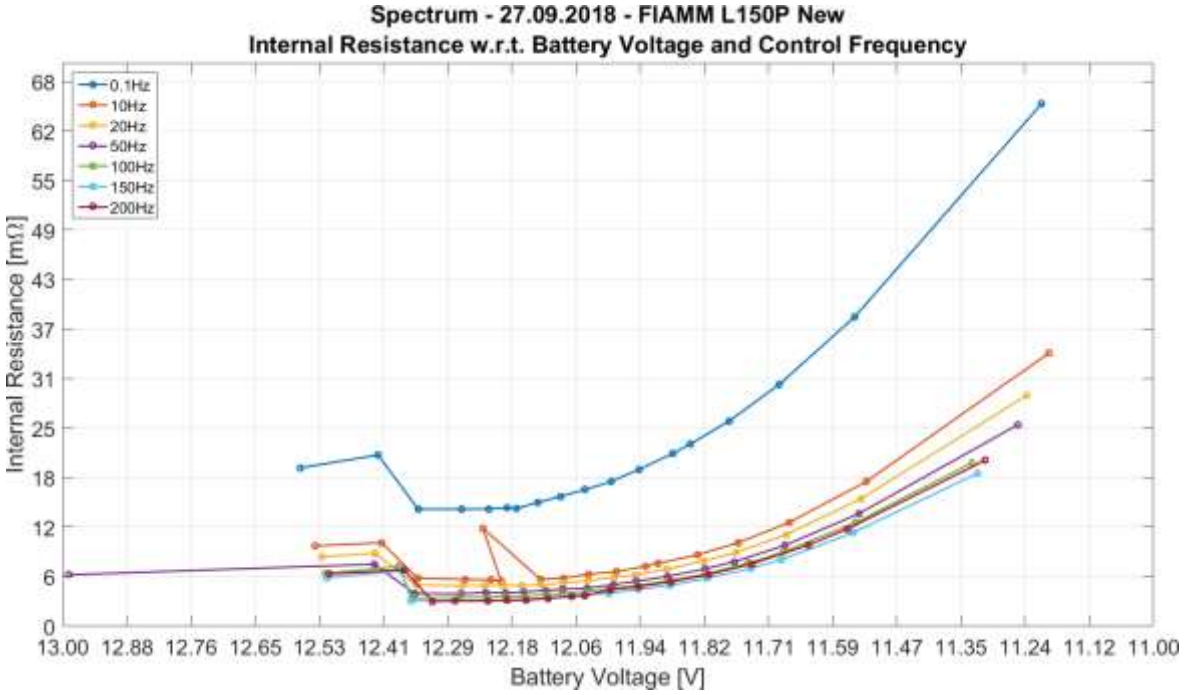


Figure 59. "Spectrum" New Fiamm Internal Resistance w.r.t. Battery Voltage and Control Frequency graphs

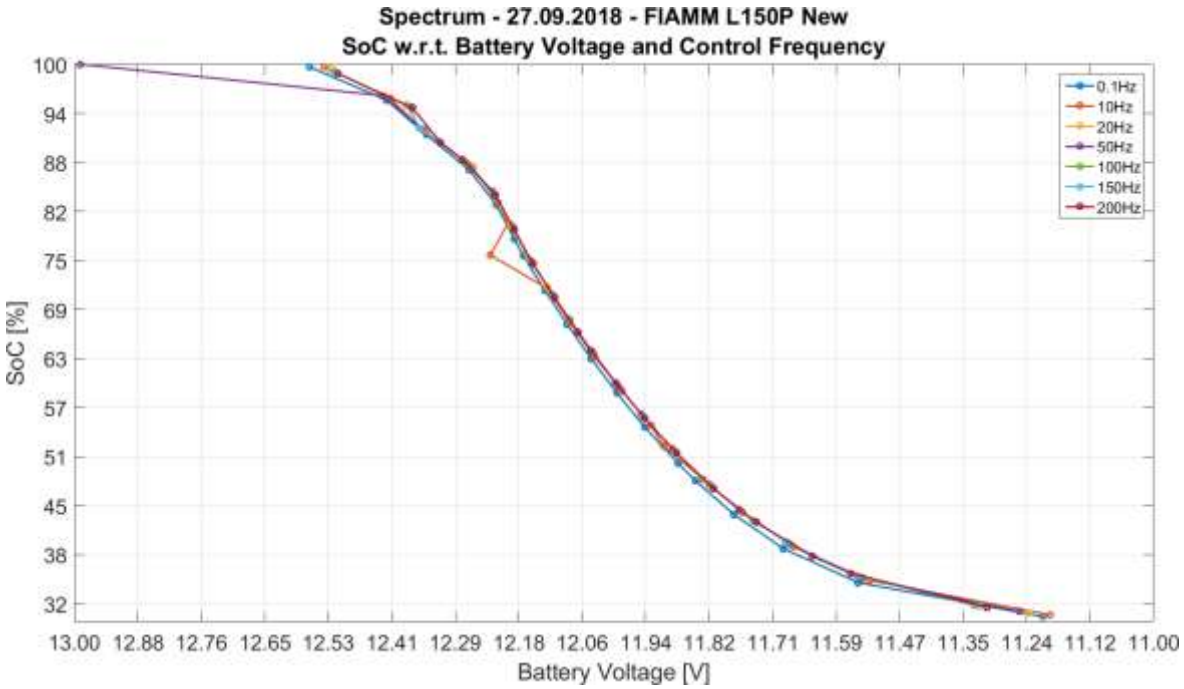


Figure 60. "Spectrum" New Fiamm SoC w.r.t. Battery Voltage and Control Frequency graphs

5.3.2.3.2 Used condition

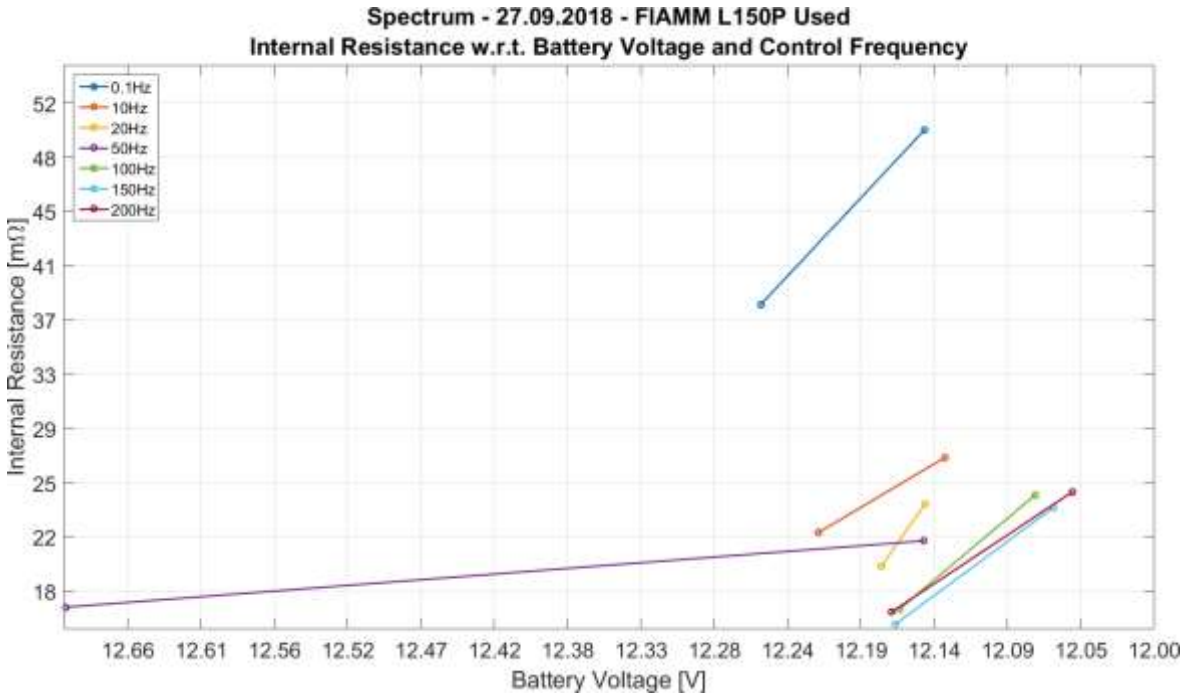


Figure 61. "Spectrum" Used Fiamm Internal Resistance w.r.t. Battery Voltage and Control Frequency graphs

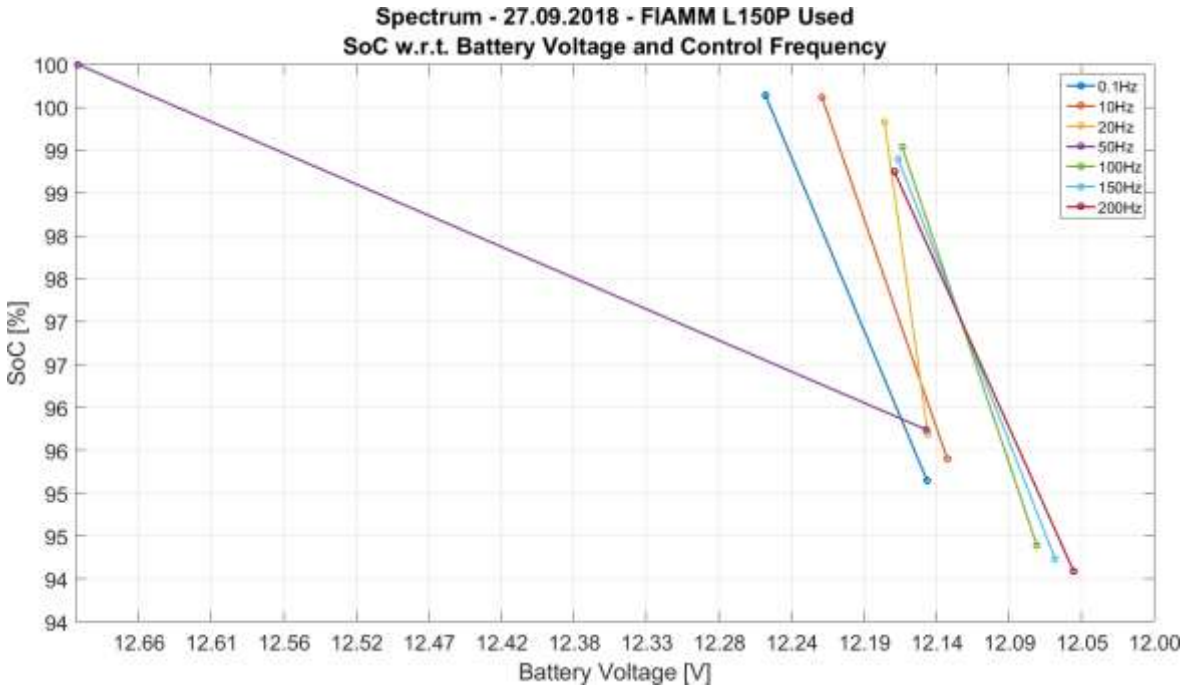


Figure 62. "Spectrum" Used Fiamm SoC w.r.t. Battery Voltage and Control Frequency graphs

5.4 "SoC" campaign

The experiment was performed 10 times on each battery, gaining a total of 90 test cycles for each SoC level, at least for those the battery has managed to reach.

5.4.1 Concept art

5.4.1.1 Plot of an acquisition cycle

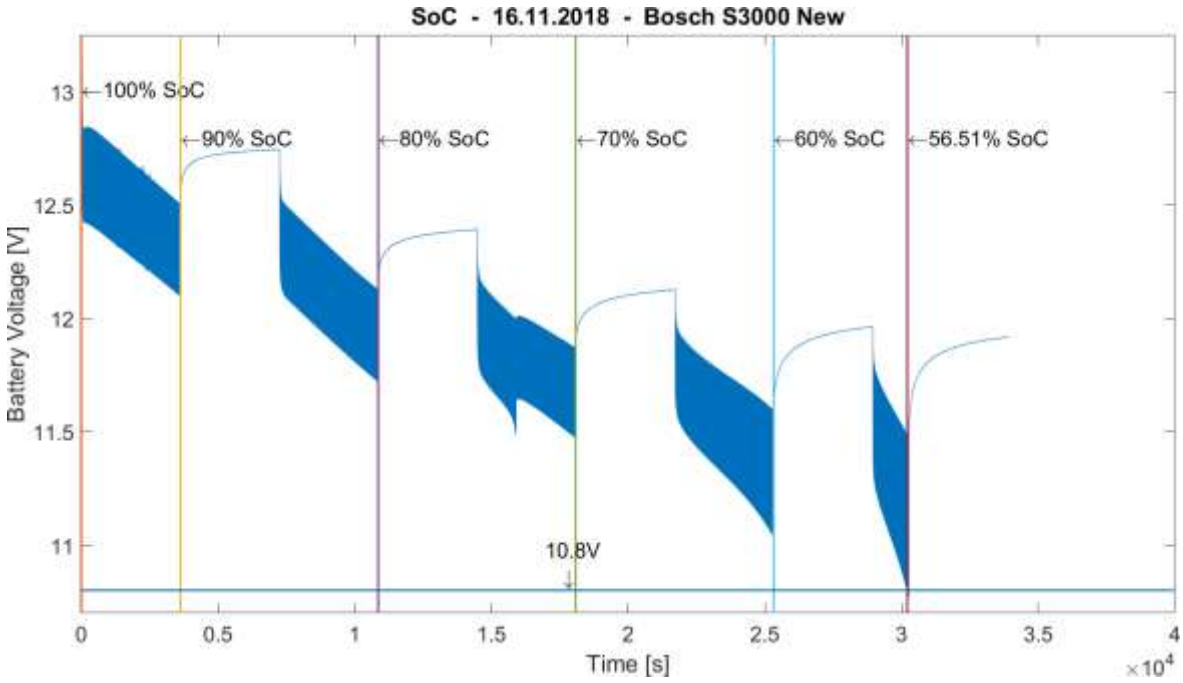


Figure 63. Plot of an acquisition cycle of "SoC" data

5.4.1.2 Details of the plot of an acquisition cycle

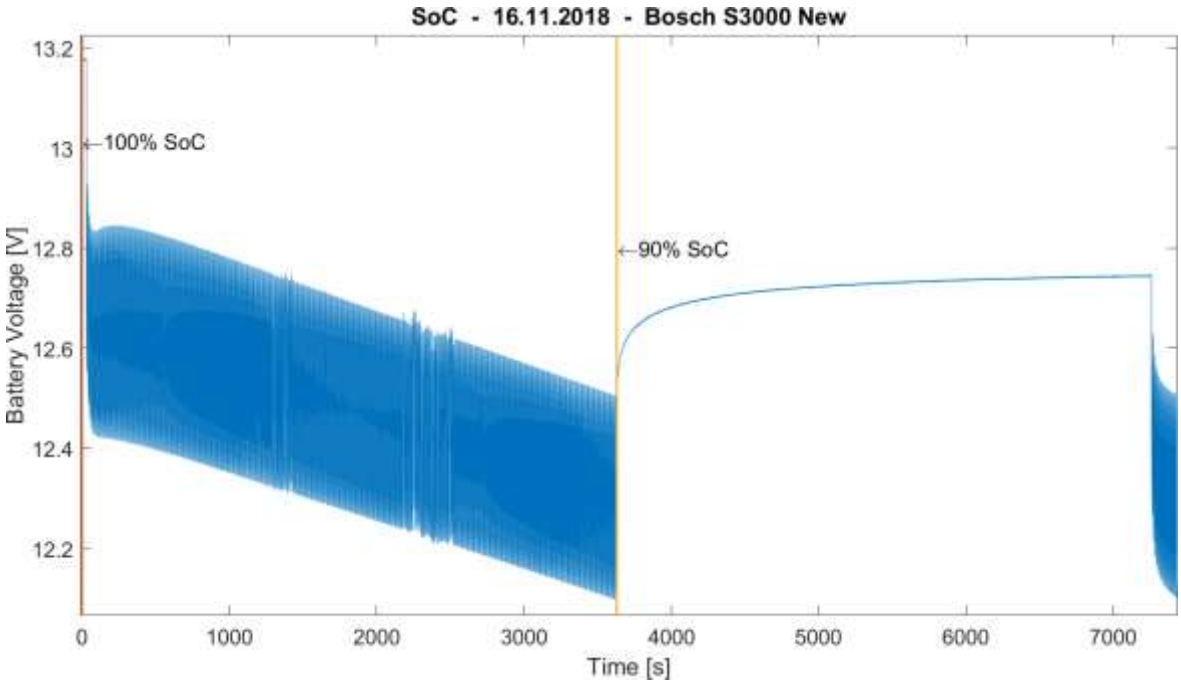


Figure 64. Detail of the plot of an acquisition of "SoC" data

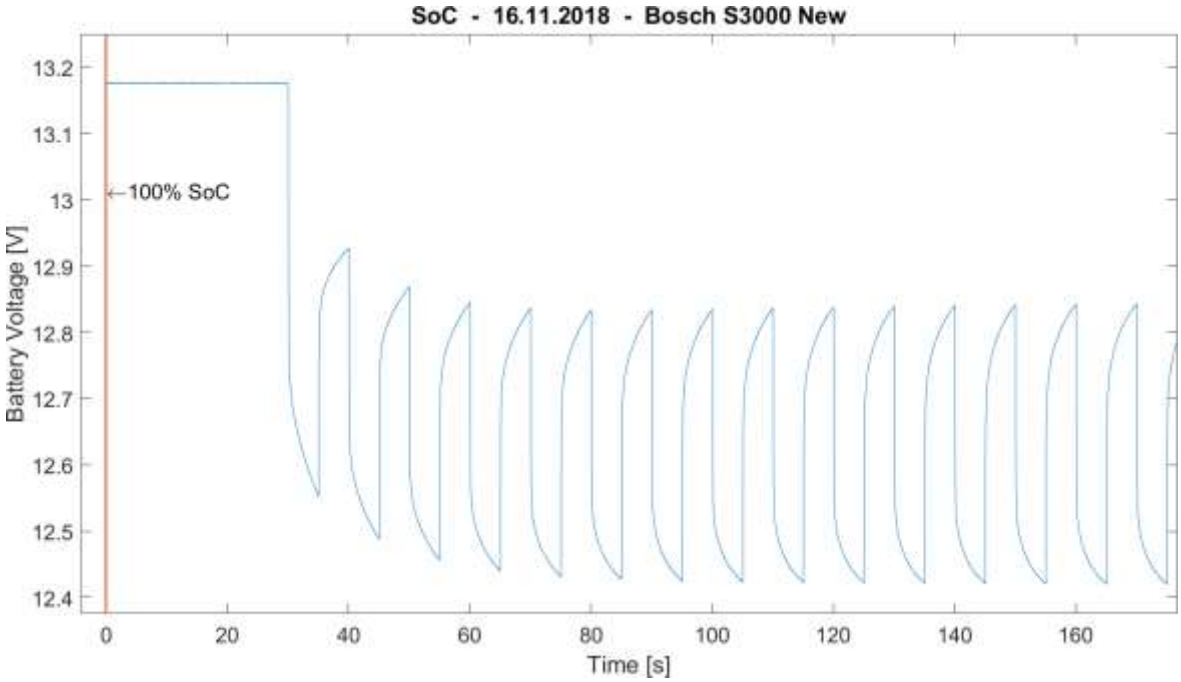


Figure 65. Zoom of the detail of the plot of an acquisition of "SoC" data

5.4.2 Battery analysis

5.4.2.1 Bosch S3000

5.4.2.1.1 New condition

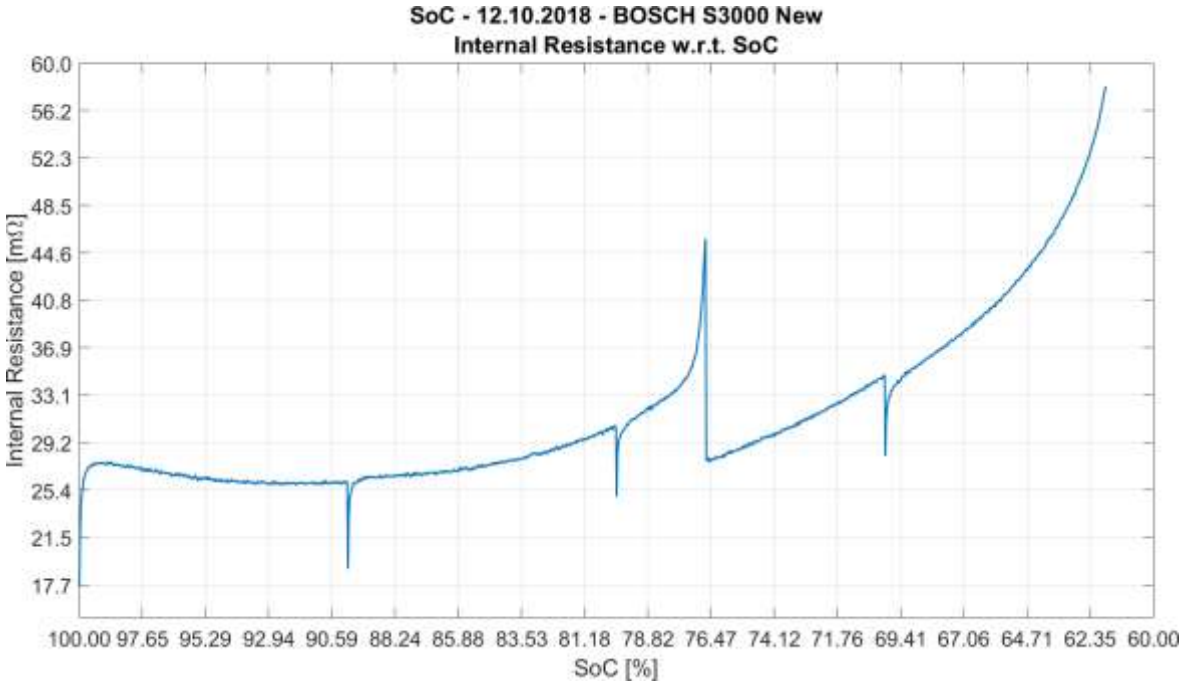


Figure 66. "SoC" New Bosch Internal Resistance w.r.t. SoC graph

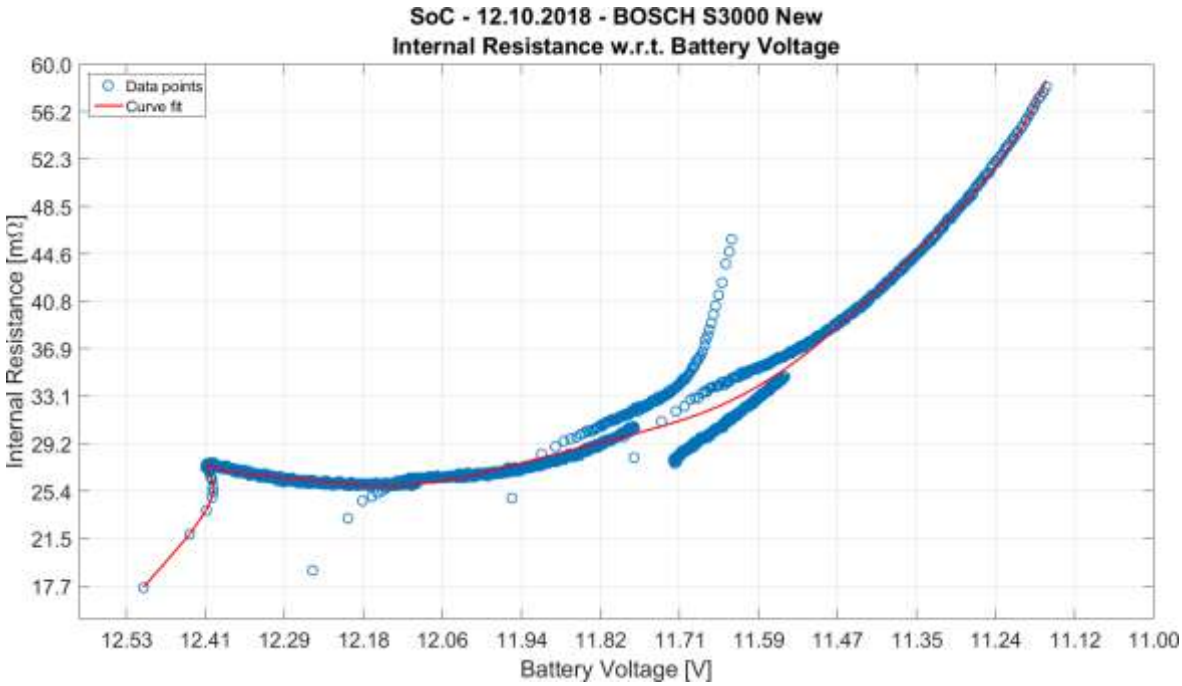


Figure 67. "SoC" New Bosch Internal Resistance w.r.t. Battery Voltage graphs

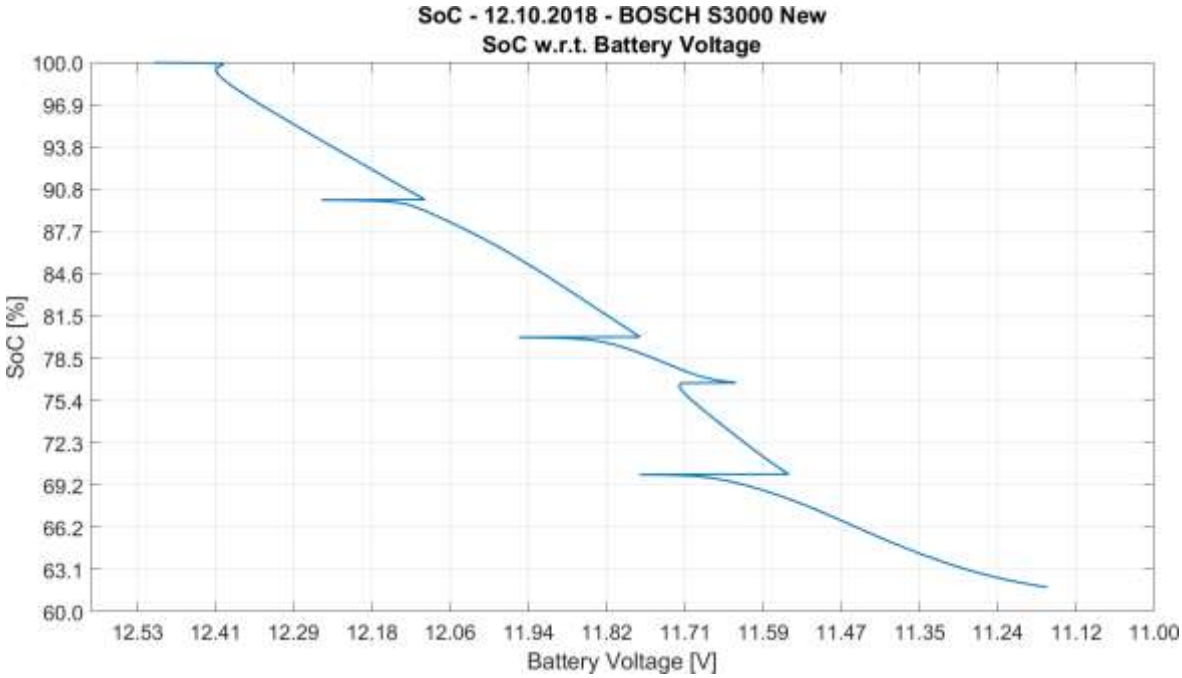


Figure 68. "SoC" New Bosch SoC w.r.t. Battery Voltage graph

5.4.2.1.2 Used condition

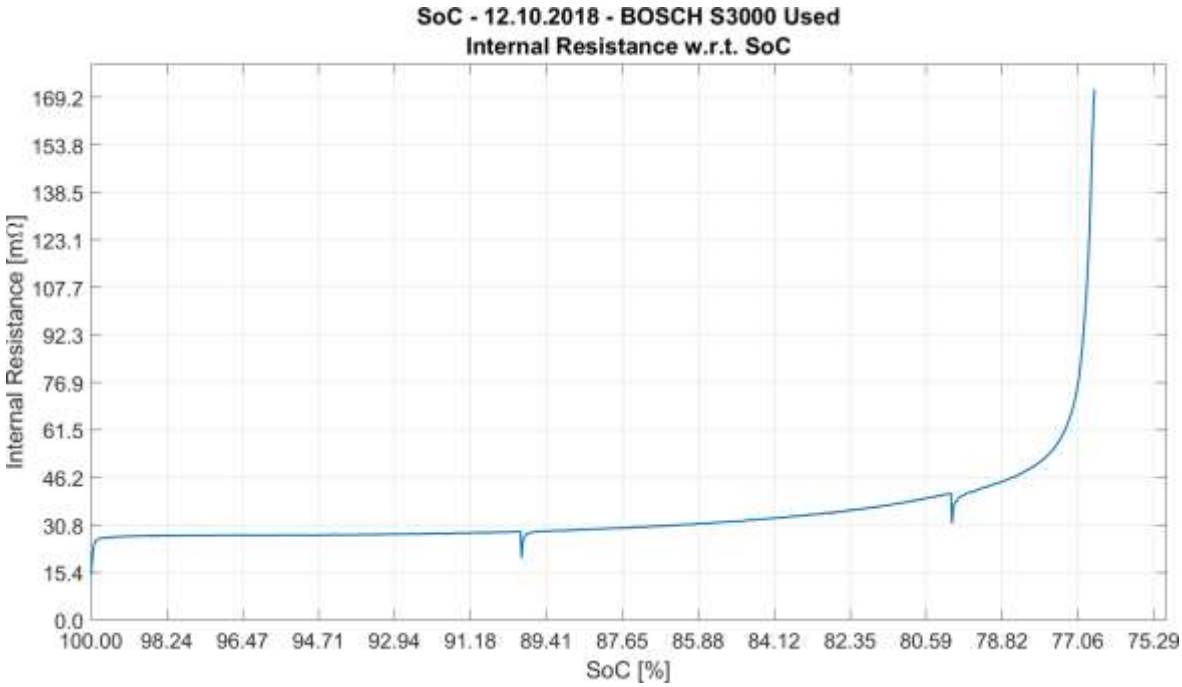


Figure 69. "SoC" Used Bosch Internal Resistance w.r.t. SoC graph

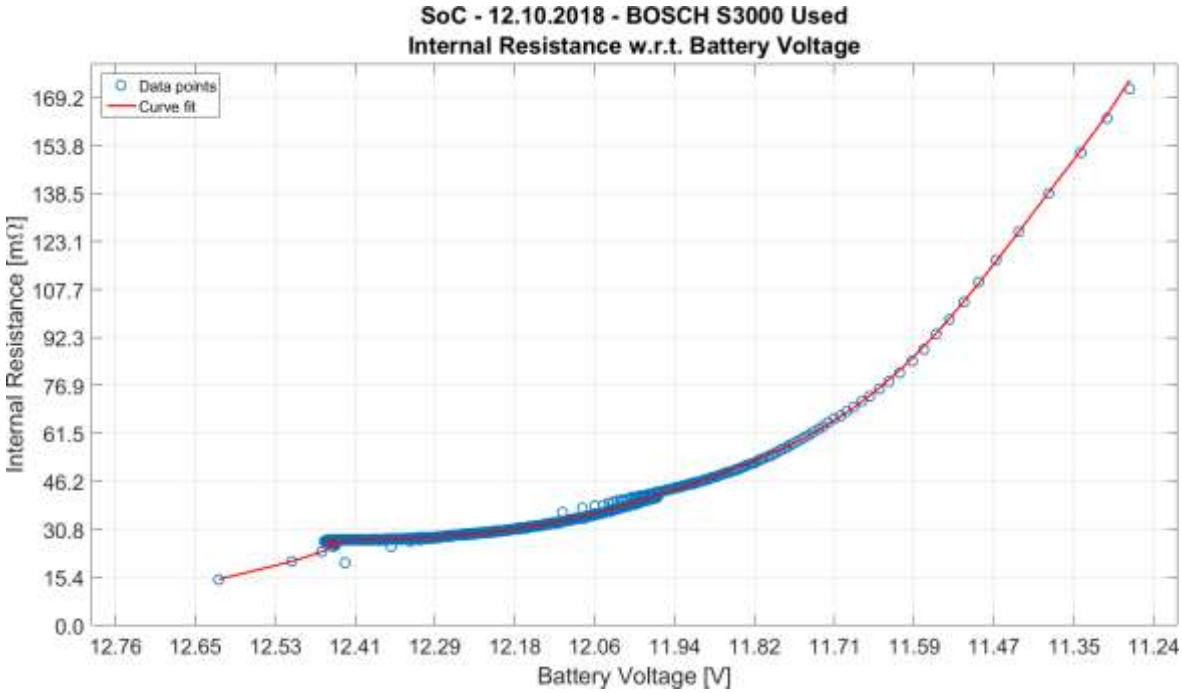


Figure 70. "SoC" Used Bosch Internal Resistance w.r.t. Battery Voltage graphs

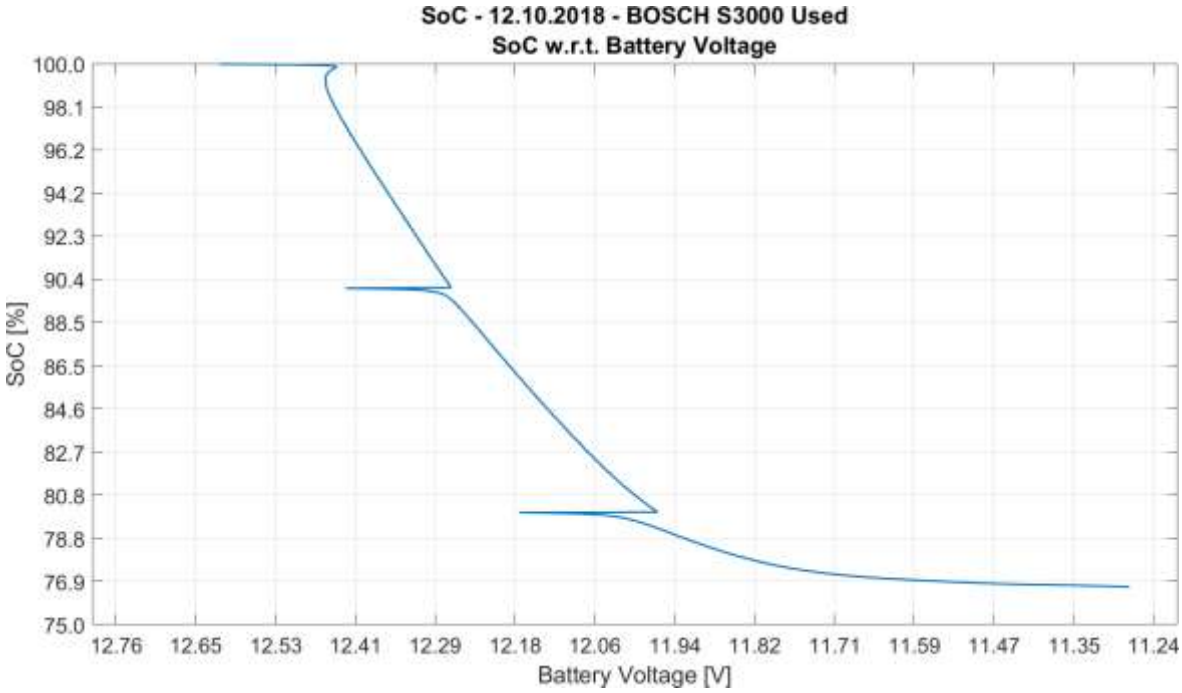


Figure 71. "SoC" Used Bosch SoC w.r.t. Battery Voltage graph

5.4.2.2 Energeco Clausum-S562.059.057

5.4.2.2.1 New condition

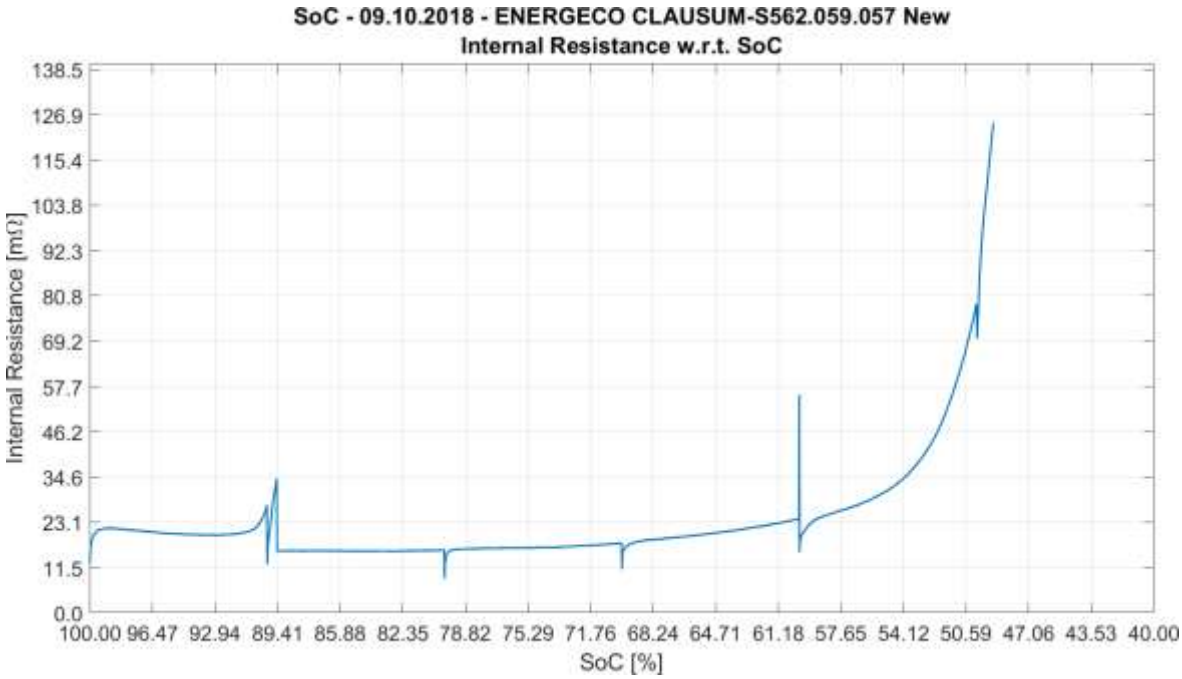


Figure 72. "SoC" New Energeco Internal Resistance w.r.t. SoC graph

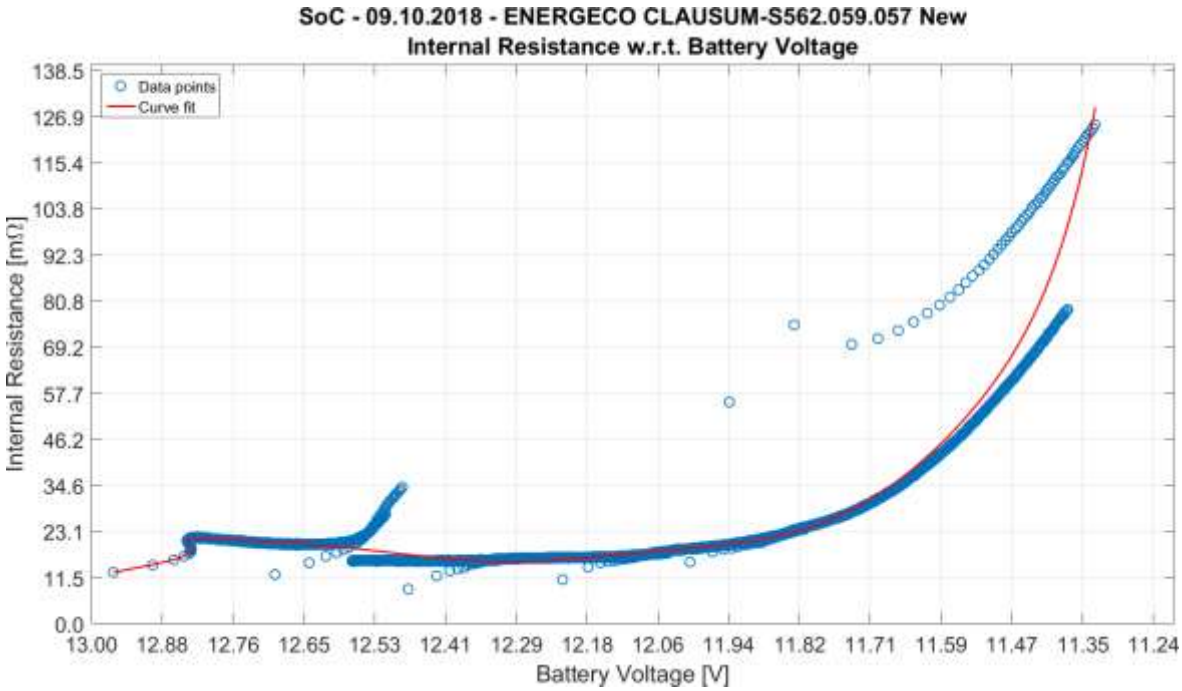


Figure 73. "SoC" New Energeco Internal Resistance w.r.t. Battery Voltage graphs

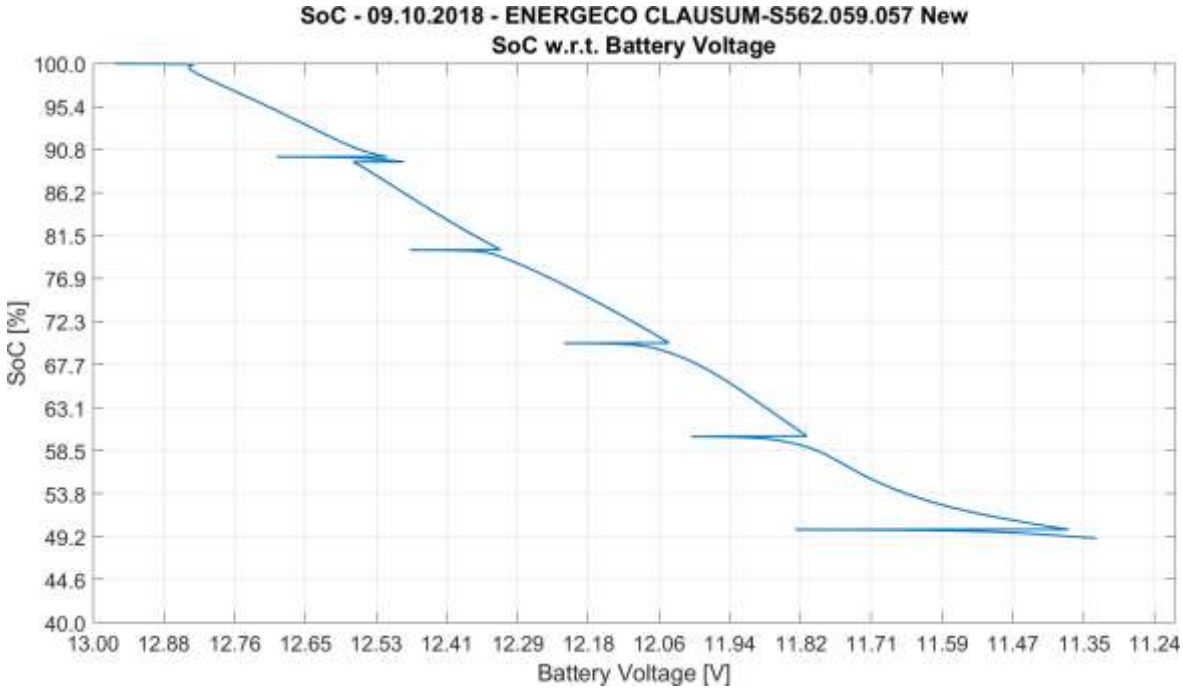


Figure 74. "SoC" New Energeco SoC w.r.t. Battery Voltage graph

5.4.2.2.2 Used condition

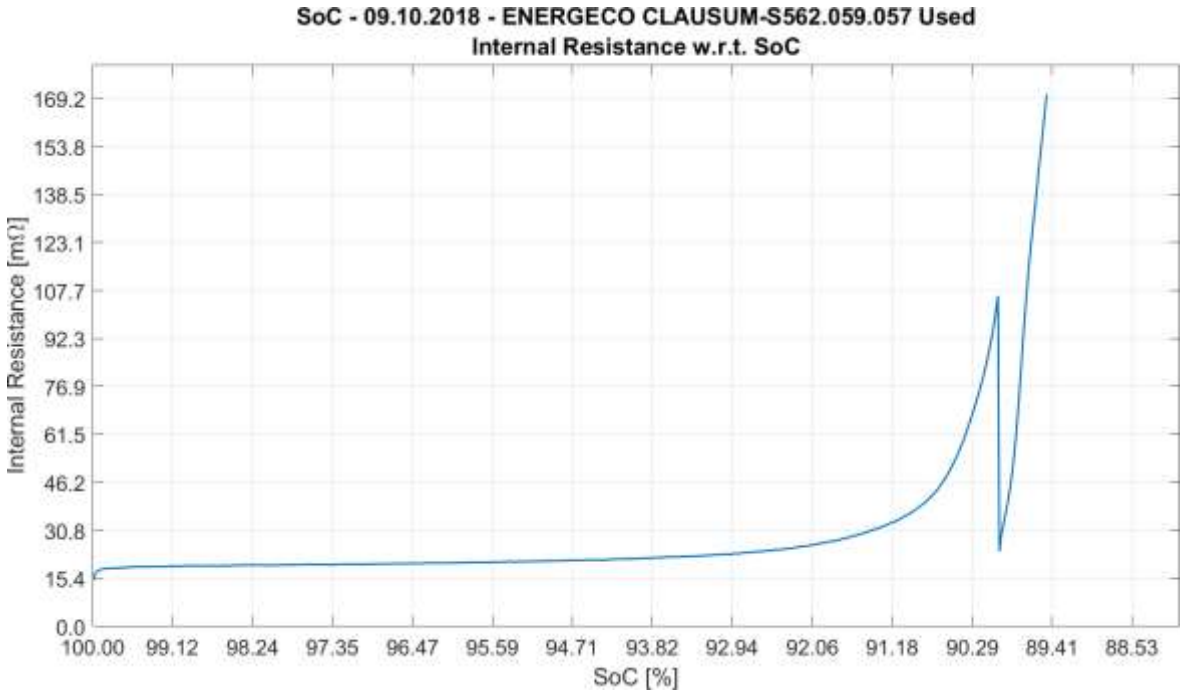


Figure 75. "SoC" Used Energeco Internal Resistance w.r.t. SoC graph

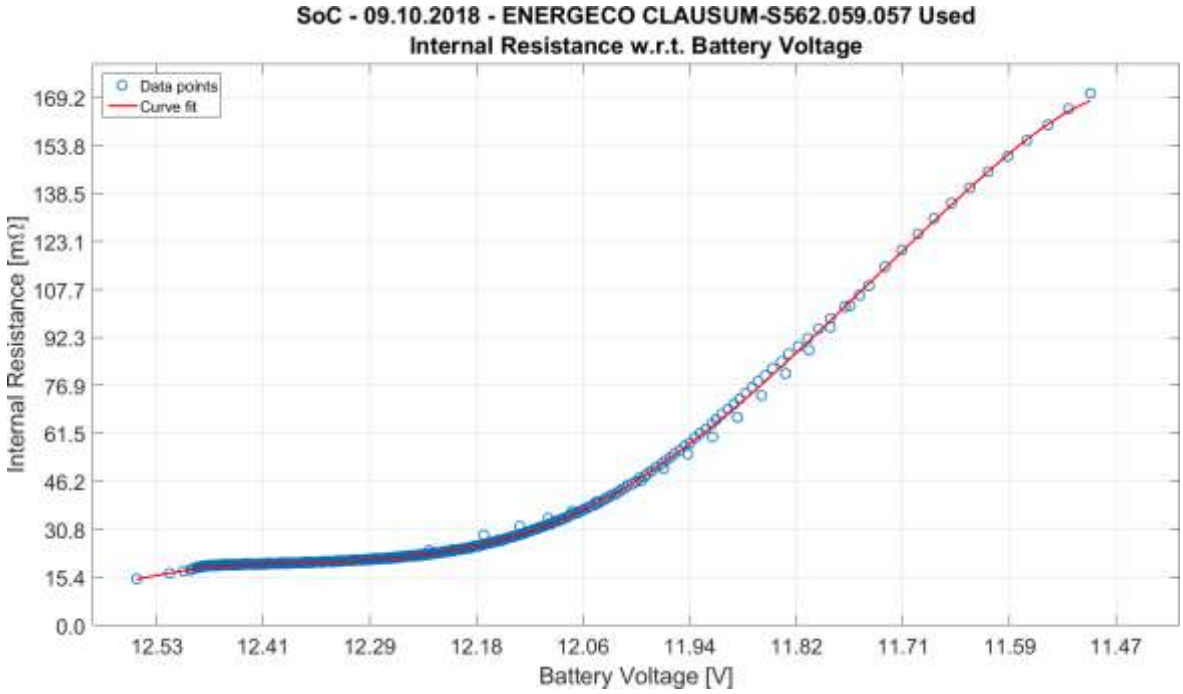


Figure 76. "SoC" Used Energeco Internal Resistance w.r.t. Battery Voltage graphs

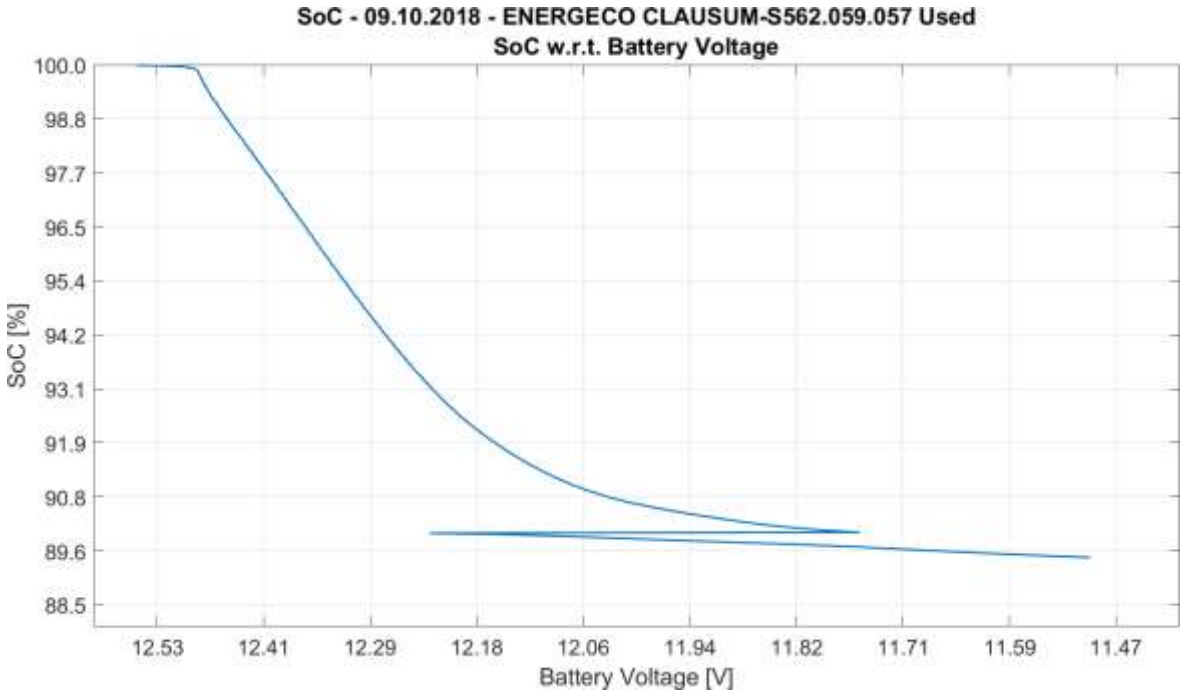


Figure 77. "SoC" Used Energeco SoC w.r.t. Battery Voltage graph

5.4.2.3 Fiamm L150P

5.4.2.3.1 New condition

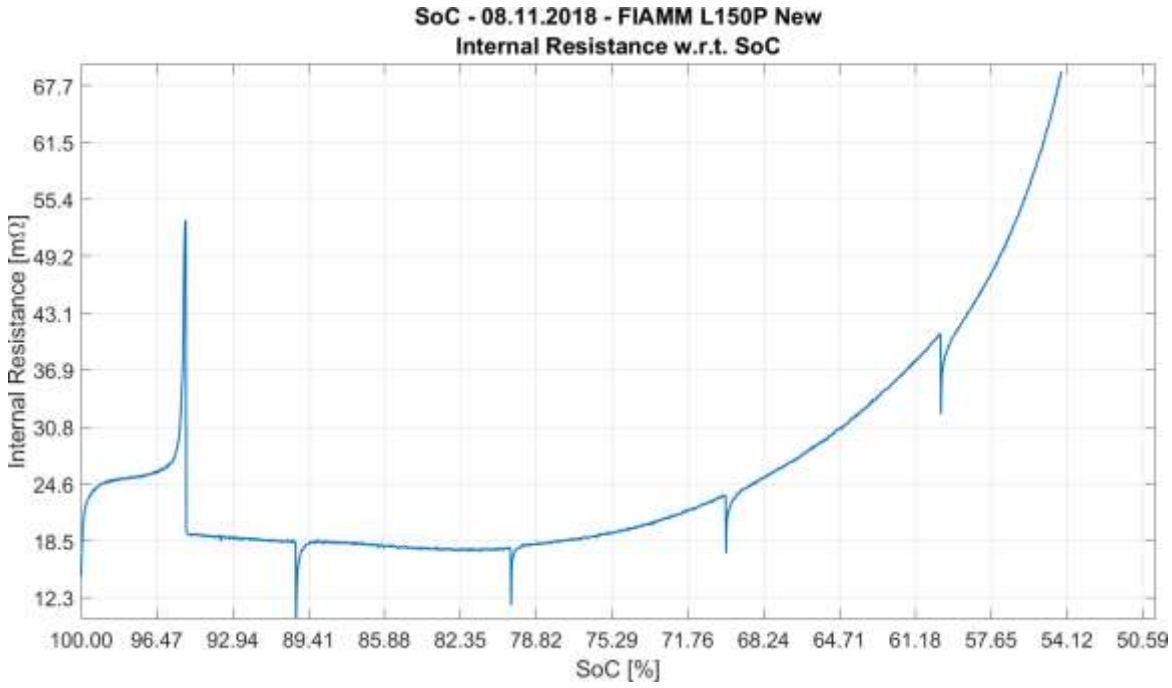


Figure 78. "SoC" New Fiamm Internal Resistance w.r.t. SoC graph

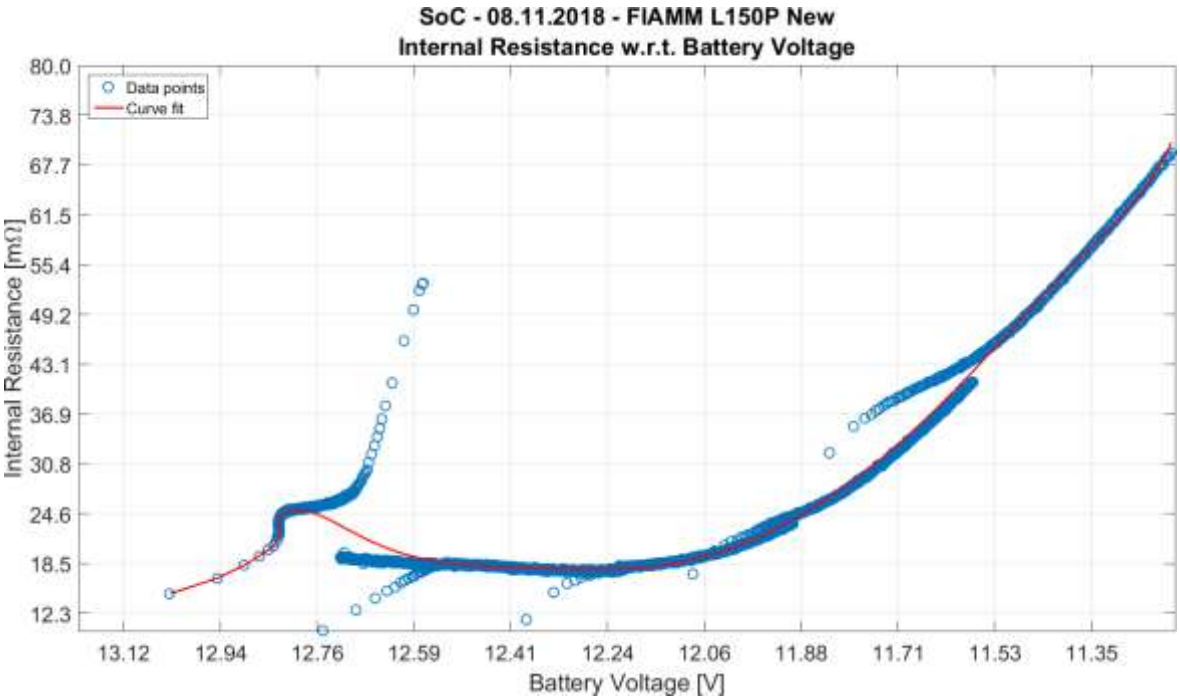


Figure 79. "SoC" New Fiamm Internal Resistance w.r.t. Battery Voltage graphs

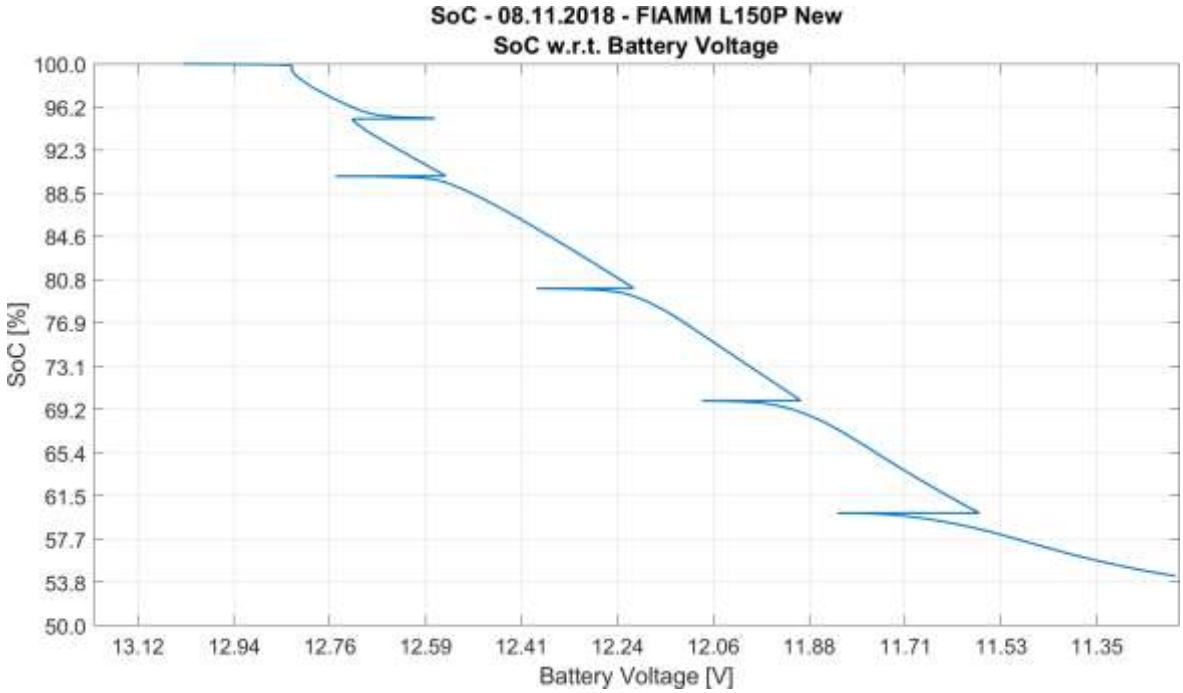


Figure 80. "SoC" New Fiamm SoC w.r.t. Battery Voltage graph

5.4.2.3.2 Used condition

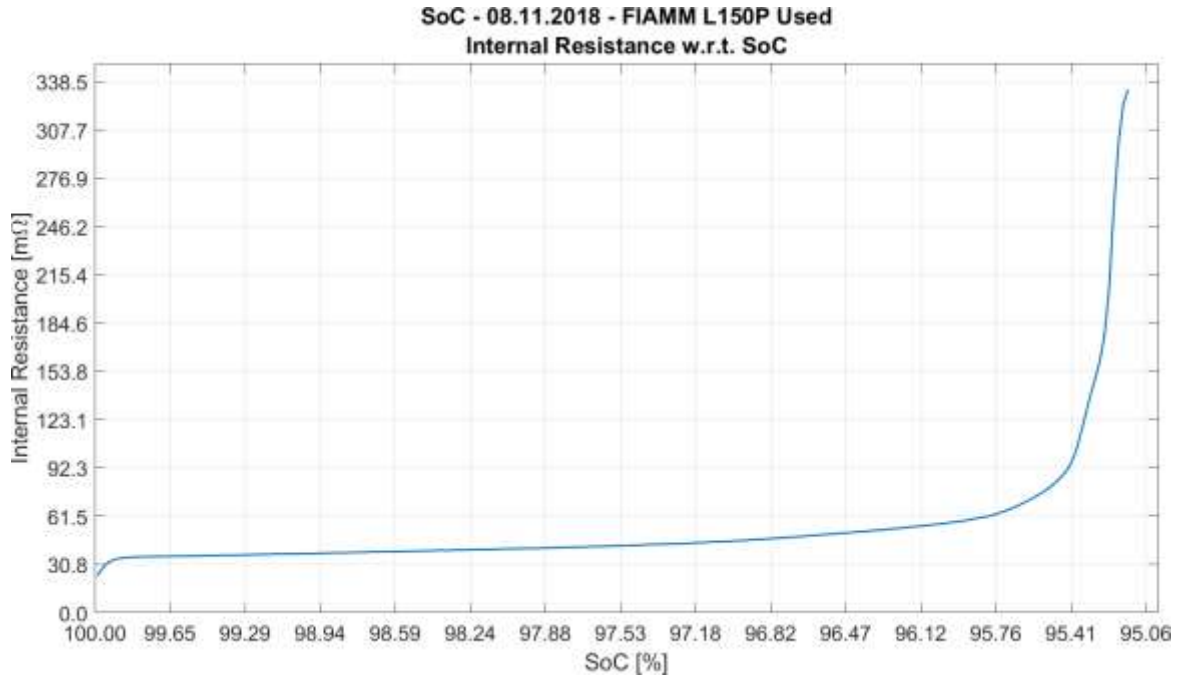


Figure 81. "SoC" Used Fiamm Internal Resistance w.r.t. SoC graph

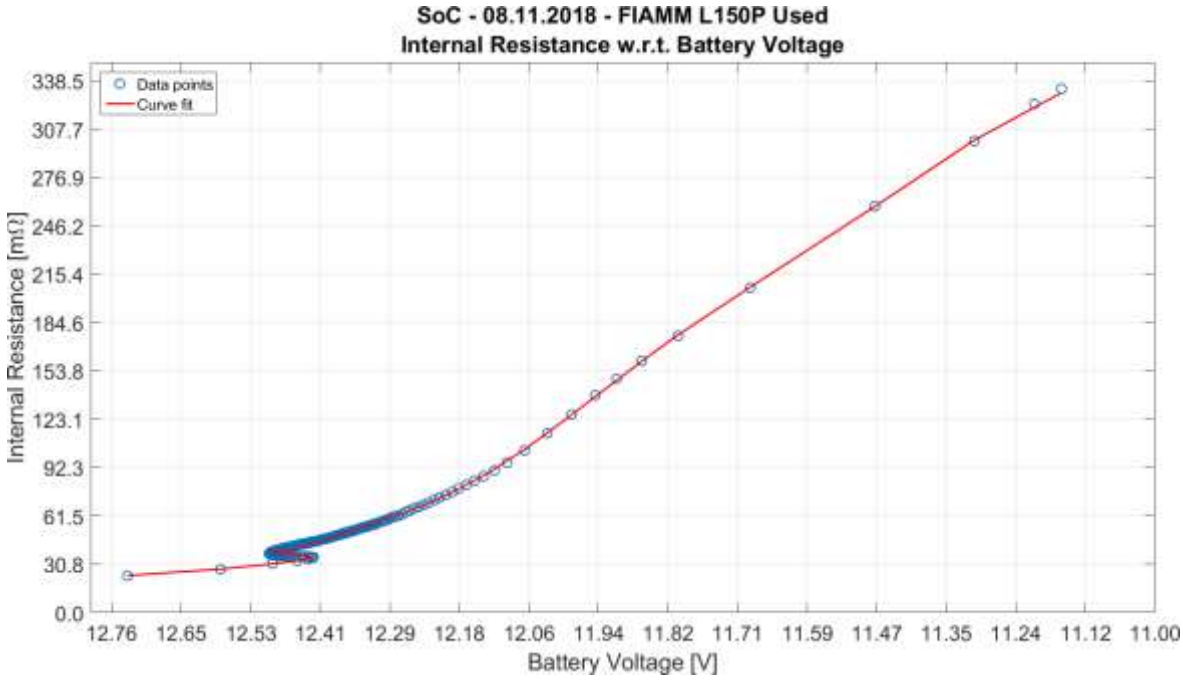


Figure 82. "SoC" Used Fiamm Internal Resistance w.r.t. Battery Voltage graphs

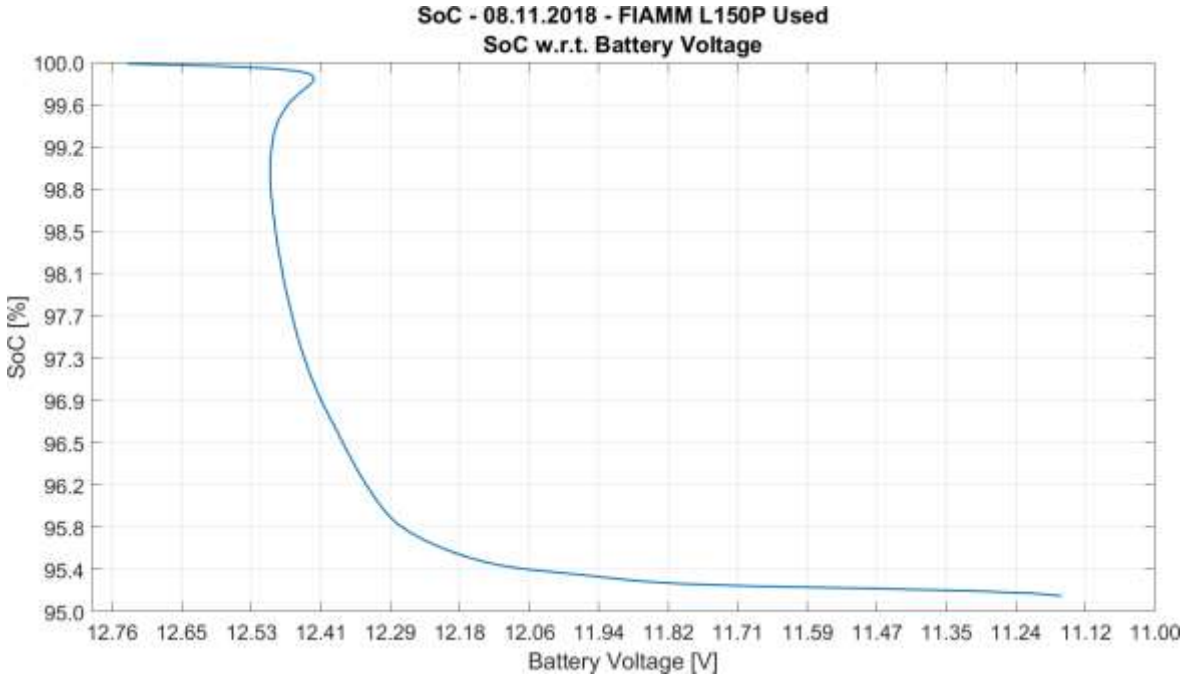


Figure 83. "SoC" Used Fiamm SoC w.r.t. Battery Voltage graph

5.5 "SoCSafe" campaign

The experiment was performed 10 times on each battery, gaining a total of 90 test cycles for each SoC level, at least for those the battery has managed to reach.

5.5.1 Concept art

5.5.1.1 Plot of an acquisition cycle

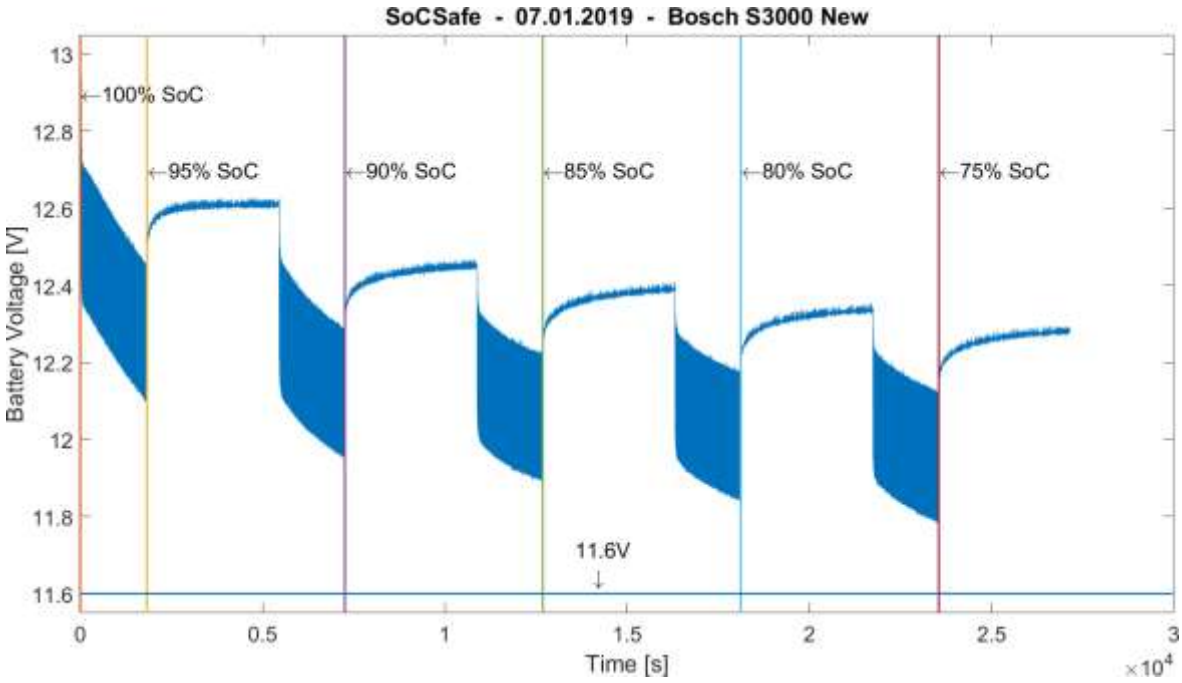


Figure 84. Plot of an acquisition cycle of "SoCSafe" data

5.5.1.2 Details of the plot of an acquisition cycle

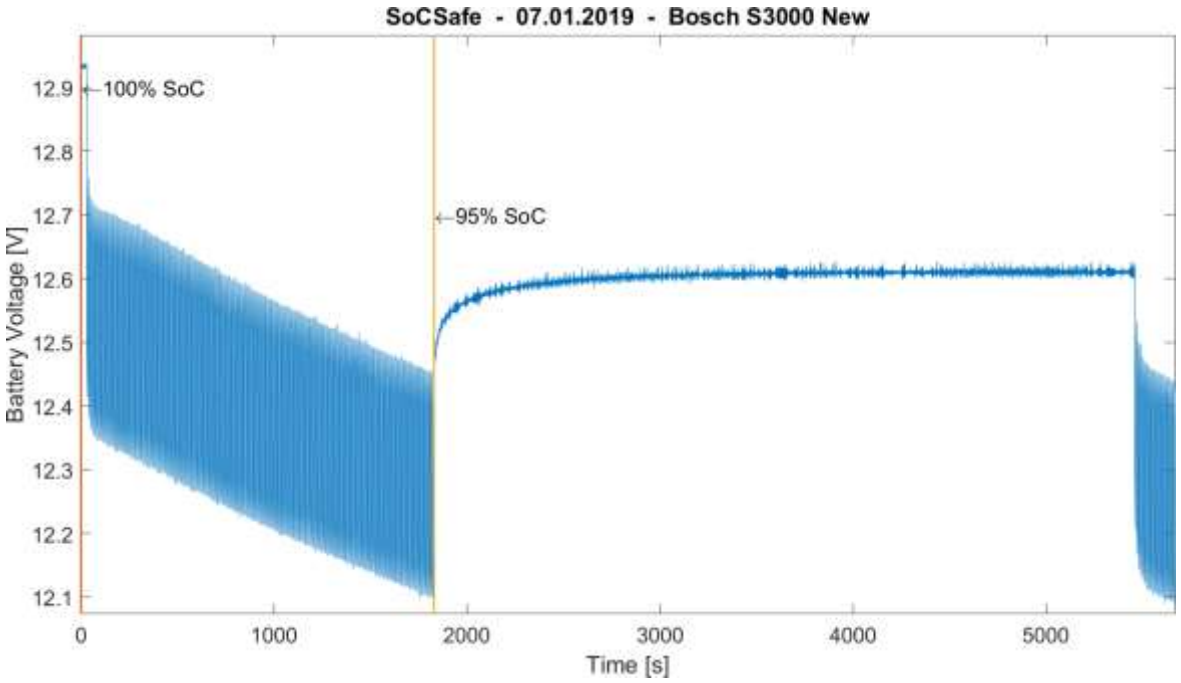


Figure 85. Detail of the plot of an acquisition of "SoCSafe" data

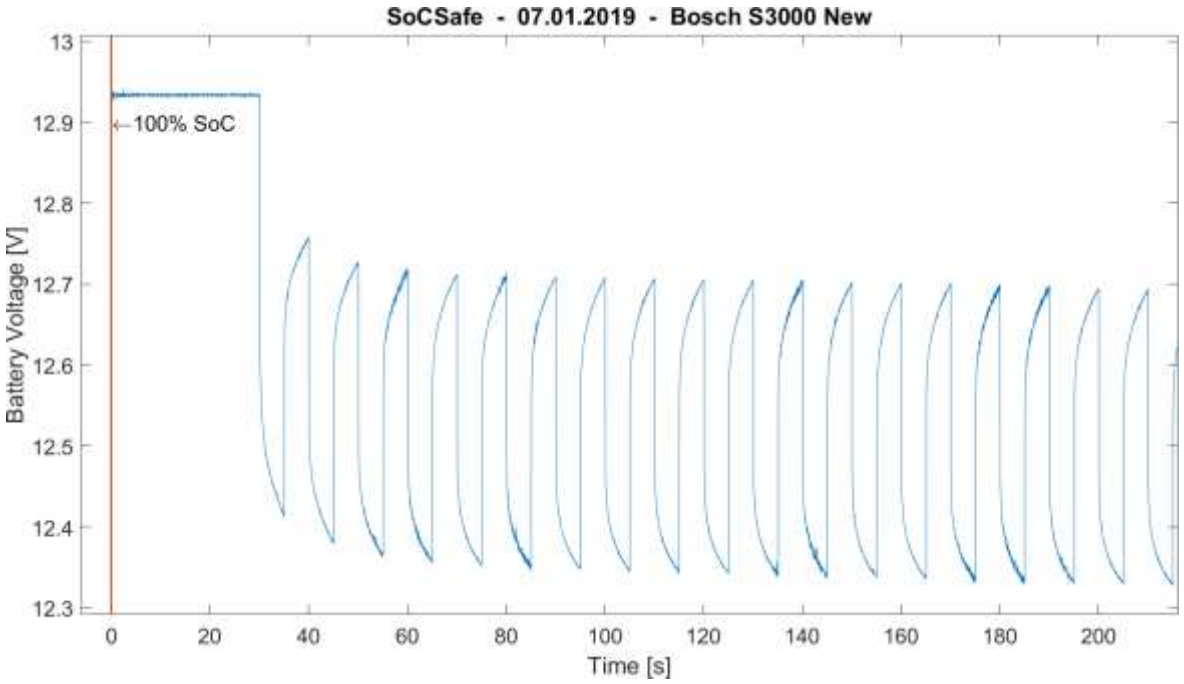


Figure 86. Zoom of the detail of the plot of an acquisition of "SoCSafe" data

5.5.2 Battery analysis

5.5.2.1 Bosch S3000

5.5.2.1.1 Pristine condition

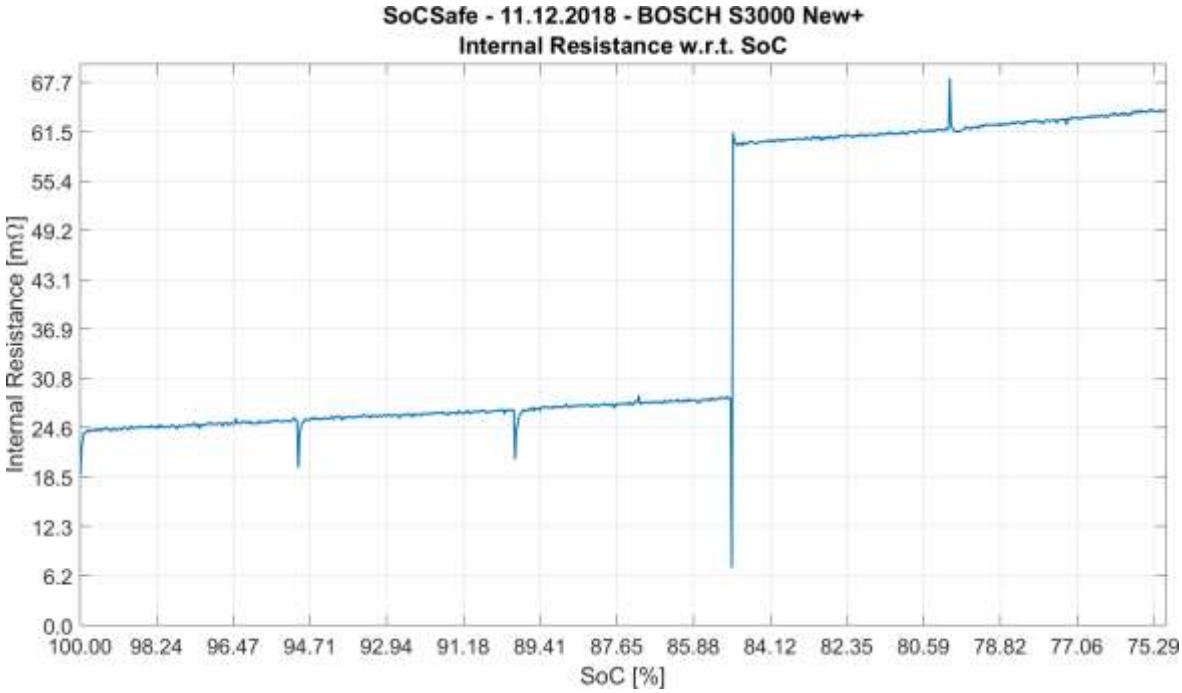


Figure 87. "SoCSafe" Pristine Bosch Internal Resistance w.r.t. SoC graph

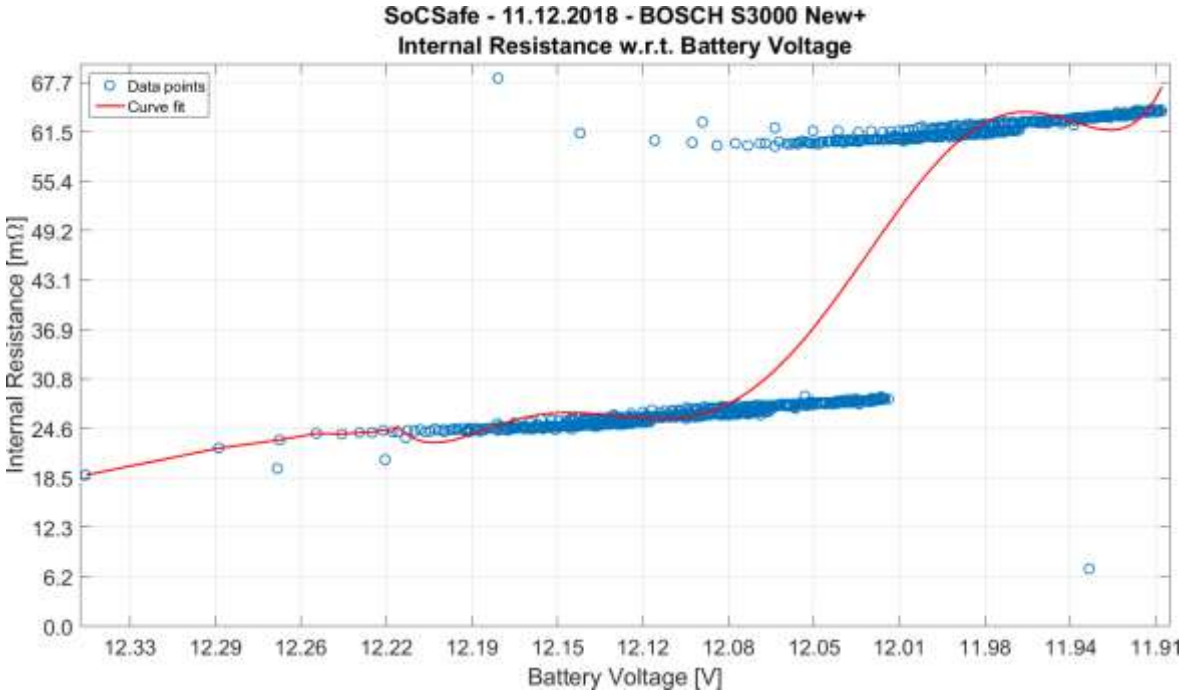


Figure 88. "SoCSafe" Pristine Bosch Internal Resistance w.r.t. Battery Voltage graphs

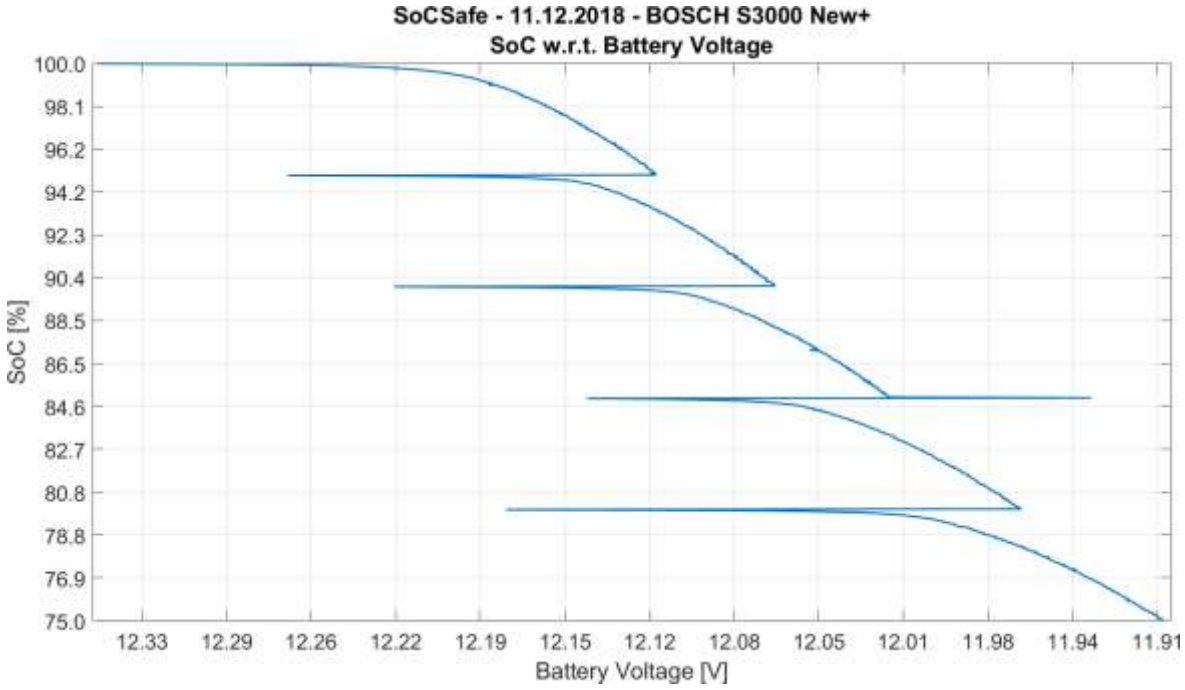


Figure 89. "SoCSafe" Pristine Bosch SoC w.r.t. Battery Voltage graph

5.5.2.2 Energeco Clausum-S562.059.057

5.5.2.2.1 Pristine condition

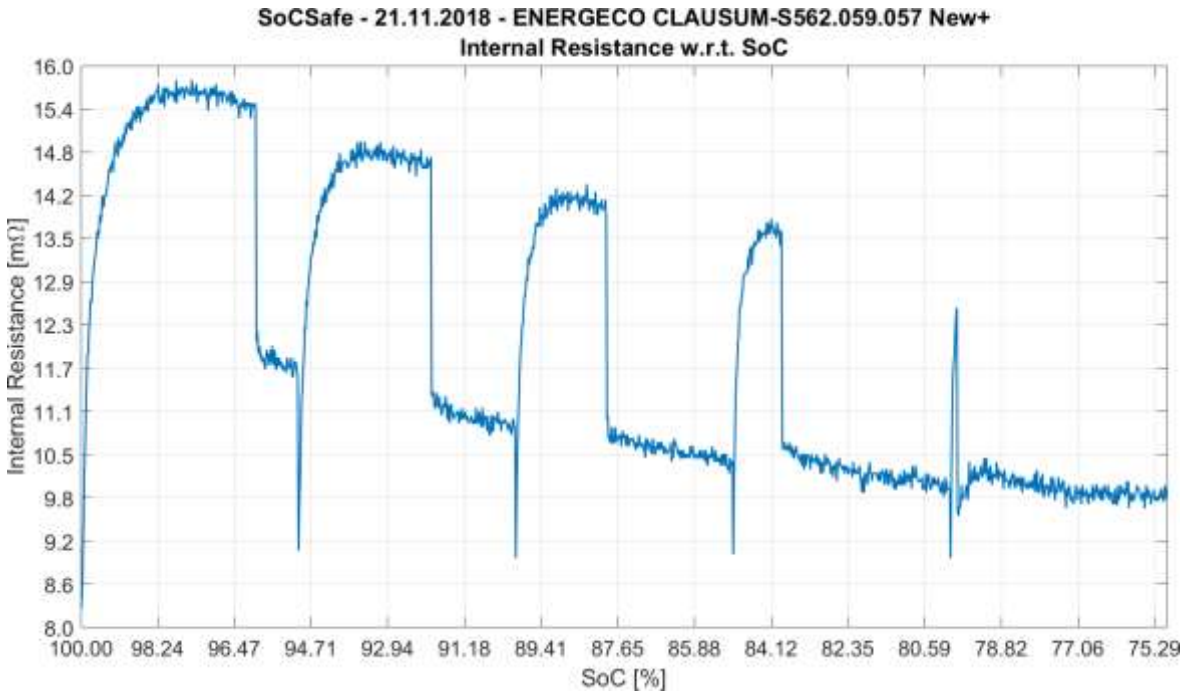


Figure 90. "SoCSafe" Pristine Energeco Internal Resistance w.r.t. SoC graph

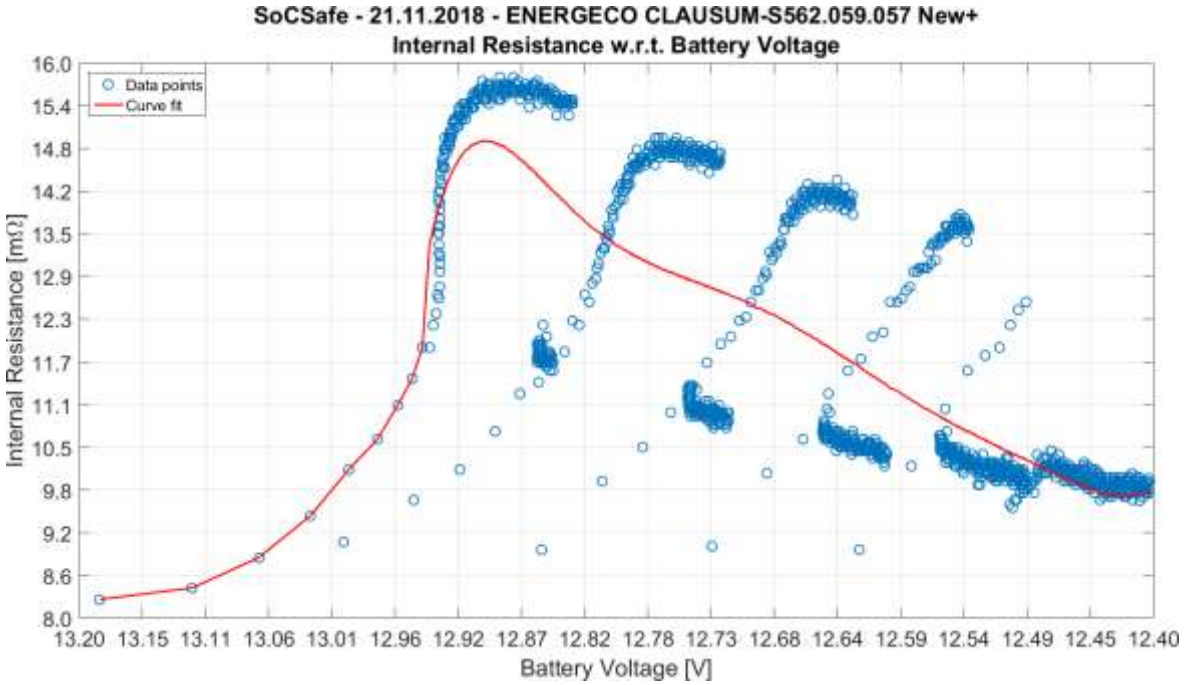


Figure 91. "SoCSafe" Pristine Energeco Internal Resistance w.r.t. Battery Voltage graphs

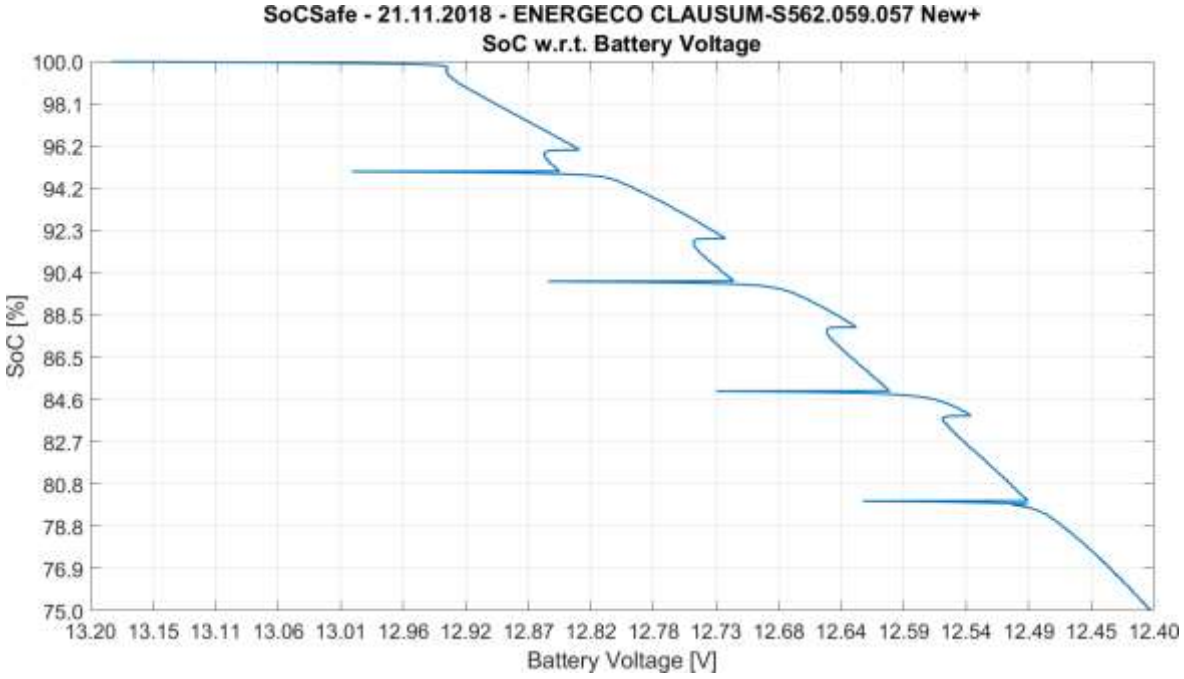


Figure 92. "SoCSafe" Pristine Energeco SoC w.r.t. Battery Voltage graph

5.5.2.3 Fiamm L150P

5.5.2.3.1 Pristine condition

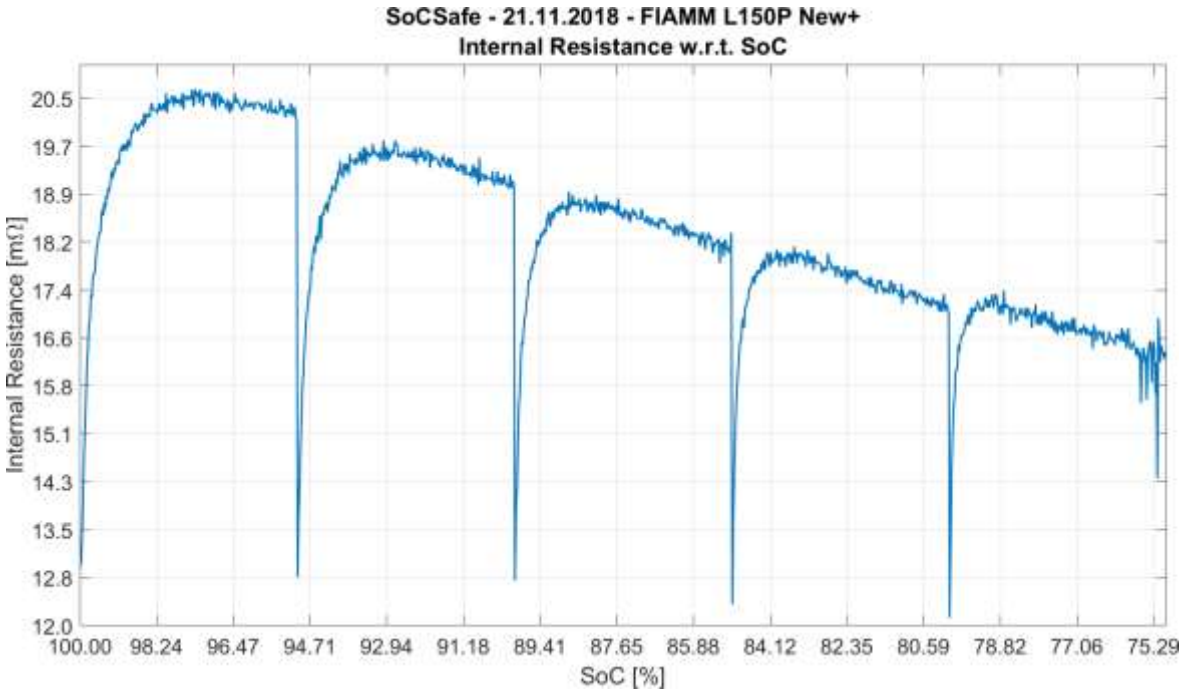


Figure 93. "SoCSafe" Pristine Fiamm Internal Resistance w.r.t. SoC graph

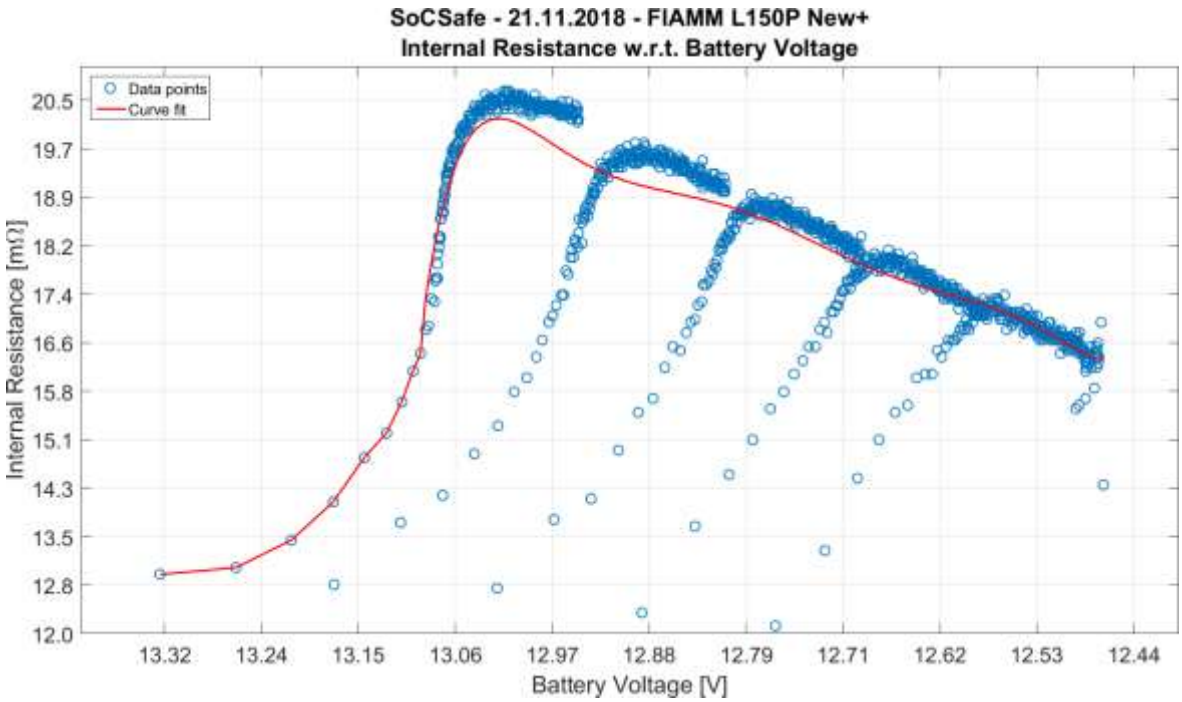


Figure 94. "SoCSafe" Pristine Fiamm Internal Resistance w.r.t. Battery Voltage graphs

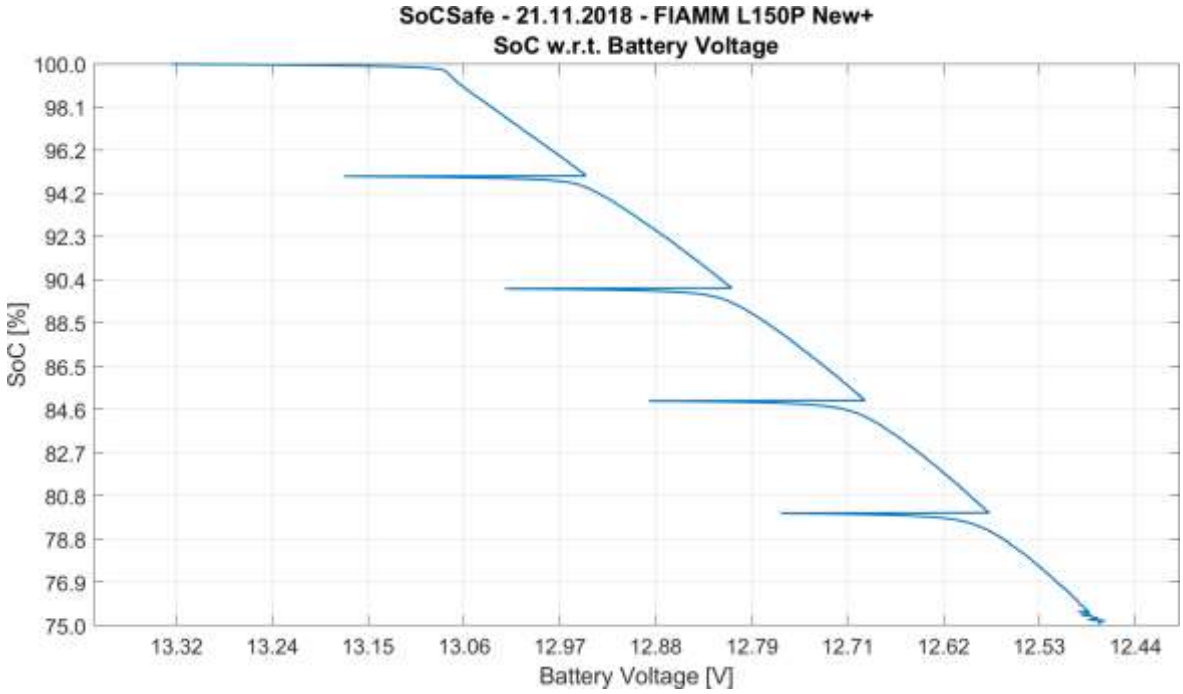


Figure 95. "SoCSafe" Pristine Fiamm SoC w.r.t. Battery Voltage graph

5.6 "SoCSafe" campaign in controlled temperature

The experiment was performed 10 times on each battery, gaining a total of 90 test cycles for each SoC level, at least for those the battery has managed to reach

5.6.1 Battery analysis

5.6.1.1 Bosch S3000 @ -10°C

5.6.1.1.1 New condition

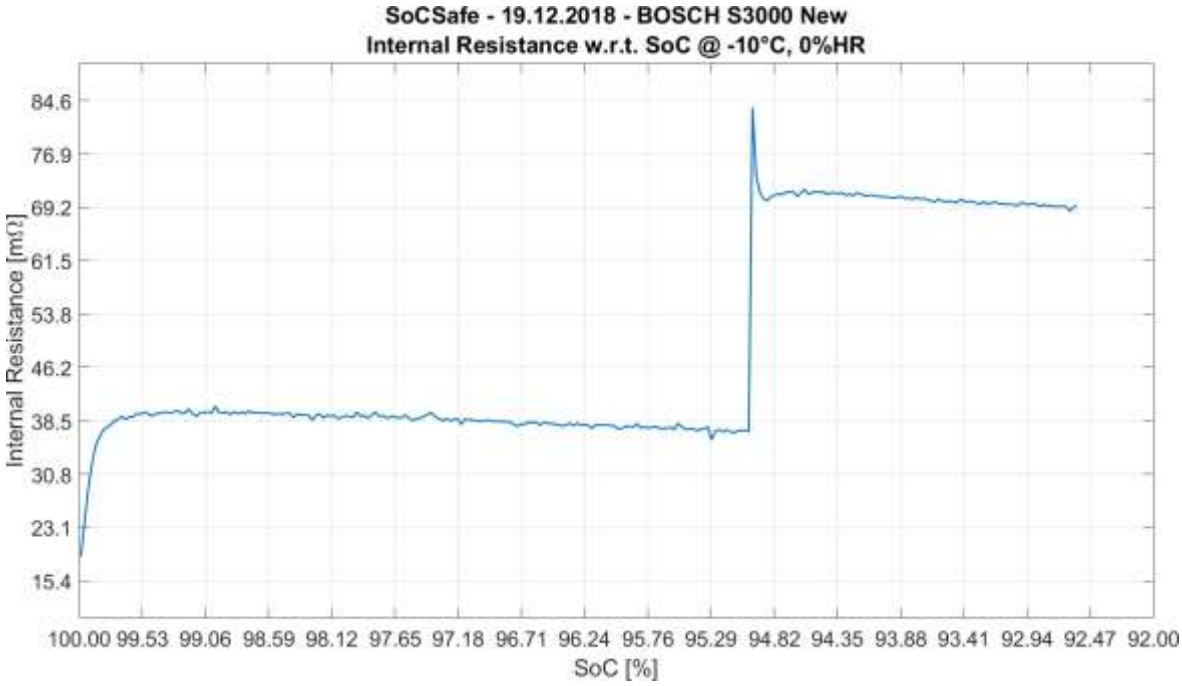


Figure 96. "SoCSafe" New Bosch Internal Resistance w.r.t. SoC at -10°C graph

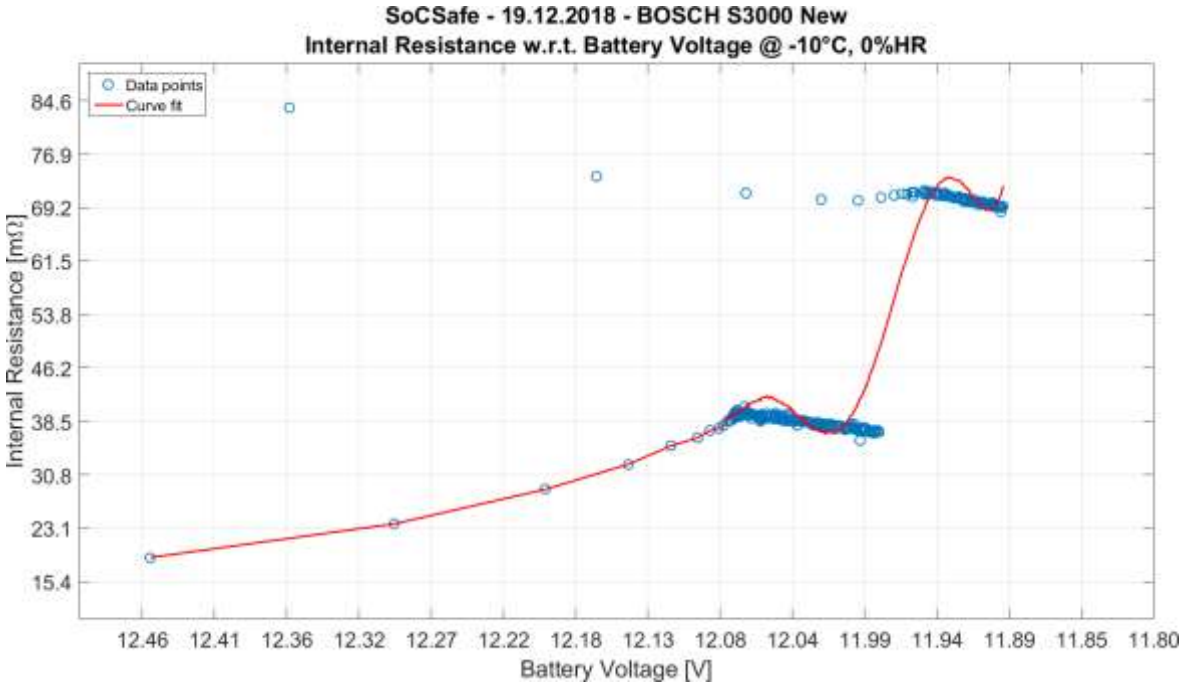


Figure 97. "SoCSafe" New Bosch Internal Resistance w.r.t. Battery Voltage at -10°C graphs

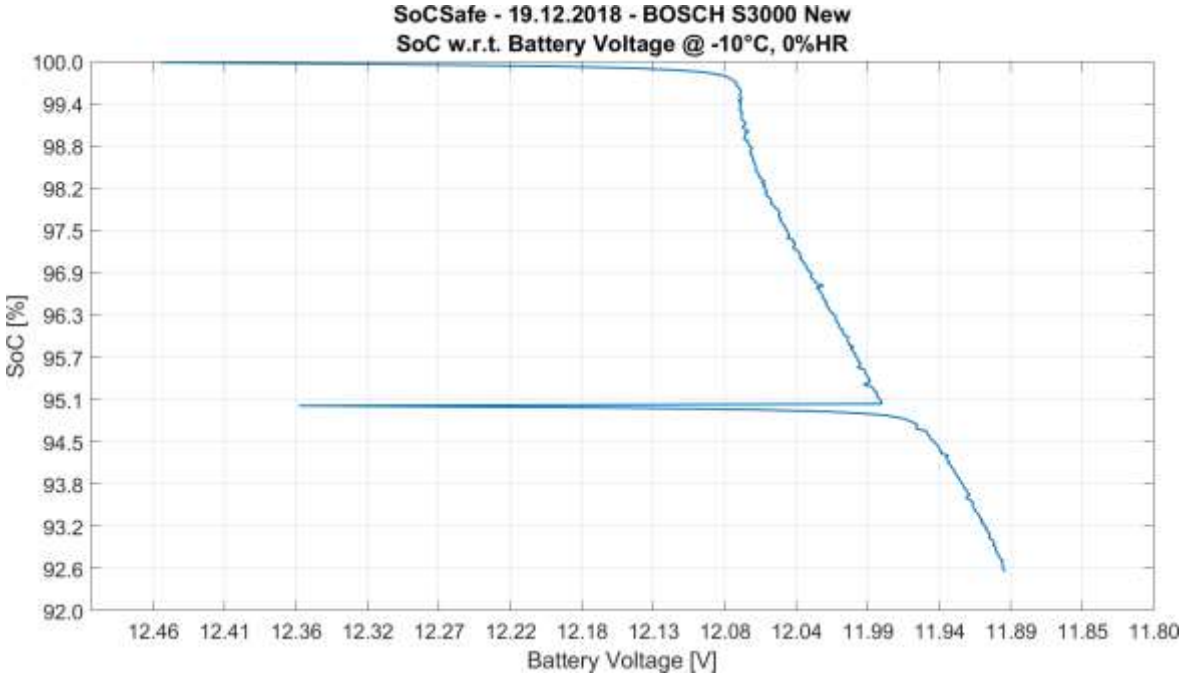


Figure 98. "SoCSafe" New Bosch SoC w.r.t. Battery Voltage at -10°C graph

5.6.1.1.2 Used condition

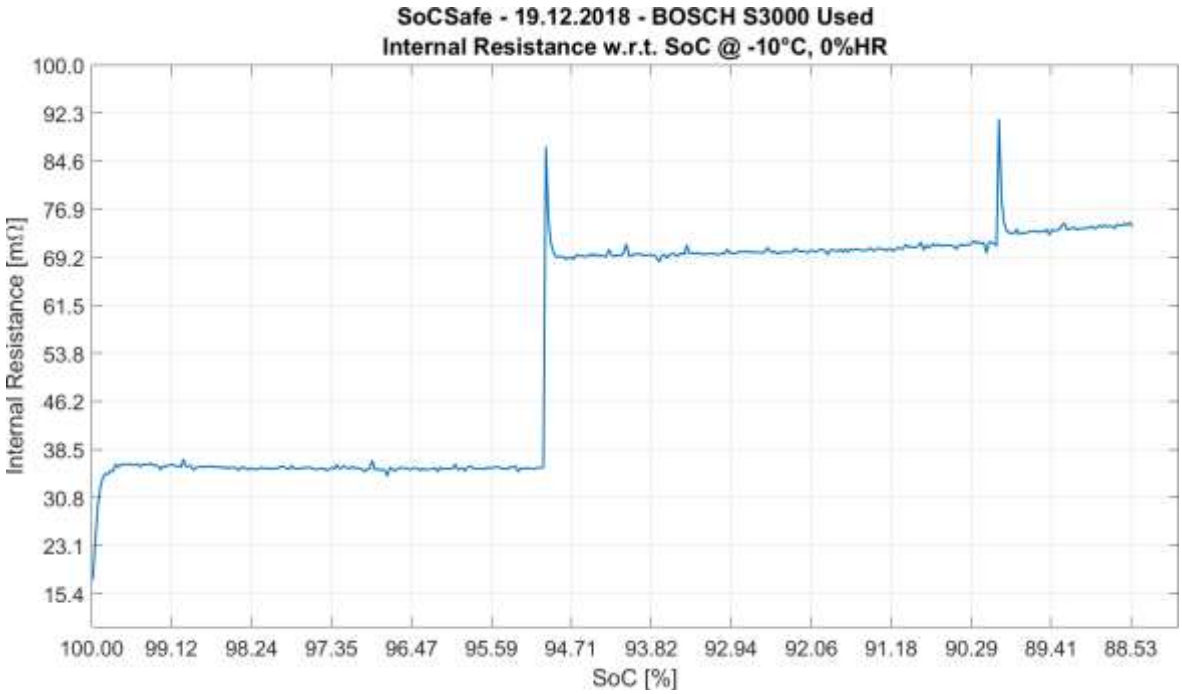


Figure 99. "SoCSafe" Used Bosch Internal Resistance w.r.t. SoC at -10°C graph

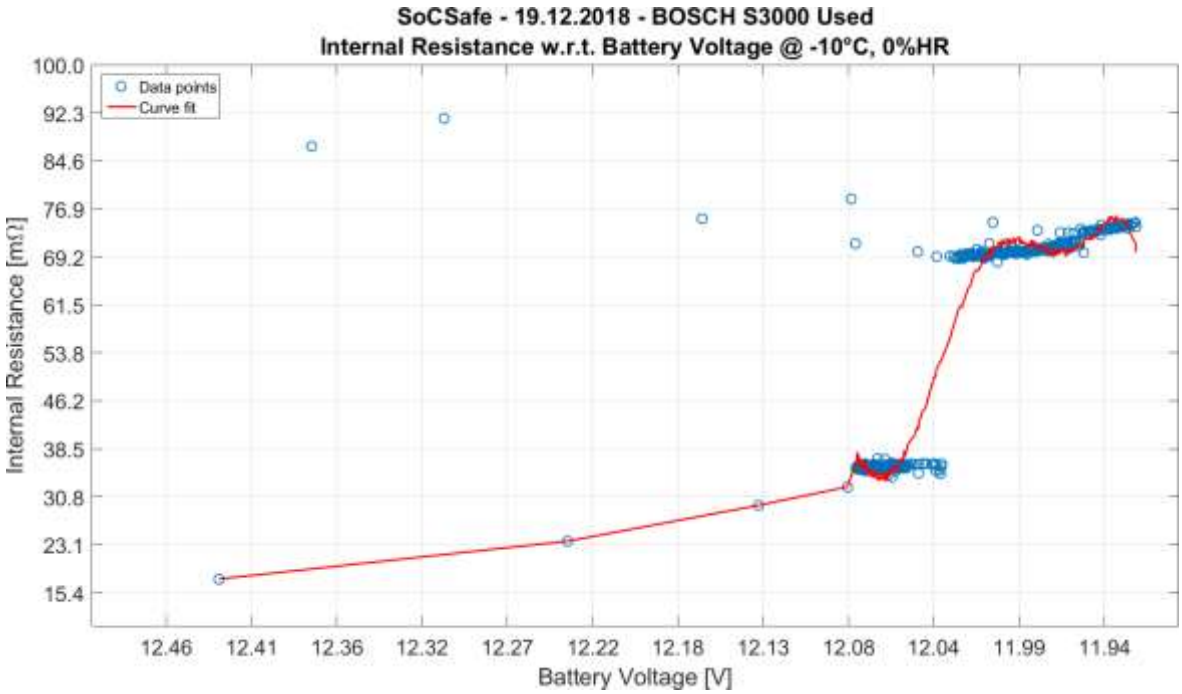


Figure 100. "SoCSafe" Used Bosch Internal Resistance w.r.t. Battery Voltage at -10°C graphs

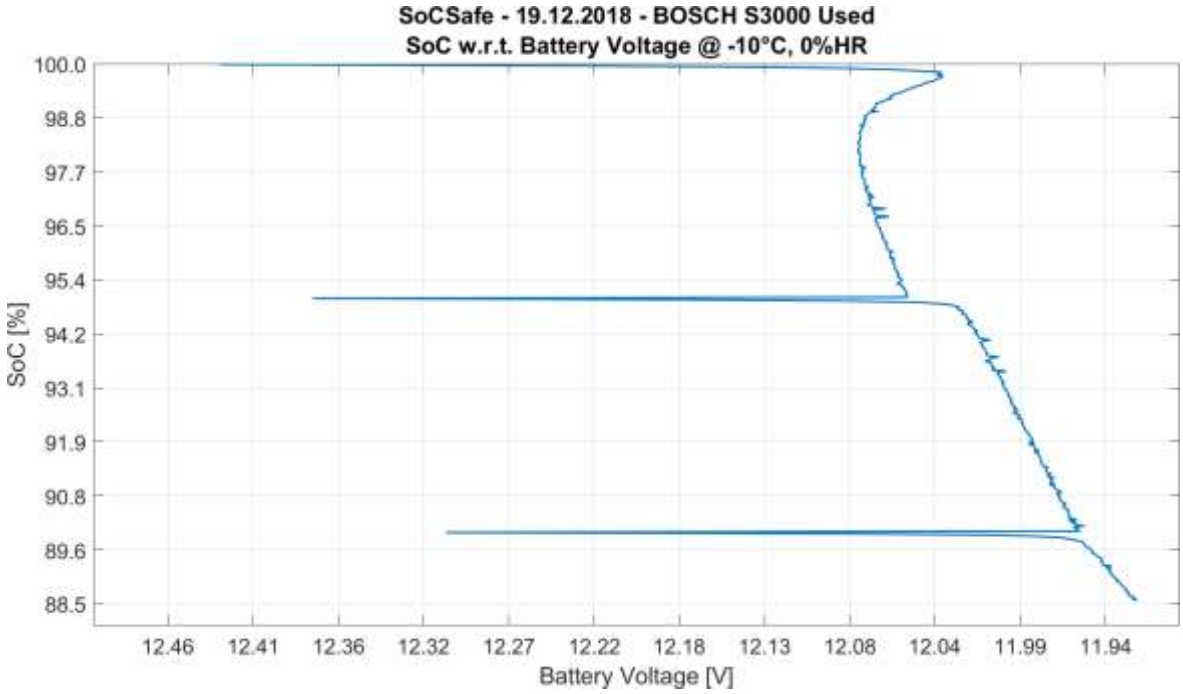


Figure 101. "SoCSafe" Used Bosch SoC w.r.t. Battery Voltage at -10°C graph

5.6.1.2 Fiamm L150P @ -10°C

5.6.1.2.1 New condition

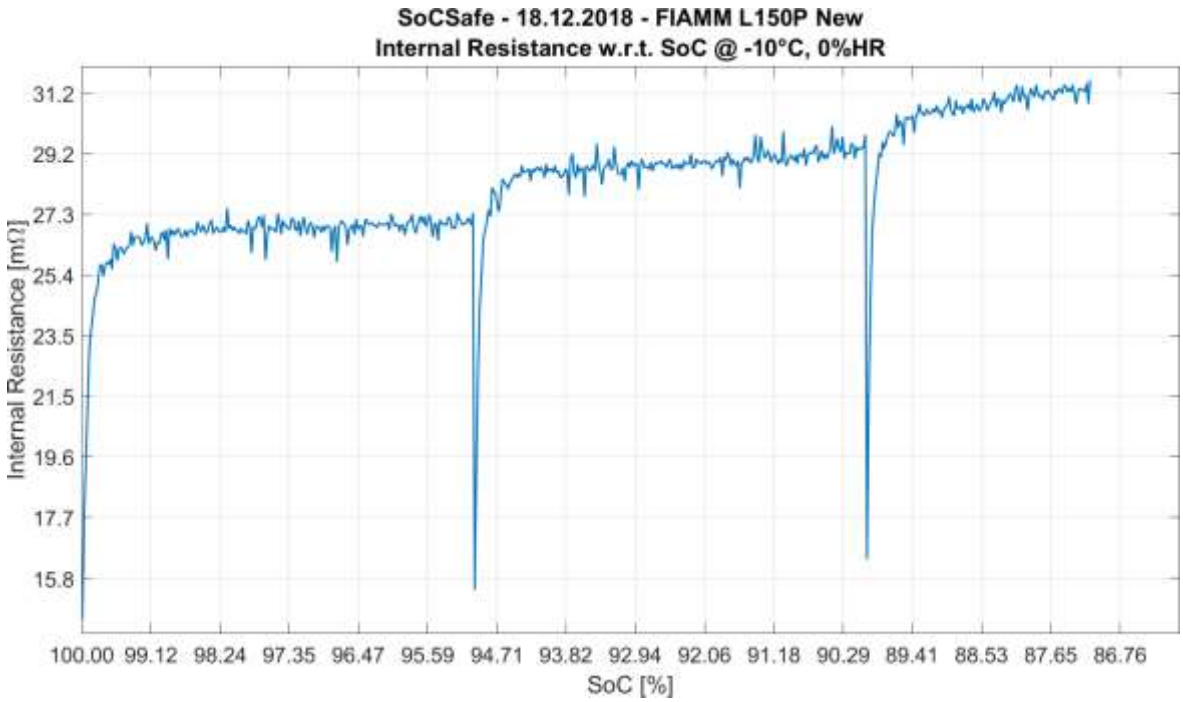


Figure 102. "SoCSafe" New Fiamm Internal Resistance w.r.t. SoC at -10°C graph

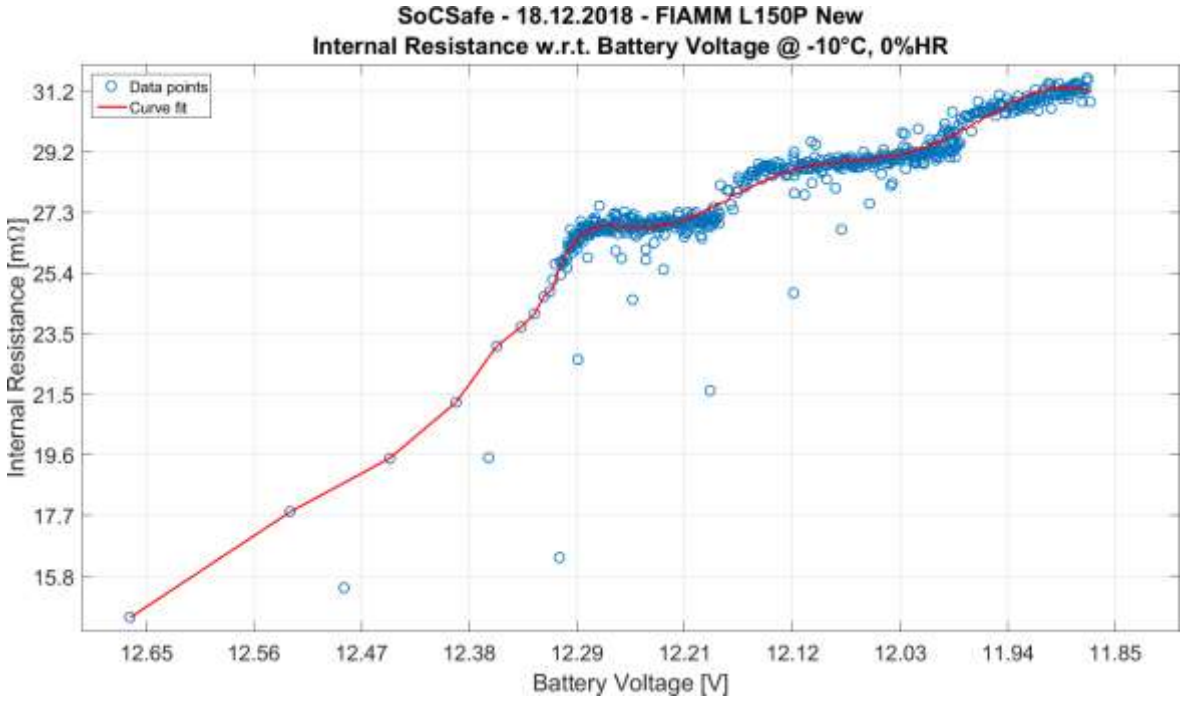


Figure 103. "SoCSafe" New Fiamm Internal Resistance w.r.t. Battery Voltage at -10°C graphs

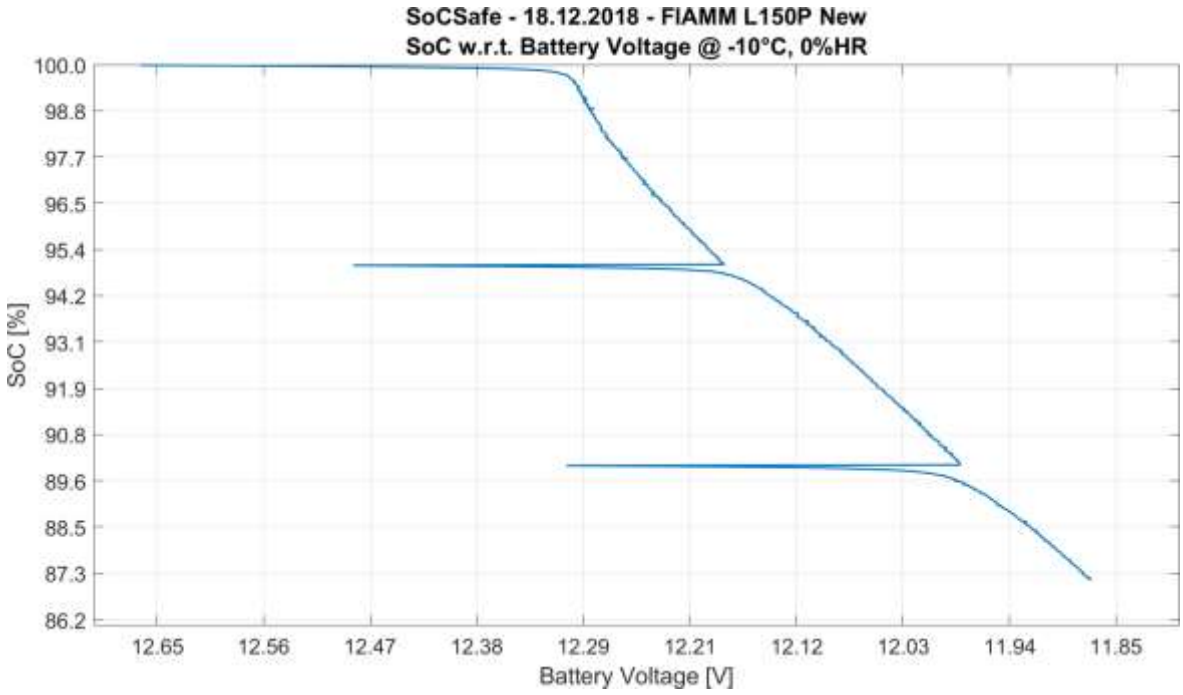


Figure 104. "SoCSafe" New Fiamm SoC w.r.t. Battery Voltage at -10°C graph

5.6.1.2.2 Used condition

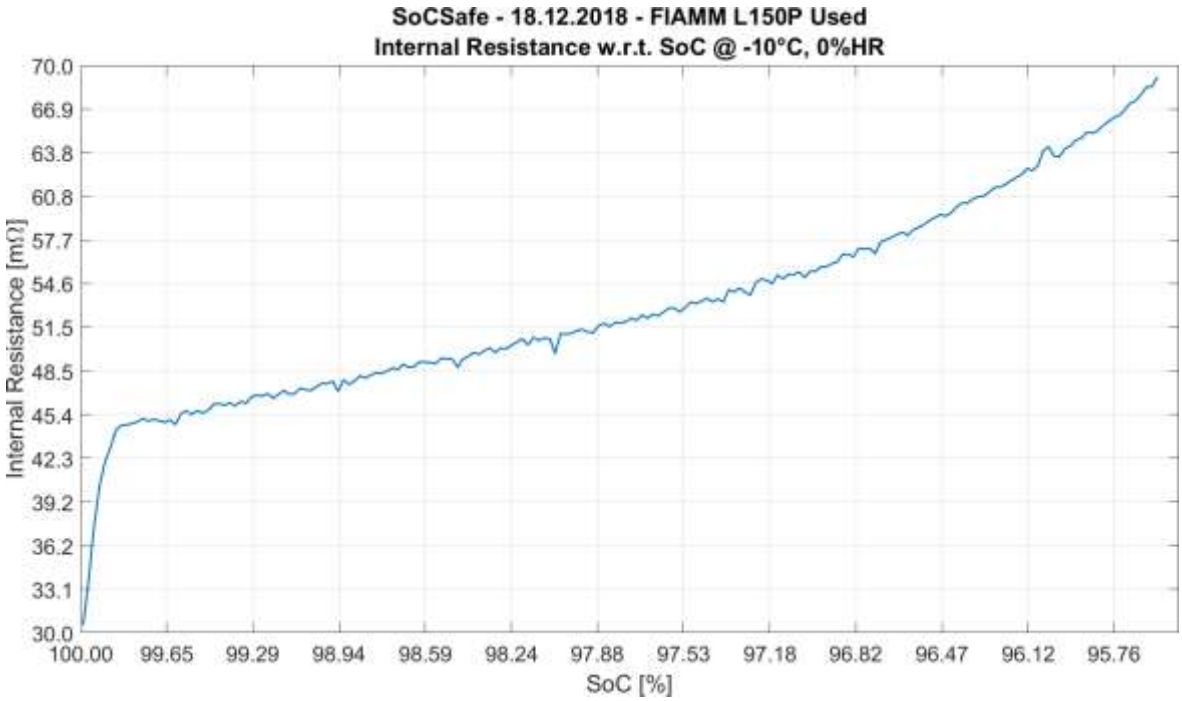


Figure 105. "SoCSafe" Used Fiamm Internal Resistance w.r.t. SoC at -10°C graph

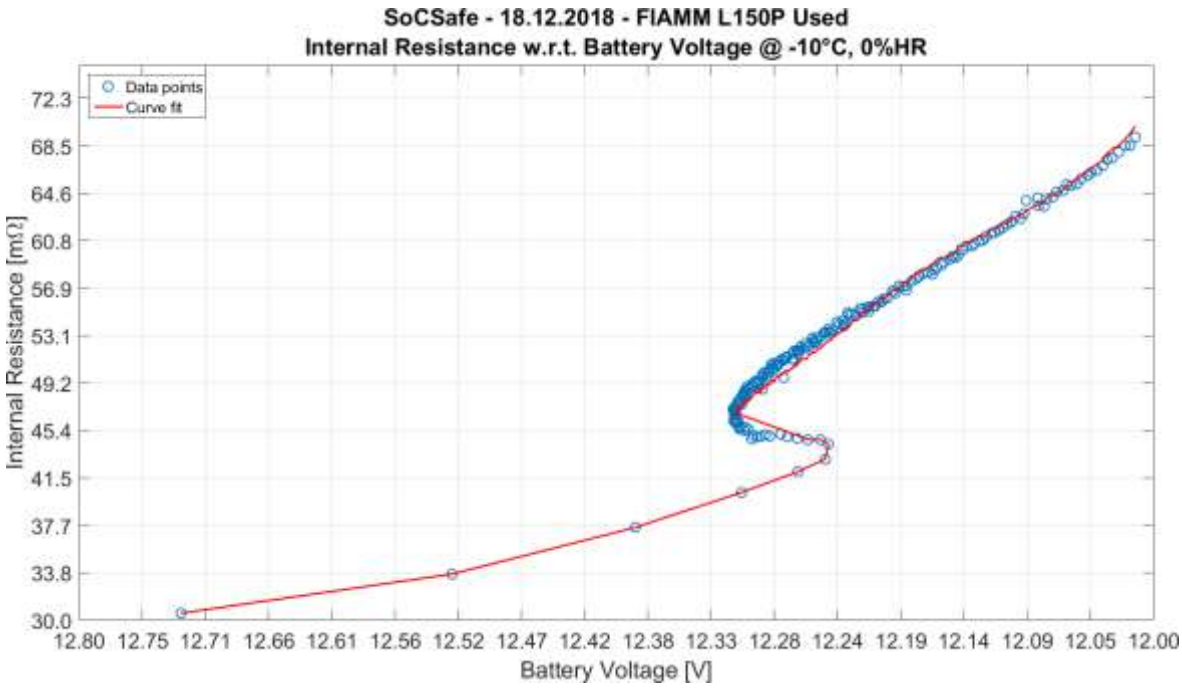


Figure 106. "SoCSafe" Used Fiamm Internal Resistance w.r.t. Battery Voltage at -10°C graphs

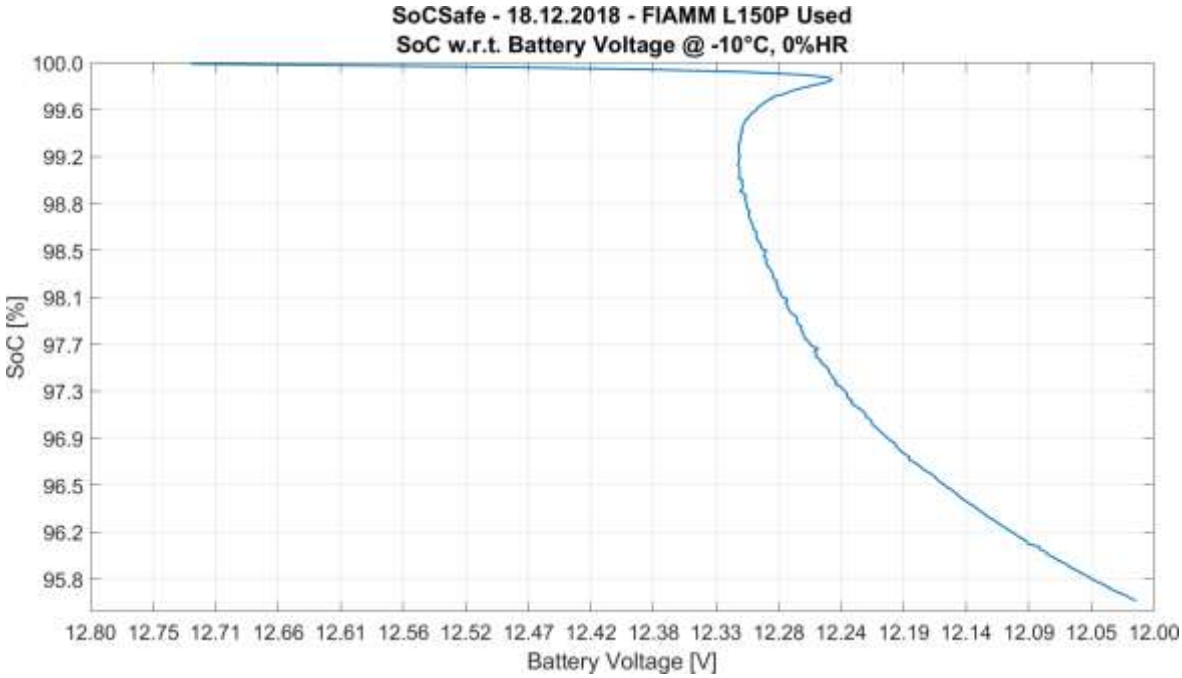


Figure 107. "SoCSafe" Used Fiamm SoC w.r.t. Battery Voltage at -10°C graph

5.6.1.3 Bosch S3000 @ +40°C

5.6.1.3.1 New condition

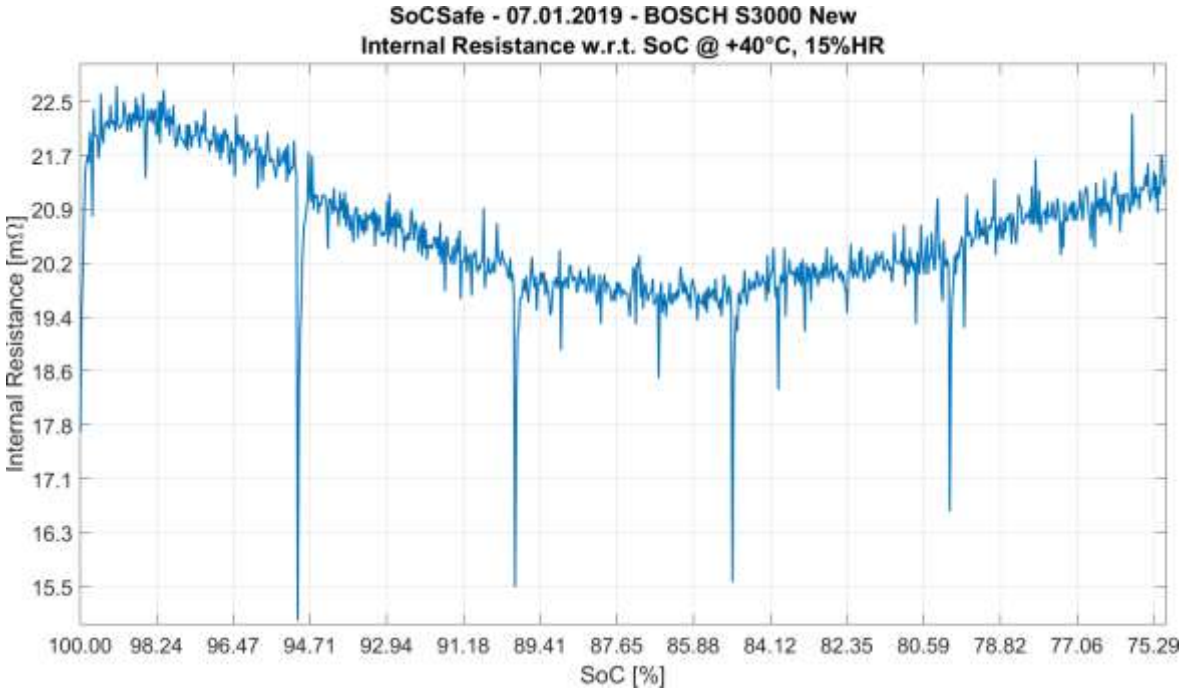


Figure 108. "SoCSafe" New Bosch Internal Resistance w.r.t. SoC at +40°C graph

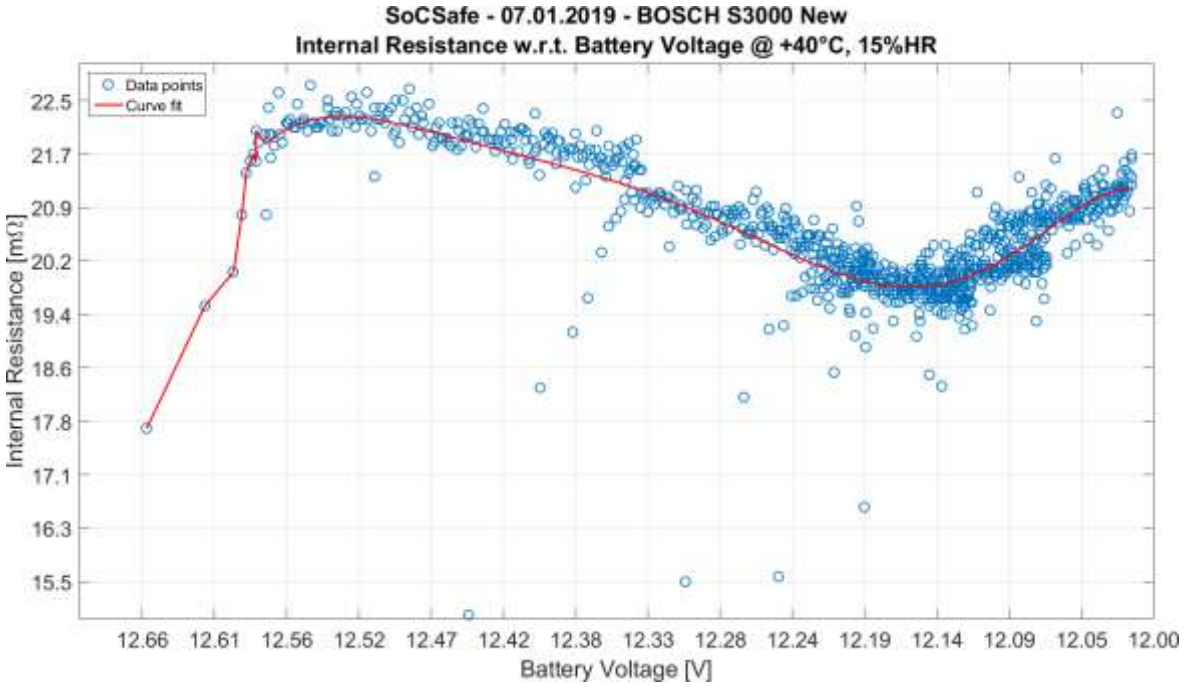


Figure 109. "SoCSafe" New Bosch Internal Resistance w.r.t. Battery Voltage +40°C graphs

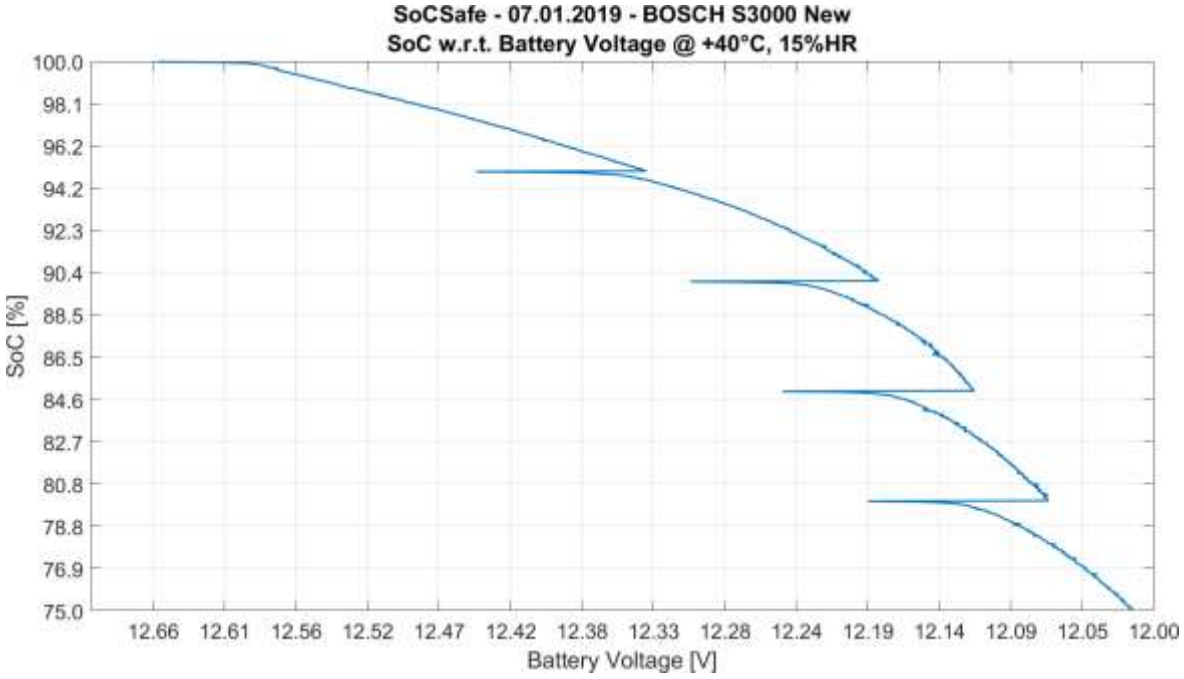


Figure 110. "SoCSafe" New Bosch SoC w.r.t. Battery Voltage +40°C graph

5.6.1.3.2 Used condition

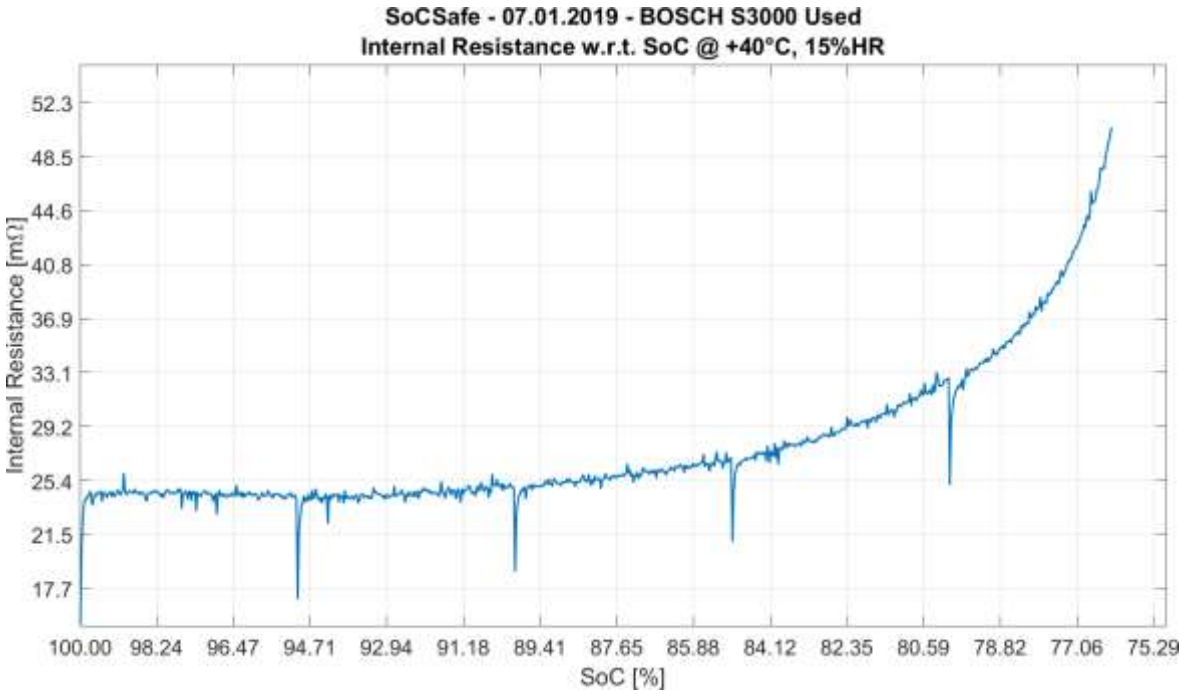


Figure 111. "SoCSafe" Used Bosch Internal Resistance w.r.t. SoC +40°C graph

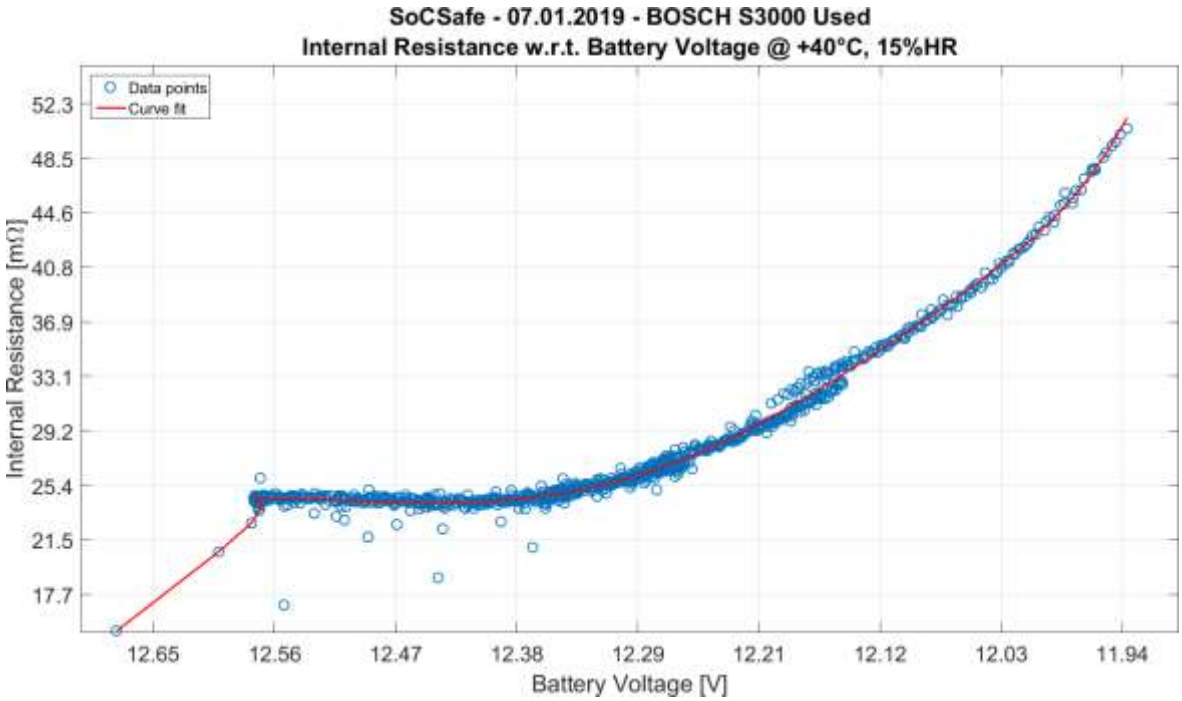


Figure 112. "SoCSafe" Used Bosch Internal Resistance w.r.t. Battery Voltage +40°C graphs

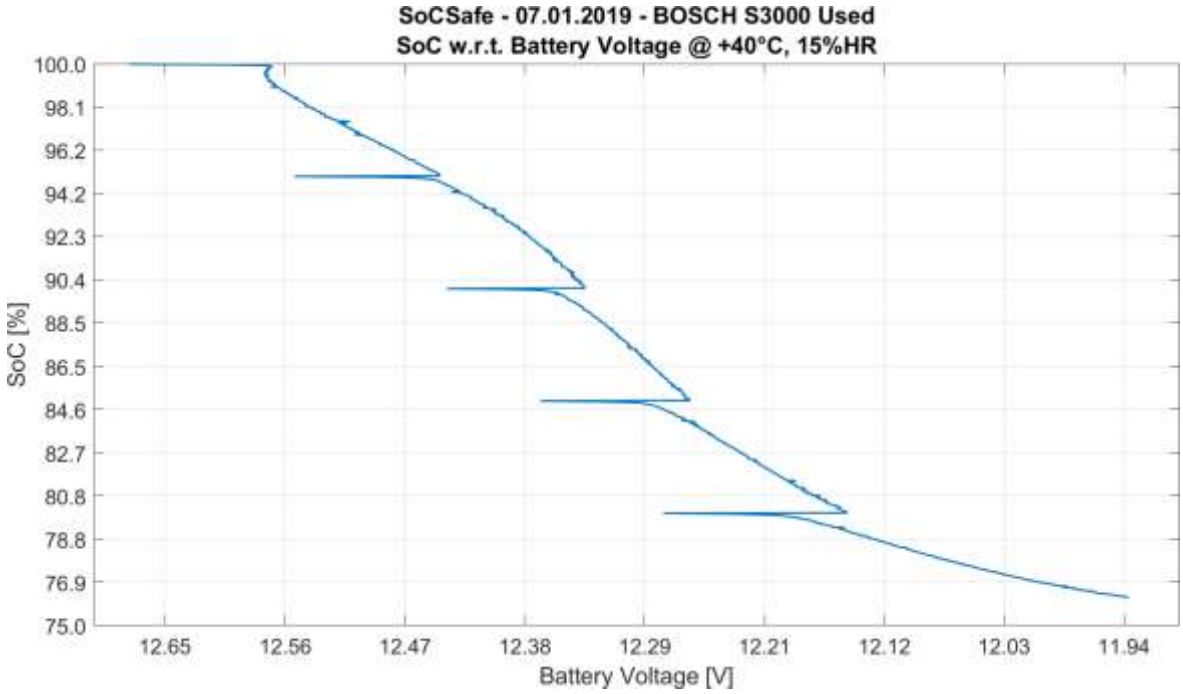


Figure 113. "SoCSafe" Used Bosch SoC w.r.t. Battery Voltage +40°C graph

5.6.1.4 Fiamm L150P @ +40°C

5.6.1.4.1 New condition

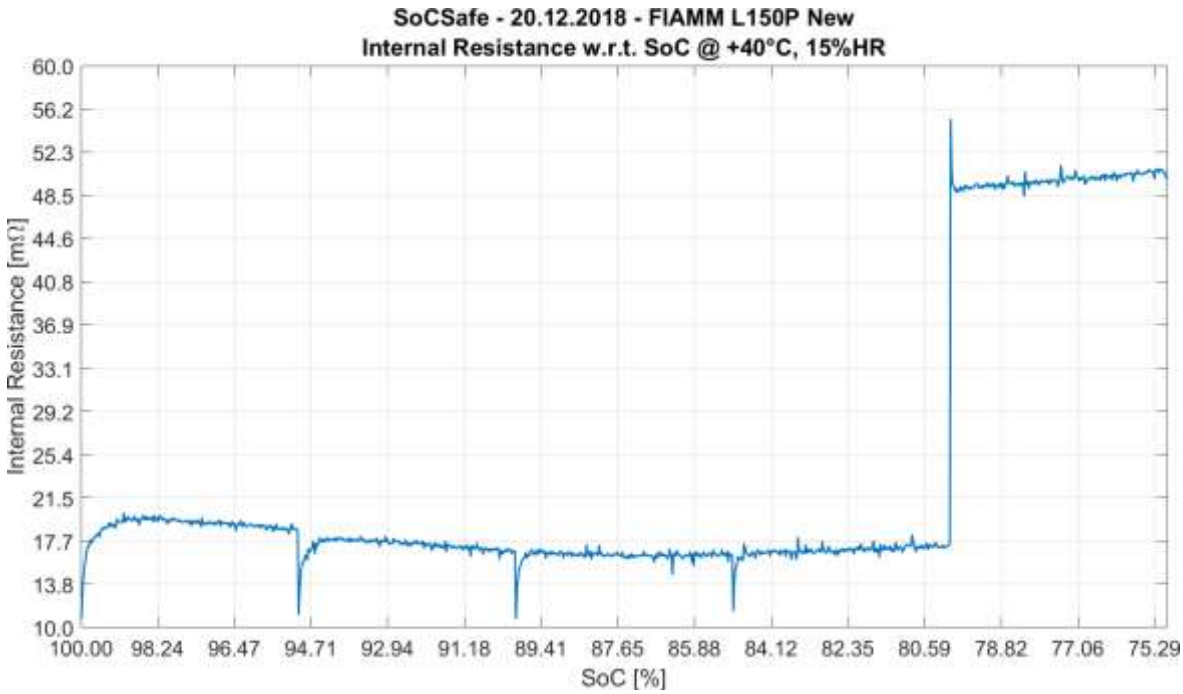


Figure 114. "SoCSafe" New Fiamm Internal Resistance w.r.t. SoC +40°C graph

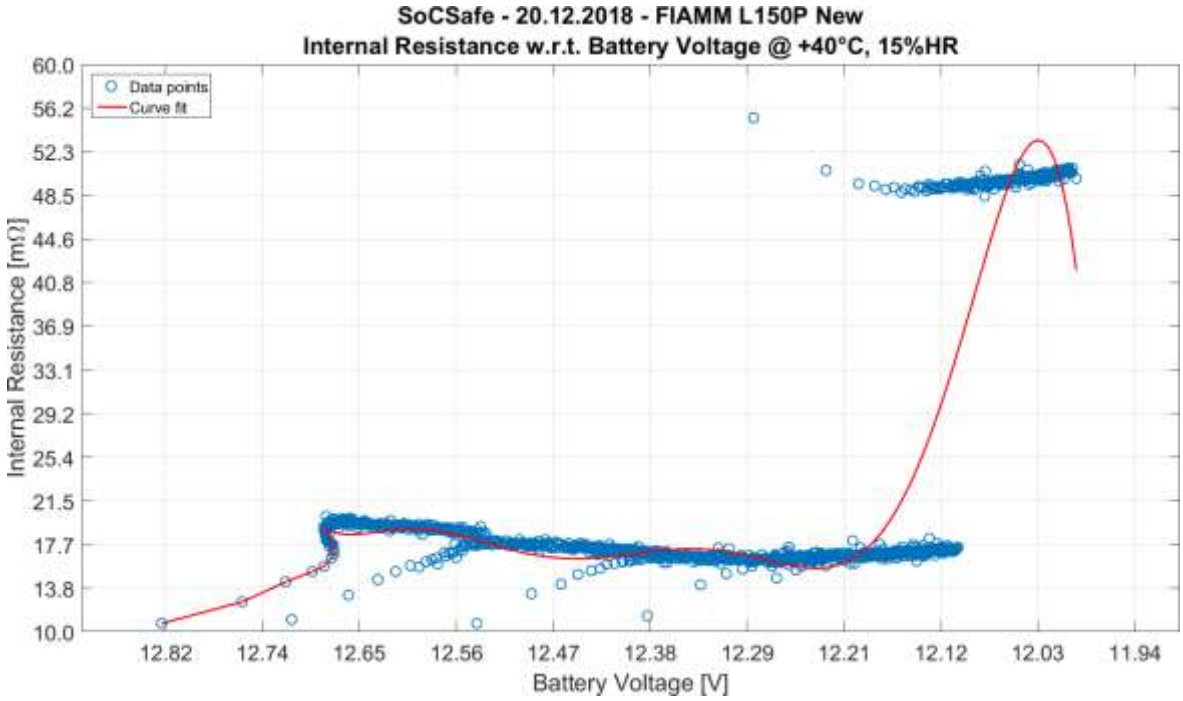


Figure 115. "SoCSafe" New Fiamm Internal Resistance w.r.t. Battery Voltage +40°C graphs

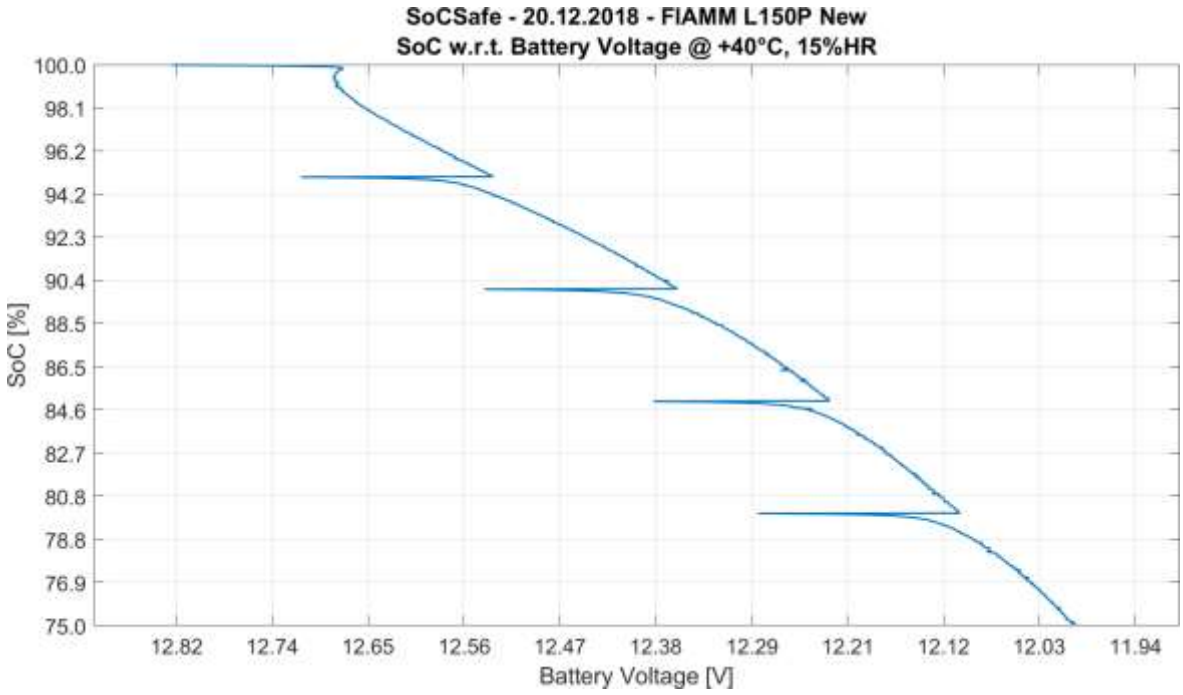


Figure 116. "SoCSafe" New Fiamm SoC w.r.t. Battery Voltage +40°C graph

5.6.1.4.2 Used condition

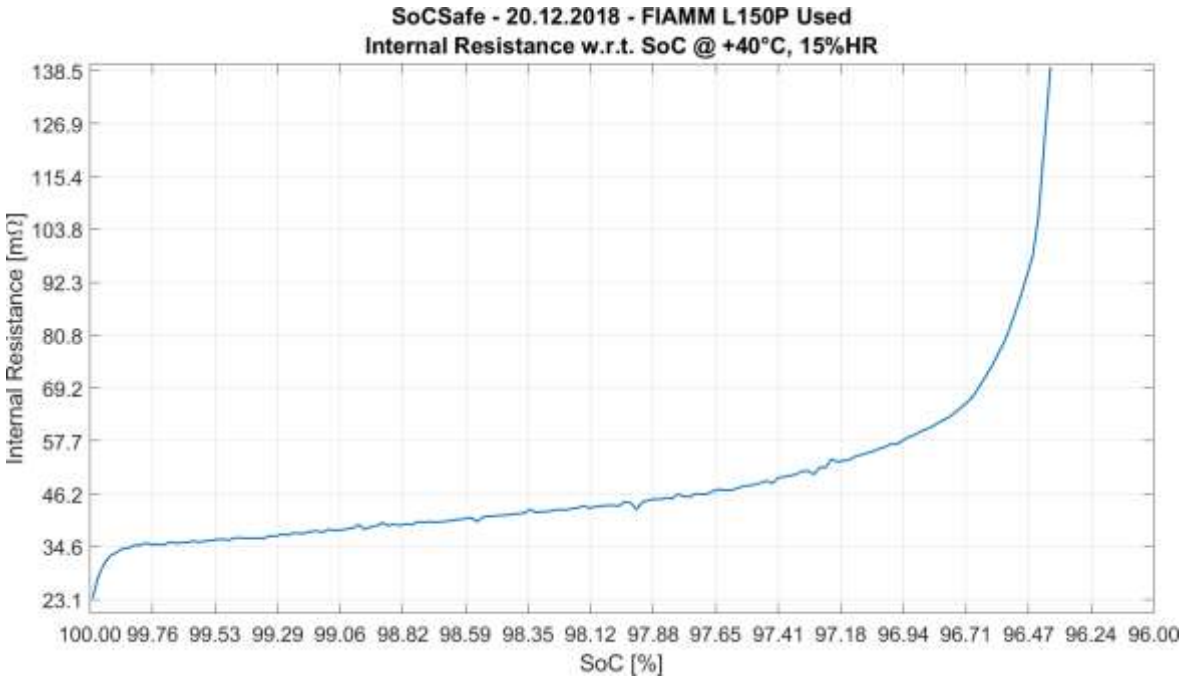


Figure 117. "SoCSafe" Used Fiamm Internal Resistance w.r.t. SoC +40°C graph

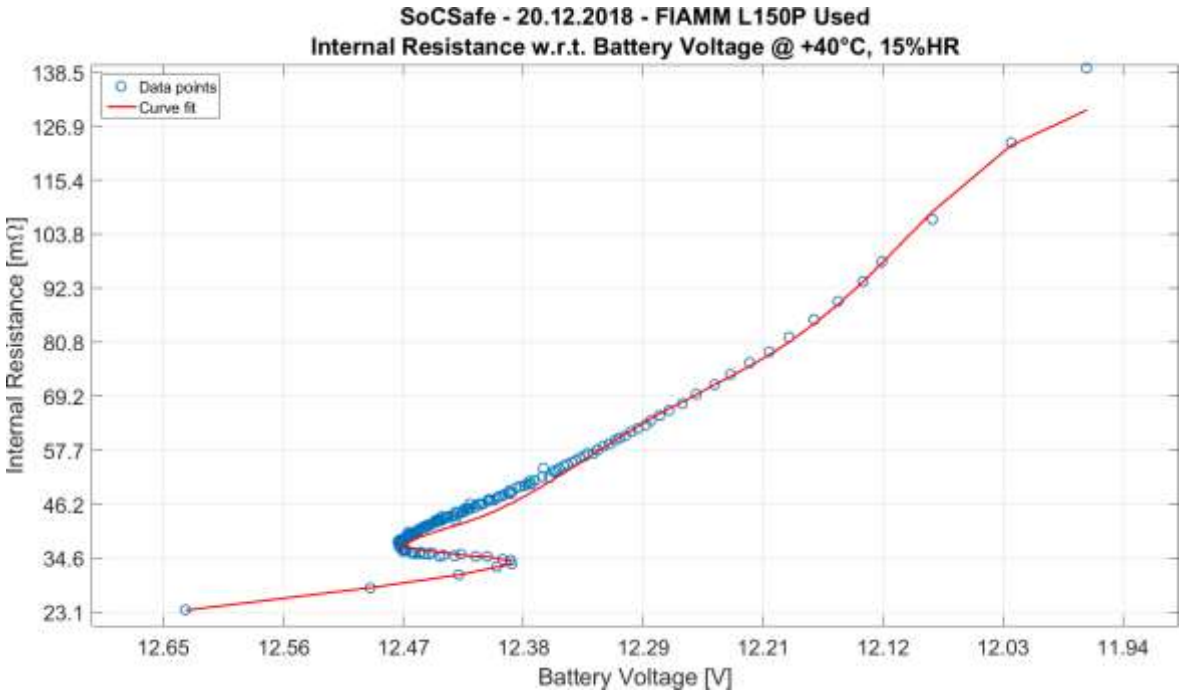


Figure 118. "SoCSafe" Used Fiamm Internal Resistance w.r.t. Battery Voltage +40°C graphs

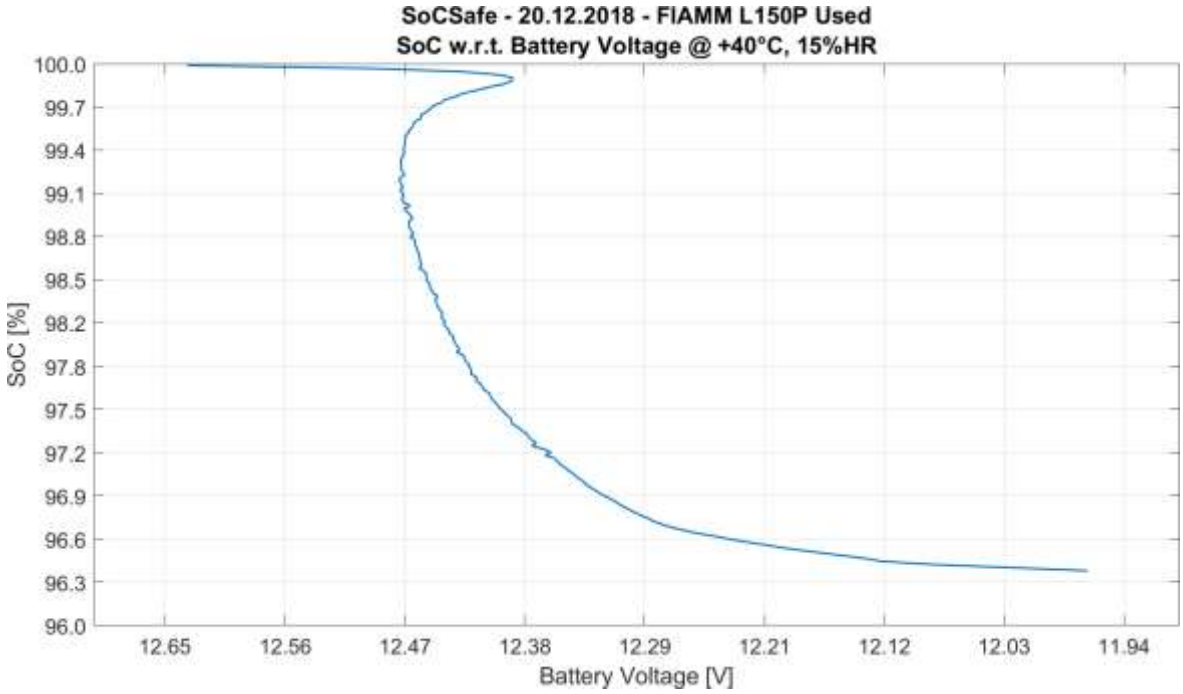


Figure 119. "SoCSafe" Used Fiamm SoC w.r.t. Battery Voltage +40°C graph

5.7 "Deep" campaign

The experiment was performed 10 times on each battery, gaining a total of 90 test cycles for each SoC level, at least for those the battery has managed to reach.

5.7.1 Concept art

5.7.1.1 Plot of an acquisition cycle

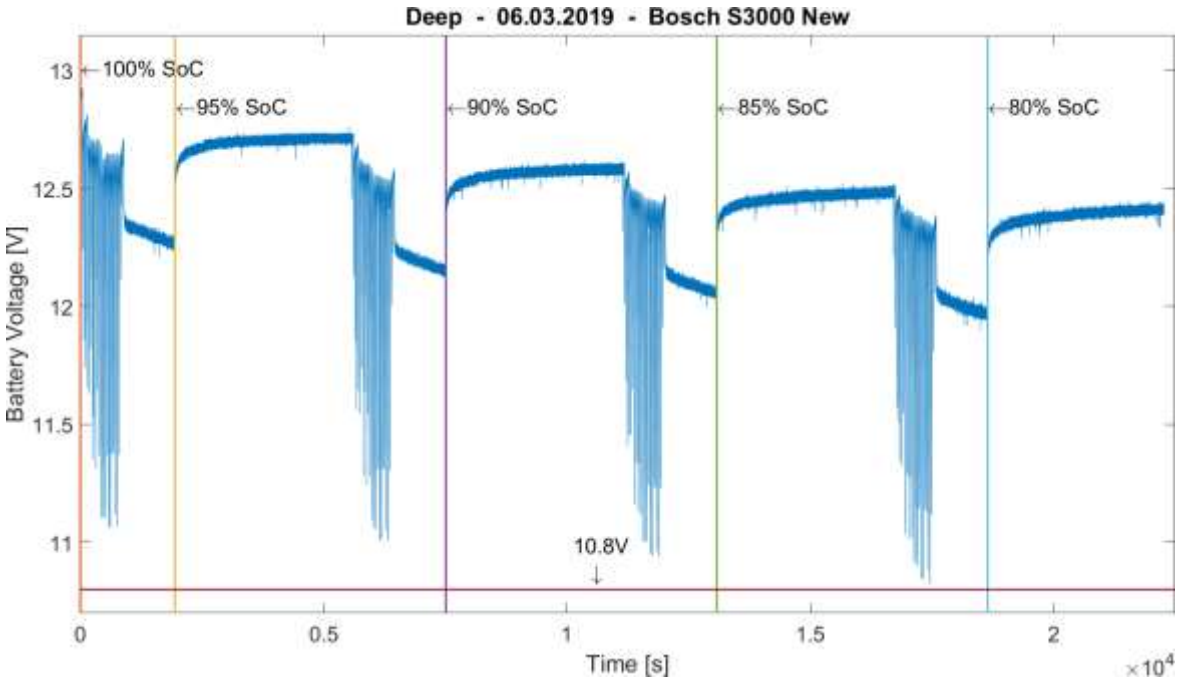


Figure 120. Plot of an acquisition cycle of "Deep" data

5.7.1.2 Details of the plot of an acquisition cycle

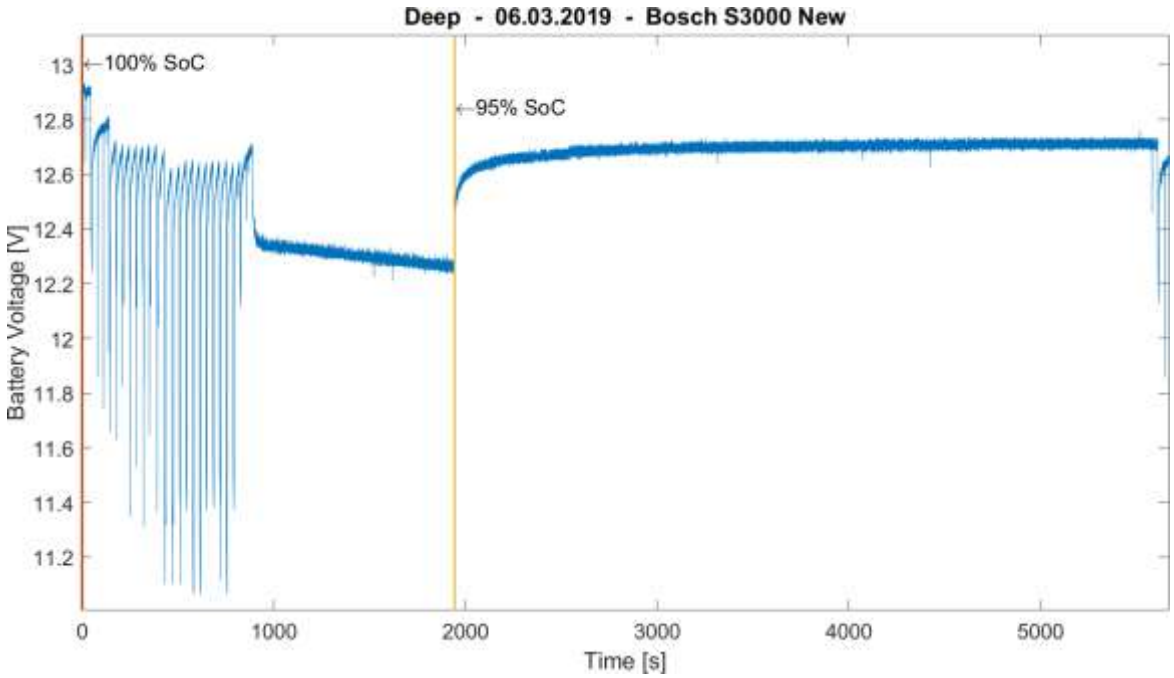


Figure 121. Detail of the plot of an acquisition of "Deep" data

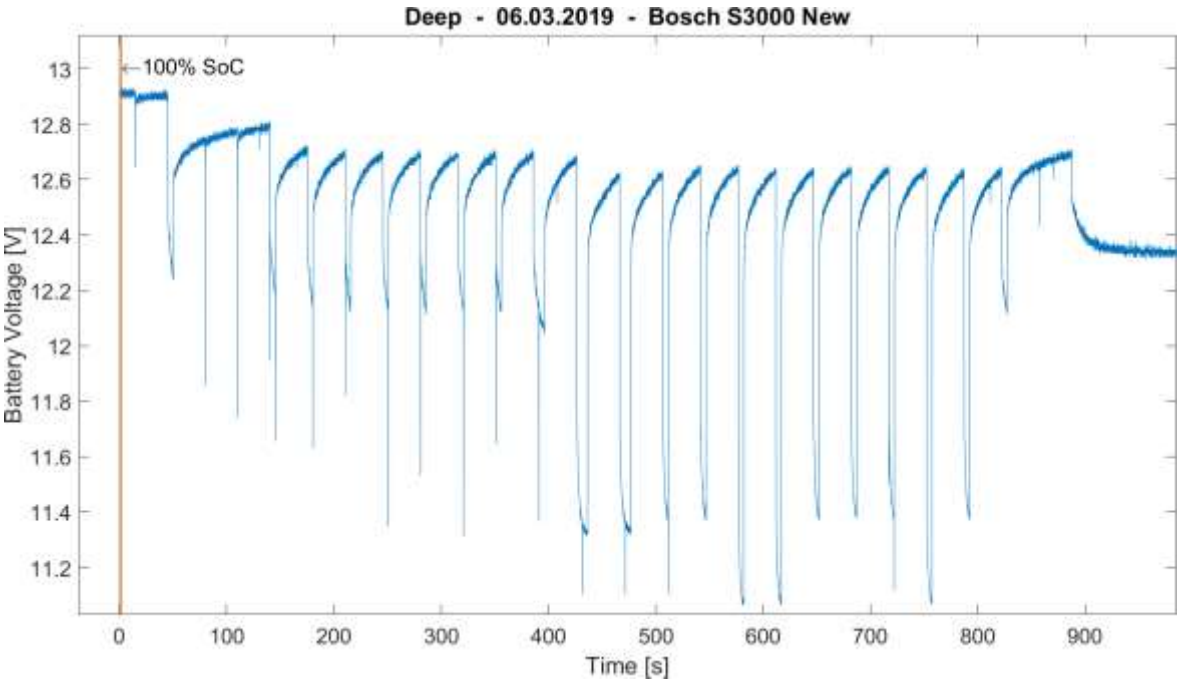


Figure 122. Zoom of the detail of the plot of an acquisition of "Deep" data

5.7.2 Battery analysis

5.7.2.1 Bosch S3000

5.7.2.1.1 New condition

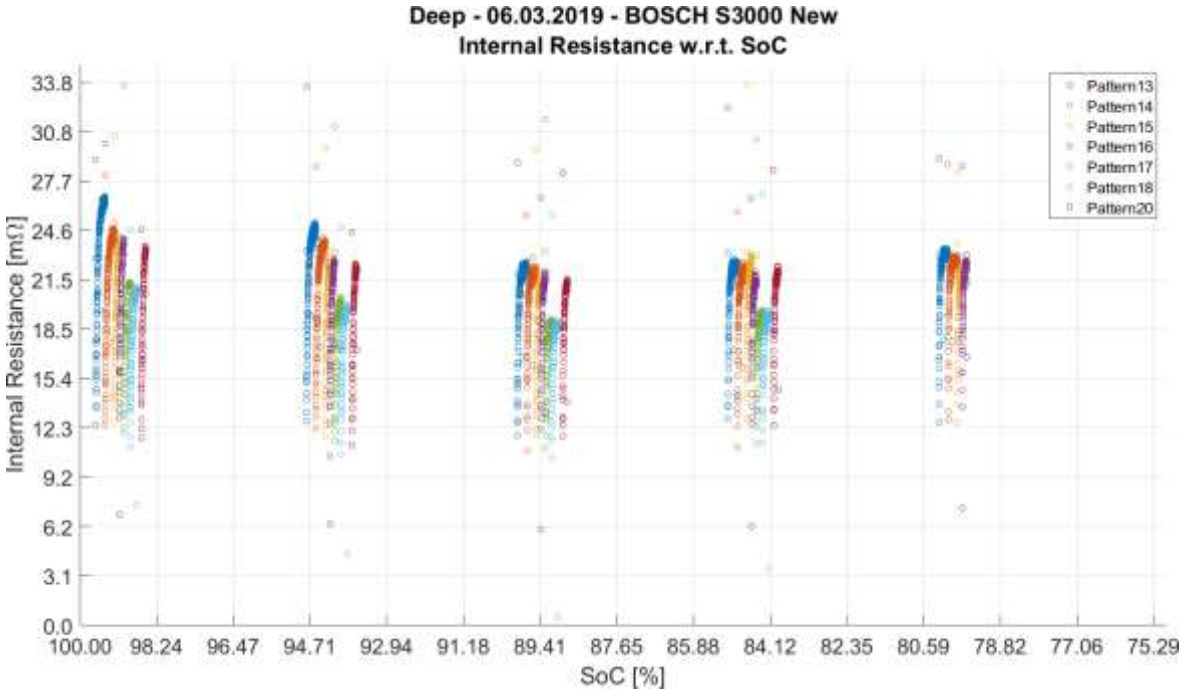


Figure 123. "Deep" New Bosch Internal Resistance w.r.t. SoC graphs

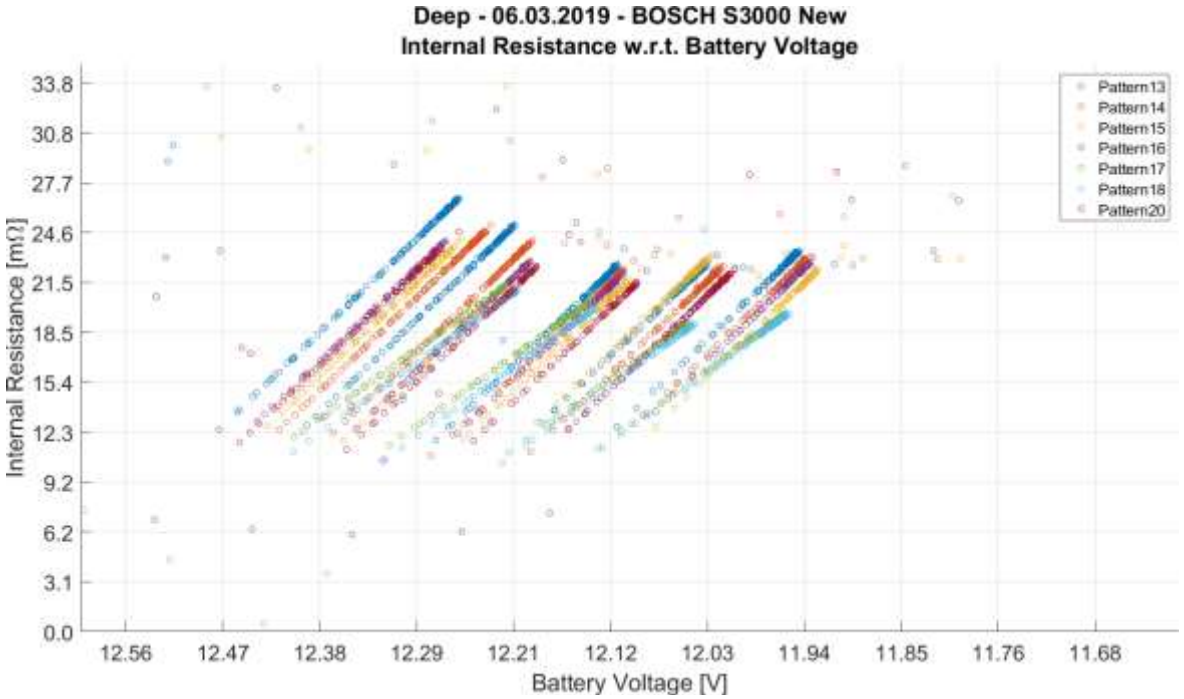


Figure 124. "Deep" New Bosch Internal Resistance w.r.t. Battery Voltage graphs

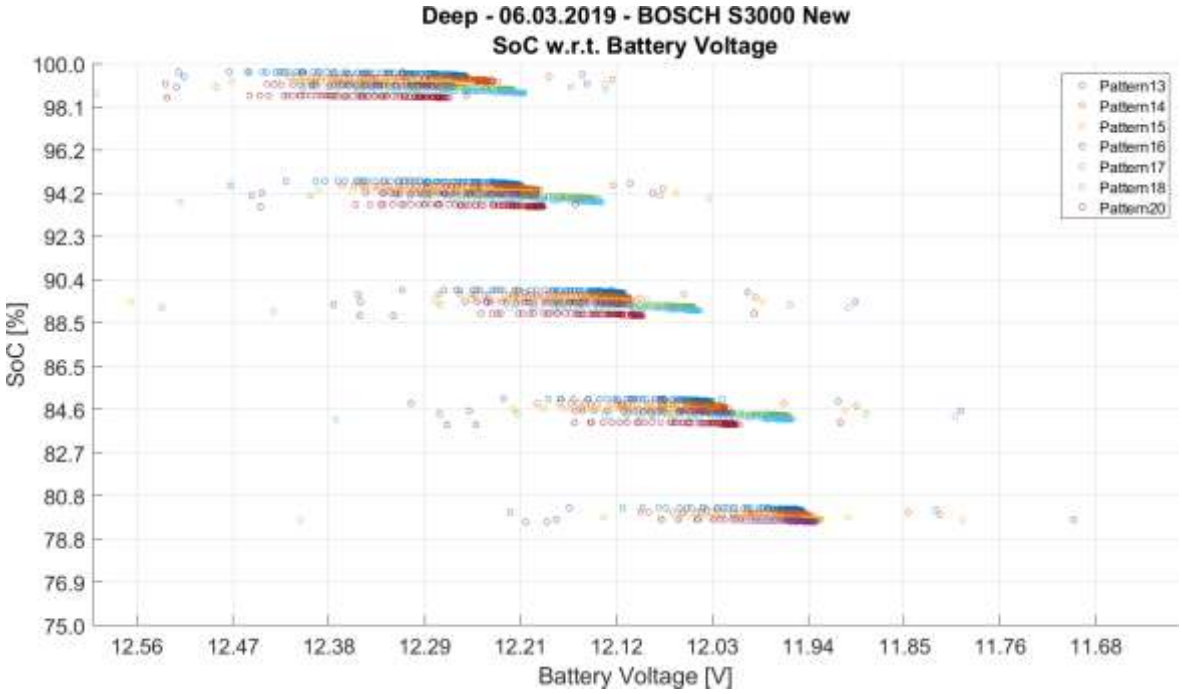


Figure 125. "Deep" New Bosch SoC w.r.t. Battery Voltage graphs

5.7.2.2 Energeco Clausum-S562.059.057

5.7.2.2.1 New condition

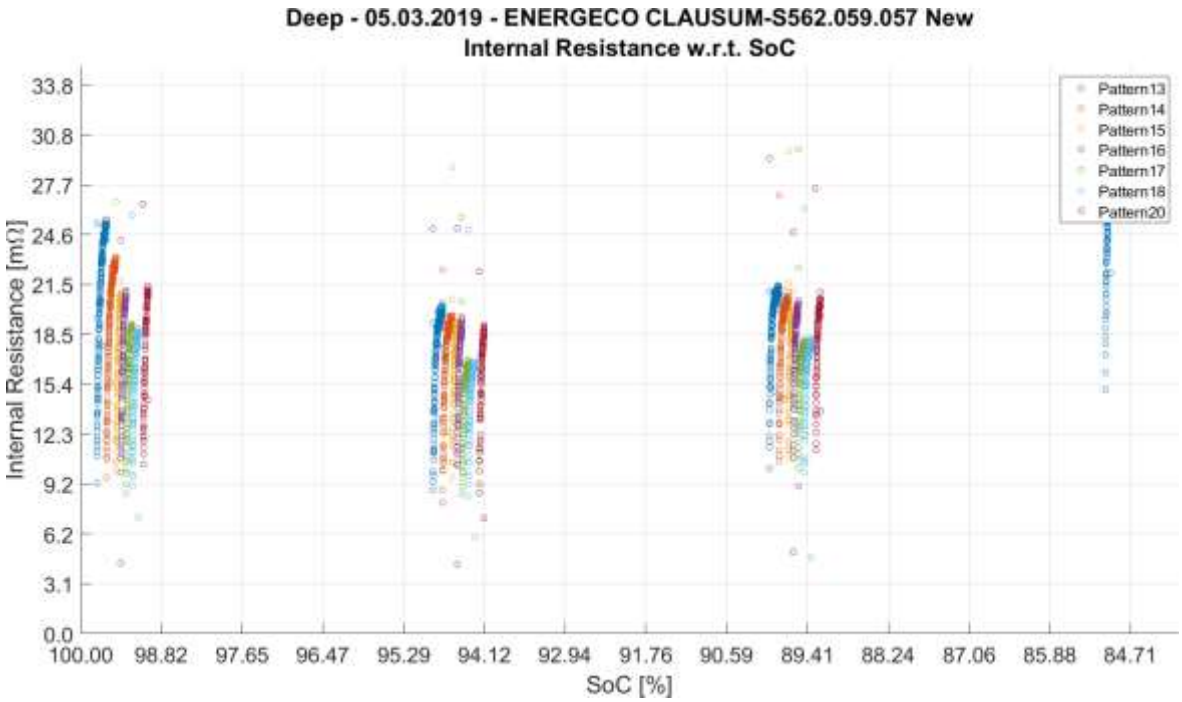


Figure 126. "Deep" New Energeco Internal Resistance w.r.t. SoC graphs

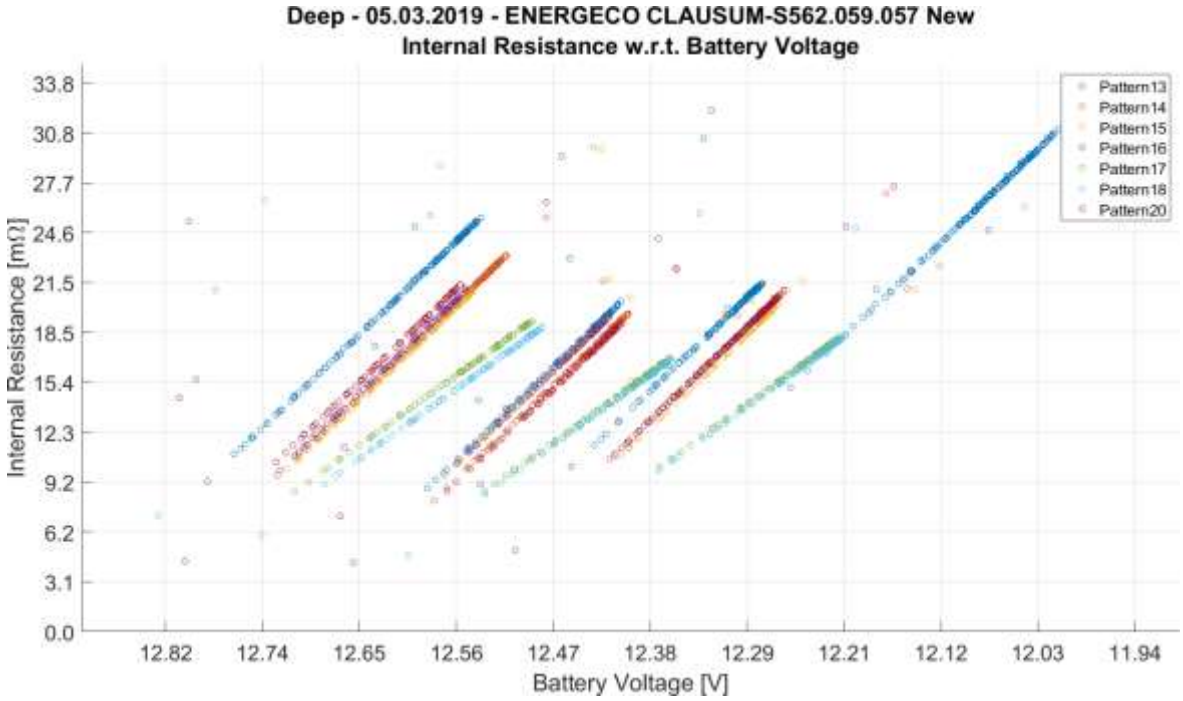


Figure 127. "Deep" New Energeco Internal Resistance w.r.t. Battery Voltage graphs

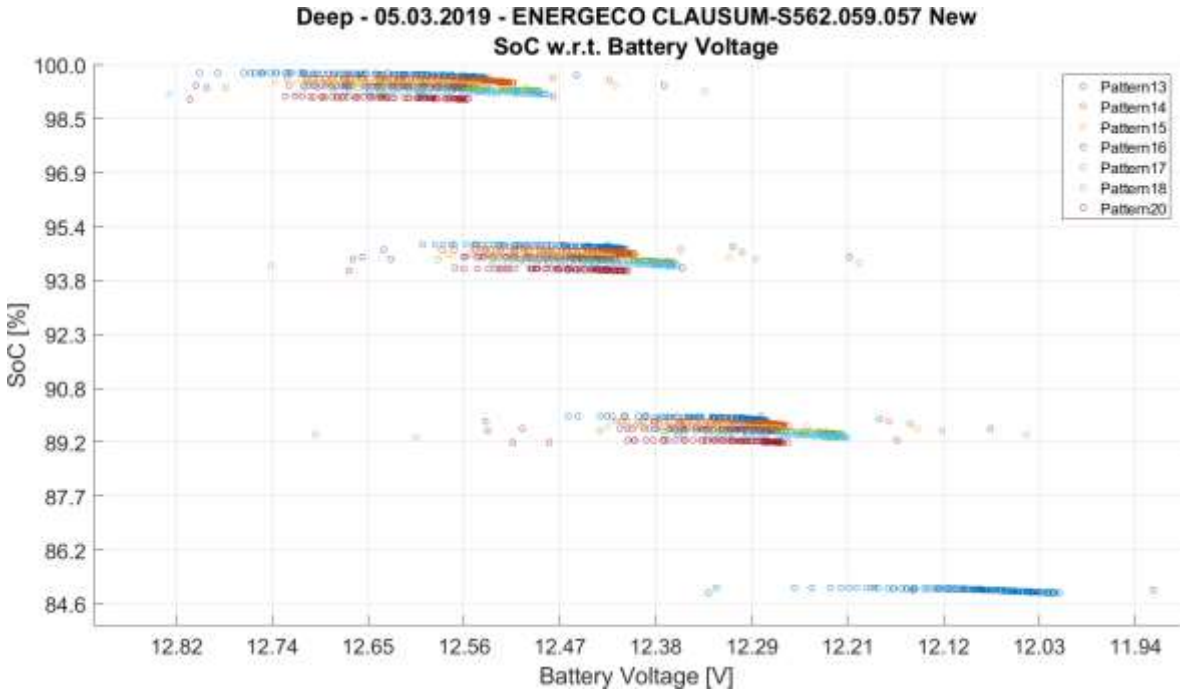


Figure 128. "Deep" New Energeco SoC w.r.t. Battery Voltage graphs

5.7.2.3 Fiamm L150P

5.7.2.3.1 New condition

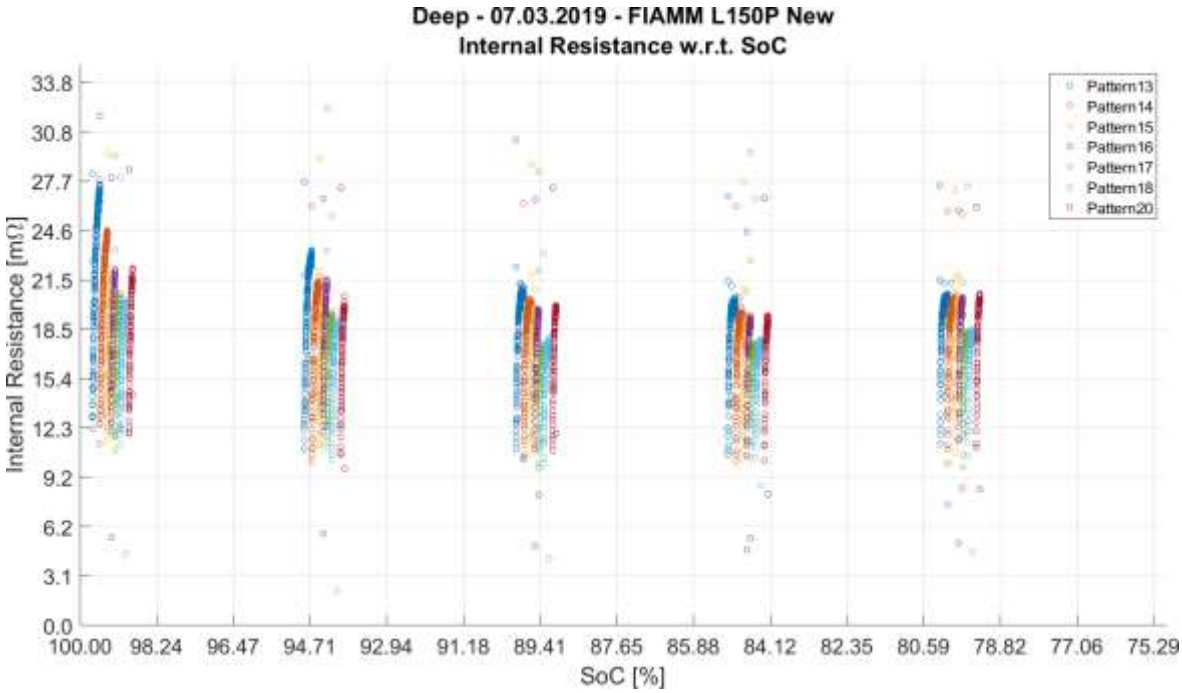


Figure 129. "Deep" New Fiamm Internal Resistance w.r.t. SoC graphs

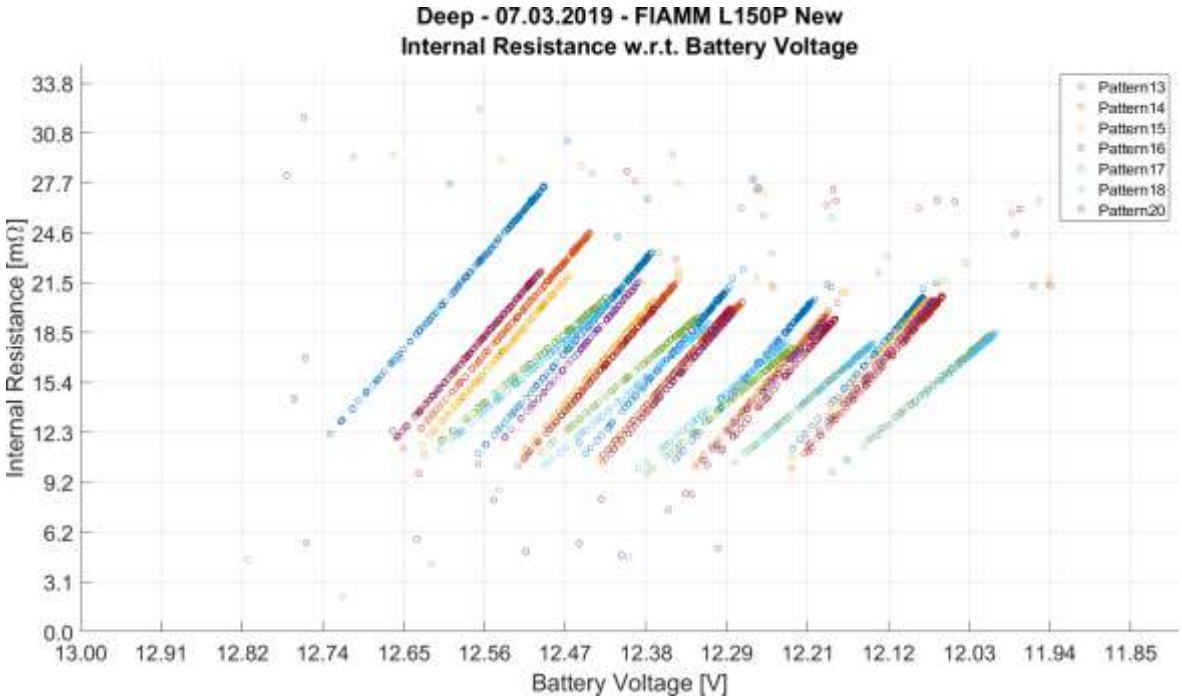


Figure 130. "Deep" New Fiamm Internal Resistance w.r.t. Battery Voltage graphs

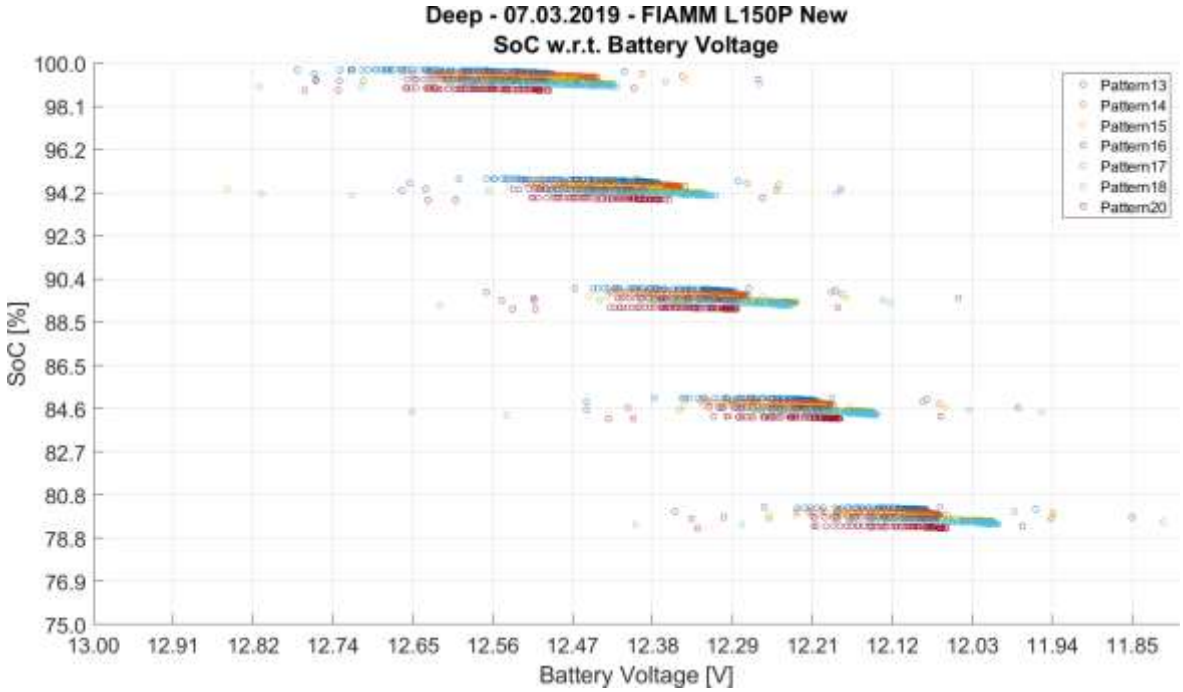


Figure 131. "Deep" New Fiamm SoC w.r.t. Battery Voltage graphs

Chapter 6.

Assessment of the results achieved

6.1 "Spectrum" campaign

The internal resistance increases when the control frequency decreases; point-by-point, the values are bigger at 0.1Hz than at all the others, which, in turn, are close together, even considering the logarithmic scale.

Model	Resistance [mΩ] @ 12.29V			
	New		Used	
	@ 0.1Hz	@ 50Hz	@ 0.1Hz	@ 50Hz
Bosch S3000	24.2	7.82	26.1	12.0
Energeco Clausum-S562.059.057	17.6	5.40	18.5	7.75
Fiamm L150P	14.5	3.96	37.8	19.9

Among all the brands, the general trend of the internal resistance is increasing with respect to the decrease of the battery voltage except for a small range in which the internal resistance decreases abruptly.

Model	Resistance [mΩ] @ 50Hz			
	New		Used	
	@ 12.5V	@ 11.3V	@ 12.5V	@ 11.3V
Bosch S3000	7.54	17.4	11.4	32.5
Energeco Clausum-S562.059.057	5.27	27.1	7.38	25.9
Fiamm L150P	7.61	23.0	18.2	> 50

As expected, the SoC decreases as the battery voltage decreases. Any difference was not noticed varying the control frequency.

6.2 "SoC" campaign

Similarly to the previous result, the general trend of the internal resistance is increasing with respect to the decrease of the battery voltage.

Considering the big amount of data collected, a curve fitting that analytically describes the trend was obtained; the points were sorted in ascending order to achieve a monotonic curve (and a better graphic result) and a polynomial curve that best fits the points.

Shortly after the test starts, a spike-shaped deviation from the trend of the internal resistance is appreciable on all the batteries except for the used Energeco Clausum-S562.059.057. This spike-shaped anomaly is observable at the same battery voltage, i.e. at the same time, on the SoC / Battery Voltage graph and it lasts for a maximum of 1% of SoC.

The slope of the SoC with respect to the battery voltage is almost linear, up until a certain voltage is reached. This voltage threshold, here named as knee voltage (V_{knee}), defines the limit of the region of linear operation, outside of which the battery continues to supply current at the expense of the voltage.

Model	Bosch S3000		Energeco Clausum-S562.059.057		Fiamm L150P	
Condition	New	Used	New	Used	New	Used
SoC_{start} [%]	99.4	99.4	99.5	100	99.5	99.2
V_{start} [V]	12.41	12.46	12.84	12.49	12.84	12.50
SoC_{knee} [%]	66.0	77.8	53.9	91.5	56.5	95.5
V_{knee} [V]	11.44	11.82	11.65	12.12	11.42	12.17
Slope [%SoC/V]	34.4	33.7	38.3	23.0	30.3	11.2

The same voltage V_{knee} represents a threshold for the internal resistance: before it, the resistance is virtually constant; after it, the resistance increases more abruptly in the used batteries than in the new ones.

Model	Bosch S3000		Energeco Clausum-S562.059.057		Fiamm L150P	
Condition	New	Used	New	Used	New	Used
Resistance [mΩ]	27	30	17	22	20	35

6.3 "SoCSafe" campaign

In each graph related to the internal resistance, a different behavior of it can be observed: the Bosch has a step increase at 85% of SoC; the Energeco shows an oscillating variation with diminishing duty cycle and period equal to the 5% SoC decreasing step; the Fiamm presents a more linear trend that decreases along with the battery voltage.

The slope of the SoC is calculated starting after the spike-shaped anomaly, that lasts for a maximum of 1% of SoC, until the test ends.

Model	Bosch S3000	Energeco Clausum-S562.059.057	Fiamm L150P
Condition	Pristine	Pristine	Pristine
SoC_{start} [%]	99.0	99.5	99.4
V_{start} [V]	12.18	12.93	13.07
SoC_{stop} [%]	75.0	75.0	75.0
V_{stop} [V]	11.90	12.40	12.47
Slope [%SoC/V]	86.7	46.3	40.9

6.4 "SoCSafe" campaign in controlled temperature

The most remarkable difference between the tests at -10°C and at +40°C is the duration of the test itself, which is lower in the former.

The temperature has different effects on the two models: it doesn't affect the SoC and internal resistance trends of the Fiamm batteries; while it is the opposite for the Bosch.

Model	Bosch S3000		Fiamm L150P	
Condition	New	Used	New	Used
SoC_{end} [%] @ -10°C	92.6	88.5	87.3	96.3
SoC_{end} [%] @ +40°C	75.0	75.8	75.0	95.5

The slope of the SoC with respect to the battery voltage is almost linear. It is calculated starting after the spike-shaped anomaly, that lasts for a maximum of 1% of SoC, until the test ends.

Temperature -10°C, Relative Humidity 0%				
Model	Bosch S3000		Fiamm L150P	
Condition	New	Used	New	Used
SoC_{start} [%]	99.3	99.2	99.2	99.4
V_{start} [V]	12.07	12.10	12.29	12.31
SoC_{stop} [%]	92.6	88.5	87.1	95.6
V_{stop} [V]	11.90	11.92	11.87	12.01
Slope [%SoC/V]	39.6	59.4	28.8	12.6

Temperature +40°C, Relative Humidity 15%				
Model	Bosch S3000		Fiamm L150P	
Condition	New	Used	New	Used
SoC_{start} [%]	99.5	99.3	99.0	99.3
V_{start} [V]	12.57	12.57	12.68	12.47
SoC_{stop} [%]	75.0	76.2	75.0	96.4
V_{stop} [V]	12.01	11.94	12.0	11.97
Slope [%SoC/V]	43.7	36.6	35.3	5.80

6.5 "Deep" campaign

Graphically speaking, it seems that all the measurements are computed punctually because of the lower sampling frequency in this campaign with respect to all the others and thus the lower number of points per time unit (see Figure 48 for further explanations).

As seen on "SoC" graphs, shortly after the test starts, a spike-shaped deviation from the trend of the internal resistance is appreciable on all the batteries. After this anomaly, the internal resistance seems to maintain the same value considering the same pattern. Near the end of the test, the internal resistance increases if the current requested is not greater than 30A and lasts more than 5 seconds (see pattern . Otherwise (see pattern 17 and 18), it remains almost constant.

The internal resistance computed during pattern 17 and 18 is lower than during the others and the dataset points are aligned with a lower slope. This happens due to the fact the battery voltage decreases more to balance the greater request of current.

The slope of the SoC is calculated starting after the spike-shaped anomaly, that lasts for a maximum of 1% of SoC, until the test ends.

Model	Bosch S3000	Energeco Clausum-S562.059.057	Fiamm L150P
Condition	New	New	New
SoC_{start} [%]	99.0	99.0	99.2
V_{start} [V]	12.67	12.56	12.50
SoC_{stop} [%]	79.6	89.2	79.7
V_{stop} [V]	11.93	12.27	12.07
Slope [%SoC/V]	57.8	33.4	45.4

Chapter 7.

Development of the device

Enough knowledge on lead acid batteries had been acquired to define a hardware architecture in which place the algorithm and compute some conversions. There were few constraints to be respected and some other conditions to fulfil.

The objective is still the same as the beginning: project a device to read some parameters and compute an algorithm to estimate the SoH. For what it was found, the parameters to read could be the voltage and the current on the battery, the algorithm could be computed as simple as possible in early versions (compute the SoC and visualize this information only) but very complex in future release (maybe integrating a machine learning system).

The device must acquire data from the battery, apply the best algorithm by means of updates for instance and store them in a database to let the machine learn. This could be possible by updating the firmware of the device using OTA (Over-The-Air) flashing but must be supported by the MCU and the device must have wireless communication, of course. Alternatively, a mobile application could be developed and easily kept updated within mobile store to apply the best algorithm available.

Building a mobile application is the easiest way to let the user visualize useful information and the updates are cost efficient with respect to the OTA firmware update that can be used to resolve bug or add features for which it is not enough to update the mobile application only.

7.1 System requirements

The hardware constraints were given by the initial assumption that the final product was intended for retail and must read the voltage of the battery:

- Compact
- Easy to use
- Easy to place
- Automotive grade components
- Analog input reading capability
- Low power consumption

- Direct supply from vehicle power network (≈12V)

Some subsequent conditions were imposed, others were laid down from personal experience and forward-looking perspective:

- Bluetooth communication
- Internal regulator of 3.3V
- Maximum available space of 4.5cm x 3.5cm
- Two layers PCB (Printed Circuit Board) with ground plane on bottom
- 2-pin connector for power supply
- Voltage and current protection (from spike and rush)
- 3.3V voltage regulator
- At least 3 fiducials points for automatic pick & place
- Spare space for further implementation of new features

It was decided that a 33% of maximum available space could be enough for the new features on the improved version; the key additions were:

- High resolution ADC
- Shunt resistor for very high currents (800A)
- Operational amplifiers for conditioning circuitry
- LIN (Local Interconnect Network) connector (for direct communication with ECU, Engine Control Unit)

Starting for the CPU (Central Processing Unit), a microcontroller with ARM architecture from a world-wide brand and a solid personal knowledge was chosen, the CC2640 from TI (Texas Instruments). It included a BT4.2 (Bluetooth version 4.2) interface compatible with BLE (Bluetooth Low Energy) protocol, two SSI (Synchronous Serial Interface) interface, an internal 12-bit ADC (Analog to Digital Converter), several GPIOs (General Purpose Input Output), internal DC-DC voltage regulator and internal temperature sensor. It was graded for automotive applications thanks to AEC-Q100 certification.

7.2 Design plan

The design of the hardware board of the device was divided in two part in which the second one is the successive development of the first. It was made to create a baseline in which, after designing it, printing it, mounting it, testing it and check it is alright, it can be established as working and the focus can be shifted to the next part to implement.

The schematic, footprints and 3D models for both versions "Basic" and "Basic Pro" are listed in the Appendix.

7.2.1 "Basic" version

The design started with a "Basic" version, a board in which the essential components were put on and linked properly to correctly read the voltage of the battery.

The "Basic" version included

- the MCU;
- a Bluetooth patch antenna;
- two crystal oscillators, one for real time clock, the other for communication clock;
- a guard system on the input in which the voltage must be read to protect from over voltage, over current and reverse current;
- a JTAG connector to flash MCU;
- a 2-pin connector to attach the battery on.

In this first prototype the power supply is given externally.

7.2.2 "Basic Pro" version

A second version was then designed that resumed the previous version and added the direct power supply from battery itself. As for the battery voltage analog input, a stronger protection must be inserted to safe the entire device from any unexpected power related behavior.

The idea for the schematic of supply and protection was taken from an existing device developed for automotive infotainment, *Remote Camera and Radar Expansion, SAT0074* [8]. Minor changes had been applied basically to fit in a smaller device that surely demands less current that an infotainment system. As it is, the components for

protection and power supply could be overestimated and a fine tuning to reduce cost ~~can~~ must be done.

This successor was labeled "Basic Pro": it relays to "Basic" version name but adds the tag "Pro" that stands for Protection. The "Basic Pro" version could be considered one of the final products, in terms of hardware development, to set the baseline for future improvements.

Chapter 8.

Conclusions

The designs of experiments were sufficiently effective to produce reliable results as the assumed trends were obtained, and what it was knew for the literature was confirmed once more.

The frequency analysis exposed that the lower the control frequency is, the more the resistive nature predominates the capacitive one.

The control frequency of 0.1Hz, chosen for the successive design, leaded to focus the attention on parameters easier to distinguish and acquire and, mostly, to reduce the complexity of all the acquisition systems.

Generally, variations of the internal resistance that diverge from the trend could be related to the chemistry inside the batteries. They are more appreciable on pristine and new batteries. Anyway, what is clearly visible from the graphs is a slow growth of the internal resistance followed by a sudden sharp fall that happens only once on the entire test and after the battery passed from one of the relaxing phases to the successive discharging one.

Analyzing a single battery during consecutive acquisitions test, this sharp fall phenomenon seems to decrease in intensity and occur almost at the same battery voltage.

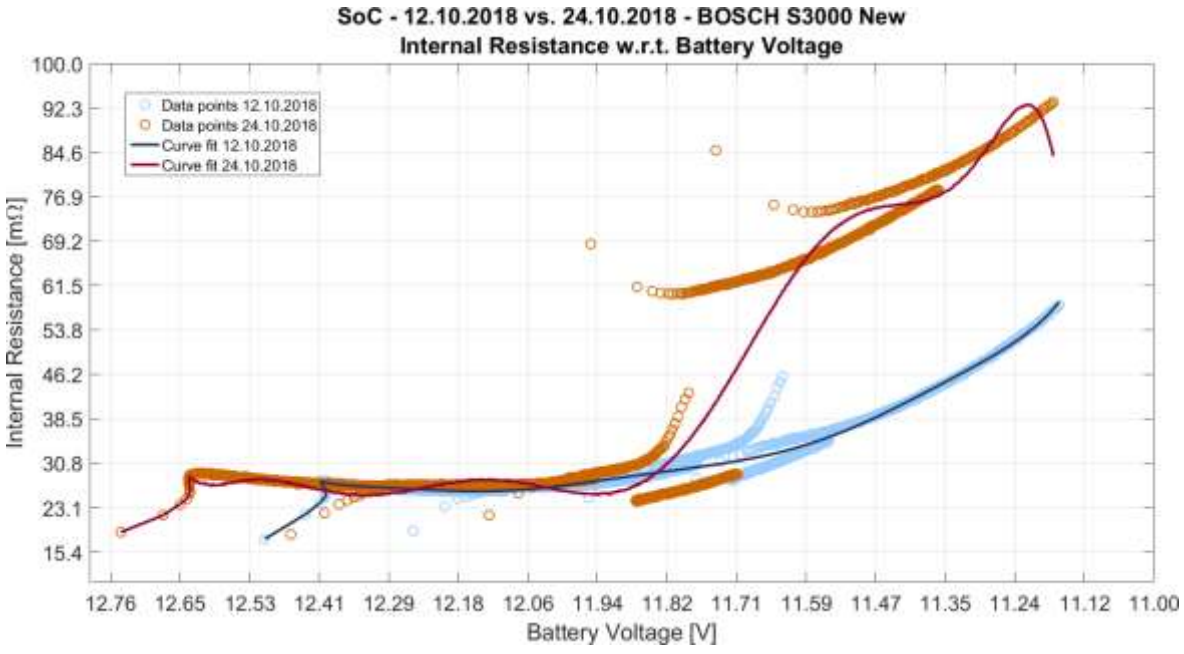


Figure 132. "SoC" New Bosch Internal Resistance w.r.t. Battery Voltage graphs of two different date datasets

The internal resistance alone is not a good direct indicator for SoH; it must be related to the open circuit voltage and to the model of the battery under test.

The Slope parameter computed in the Chapter 6 - Assessment of the results achieved, whose value indicates how much charge, in percentage, is able to provide at the expense of 1V decrement of the battery voltage, seems to be a good indicator for SoH. The Slope values are greater on those batteries that are in better condition.

The temperature seems to have dissimilar effects on batteries from different brands. Instead, the trend is maintained on the batteries from the same brand: on the Bosch the Slope increases over the entire temperature range; on the Fiamm it has slighter variations around a mean value.

The trend of SoC with respect to the battery voltage, a.k.a. the Slope parameter, is maintained between the "SoC" and "Deep" datasets of the same battery except for the Bosch that seems to behave better with pulses at higher current.

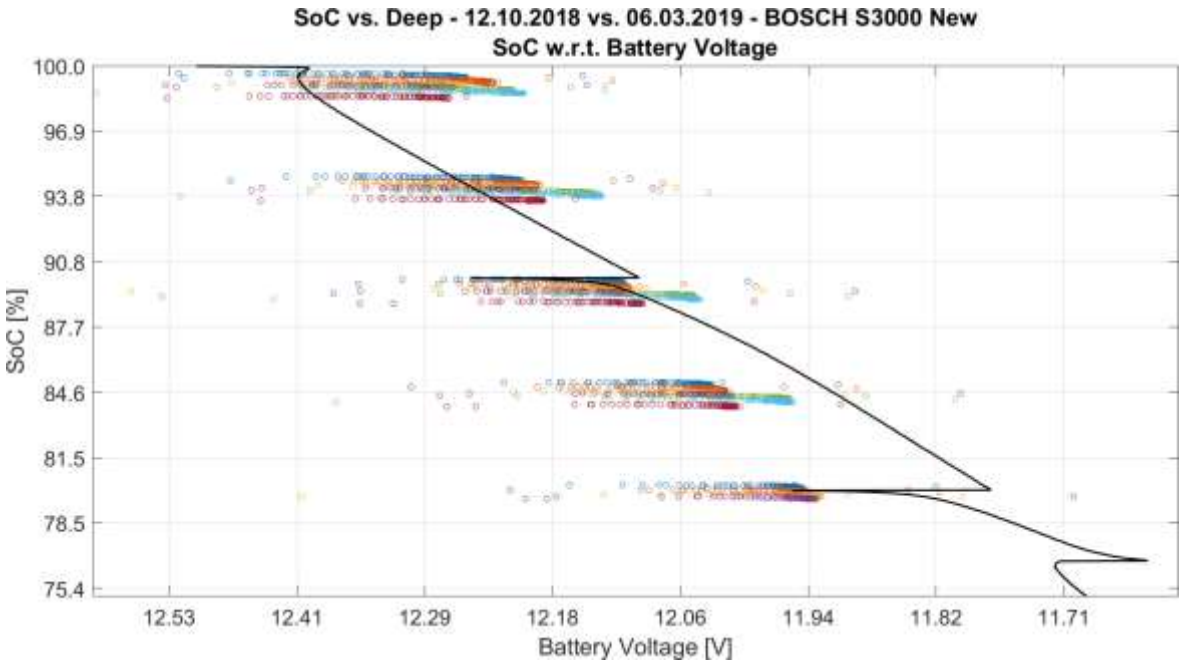


Figure 133. "SoC" and "Deep" New Bosch SoC w.r.t. Battery Voltage graphs of two different test datasets

Analyzing the knee voltage as a parameter that could be related to the SoH, it is noticeable that the lower the battery voltage is with respect to the V_{knee}, the more the battery is stressed and, accordingly, the SoH inevitably decreases. Moreover, the knee voltage is greater on whose batteries that are more aged, which is another evidence that SoH and V_{knee} are inversely proportional to each other.

During high discharge current (at 40A), the angle of the dataset points on a single pattern is greater when the slew rate of the current is lower, i.e. when the resistive

effect largely prevails the capacitive one, that is considered negligible. In this case, the Ohmic model (see 2.3 - Equivalent models for further explanations) best reflects the electric behavior of the battery, making the internal resistance a good parameter for SoH indication.

On the other hand, the angle is smaller when the slew rate is higher, i.e. when the capacitive effect is no longer negligible. In this opposite case, it's not the internal resistance to grow but the battery voltage to shrink. An analysis of the cranking of a vehicle engine could require the study and the calculation of the impedance of the battery (focusing on the internal capacitance), further complicating the DoE.

Chapter A

Appendix

A.1. Spectrum DoE: extended analysis for hardware design

A.1.1. MOSFET parasitic capacitances

A.1.2. Feedback resistor value: offset minimization

A.1.3. Op amp output impedance simulation

A.2. "DeepDischarge" code

A.2.1. MATLAB script

```
1. % Title: Connect to STM32F303K8 for communicating with an instrument.
2. % Author: di Simone Giuseppe
3. % Company: Politecnico di Torino
4. % Based on: Automatic VISA by MATLAB Team
5. % Description: Connect to STM32F303K8 and acquire current and voltage
   every
6. %              n-seconds (defined by the user) saving them in a .mat file
7.
8. clear variables;
9. close all;
10. clc;
11.
12. COMM_PERIOD = 0.05;
13. SLOW_COMM_PERIOD = 1;
14.
15. % Find a serial port object.
16. stm32 = instrfind('Type', 'serial', 'Port', 'COM12', 'Tag', '');
17.
18. % Create the serial port object if it does not exist
19. % otherwise use the object that was found.
20. if isempty(stm32)
21.     stm32 = serial('COM12', 'BaudRate', 921600);
22. else
23.     fclose(stm32);
24.     stm32 = stm32(1);
25. end
26.
27. % Connect to instrument object, stm32.
28. stm32Attempts = 0;
29. stm32MaxAttempts = 5;
30.
31. for stm32Attempts = 1 : stm32MaxAttempts
32.     try
33.         fopen(stm32);
34.         break;
35.     catch
36.         if(stm32Attempts == 5)
37.             err = msgbox('Exceeded attempts to connect to
   STM32F303K8','Error','error');
38.             return
39.         end
40.         uiwait(msgbox('STM32F303K8 not recognized by this PC. Try
   again','Warning','warn','modal'));
41.     end
42. end
43.
44. flushinput(stm32);
45.
46. % Communicating with instrument object, stm32.
47. endStringRx = 1;
48. exitCmd = 0;
49. overheatStringRx = 0;
50. noCurrentStringRx = 0;
51. vbattLowStringRx = 0;
52. stopStringRx = 0;
53. sendStopString = 0;
54. wait4reset = 3;
55. WorkingOnString = ['Discharge', '    Relax'];
```

```
56.
57. %while(~exitCmd && ~overheatStringRx && ~stopStringRx)
58. while(~exitCmd)
59.
60.     flushinput(stm32);
61.
62.     ackChar = '1';
63.
64.     stringInit = '';
65.     stringStart = '';
66.     stringFirst = '';
67.     stringEnd = '';
68.
69.     %% Instrument Configuration and Control
70.     if(endStringRx == 1)
71.
72.         listBattery = {'BOSCH S3000 - 40Ah','ENERGECO CLAUSUM-S562.059.057
    - 62Ah','FIAMM L044P - 44Ah',...
73.                        'FIAMM L150P - 50Ah'};
74.         listCapacity = {40, 62, 44, 50};
75.         listTimeConst = {1055000, 2043000, 1236000, 1504000};
76.         indexBattery = listdlg('PromptString','Select a Brand/Model of
    battery:',...
77.                                'SelectionMode','single',...
78.                                'ListString',listBattery);
79.         if isempty(indexBattery)
80.             return
81.         end
82.
83.         listCondition = {'New','Used','New+'};
84.         indexCondition = listdlg('PromptString','Select the condition of
    battery:',...
85.                                  'SelectionMode','single',...
86.                                  'ListString',listCondition);
87.         if isempty(indexCondition)
88.             return
89.         end
90.
91.         TimeConst = listTimeConst{1,indexBattery};
92.     else
93.         pause(wait4reset);
94.     end
95.
96.     endStringRx = 0;
97.     overheatStringRx = 0;
98.     noCurrentStringRx = 0;
99.     vbattLowStringRx = 0;
100.      commErrorStringRx = 0;
101.      stopStringRx = 0;
102.      sendStopString = 0;
103.      ackChar = '1';
104.      Invalid_Data_Counter = 0;
105.      Start = 0;
106.
107.      %pause(2);
108.      fprintf(stm32, 'S'); %Acknowledge for NUCLEO
109.
110.      while(isempty(stringInit))
111.          stringInit = fscanf(stm32);
112.      end
113.
114.      if strncmp(stringInit, 'Init', 4) %If I receive "Init error", end
    the execution
115.          commErrorStringRx = 1;
116.          fprintf(1,'%s',stringInit); %"Init error" from NUCLEO
```

```
117.      end
118.
119.      answer = questdlg('What to do next?', ...
120.          'Select Operation', ...
121.          'Start Test','Monitor','Exit','Exit');
122.      % Handle response
123.      switch answer
124.          case 'Start Test'
125.              charAction = 'T';
126.          case 'Monitor'
127.              charAction = 'M';
128.          case 'Exit'
129.              charAction = 'E';
130.      end
131.      if isempty(charAction)
132.          return
133.      end
134.
135.      pause(wait4reset); %wait for NUCLEO initialization
136.
137.      %% Pattern selection
138.      % Test mode
139.      if(charAction == 'T')
140.
141.          % Dialog box
142.          counter = 0;
143.          stopButton = 0;
144.          stopCounter = 0;
145.          Time = [];
146.
147.          measurementString = ['Test Start:';'           ';...
148.                               'Test Time: ';'           ';...
149.                               'SoC level: ';'           ';...
150.                               'Working on:';'           ';...
151.                               'Vbattery:  ';'           ';...
152.                               'Ibattery:  ';'           ';...
153.                               'Ich1:      ';'           ';...
154.                               'Ich2:      ';'           ';...
155.                               'Temp1:     ';'           ';...
156.                               'Temp2:     '];
157.
158.          measurementUnitsString = ['  ';'  ';...
159.                                    ' s';'  ';...
160.                                    ' %';'  ';...
161.                                    '  ';'  ';...
162.                                    ' V';'  ';...
163.                                    ' A';'  ';...
164.                                    ' A';'  ';...
165.                                    ' A';'  ';...
166.                                    '°C';'  ';...
167.                                    '°C'];
168.
169.          c = dialog('Position',[736 305 250 320],'Name','Real-Time
    acquisition');
170.          movegui(c,'center');
171.
172.          %txt = uicontrol('Parent',c,...
173.          %                'Style','text',...
174.          %                'Position',[20 40 210 40],...
175.          %                'String','Click the Stop button to end the
    acquisition');
176.
177.          %btn = uicontrol('Parent',c,...
178.          %                'Position',[85 15 70 25],...
179.          %                'String','Close',...
```

```
180.          %                'Callback','stopButton=1;delete(c);');
181.
182.          measurementstxt = uicontrol('Parent',c,...
183.                                      'Style','text',...
184.                                      'HorizontalAlignment', 'left',...
185.                                      'Position',[28 20 60 280],...
186.                                      'String', {measurementString});
187.
188.          measurementunitstxt = uicontrol('Parent',c,...
189.                                      'Style','text',...
190.                                      'HorizontalAlignment', 'right',...
191.                                      'Position',[202 20 20 280],...
192.                                      'String', {measurementUnitsString});
193.
194.          numElements = max(size(measurementString));
195.
196.          Test_Time = [];
197.          Vbatt = [];
198.          Ibatt = [];
199.          Ich1 = [];
200.          Ich2 = [];
201.          Temp1 = [];
202.          Temp2 = [];
203.
204.          updateIndex = 1;
205.          NFile = 0;
206.                   Soc_Level = 100;
207.          start_test = 1;
208.          sel_pattern = 0;
209.
210.          fprintf(stm32, 'R'); %send NUCLEO reset command
211.          stringInit = '';
212.          while(isempty(stringInit))
213.              stringInit = fscanf(stm32);
214.          end
215.          fprintf(1,'%s',stringInit);%"Reset received\n"
216.
217.        while(~(NFile >= 5 && sel_pattern == 2) && ~overheatStringRx &&
    ~stopStringRx && ~noCurrentStringRx && ~vbattLowStringRx &&
    ~commErrorStringRx)
218.
219.          updateIndex = 1;
220.
221.          if(sel_pattern == 1)
222.              sel_pattern = 2;
223.          else
224.              sel_pattern = 1;
225.              NFile = NFile + 1;
226.              Test_Time = [];
227.              Vbatt = [];
228.              Ibatt = [];
229.              Ich1 = [];
230.              Ich2 = [];
231.              Temp1 = [];
232.              Temp2 = [];
233.                           Soc_Level = 100 - ((NFile-1)*5);
234.          end
235.
236.          pause(wait4reset); %wait for NUCLEO initialization
237.          fprintf(stm32, 'S'); %Acknowledge for NUCLEO
238.          stringInit = '';
239.          while(isempty(stringInit))
240.              stringInit = fscanf(stm32);
241.          end
```

```
242.          fprintf(1,'%s',stringInit); %"Select what to do next:" from
    NUCLEO
243.
244.          stringStart = '';
245.          fprintf(stm32,'%d\n', sel_pattern); %Send pattern 1 character to
    NUCLEO
246.          while(isempty(stringStart))
247.              stringStart = fscanf(stm32);
248.          end
249.          fprintf(1,'%s',stringStart); %"Start pattern dicharge\n"
250.
251.          if(sel_pattern == 1)
252.              fprintf(stm32,'%d\n', indexBattery); %Send time constant
253.
254.              stringStart = '';
255.              while(isempty(stringStart))
256.                  stringStart = fscanf(stm32);
257.              end
258.              if strncmp(stringStart, 'Index', 5) %If I receive "End",
    update the counter endStringRx
259.                  commErrorStringRx = 1;
260.                  fprintf(1,'%s', stringStart);
261.              end
262.          end
263.
264.          % Pattern 'sel_pattern'
265.          while(~endStringRx && ~overheatStringRx && ~stopStringRx &&
    ~noCurrentStringRx && ~vbattLowStringRx && ~commErrorStringRx)
266.              stringEnd = '';
267.              while(isempty(stringEnd))
268.                  stringEnd = fscanf(stm32);
269.              end
270.              %fprintf(1,'%s', stringEnd);
271.
272.              if ~ishandle(c)
273.                  sendStopString = 1; %If user close dialog box
274.              end
275.
276.              if strncmp(stringEnd, 'End', 3) %If I receive "End", update
    the counter endStringRx
277.                  endStringRx = 1;
278.                  fprintf(1,'%s', stringEnd);
279.                  fprintf(stm32, '1'); % Acknowledge
280.              elseif strncmp(stringEnd, 'Overheating', 11)
281.                  overheatStringRx = 1;
282.                  err = msgbox(stringEnd,'Error','error');
283.                  waitfor(err);
284.                  fprintf(stm32, '1'); % Acknowledge
285.              elseif strncmp(stringEnd, 'No current', 10)
286.                  noCurrentStringRx = 1;
287.                  err = msgbox(stringEnd,'Error','error');
288.                  waitfor(err);
289.                  fprintf(stm32, '1'); % Acknowledge
290.              elseif strncmp(stringEnd, 'Vbatt too', 9)
291.                  vbattLowStringRx = 1;
292.                  err = msgbox(stringEnd,'Error','error');
293.                  waitfor(err);
294.                  fprintf(stm32, '1'); % Acknowledge
295.              else
296.                  dataString = strsplit(stringEnd, ', ');
297.
298.                  if(start_test == 1)
299.                      start_test = 0;
300.                      Start = now;
301.                  end
```

```
302.
303.                 if(max(size(dataString)) >= 5)
304.                     Test_Time_Temp = diff([Start, now])*(24*60*60);
305.                     Vbatt_Temp =
    (str2double(dataString{1,4})*(3.340/4095))/0.235849 - 0.016;
306.                     Ich1_Temp =
    (str2double(dataString{1,5})*(3.340/4095))/0.1;           %All 4 MOS
    active Ich1 = (Vres_ch1/(4*0.1))*4
307.                     Ich2_Temp =
    (str2double(dataString{1,3})*(3.340/4095))/0.1;           %All 4 MOS
    active Ich2 = (Vres_ch2/(4*0.1))*4
308.                     Temp1_Temp =
    (str2double(dataString{1,1})*(3.340/4095)-0.376)*51.28;
309.                     Temp2_Temp =
    (str2double(dataString{1,2})*(3.340/4095)-0.384)*51.28;
310.                     Ibatt_Temp = Ich1_Temp + Ich2_Temp;
311.
312.                     Test_Time = [Test_Time; Test_Time_Temp];
313.                     Vbatt = [Vbatt; Vbatt_Temp];
314.                     Ibatt = [Ibatt; Ibatt_Temp];
315.                     Ich1 = [Ich1; Ich1_Temp];
316.                     Ich2 = [Ich2; Ich2_Temp];
317.                     Temp1 = [Temp1; Temp1_Temp];
318.                     Temp2 = [Temp2; Temp2_Temp];
319.                 else
320.                     Invalid_Data_Counter = Invalid_Data_Counter + 1;
321.                 end
322.
323.                 if(updateIndex == (SLOW_COMM_PERIOD/COMM_PERIOD))
324.                     updateIndex = 1;
325.
326.                     if (exist('measurestxt', 'var') == 1)
327.                         delete(measurestxt);
328.                     end
329.
330.                     if(sendStopString == 1)
331.                         ackChar = '0';
332.                         %stopStringRx = 1;
333.                     else
334.                         ackChar = '1';
335.                         %stopStringRx = 0;
336.                         measurestxt = uicontrol('Parent',c,...
337.                                                 'Style','text',...
338.                                                 'HorizontalAlignment',
    'right',...
339.                                                 'Position',[85 20 120
    280],...
340.                                                 'String',
    {datestr(Start);' ';...
341.
    num2str(Test_Time_Temp,'   %3.3f');' ';...
342.
    num2str(Soc_Level,'    %d');' ';...
343.
    WorkingOnString((9*(sel_pattern-1)+1):(9*(sel_pattern-1)+9));' ';...
344.
    num2str(Vbatt_Temp,'   %3.3f');' ';...
345.
    num2str(Ibatt_Temp,'   %3.3f');' ';...
346.
    num2str(Ich1_Temp,'   %3.3f');' ';...
347.
    num2str(Ich2_Temp,'   %3.3f');' ';...
348.
    num2str(Temp1_Temp,'     %3.1f');' ';...
```

```
349.
   num2str(Temp2_Temp,'     %3.1f')});
350.                        end
351.                        %fprintf(stm32, ackChar); % Acknowledge
352.                    else
353.                        updateIndex = updateIndex+1;
354.                    end
355.                    pause(COMM_PERIOD/10000000);
356.                end
357.            end
358.
359.            endStringRx = 0;
360.
361.            % Saving the acquired data in the corresponding file
362.            if(NFile > 0 && (sel_pattern == 2 || (sel_pattern == 1 &&
   (overheatStringRx || stopStringRx || noCurrentStringRx ||
   vbattLowStringRx))))
363.                if(NFile == 1)
364.                    selectedBattery = listBattery{1,indexBattery};
365.                    subFolders = strsplit(selectedBattery,' ');
366.                    Brand = subFolders{1,1};
367.                    Model = subFolders{1,2};
368.                    Capacity = listCapacity{1,indexBattery};
369.                    batteryPath = [Brand,'\',Model];
370.                    DateDirCreation = datestr(Start,'yyyy.mm.dd HH.MM.ss ');
371.                    finalFolder = [DateDirCreation,'- Deep (Var Discharge -
   Const Discharge - Wait, Step = 5%)'];
372.                    Condition = listCondition{1,indexCondition};
373.                    ExtendedPath = [pwd, '\', batteryPath,'\',finalFolder];
374.                    mkdir(ExtendedPath);
375.                end
376.
377.                FileName = ['Deep-',subFolders{1,1},'-',subFolders{1,2},'-
   ',Condition,'_', sprintf('%02d', NFile),'.mat'];
378.                ExtendedFileName = [ExtendedPath,'\',FileName];
379.                StartTest = Start;
380.                Time = [];
381.                Time = Test_Time;
382.                SamplePeriod = Time(2)-Time(1);
383.
384.                if exist(ExtendedFileName, 'file')
385.                    delete(ExtendedFileName);
386.                end
387.
388.                            % Cancel wrong sample in Temp2 occurring after
   reset
389.                            k = 0;
390.                            for k=1:(size(Temp2)-1)
391.                                    if Temp2(k) < 0
392.                                            Temp2(k) = Temp2(k+1);
393.                                    end
394.                            end
395.
396.                save(ExtendedFileName,
   'Time','Vbatt','Ibatt','Ich1','Ich2','Temp1',...
397.
   'Temp2','StartTest','Brand','Model',...
398.
   'Condition','DateDirCreation','FileName','SamplePeriod');
399.
400.            end
401.
402.            if(stopStringRx)
403.                while ~strncmp(fscanf(stm32), 'Stop', 4)
404.                end
```

```
405.          elseif ((ishandle(c) && NFile == 5 && sel_pattern == 2) ||
    (overheatStringRx || noCurrentStringRx || vbattLowStringRx ||
    commErrorStringRx))
406.              delete(c);
407.          end
408.          flushinput(stm32);
409.        end
410.
411.      % Monitor mode
412.      elseif(charAction == double('M'))
413.
414.          % Dialog box
415.          counter = 0;
416.          stopButton = 0;
417.          stopCounter = 0;
418.          %Time = [];
419.
420.          measurementString = ['Local Time:';'            ';...
421.                               'Test Time: ';'            ';...
422.                               'Vbattery:  ';'            ';...
423.                               'Ibattery:  ';'            ';...
424.                               'Ich1:      ';'            ';...
425.                               'Ich2:      ';'            ';...
426.                               'Temp1:     ';'            ';...
427.                               'Temp2:     '];
428.
429.          measurementUnitsString = ['  ';'  ';...
430.                                    ' s';'  ';...
431.                                    ' V';'  ';...
432.                                    ' A';'  ';...
433.                                    ' A';'  ';...
434.                                    ' A';'  ';...
435.                                    '°C';'  ';...
436.                                    '°C'];
437.
438.          d = dialog('Position',[736 305 250 320],'Name','Real-Time
    monitoring');
439.          movegui(d,'center');
440.
441.          txt = uicontrol('Parent',d,...
442.                          'Style','text',...
443.                          'Position',[20 40 210 40],...
444.                          'String','Click the Stop button to end the
    monitor');
445.
446.          btn = uicontrol('Parent',d,...
447.                          'Position',[85 15 70 25],...
448.                          'String','Stop',...
449.                          'Callback','stopButton=1;delete(gcf);');
450.
451.          measurementstxt = uicontrol('Parent',d,...
452.                                      'Style','text',...
453.                                      'HorizontalAlignment', 'left',...
454.                                      'Position',[28 80 60 220],...
455.                                      'String', {measurementString});
456.
457.          measurementunitstxt = uicontrol('Parent',d,...
458.                                      'Style','text',...
459.                                      'HorizontalAlignment', 'right',...
460.                                      'Position',[202 80 20 220],...
461.                                      'String', {measurementUnitsString});
462.
463.          numElements = max(size(measurementString));
464.
465.          Test_Time = [];
```

```
466.          Vbatt = [];
467.          Ibatt = [];
468.          Ich1 = [];
469.          Ich2 = [];
470.          Temp1 = [];
471.          Temp2 = [];
472.
473.          updateIndex = 1;
474.          start_test = 1;
475.
476.          fprintf(stm32, 'M');
477.          stringStart = '';
478.          while(isempty(stringStart))
479.             stringStart = fscanf(stm32);
480.          end
481.          fprintf(1,'%s',stringStart); %"Monitor mode\n" from NUCLEO
482.
483.          while(~stopStringRx) % Button pressed on the dialogbox
    (ButtonPressedFunction)
484.              stringEnd = '';
485.              while(isempty(stringEnd))
486.                  stringEnd = fscanf(stm32);
487.              end
488.              %fprintf(1,'%s', stringEnd);
489.
490.              if ~ishandle(d)
491.                  sendStopString = 1;
492.              end
493.
494.              dataString = strsplit(stringEnd, ', ');
495.                           if(start_test == 1)
496.                                   start_test = 0;
497.                                   Start = now;
498.              end
499.
500.              if(max(size(dataString)) >= 5)
501.                  Test_Time_Temp = diff([Start, now])*(24*60*60);
502.                  Vbatt_Temp =
    (str2double(dataString{1,4})*(3.340/4095))/0.235849 - 0.016;
503.                  Ich1_Temp =
    (str2double(dataString{1,5})*(3.340/4095))/0.1;           %All 4 MOS
    active Ich1 = (Vres_ch1/(4*0.1))*4
504.                  Ich2_Temp =
    (str2double(dataString{1,3})*(3.340/4095))/0.1;           %All 4 MOS
    active Ich2 = (Vres_ch2/(4*0.1))*4
505.                  Temp1_Temp = (str2double(dataString{1,1})*(3.340/4095)-
    0.376)*51.28;
506.                  Temp2_Temp = (str2double(dataString{1,2})*(3.340/4095)-
    0.384)*51.28;
507.                  Ibatt_Temp = Ich1_Temp + Ich2_Temp;
508.
509.                  Test_Time = [Test_Time; Test_Time_Temp];
510.                  Vbatt = [Vbatt; Vbatt_Temp];
511.                  Ibatt = [Ibatt; Ibatt_Temp];
512.                  Ich1 = [Ich1; Ich1_Temp];
513.                  Ich2 = [Ich2; Ich2_Temp];
514.                  Temp1 = [Temp1; Temp1_Temp];
515.                  Temp2 = [Temp2; Temp2_Temp];
516.              else
517.                  Invalid_Data_Counter = Invalid_Data_Counter + 1;
518.              end
519.
520.              if(updateIndex == SLOW_COMM_PERIOD/COMM_PERIOD)
521.
522.                  if (exist('measurestxt', 'var') == 1)
```

```
523.                     delete(measurestxt);
524.                 end
525.
526.                 if(sendStopString == 1)
527.                     ackChar = '0';
528.                     stopStringRx = 1;
529.                 else
530.                     ackChar = '1';
531.                     stopStringRx = 0;
532.                     updateIndex = 1;
533.                     measurestxt = uicontrol('Parent',d,...
534.                                             'Style','text',...
535.                                             'HorizontalAlignment',
    'right',...
536.                                             'Position',[85 80 120
    220],...
537.                                             'String', {datestr(Start);'
    ';...
538.
    num2str(Test_Time_Temp,'   %3.3f');' ';...
539.
    num2str(Vbatt_Temp,'   %3.3f');' ';...
540.
    num2str(Ibatt_Temp,'   %3.3f');' ';...
541.
    num2str(Ich1_Temp,'   %3.3f');' ';...
542.
    num2str(Ich2_Temp,'   %3.3f');' ';...
543.
    num2str(Temp1_Temp,'     %3.1f');' ';...
544.
    num2str(Temp2_Temp,'     %3.1f')});
545.                 end
546.                 fprintf(stm32, ackChar); % Acknowledge
547.             else
548.                 updateIndex = updateIndex+1;
549.             end
550.             pause(COMM_PERIOD/10000000);
551.         end
552.
553.         if(stopStringRx)
554.             while ~strncmp(fscanf(stm32), 'Stop', 4)
555.             end
556. %             stringEnd = fscanf(stm32);
557. %             fprintf(1,'%s', stringEnd);
558. %             stringEnd = fscanf(stm32);
559. %             fprintf(1,'%s', stringEnd);
560. %             stringEnd = strcat(stringEnd);
561. %             fprintf(1,'%s', stringEnd);
562.         end
563.
564.     % End execution (no error)
565.     elseif(charAction == double('E'))
566.         fprintf(stm32, 'E');
567.         stringEnd = '';
568.         while(isempty(stringEnd))
569.            stringEnd = fscanf(stm32);
570.         end
571.         fprintf(1,'%s',stringEnd);
572.         exitCmd = 1;
573.     else
574.         fprintf(stm32, 'E');
575.         stringEnd = '';
576.         while(isempty(stringEnd))
577.             stringEnd = fscanf(stm32);
```

```
578.         end
579.         fprintf(1,'%s\n',stringEnd);
580.         exitCmd = 1;
581.     end
582.
583.     flushinput(stm32);
584. end
585.
586. % Disconnect from instrument object, stm32.
587. fclose(stm32);
588.
589. % Clean up all objects.
590. delete(stm32);
591.
592. %% Plot useful graphs
593.
594. figure
595. p1 = subplot(4,1,1);
596. plot(Test_Time, Vbatt)
597. ylabel(p1, 'Vbatt [V]');
598.
599. p2 = subplot(4,1,2);
600. plot(Test_Time, Ibatt)
601. ylabel(p2, 'Ibatt [A]');
602.
603. p3 = subplot(4,1,3);
604. ax3 = plotyy(Test_Time, Ich1, Test_Time, Ich2, 'plot');
605. ylabel(ax3(1), 'Ich1 [A]');
606. ylabel(ax3(2), 'Ich2 [A]');
607.
608. p4 = subplot(4,1,4);
609. ax4 = plotyy(Test_Time, Temp1, Test_Time, Temp2, 'plot');
610. ylabel(ax4(1), 'Tch1 [°C]');
611. ylabel(ax4(2), 'Tch2 [°C]');
612. xlabel(p4, 'Time [s]');
613.
614. % figure
615. % p1 = subplot(4,1,1);
616. % plot(Time, Vbatt)
617. % ylabel(p1, 'Vbatt [V]');
618. %
619. % p2 = subplot(4,1,2);
620. % plot(Time, Ibatt)
621. % ylabel(p2, 'Ibatt [A]');
622. %
623. % p3 = subplot(4,1,3);
624. % ax3 = plotyy(Time, Ich1, Time, Ich2, 'plot');
625. % ylabel(ax3(1), 'Ich1 [A]');
626. % ylabel(ax3(2), 'Ich2 [A]');
627. %
628. % p4 = subplot(4,1,4);
629. % ax4 = plotyy(Time, Temp1, Time, Temp2, 'plot');
630. % ylabel(ax4(1), 'Tch1 [°C]');
631. % ylabel(ax4(2), 'Tch2 [°C]');
632. % xlabel(p4, 'Time [s]');
```

A.2.2. STM32 firmware

A.2.2.1. main.c

```
#include "patterns.h"

// Global variables
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
DAC_HandleTypeDef hdac1;
DAC_HandleTypeDef hdac2;

Serial gui(D1, D0, BAUDRATE);

// ADC Clock Configuration
void SystemClock_Config(void)
{
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC12;
  PeriphClkInit.Adc12ClockSelection = RCC_ADC12PLLCLK_DIV16; // 64MHz/16 = 4MHz
  while(HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK);
}

// ADC1 Initialization Function
static void MX_ADC1_Init(void)
{
    ADC_MultiModeTypeDef multimode = {0};
    ADC_InjectionConfTypeDef sConfigInjected = {0};

    // Common configuration
    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
    hadc1.Init.ContinuousConvMode = DISABLE;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion = 1;
    hadc1.Init.DMAContinuousRequests = DISABLE;
    hadc1.Init.EOCSelection = ADC_EOC_SEQ_CONV;
    hadc1.Init.LowPowerAutoWait = DISABLE;
    hadc1.Init.Overrun = ADC_OVR_DATA_OVERWRITTEN;
    while(HAL_ADC_Init(&hadc1) != HAL_OK);

    // Configure the ADC multi-mode
    multimode.Mode = ADC_DUALMODE_INJECSIMULT;
    multimode.TwoSamplingDelay = ADC_TWOSAMPLINGDELAY_10CYCLES;
    while(HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) != HAL_OK);

    // Configure Injected Channel 4 = PA3 (Vtemp1)
    sConfigInjected.InjectedChannel = ADC_CHANNEL_4;
    sConfigInjected.InjectedRank = ADC_INJECTED_RANK_1;
    sConfigInjected.InjectedSingleDiff = ADC_SINGLE_ENDED;
    sConfigInjected.InjectedNbrOfConversion = 3;
    sConfigInjected.InjectedSamplingTime = ADC_SAMPLETIME_601CYCLES_5;
    sConfigInjected.ExternalTrigInjecConvEdge = ADC_EXTERNALTRIGINJECCONV_EDGE_NONE;
    sConfigInjected.ExternalTrigInjecConv = ADC_INJECTED_SOFTWARE_START;
    sConfigInjected.AutoInjectedConv = DISABLE;
    sConfigInjected.InjectedDiscontinuousConvMode = DISABLE;
    sConfigInjected.QueueInjectedContext = DISABLE;
```

```
58.     sConfigInjected.InjectedOffset = 0;
59.     sConfigInjected.InjectedOffsetNumber = ADC_OFFSET_NONE;
60.     while(HAL_ADCEx_InjectedConfigChannel(&hadc1, &sConfigInjected) !=
    HAL_OK);
61.
62.     // Configure Channel 2 = PA1 (Vre2)
63.     sConfigInjected.InjectedChannel = ADC_CHANNEL_2;
64.     sConfigInjected.InjectedRank = ADC_INJECTED_RANK_2;
65.     sConfigInjected.InjectedSamplingTime = ADC_SAMPLETIME_601CYCLES_5;
66.     while(HAL_ADCEx_InjectedConfigChannel(&hadc1, &sConfigInjected) !=
    HAL_OK);
67.
68.     // Configure Channel 1 = PA0 (Vres1)
69.     sConfigInjected.InjectedChannel = ADC_CHANNEL_1;
70.     sConfigInjected.InjectedRank = ADC_INJECTED_RANK_3;
71.     sConfigInjected.InjectedSamplingTime = ADC_SAMPLETIME_601CYCLES_5;
72.     while(HAL_ADCEx_InjectedConfigChannel(&hadc1, &sConfigInjected) !=
    HAL_OK);
73. }
74.
75. //ADC2 Initialization Function
76. static void MX_ADC2_Init(void)
77. {
78.     ADC_InjectionConfTypeDef sConfigInjected = {0};
79.
80.     // Common configuration
81.     hadc2.Instance = ADC2;
82.     hadc2.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
83.     hadc2.Init.Resolution = ADC_RESOLUTION_12B;
84.     hadc2.Init.ScanConvMode = ADC_SCAN_ENABLE;
85.     hadc2.Init.ContinuousConvMode = DISABLE;
86.     hadc2.Init.DiscontinuousConvMode = DISABLE;
87.     hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
88.     hadc2.Init.NbrOfConversion = 1;
89.     hadc2.Init.DMAContinuousRequests = DISABLE;
90.     hadc2.Init.EOCSelection = ADC_EOC_SEQ_CONV;
91.     hadc2.Init.LowPowerAutoWait = DISABLE;
92.     hadc2.Init.Overrun = ADC_OVR_DATA_OVERWRITTEN;
93.     while(HAL_ADC_Init(&hadc2) != HAL_OK);
94.
95.     // Configure Channel 4 = PA7 (Vbatt)
96.     sConfigInjected.InjectedChannel = ADC_CHANNEL_4;
97.     sConfigInjected.InjectedRank = ADC_INJECTED_RANK_1;
98.     sConfigInjected.InjectedSingleDiff = ADC_SINGLE_ENDED;
99.     sConfigInjected.InjectedNbrOfConversion = 2;
100.      sConfigInjected.InjectedSamplingTime = ADC_SAMPLETIME_19CYCLES_5;
101.      sConfigInjected.AutoInjectedConv = DISABLE;
102.      sConfigInjected.InjectedDiscontinuousConvMode = DISABLE;
103.      sConfigInjected.QueueInjectedContext = DISABLE;
104.      sConfigInjected.InjectedOffset = 0;
105.      sConfigInjected.InjectedOffsetNumber = ADC_OFFSET_NONE;
106.      while(HAL_ADCEx_InjectedConfigChannel(&hadc2, &sConfigInjected) !=
    HAL_OK);
107.
108.      // Configure Channel 1 = PA4 (Vtemp2)
109.      sConfigInjected.InjectedChannel = ADC_CHANNEL_1;
110.      sConfigInjected.InjectedRank = ADC_INJECTED_RANK_2;
111.      sConfigInjected.InjectedSamplingTime = ADC_SAMPLETIME_601CYCLES_5;
112.      while(HAL_ADCEx_InjectedConfigChannel(&hadc2, &sConfigInjected) !=
    HAL_OK);
113. }
114.
115. // DAC1 Initialization Function
116. static void MX_DAC1_Init(void)
117. {
```

```
118.    DAC_ChannelConfTypeDef sConfig = {0};
119.
120.    // DAC Initialization
121.    hdac1.Instance = DAC1;
122.    while(HAL_DAC_Init(&hdac1) != HAL_OK);
123.
124.    // DAC channel OUT2 configuration
125.    sConfig.DAC_Trigger = DAC_TRIGGER_SOFTWARE;
126.    sConfig.DAC_OutputSwitch = DAC_OUTPUTSWITCH_ENABLE;
127.    while(HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_2) !=
    HAL_OK);
128. }
129.
130. // DAC2 Initialization Function
131. static void MX_DAC2_Init(void)
132. {
133.    DAC_ChannelConfTypeDef sConfig = {0};
134.
135.    // DAC Initialization
136.    hdac2.Instance = DAC2;
137.    while(HAL_DAC_Init(&hdac2) != HAL_OK);
138.
139.    // DAC channel OUT1 config
140.    sConfig.DAC_Trigger = DAC_TRIGGER_SOFTWARE;
141.    sConfig.DAC_OutputSwitch = DAC_OUTPUTSWITCH_ENABLE;
142.    while(HAL_DAC_ConfigChannel(&hdac2, &sConfig, DAC_CHANNEL_1) !=
    HAL_OK);
143. }
144.
145. // GPIO Initialization
146. static void MX_GPIO_Init(void)
147. {
148.    GPIO_InitTypeDef GPIO_InitStruct = {0};
149.
150.    // Configure GPIO pins : PA8 PA11
151.    GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_11;
152.    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
153.    GPIO_InitStruct.Pull = GPIO_NOPULL;
154.    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
155. }
156.
157. // Main
158. int main()
159. {
160.     char buffer = '0';
161.     char pattern = '0';
162.     unsigned int time_index = '0';
163.     unsigned int time_const = 0;
164.
165.     // Pheripheral Initialization
166.     SystemClock_Config();
167.     MX_GPIO_Init();
168.     MX_ADC1_Init(); // Initialize ADC1
169.     while(HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED) !=
    HAL_OK); // Calibrate AD converter 1
170.     MX_ADC2_Init(); // Initialize ADC2
171.     while(HAL_ADCEx_Calibration_Start(&hadc2, ADC_SINGLE_ENDED) !=
    HAL_OK); // Calibrate AD converter 2
172.     MX_DAC1_Init(); // Initialize DAC1
173.     MX_DAC2_Init(); // Initialize DAC2
174.
175.     buffer = gui.getc();
176.     gui.getc();
177.
178.     if(buffer != 'S'){
```

```
179.          gui.printf("Init error\n");
180.          wait(1);
181.          HAL_DeInit();
182.          HAL_NVIC_SystemReset();
183.          NVIC_SystemReset();
184.      }
185.
186.      gui.printf("Select what to do next\n");
187.
188.      buffer = gui.getc();
189.      gui.getc();
190.      pattern = buffer;
191.
192.      switch (pattern){
193.          case '1':
194.              gui.printf("Start pattern dicharge\n");
195.              buffer = gui.getc();
196.              gui.getc();
197.              time_index = buffer - '0';
198.              if(time_index > 0) {
199.                  gui.printf("OK\n");
200.                  time_const = time_const_list[time_index];
201.                  pattern1(time_const);
202.              }
203.              else {
204.                  gui.printf("Index not valid\n");
205.              }
206.              wait(1);
207.              HAL_DeInit();
208.              HAL_NVIC_SystemReset();
209.              NVIC_SystemReset();
210.          break;
211.
212.          case '2':
213.              gui.printf("Start 1 hour relax\n");
214.              pattern2();
215.              wait(1);
216.              HAL_DeInit();
217.              HAL_NVIC_SystemReset();
218.              NVIC_SystemReset();
219.          break;
220.
221.          case 'M': // Monitor - Debug mode
222.              gui.printf("Monitor mode\n");
223.              monitor_mode();
224.              while(1);
225.          break;
226.
227.          case 'E':
228.              gui.printf("Goodbye!\n");
229.              wait(1);
230.              HAL_DeInit();
231.              HAL_NVIC_SystemReset();
232.              NVIC_SystemReset();
233.          break;
234.
235.          case 'R':
236.              gui.printf("Reset received\n");
237.              wait(1);
238.              HAL_DeInit();
239.              HAL_NVIC_SystemReset();
240.              NVIC_SystemReset();
241.          break;
242.
243.          default:
```

```
244.             gui.printf("The selected action is not valid\n");
245.             wait(1);
246.             HAL_DeInit();
247.             HAL_NVIC_SystemReset();
248.             NVIC_SystemReset();
249.         break;
250.     }
251. }
```

A.2.2.2. patterns.h

```
#ifndef PATTERNS_H

    #define PATTERNS_H
    #define USE_TIMEOUT 0

    #include "mbed.h"

// Constant variables

 /* // DEBUG CONSTANTS
    const unsigned int time_const_list[] = {0, 1000, 2000, 2500, 3000}; // N.A., 40, 62, 44, 50 Ah
    const unsigned int STARTTIME    =      1000;
    const unsigned int RELAXTIME    =       100;
    const unsigned int RELAXTIME2   =      2000;
    const unsigned int WAITTIME     =     10000;
    const int POWERTIME1            =        50;
    const int POWERTIME2            =       100;
    const int LOWBATT_HEXVOLT       =         0;
*/

    const unsigned int time_const_list[] = {0, 1055000, 2043000, 1236000, 1504000}; // N.A., 40, 62, 44, 50 Ah
    const unsigned int STARTTIME    =     15000; //   15 s
    const unsigned int RELAXTIME    =     30000; //   30 s
    const unsigned int RELAXTIME2   =    600000; //   10 min
    const unsigned int WAITTIME     =   3000000; //   50 min
    const int POWERTIME1            =        50; //   0.05 s
    const int POWERTIME2            =      5000; //    5 s
    const int LOWBATT_HEXVOLT       =      3122; // (10.8V * 0.235849 * 4095)/3.3V = 3122

    const int POWEROFF              =         0; //    0 mV ->    0 A
    const int POWER_LOW             =       810; //  810 mV ->   10 A
    const int POWER_MED             =      2430; // 2430 mV ->   30 A
    const int POWER_HIGH            =      3240; // 3240 mV ->   40 A
    const int POWER_CONST           =       340; //  324 mV ->    4 A

    const int OVERHEAT_TEMP         =        55; // °C
    const int OVERHEAT_HEXTEMP      =     29243; // 0xFFFF = 3.3V; 2.85V = 125°C , 0.2V = -10°C , 75°C = 1.85V = 0x8F84 = 36988, 50°C = 27307, 55°C = 29243
    const float MIN_FLOATVCTRL      =       0.1; // 0.1f = 10% * Vref = 10% * 3.3V = 0.33V < 0.408V = minimum control voltage (for 5A)
    const int MIN_HEXVRES           =      3000; // 150mV < 0.25V = minimum voltage drop on power resistor (for 5A)
    const int MAX_HEXVRES           =     40314; // 2030 mV > 2V = maximum voltage drop on power resistor (for 40A)

    const float COMM_PERIOD         =     0.050; // 50ms = 2500us is the fastest time between two consecutive transmissions (found by tests)
    const float SLOW_COMM_PERIOD    =         1; // Time to check if the user sent a stop execution
    const int BAUDRATE              =    921600; // Max admitted baud rate via UART


    // Function prototypes
    void monitor_mode(void);
    void pattern1(unsigned int);
    void pattern2(void);
    void force_dac(int);
```

```
52.     void send_data(void);
53.     void receive_ack(void);
54.     void check_if_hextemp_is_safe(void);
55.     void check_if_hexvbatt_is_low(void);
56.     void check_if_hexvolt_is_zero(void);
57.
58. #endif
```

A.2.2.3. patterns.c

```
#include "patterns.h"

ADC_HandleTypeDef hadc1_prime;
ADC_HandleTypeDef hadc2_prime;

AnalogIn Vbatt(A6);
AnalogIn Vtemp1(A2);
AnalogIn Vtemp2(A3);
AnalogIn Vres1(A0);
AnalogIn Vres2(A1);
AnalogOut Vctrl1(A4);
AnalogOut Vctrl2(A5);

DigitalOut Vbjt1(D3);
DigitalOut Vbjt2(D4);
DigitalOut Vbjt3(D5);

Serial pc(D1, D0, BAUDRATE);

Timer timer;

LowPowerTicker send_ticker;
LowPowerTicker safe_temp_ticker;
LowPowerTicker zero_volt_ticker;
LowPowerTicker receive_ticker;
LowPowerTicker low_volt_ticker;

char bufferprime = '1';
unsigned short Vbatt_16;
unsigned short Temp1_16;
unsigned short Temp2_16;
unsigned short Vres1_16;
unsigned short Vres2_16;

void monitor_mode(void)
{
    hadc1_prime.Instance = ADC1;
    hadc2_prime.Instance = ADC2;

    send_ticker.attach(&send_data, COMM_PERIOD);
    receive_ticker.attach(&receive_ack, SLOW_COMM_PERIOD);
    return;
}

void pattern1(unsigned int time_const)
{
    hadc1_prime.Instance = ADC1;
    hadc2_prime.Instance = ADC2;

    int power_low = POWER_LOW;
    int power_medium = POWER_MED;
    int power_high = POWER_HIGH;

    send_ticker.attach(&send_data, COMM_PERIOD);
    //receive_ticker.attach(&receive_ack, SLOW_COMM_PERIOD);

    safe_temp_ticker.attach(check_if_hextemp_is_safe, SLOW_COMM_PERIOD);
    //zero_volt_ticker.attach(check_if_hexvolt_is_zero, SLOW_COMM_PERIOD);
    low_volt_ticker.attach(check_if_hexvbatt_is_low, SLOW_COMM_PERIOD);

    timer.start();

```

```
63.     force_dac(POWEROFF);
64.     while(timer.read_ms() < STARTTIME);
65.
66. // 0
67.     timer.reset();
68.     force_dac(power_low);
69.     while(timer.read_ms() < POWERTIME1);
70.     timer.reset();
71.     force_dac(POWEROFF);
72.     while(timer.read_ms() < RELAXTIME);
73. // 1
74.     timer.reset();
75.     force_dac(power_low);
76.     while(timer.read_ms() < POWERTIME2);
77.     timer.reset();
78.     force_dac(POWEROFF);
79.     while(timer.read_ms() < RELAXTIME);
80. // 2
81.     timer.reset();
82.     force_dac(power_medium);
83.     while(timer.read_ms() < POWERTIME1);
84.     timer.reset();
85.     force_dac(POWEROFF);
86.     while(timer.read_ms() < RELAXTIME);
87. // 3
88.     timer.reset();
89.     force_dac(power_high);
90.     while(timer.read_ms() < POWERTIME1);
91.     timer.reset();
92.     force_dac(POWEROFF);
93.     while(timer.read_ms() < RELAXTIME);
94. // 4
95.     timer.reset();
96.     force_dac(power_medium);
97.     while(timer.read_ms() < POWERTIME1);
98.     timer.reset();
99.     force_dac(power_low);
100.      while(timer.read_ms() < POWERTIME2);
101.      timer.reset();
102.      force_dac(power_medium);
103.      while(timer.read_ms() < POWERTIME1);
104.      timer.reset();
105.      force_dac(POWEROFF);
106.      while(timer.read_ms() < RELAXTIME);
107.  // 5
108.      timer.reset();
109.      force_dac(power_low);
110.      while(timer.read_ms() < POWERTIME2);
111.      timer.reset();
112.      force_dac(power_medium);
113.      while(timer.read_ms() < POWERTIME1);
114.      timer.reset();
115.      force_dac(POWEROFF);
116.      while(timer.read_ms() < RELAXTIME);
117.  // 6
118.      timer.reset();
119.      force_dac(power_medium);
120.      while(timer.read_ms() < POWERTIME1);
121.      timer.reset();
122.      force_dac(power_low);
123.      while(timer.read_ms() < POWERTIME2);
124.      timer.reset();
125.      force_dac(POWEROFF);
126.      while(timer.read_ms() < RELAXTIME);
127.  // 7
```

```
128.      timer.reset();
129.      force_dac(power_low);
130.      while(timer.read_ms() < POWERTIME2);
131.      timer.reset();
132.      force_dac(power_high);
133.      while(timer.read_ms() < POWERTIME1);
134.      timer.reset();
135.      force_dac(POWEROFF);
136.      while(timer.read_ms() < RELAXTIME);
137.  // 8
138.      timer.reset();
139.      force_dac(power_high);
140.      while(timer.read_ms() < POWERTIME1);
141.      timer.reset();
142.      force_dac(power_low);
143.      while(timer.read_ms() < POWERTIME2);
144.      timer.reset();
145.      force_dac(POWEROFF);
146.      while(timer.read_ms() < RELAXTIME);
147.  // 9
148.      timer.reset();
149.      force_dac(power_low);
150.      while(timer.read_ms() < POWERTIME2);
151.      timer.reset();
152.      force_dac(power_medium);
153.      while(timer.read_ms() < POWERTIME1);
154.      timer.reset();
155.      force_dac(power_high);
156.      while(timer.read_ms() < POWERTIME1);
157.      timer.reset();
158.      force_dac(POWEROFF);
159.      while(timer.read_ms() < RELAXTIME);
160.  // 10
161.      timer.reset();
162.      force_dac(power_high);
163.      while(timer.read_ms() < POWERTIME1);
164.      timer.reset();
165.      force_dac(power_medium);
166.      while(timer.read_ms() < POWERTIME1);
167.      timer.reset();
168.      force_dac(power_low);
169.      while(timer.read_ms() < POWERTIME2);
170.      timer.reset();
171.      force_dac(POWEROFF);
172.      while(timer.read_ms() < RELAXTIME);
173.  // 11
174.      timer.reset();
175.      force_dac(power_low);
176.      while(timer.read_ms() < POWERTIME2);
177.      timer.reset();
178.      force_dac(power_medium);
179.      while(timer.read_ms() < POWERTIME1);
180.      timer.reset();
181.      force_dac(power_high);
182.      while(timer.read_ms() < POWERTIME1);
183.      timer.reset();
184.      force_dac(power_medium);
185.      while(timer.read_ms() < POWERTIME1);
186.      timer.reset();
187.      force_dac(power_low);
188.      while(timer.read_ms() < POWERTIME2);
189.      timer.reset();
190.      force_dac(POWEROFF);
191.      while(timer.read_ms() < RELAXTIME);
192.  // 12
```

```
193.     timer.reset();
194.     force_dac(power_low);
195.     while(timer.read_ms() < POWERTIME1);
196.     timer.reset();
197.     force_dac(power_medium);
198.     while(timer.read_ms() < POWERTIME2);
199.     timer.reset();
200.     force_dac(power_high);
201.     while(timer.read_ms() < POWERTIME1);
202.     timer.reset();
203.     force_dac(power_medium);
204.     while(timer.read_ms() < POWERTIME2);
205.     timer.reset();
206.     force_dac(power_low);
207.     while(timer.read_ms() < POWERTIME1);
208.     timer.reset();
209.     force_dac(POWEROFF);
210.     while(timer.read_ms() < RELAXTIME);
211. // 13
212.     timer.reset();
213.     force_dac(power_medium);
214.     while(timer.read_ms() < POWERTIME2);
215.     timer.reset();
216.     force_dac(power_high);
217.     while(timer.read_ms() < POWERTIME1);
218.     timer.reset();
219.     force_dac(power_medium);
220.     while(timer.read_ms() < POWERTIME2);
221.     timer.reset();
222.     force_dac(POWEROFF);
223.     while(timer.read_ms() < RELAXTIME);
224. // 14
225.     timer.reset();
226.     force_dac(power_low);
227.     while(timer.read_ms() < POWERTIME1);
228.     timer.reset();
229.     force_dac(power_medium);
230.     while(timer.read_ms() < POWERTIME2);
231.     timer.reset();
232.     force_dac(power_high);
233.     while(timer.read_ms() < POWERTIME1);
234.     timer.reset();
235.     force_dac(POWEROFF);
236.     while(timer.read_ms() < RELAXTIME);
237. // 15
238.     timer.reset();
239.     force_dac(power_high);
240.     while(timer.read_ms() < POWERTIME1);
241.     timer.reset();
242.     force_dac(power_medium);
243.     while(timer.read_ms() < POWERTIME2);
244.     timer.reset();
245.     force_dac(power_low);
246.     while(timer.read_ms() < POWERTIME1);
247.     timer.reset();
248.     force_dac(POWEROFF);
249.     while(timer.read_ms() < RELAXTIME);
250. // 16
251.     timer.reset();
252.     force_dac(power_low);
253.     while(timer.read_ms() < POWERTIME1);
254.     timer.reset();
255.     force_dac(power_high);
256.     while(timer.read_ms() < POWERTIME2);
257.     timer.reset();
```

```
258.      force_dac(POWEROFF);
259.      while(timer.read_ms() < RELAXTIME);
260.  // 17
261.      timer.reset();
262.      force_dac(power_high);
263.      while(timer.read_ms() < POWERTIME2);
264.      timer.reset();
265.      force_dac(power_low);
266.      while(timer.read_ms() < POWERTIME1);
267.      timer.reset();
268.      force_dac(POWEROFF);
269.      while(timer.read_ms() < RELAXTIME);
270.  // 18
271.      timer.reset();
272.      force_dac(power_low);
273.      while(timer.read_ms() < POWERTIME1);
274.      timer.reset();
275.      force_dac(power_medium);
276.      while(timer.read_ms() < POWERTIME2);
277.      timer.reset();
278.      force_dac(POWEROFF);
279.      while(timer.read_ms() < RELAXTIME);
280.  // 19
281.      timer.reset();
282.      force_dac(power_medium);
283.      while(timer.read_ms() < POWERTIME2);
284.      timer.reset();
285.      force_dac(power_low);
286.      while(timer.read_ms() < POWERTIME1);
287.      timer.reset();
288.      force_dac(POWEROFF);
289.      while(timer.read_ms() < RELAXTIME);
290.  // 20
291.      timer.reset();
292.      force_dac(power_high);
293.      while(timer.read_ms() < POWERTIME1);
294.      timer.reset();
295.      force_dac(power_medium);
296.      while(timer.read_ms() < POWERTIME2);
297.      timer.reset();
298.      force_dac(power_high);
299.      while(timer.read_ms() < POWERTIME1);
300.      timer.reset();
301.      force_dac(POWEROFF);
302.      while(timer.read_ms() < RELAXTIME);
303.  // 21
304.      timer.reset();
305.      force_dac(power_high);
306.      while(timer.read_ms() < POWERTIME2);
307.      timer.reset();
308.      force_dac(POWEROFF);
309.      while(timer.read_ms() < RELAXTIME);
310.  // 22
311.      timer.reset();
312.      force_dac(power_medium);
313.      while(timer.read_ms() < POWERTIME2);
314.      timer.reset();
315.      force_dac(POWEROFF);
316.      while(timer.read_ms() < RELAXTIME);
317.  // 23
318.      timer.reset();
319.      force_dac(power_low);
320.      while(timer.read_ms() < POWERTIME2);
321.      timer.reset();
322.      force_dac(POWEROFF);
```

```
323.     while(timer.read_ms() < RELAXTIME);
324. // 24
325.     timer.reset();
326.     force_dac(power_low);
327.     while(timer.read_ms() < POWERTIME1);
328.     timer.reset();
329.     force_dac(POWEROFF);
330.     while(timer.read_ms() < RELAXTIME);
331. // 25
332.
333. // Start constant discharge
334.     timer.reset();
335.     force_dac(POWER_CONST);
336.     while(timer.read_ms() < time_const);
337.     timer.reset();
338.     force_dac(POWEROFF);
339.     while(timer.read_ms() < RELAXTIME2);
340.
341.     send_ticker.detach();
342.     //receive_ticker.detach();
343.     fflush(pc);
344.     timer.stop();
345.
346.     pc.printf("End\n");
347.
348.     char ack = pc.getc();
349.     pc.getc();
350.     if(ack == '1'){
351.         return;
352.     }
353. }
354.
355. void pattern2(void){
356.
357.     hadc1_prime.Instance = ADC1;
358.     hadc2_prime.Instance = ADC2;
359.
360.     send_ticker.attach(&send_data, COMM_PERIOD);
361.     //receive_ticker.attach(&receive_ack, SLOW_COMM_PERIOD);
362.
363.     safe_temp_ticker.attach(check_if_hextemp_is_safe, SLOW_COMM_PERIOD);
364.     //zero_volt_ticker.attach(check_if_hexvolt_is_zero,
    SLOW_COMM_PERIOD);
365.     low_volt_ticker.attach(check_if_hexvbatt_is_low, SLOW_COMM_PERIOD);
366.
367.     timer.start();
368.
369.     force_dac(POWEROFF);
370.     while(timer.read_ms() < STARTTIME);
371.
372. // Wait
373.     timer.reset();
374.     force_dac(POWEROFF);
375.     while(timer.read_ms() < WAITTIME);
376.
377.     send_ticker.detach();
378.     //receive_ticker.detach();
379.     fflush(pc);
380.     timer.stop();
381.
382.     pc.printf("End\n");
383.
384.     char ack = pc.getc();
385.     pc.getc();
386.     if(ack == '1'){
```

```
387.         return;
388.     }
389. }
390.
391. void force_dac(int millivolt)
392. {
393.     int dac_out = (millivolt << 4) + (millivolt << 2) - (millivolt >>
   3);
394.
395.     Vctrl1.write_u16(dac_out);
396.     Vctrl2.write_u16(dac_out);
397. }
398.
399. void send_data(void)
400. {
401.     HAL_ADCEx_InjectedStart(&hadc2_prime); // Slave first
402.     HAL_ADCEx_InjectedStart(&hadc1_prime);
403.
404.     // Wait for the conversions to finish
405.     while(HAL_ADCEx_InjectedPollForConversion(&hadc2_prime, 0.5) !=
   HAL_OK); // Timeout in ms
406.     while(HAL_ADCEx_InjectedPollForConversion(&hadc1_prime, 0.5) !=
   HAL_OK); // Timeout in ms
407.
408.     Temp1_16 = HAL_ADCEx_InjectedGetValue(&hadc1_prime,
   ADC_INJECTED_RANK_1);
409.     Vres2_16 = HAL_ADCEx_InjectedGetValue(&hadc1_prime,
   ADC_INJECTED_RANK_2);
410.     Vres1_16 = HAL_ADCEx_InjectedGetValue(&hadc1_prime,
   ADC_INJECTED_RANK_3);
411.
412.     Vbatt_16 = HAL_ADCEx_InjectedGetValue(&hadc2_prime,
   ADC_INJECTED_RANK_1);
413.     Temp2_16 = HAL_ADCEx_InjectedGetValue(&hadc2_prime,
   ADC_INJECTED_RANK_2);
414.
415.     //HAL_ADCEx_InjectedStop(&hadc1_prime); //Master first
416.     //HAL_ADCEx_InjectedStop(&hadc2_prime);
417.
418.     pc.printf("%d, %d, %d, %d, %d\n",
419.         Temp1_16,
420.         Temp2_16,
421.         Vres2_16,
422.         Vbatt_16,
423.         Vres1_16);
424.     return;
425. }
426.
427. void receive_ack(void)
428. {
429.     bufferprime = pc.getc();
430.     pc.getc();
431.     if(bufferprime != '1'){
432.         send_ticker.detach();
433.         receive_ticker.detach();
434.         fflush(pc);
435.         pc.printf("Stop\n");
436.         Vctrl1.write_u16(POWEROFF);
437.         Vctrl2.write_u16(POWEROFF);
438.         wait(1);
439.         NVIC_SystemReset();
440.     }
441.     return;
442. }
443.
```

```
444. void check_if_hextemp_is_safe(void)
445. {
446.     if(Temp1_16 > OVERHEAT_HEXTEMP){
447.         send_ticker.detach();
448.         fflush(pc);
449.         pc.printf("Overheating on Branch 1\n");
450.         Vbjt1 = 1;
451.         Vctrl1.write_u16(POWEROFF);
452.         Vctrl2.write_u16(POWEROFF);
453.
454.         char ack = pc.getc();
455.         pc.getc();
456.         if(ack == '1'){
457.             wait(1);
458.             NVIC_SystemReset();
459.         }
460.     }
461.
462.     if(Temp2_16 > OVERHEAT_HEXTEMP){
463.         send_ticker.detach();
464.         fflush(pc);
465.         pc.printf("Overheating on Branch 2\n");
466.         Vbjt2 = 1;
467.         Vctrl1.write_u16(POWEROFF);
468.         Vctrl2.write_u16(POWEROFF);
469.
470.         char ack = pc.getc();
471.         pc.getc();
472.         if(ack == '1'){
473.             wait(1);
474.             NVIC_SystemReset();
475.         }
476.     }
477. }
478.
479. void check_if_hexvbatt_is_low(void)
480. {
481.     if(Vbatt_16 < LOWBATT_HEXVOLT){
482.         send_ticker.detach();
483.         fflush(pc);
484.         pc.printf("Vbatt too low\n");
485.         Vbjt1 = 1;
486.         Vbjt2 = 1;
487.         Vbjt3 = 1;
488.         Vctrl1.write_u16(POWEROFF);
489.         Vctrl2.write_u16(POWEROFF);
490.
491.         char ack = pc.getc();
492.         pc.getc();
493.         if(ack == '1'){
494.             wait(1);
495.             NVIC_SystemReset();
496.         }
497.     }
498. }
499.
500. void check_if_hexvolt_is_zero(void)
501. {
502.     if(Vctrl1.read() > MIN_FLOATVCTRL && Vres1_16 < MIN_HEXVRES &&
   Vctrl2.read() > MIN_FLOATVCTRL && Vres2_16 < MIN_HEXVRES){
503.         send_ticker.detach();
504.         fflush(pc);
505.         pc.printf("No current was draining. Check the fuse if
   persists\n");
506.         Vbjt1 = 1;
```

```
507.          Vbjt2 = 1;
508.          Vbjt3 = 1;
509.          Vctrl1.write_u16(POWEROFF);
510.          Vctrl2.write_u16(POWEROFF);
511.
512.          char ack = pc.getc();
513.          pc.getc();
514.          if(ack == '1'){
515.              wait(1);
516.              NVIC_SystemReset();
517.          }
518.      }
519.
520.      if(Vctrl1.read() > MIN_FLOATVCTRL && Vres1_16 < MIN_HEXVRES){
521.          send_ticker.detach();
522.          fflush(pc);
523.          pc.printf("No current was draining. Check the Branch 1 if
    persists\n");
524.          Vbjt1 = 1;
525.          Vbjt3 = 1;
526.          Vctrl1.write_u16(POWEROFF);
527.          Vctrl2.write_u16(POWEROFF);
528.
529.          char ack = pc.getc();
530.          pc.getc();
531.          if(ack == '1'){
532.              wait(1);
533.              NVIC_SystemReset();
534.          }
535.      }
536.
537.      if(Vctrl2.read() > MIN_FLOATVCTRL && Vres2_16 < MIN_HEXVRES){
538.          send_ticker.detach();
539.          fflush(pc);
540.          pc.printf("No current was draining. Check the Branch 2 if
    persists\n");
541.          Vbjt2 = 1;
542.          Vbjt3 = 1;
543.          Vctrl1.write_u16(POWEROFF);
544.          Vctrl2.write_u16(POWEROFF);
545.
546.          char ack = pc.getc();
547.          pc.getc();
548.          if(ack == '1'){
549.              wait(1);
550.              NVIC_SystemReset();
551.          }
552.      }
553.  }
```

A.3. "Basic" (a.k.a. PROTO0) hardware design

A.3.1. Schematic

A.3.1.1. Microcontroller

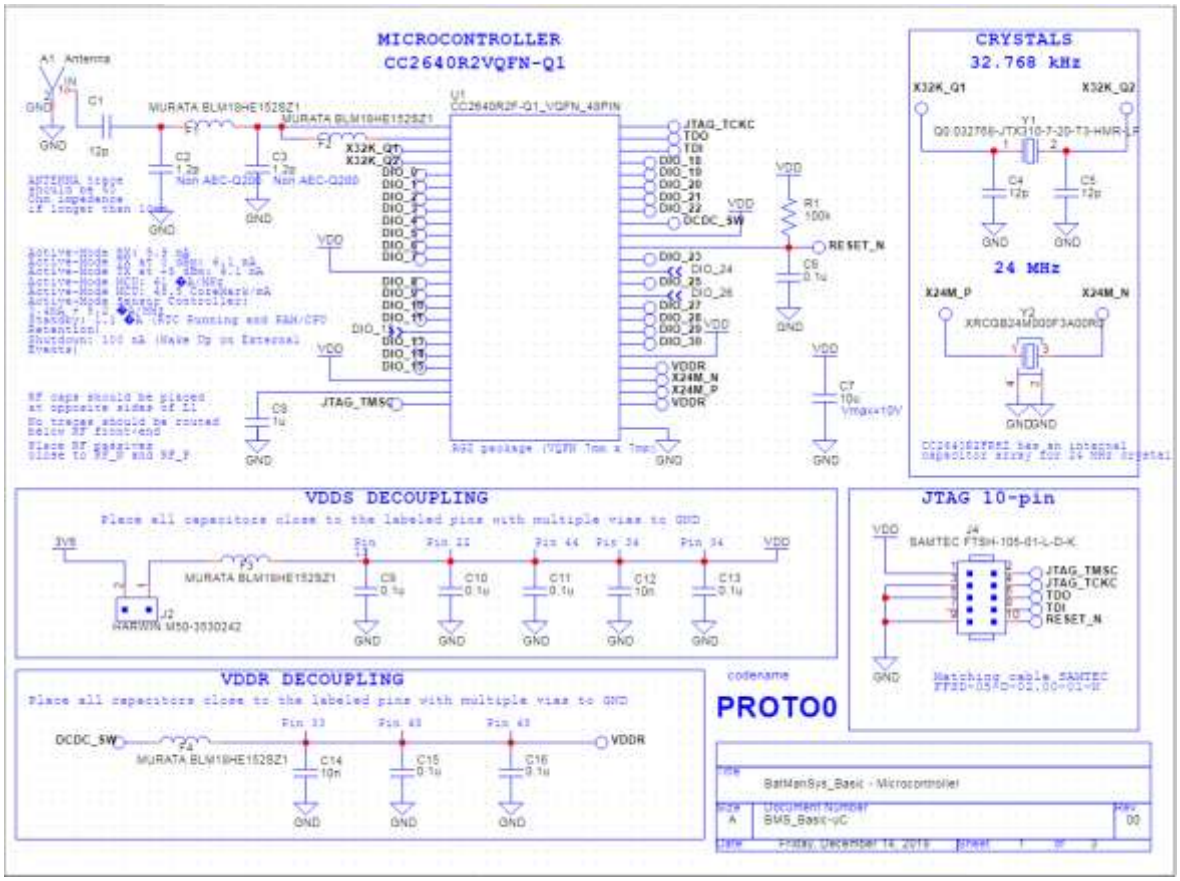


Figure 134. Microcontroller page of schematic for "Basic" hardware design

A.3.1.2. Power distribution

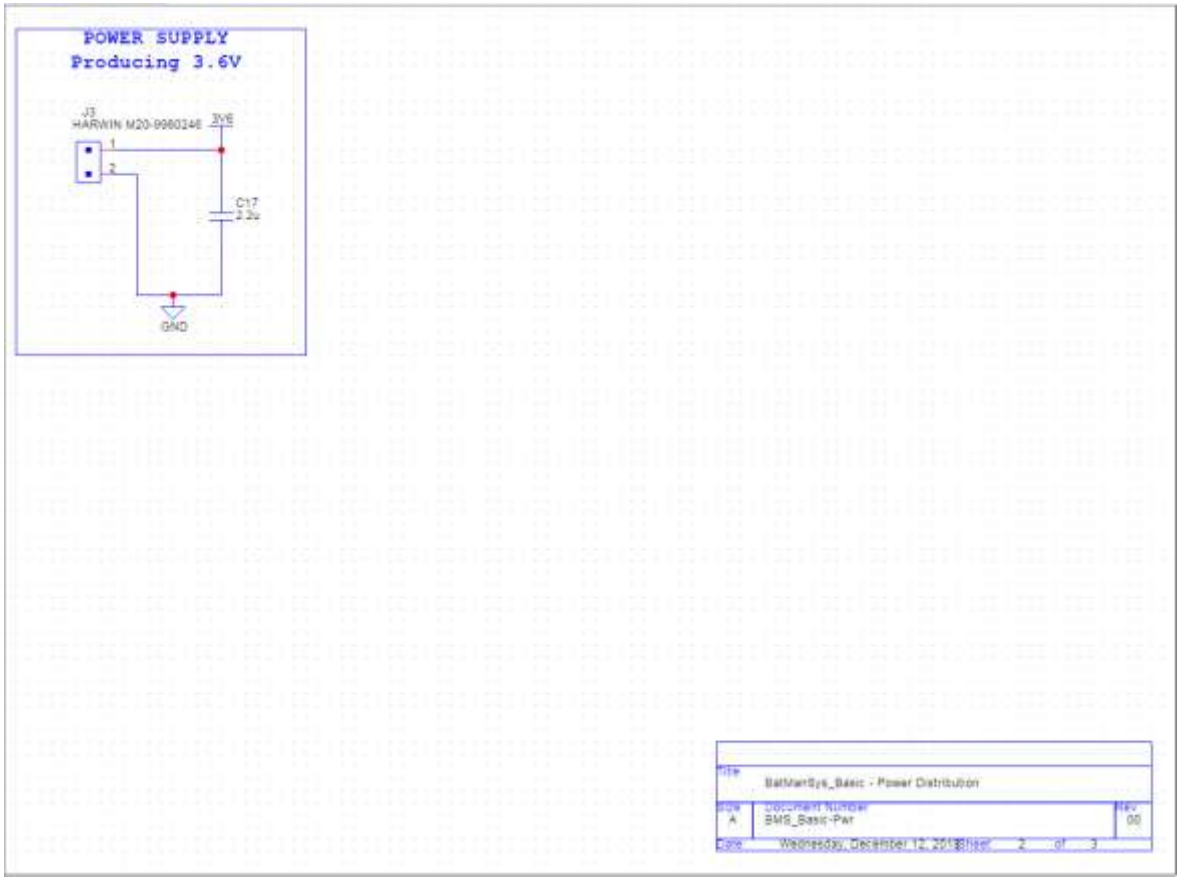


Figure 135. Power distribution page of the schematic for the "Basic" hardware design

A.3.1.3. Digital

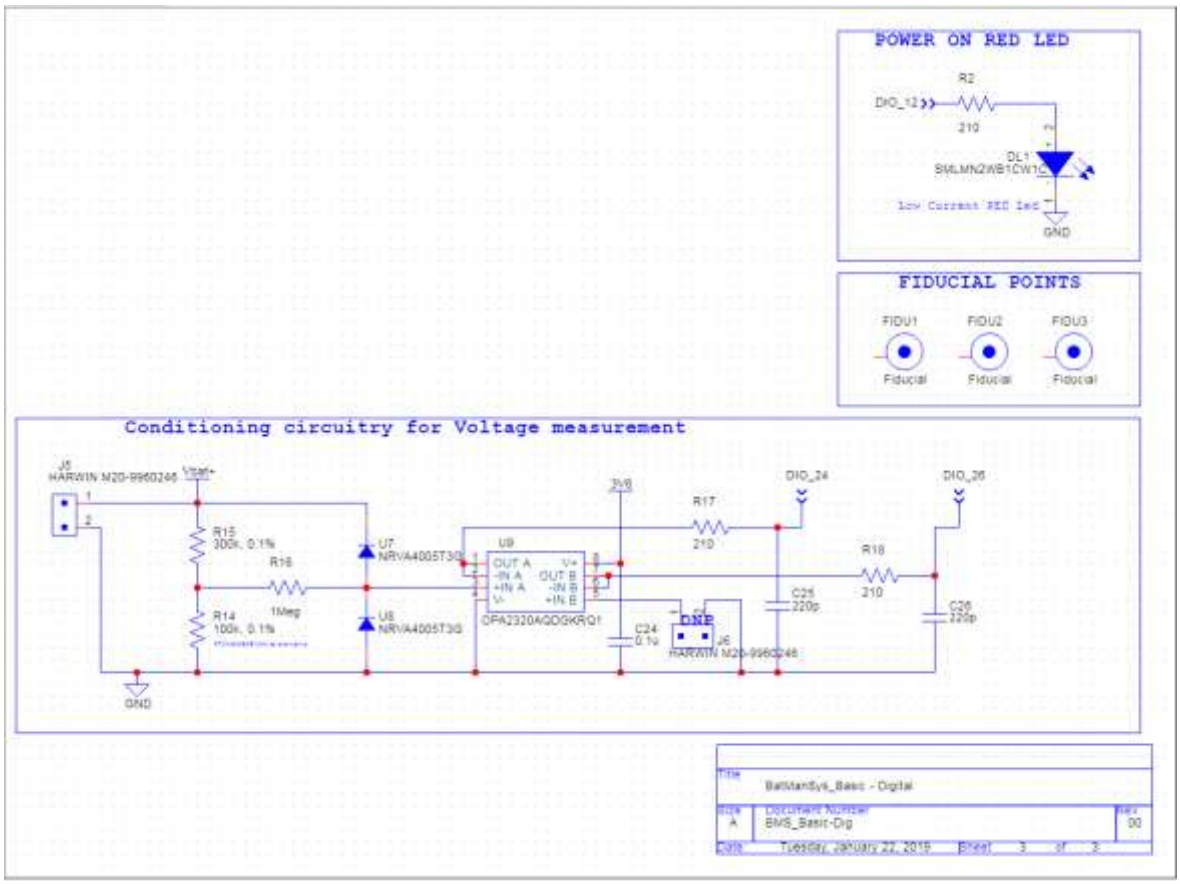


Figure 136. Digital page of the schematic for the "Basic" hardware design

A.3.2. Footprint

A.3.2.1. Top side

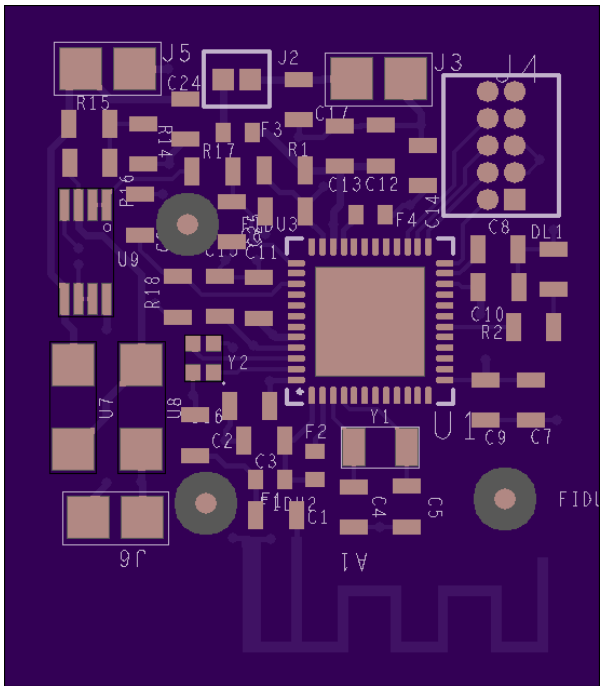


Figure 137. Top side of the footprint for the "Basic" hardware design

A.3.2.2. Bottom side

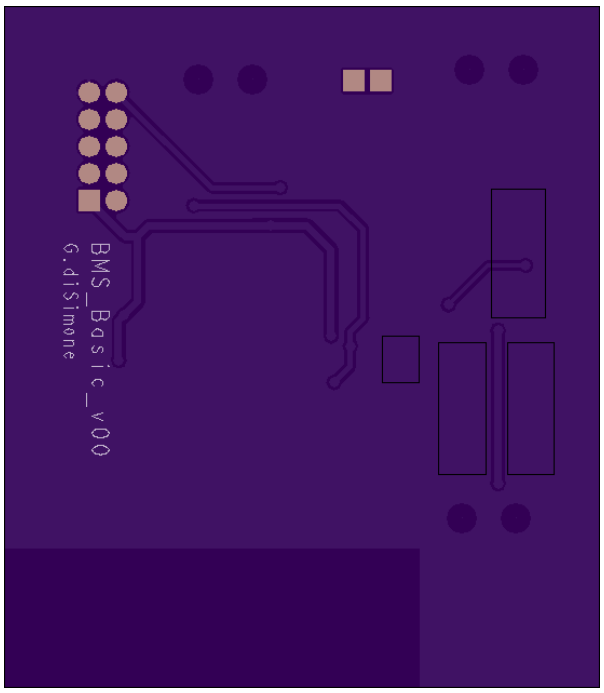


Figure 138. Bottom side of the footprint for the "Basic" hardware design

A.3.3. 3D step model

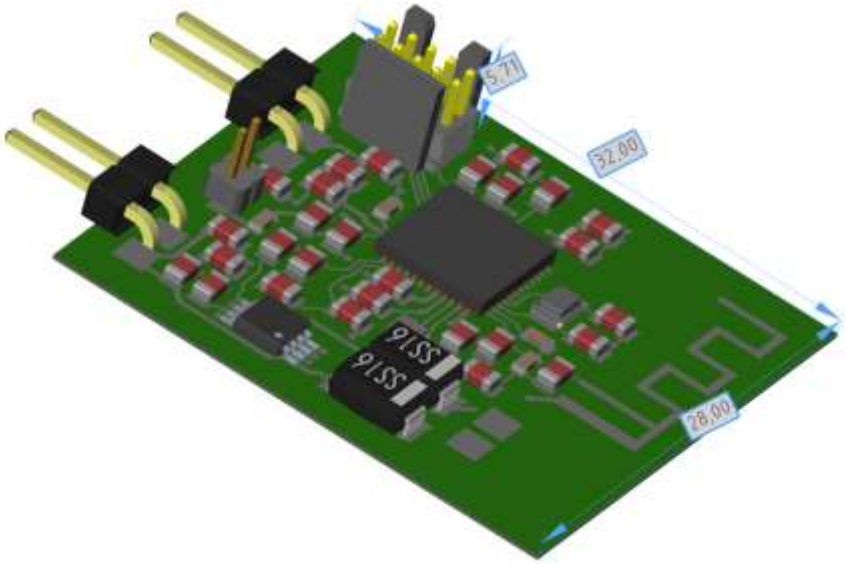


Figure 139. 3D step model for the "Basic" hardware design

A.4. "Basic Pro" (a.k.a. PROTO1) hardware design

A.4.1. Schematic

A.4.1.1. Microcontroller

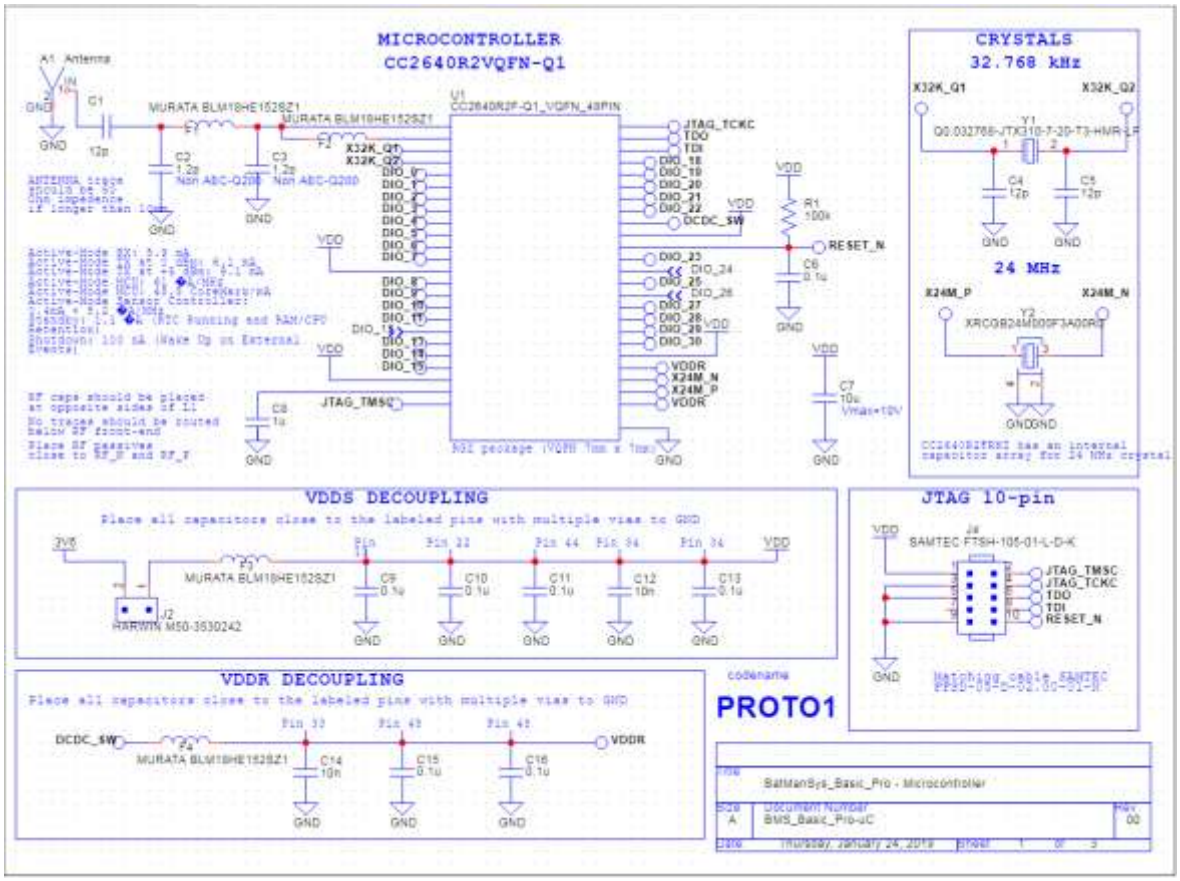


Figure 140. Microcontroller page of the schematic for the "Basic Pro" hardware design

A.4.1.2. Power distribution

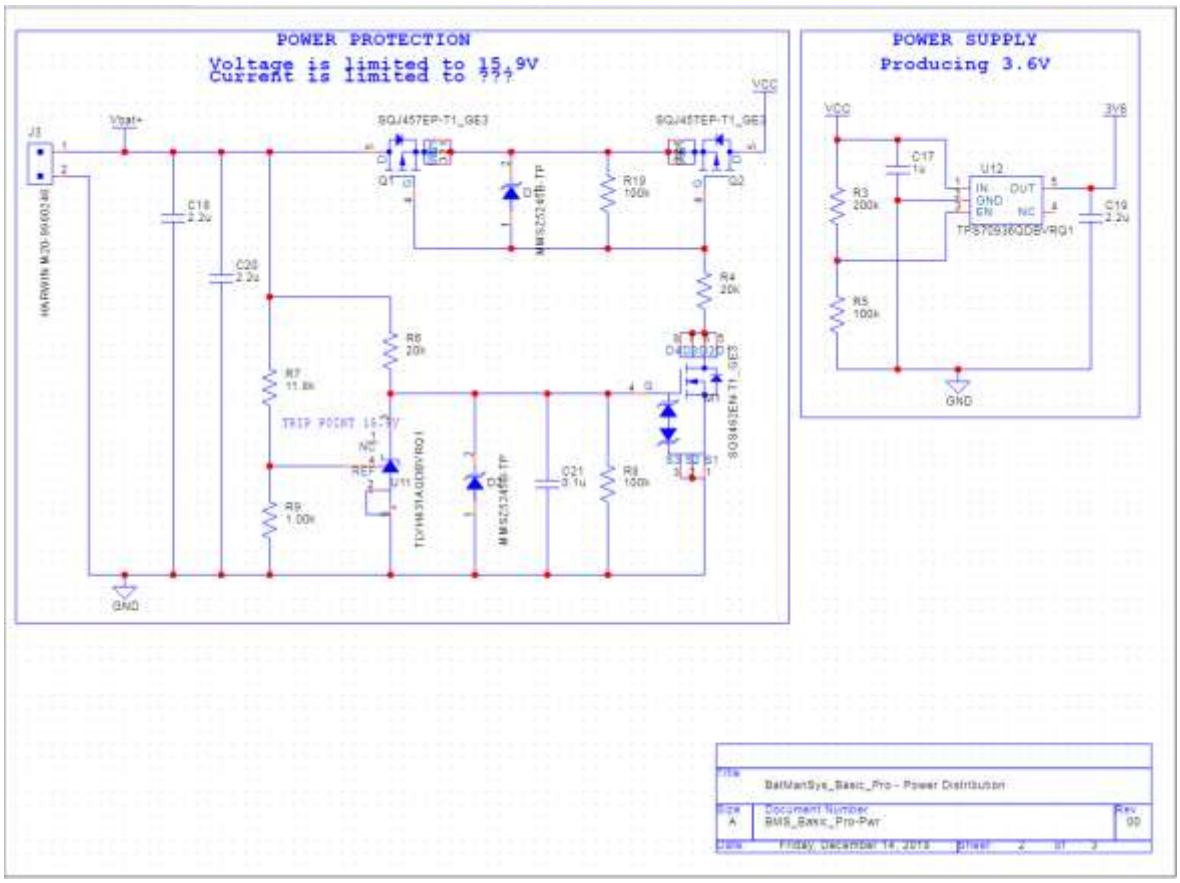


Figure 141. Power distribution page of the schematic for the "Basic Pro" hardware design

A.4.1.3. Digital

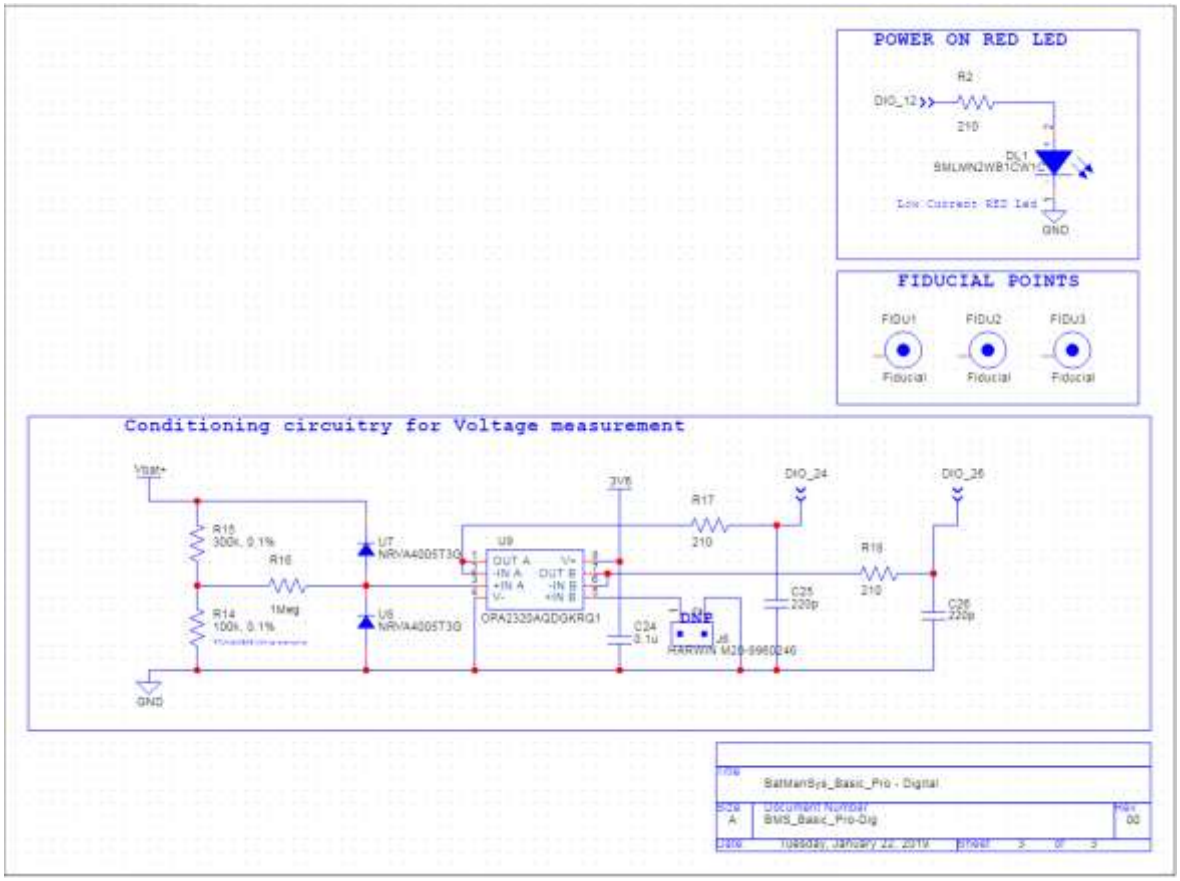


Figure 142. Digital page of the schematic for the "Basic Pro" hardware design

A.4.2. Footprint

A.4.2.1. Top side

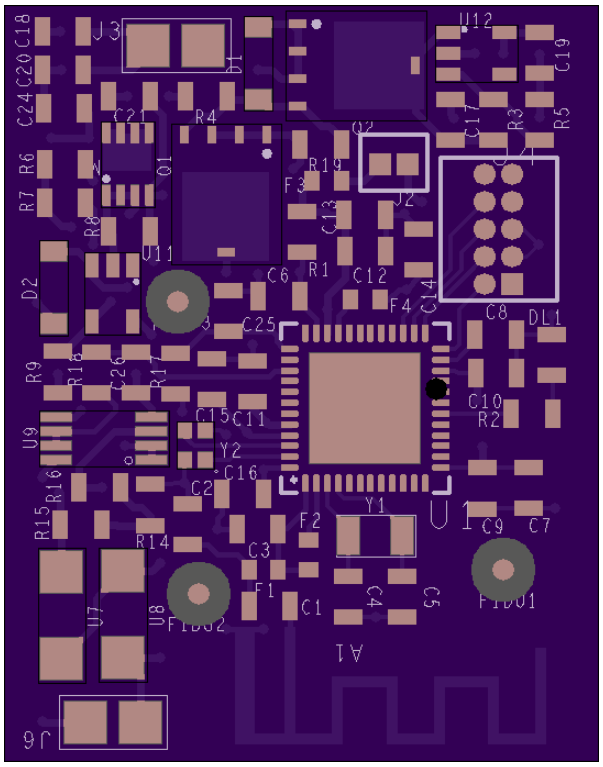


Figure 143. Top side of the footprint for the "Basic Pro" hardware design

A.4.2.2. Bottom side

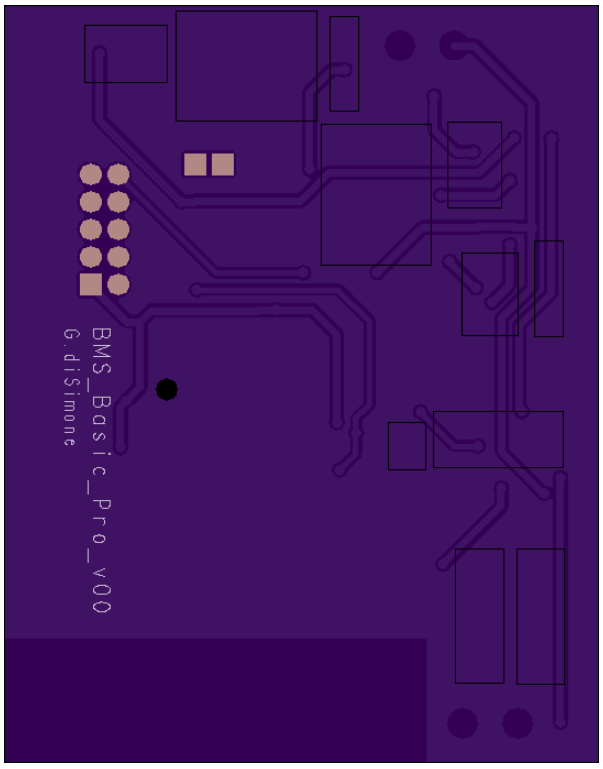


Figure 144. Bottom side of the footprint for the "Basic Pro" hardware design

A.4.3. 3D step model

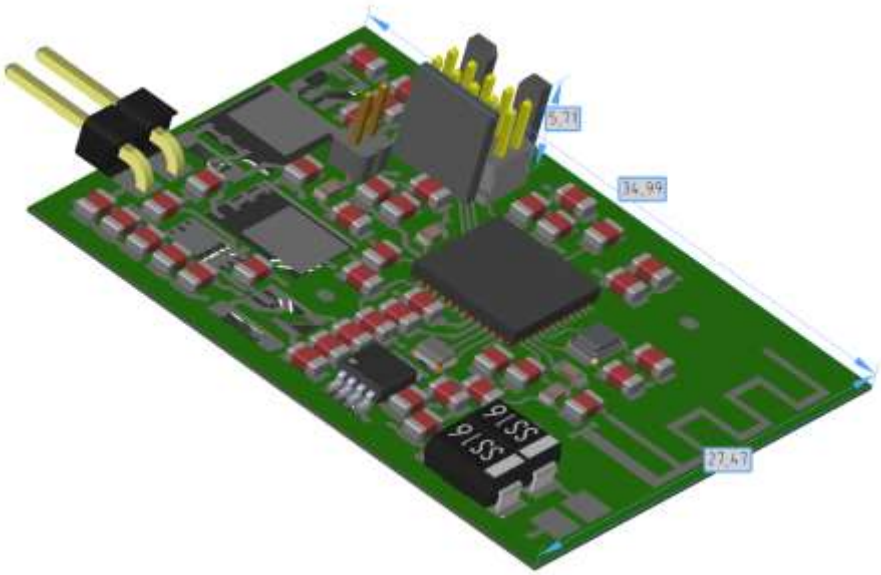


Figure 145. 3D step model for the "Basic Pro" hardware design

Chapter B

Bibliography

[1] T. I-Kun, M. Inamori and M. Morimoto, "SOH estimation of lead acid battery by artificial deterioration," in *2014 17th International Conference on Electrical Machines and Systems (ICEMS)*, 2014.

[2] Y. Kohya, T. Takeda, K. Takano, M. Kohno, K. Yotsumoto and T. Ogata, "A deterioration estimating system for 200-Ah sealed lead-acid batteries," in *Proceedings of Intelec 94*, 1994.

[3] X. Zhang, R. Grube, K.-K. Shin, M. Salman and R. Conell, "Automotive Battery State-of-Health Monitoring: a Parity Relation Based Approach," *IFAC Proceedings Volumes,* vol. 42, no. 8, pp. 552-557, 2009.

[4] J. Xie, W. Li and Y. Hu, "Aviation lead-acid battery state-of-health assessment using PSO-SVM technique," in *2014 IEEE 5th International Conference on Software Engineering and Service Science*, 2014.

[5] T.-T. Nguyen, V.-L. Tran and W. Choi, "Development of the intelligent charger with battery State-Of-Health estimation using online impedance spectroscopy," in *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*, 2014.

[6] M. Dürr, A. Cruden, S. Gair and J. R. McDonald, "Dynamic model of a lead acid battery for use in a domestic fuel cell system," *Journal of Power Sources,* vol. 161, no. 2, pp. 1400-1411, 2006.

[7] J. Zeitouny, J. B. Merhi, J. Chalak and S. Karaki, "Estimation of battery internal parameters," in *2015 IEEE International Conference on Industrial Technology (ICIT)*, 2015.

[8] Wikipedia contributors, "American wire gauge," 20 03 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=American_wire_gauge&oldid=888626831.

[9] International Rectifier®, "IRFP250MPbF Product Data Sheet," 01 03 2010. [Online]. Available: https://www.infineon.com/dgdl/irfp250mpbf.pdf?fileId=5546d462533600a4015356287bc71fda.

[10] Texas Instruments® Incorporated, "LMC6482 CMOS Dual Rail-To-Rail Input and Output Operational Amplifier datasheet," 30 06 2017. [Online]. Available: http://www.ti.com/lit/ds/symlink/lmc6482.pdf.

[11] CADDOCK® ELECTRONICS, INC., "MP900 and MP9000 Series Kool-Pak® Power Film Resistors TO-126, TO-220 and TO-247 Style," 28 07 2017. [Online]. Available: http://www.caddock.com/Online_catalog/Mrktg_Lit/MP9000_Series.pdf.

[12] Toshiba® Electronic Devices & Storage Corporation, "Power MOSFET Thermal Design and Attachment of a Thermal Fin," 30 07 2018. [Online]. Available: https://toshiba.semicon-storage.com/info/docget.jsp?did=13417.

[13] Crydom®, Inc, "The Effect of Forced Air Cooling on Heat Sink Thermal Ratings," 08 04 2011. [Online]. Available: http://www.crydom.com/en/tech/hs_wp_fa.pdf.

[14] Texas Instruments® Incorporated, "Implementation and Applications of Current Sources and Current Receivers," 28 03 2001. [Online]. Available: http://www.ti.com/lit/an/sboa046/sboa046.pdf.

[15] Texas Instruments® Incorporated, "Operational Amplifier Stability Part 10 of 15: Capacitor Load Stability: RISO with Dual Feedback," 23 09 2007. [Online]. Available: http://read.pudn.com/downloads140/ebook/606849/%E8%BF%90%E6%94%BE%E7%A8%B3%E5%AE%9A%E6%80%A7%EF%BC%8810%EF%BC%89.pdf.

[16] Linear Technology® Corporation, "LTC1152 - Rail-to-Rail Input Rail-to-Rail Output Zero-Drift Op Amp," 16 08 2007. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/lt1152.pdf.

[17] Texas Instruments® Incorporated, "FilterPro™ MFB and Sallen-Key Low-Pass Filter Design Program User Guide," 24 02 2011. [Online]. Available: http://www.ti.com/lit/pdf/SBFA001.

[18] National Instruments®, "PXIe-6363 Specifications - National Instruments," 25 10 2018. [Online]. Available: http://www.ni.com/pdf/manuals/377776a.pdf.

[19] Texas Instruments® Incorporated, "Optimize Your SAR ADC Design," 20 01 2010. [Online]. Available: http://e2e.ti.com/cfs-file/__key/communityserver-discussions-components-files/14/4478.PA_2D00_001--Optimize_5F00_SAR_5F00_converter_5F00_design-REV-b.pdf.

[20] OHMITE® MANUFACTURING COMPANY, INC, "10 Series Axial Wire Element Current Sense Two Terminal Axial," 29 08 2018. [Online]. Available: https://www.ohmite.com/assets/docs/res_10.pdf.

[21] Fischer® Elektronik GmbH & Co. KG, "Fischerelektronik LAM 3 K," 31 05 2018. [Online]. Available: https://www.fischerelektronik.de/web_fischer/en_GB/PR/LAM3K_/datasheet.xhtml.

[22] Texas Instruments® Incorporated, "OPAx350 High-Speed, Single-Supply, Rail-to-Rail Op Amps datasheet (Rev. D)," 21 03 2019. [Online]. Available: http://www.ti.com/lit/ds/sbos099d/sbos099d.pdf.

[23] Texas Instruments® Incorporated, "Single-Supply, Rail-to-Rail Operational Amplifiers MicroAmplifier™ Series datasheet (Rev. C)," 28 03 2019. [Online]. Available: http://www.ti.com/lit/ds/symlink/opa340.pdf.

[24] Microchip Technology Inc., "Low-Power Linear Active Thermistor ICs," 03 06 2016. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/20001942G.pdf.

[25] STMicroelectronics®, "STM32 Nucleo-32 boards (MB1180) - User manual," 12 11 2018. [Online]. Available: https://www.st.com/resource/en/user_manual/dm00231744.pdf.

[26] ON Semiconductor®, "BC337 - NPN Amplifier Transistors," 27 11 2013. [Online]. Available: https://www.onsemi.com/pub/Collateral/BC337-D.PDF.

[27] Texas Instruments® Incorporated, "Remote Camera and Radar Expansion, SAT0074," 18 08 2014. [Online]. Available: http://www.ti.com/general/docs/lit/getliterature.tsp?baseLiteratureNumber=tidra74&fileType=pdf.