Entwicklung eines Tools zur Validierung eines Akkumulators für den Motorsport

Bachelorarbeit

zur Erlangung des akademischen Grades

"Bachelor of Science in Engineering"

Studiengang:

Mechatronik

Management Center Innsbruck

Betreuende/r:

Bernhard Hollaus, PhD.

Verfasser/-in:

Roland Kummer

1910602092

Eidesstattliche Erklärung

„Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.“

Ort, Datum

Unterschrift

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, welche mich bei der Umsetzung meiner Bachelorarbeit unterstützt haben.

Zunächst möchte ich mich bei meinem Betreuer Bernhard Hollaus, PhD. bedanken, welcher bei Fragen bzw. Anliegen stets ein offenes Ohr hatte und hilfreiche Vorschläge eingebracht hat.

Des Weiteren möchte ich mich beim Team von Campus Tirol Motorsport bedanken. Mir wurde durch die Faszination des Teams nicht nur eine spannende Plattform zum Ausarbeiten meiner Bachelorarbeit geboten, sondern durch den vorherrschenden Teamgeist und der damit einhergehenden ständigen Hilfsbereitschaft der Teammitglieder eine tolle Arbeitsumgebung geschaffen.

Hierbei möchte ich mich besonders bei Philipp Stocker bedanken, welcher mich über den Verlauf der gesamten Arbeit auf mehrfacher Art und Weise unterstützt hat. Von der Bestimmung des Bachelorarbeitsthemas, über die Betreuung in- und außerhalb der Werkstatt bis hin zum Networking mit Unternehmen zur Umsetzung der Bachelorarbeit. Das von seiner Seite gegebene Ausmaß an Engagement und Hilfsbereitschaft war für mich nicht selbstverständlich. Die Zusammenarbeit mit ihm war für mich nicht nur eine technische sondern auch eine menschliche Wissensbereicherung.

In diesem Zusammenhang möchte ich mich beim Unternehmen ALPITRONIC für die Bereitstellung von für die Umsetzung der Bachelorarbeit benötigten Hardware und beim Unternehmen MATTRO für die Bereitstellung von Räumlichkeiten für entsprechende Versuchsdurchführungen bedanken. Der Kontaktaufbau zu beiden Unternehmen wurde mit Unterstützung von Philipp Stocker ermöglicht.

Abschließend möchte ich mich bei Alexander Sellemond bedanken, welcher das Testequipment von ALPITRONIC für die Erfordernisse dieser Arbeit entsprechend umprogrammiert hat.

Roland Kummer

Innsbruck, 08.06.2022

Kurzfassung

Die Elektrifizierung von Fahrzeugen und die damit einhergehende Entwicklung nimmt fortlaufend größere Ausmaße an. Dabei wirft vor allem die wichtigste Komponente der Elektrofahrzeuge, der Akkumulator (kurz Akku), die meisten Fragen auf. Hierbei sind vor allem deren Kapazität und Empfindlichkeit auf äußere Einflüsse wie die Umgebungstemperatur ein bekanntes Thema. Für die Validierung von Akkus werden am Markt diverse Akkuvalidierungssysteme angeboten, welche jedoch zumeist sehr komplex sind. Diese Arbeit präsentiert eine verhältnismäßig simple Alternative zur Validierung eines Akkuprototypen für einen Rennwagen der Formula Student. Hierbei wird ein Validierungstool über die Programmierplattform MATLAB und dessen Blockdiagrammumgebung SIMULINK implementiert. Durch das Tool kann über einen CAN-Bus ein Lastprofil an eine Senke übermittelt werden, welche für den Akku eine entsprechende Last simuliert. Vom Akku gesendete Sensordaten können wiederum in das Programm eingelesen und ausgewertet werden. Das Ziel der Implementierung des Validierungstools ist die Optimierung der Nutzung der verfügbaren Akkukapazität für den Formula Student Wettbewerb. Diese Bachelorarbeit beginnt mit der Darlegung der notwendigen Grundlagen zu den angewandten Methoden. Daraufhin wird die Herangehensweise zur Implementierung des Tools beschrieben. Abschließend wird das Tool am Akkuprototypen getestet, woraufhin die daraus resultierenden Ergebnisse analysiert werden.

Schlagwörter: Akkuvalidierung, Batteriesimulator, CAN-Bus, MATLAB

Abstract

The electrification of vehicles and the associated development is continuously taking on greater proportions. The most important component of electric vehicles, the accumulator, raises the most questions. In particular, its capacity and sensitivity to external influences such as ambient temperature are well-known issues. For the validation of accumulators, various accumulator validation systems are offered on the market, which are, however, mostly very complex. This thesis presents a relatively simple alternative for the validation of a accumulator prototype for a Formula Student race car. A validation tool is implemented using the MATLAB programming platform and its block diagram environment SIMULINK. The tool can transmit a load profile to a sink via a CAN bus, which simulates a corresponding load for the accumulator. Sensor data sent by the accumulator can in turn be read into the program and evaluated. The goal of implementing the validation tool is to optimize the use of the available accumulator capacity for the Formula Student competition. This bachelor thesis starts with the presentation of the necessary basics for the applied methods. Then the approach for the implementation of the tool is described. Finally, the tool is tested on the accumulator prototype, after which the corresponding results are analyzed.

Keywords: Accumulatorvalidation, Battery Simulator, CAN bus, MATLAB

Inhaltsverzeichnis

1. Einleitung **1**
- 1.1. Motivation und Problemstellung . . . 1
- 1.2. Stand der Technik . . . 1
- 1.3. Zielsetzung . . . 2

2. Grundlagen **3**
- 2.1. Bus (Datenverarbeitung) . . . 3
  - 2.1.1. CAN-Bus . . . 4
- 2.2. Batterie/Akkumulator . . . 6
  - 2.2.1. Allgemein . . . 6
  - 2.2.2. Lithium-Ionen Akkumulator . . . 6
  - 2.2.3. Ladezustandsermittlung . . . 7
- 2.3. Batteriesimulator . . . 9

3. Konzept **10**

4. Umsetzung **13**
- 4.1. Definiton der CAN-Nachrichten . . . 13
- 4.2. Definition der Stromprofile . . . 18
  - 4.2.1. Entladekurve des Akkus . . . 18
  - 4.2.2. Fahrtsimulation . . . 19
- 4.3. SIMULINK-Modell . . . 20
  - 4.3.1. Generierung der Stromprofile . . . 20
  - 4.3.2. CAN-Kommunikation . . . 22
  - 4.3.3. SOC-Berechnung . . . 27
  - 4.3.4. Visualisierung . . . 29
- 4.4. Kontrollpanel . . . 30
- 4.5. Testequipment . . . 33
- 4.6. Verbindung von Soft- und Hardware . . . 35
- 4.7. Inbetriebnahme . . . 39
- 4.8. Ermittlung der Ersatzschaltbildparameter für die SOC-Berechnung . . . 41

5. Resultate **44**

6. Zusammenfassung und Ausblick **54**
- 6.1. Zusammenfassung . . . 54
- 6.2. Reflexion und Ausblick . . . 54

Literaturverzeichnis **IX**

Abbildungsverzeichnis **XI**

Tabellenverzeichnis **XII**

Abkürzungsverzeichnis **XIII**

A. DBC-Datei **XIV**

B. SIMULINK-Modell **XXI**

B.1. Oberste Ebene XXI
B.2. Function-Call-Subsystem XXII
B.3. SOC-Subsystem XXIII
B.4. Zustandsübergangsfunktion XXIV
B.5. Messfunktion XXV
B.6. Visualisierungs-Subsystem XXVI

C. MATLAB-Code für Kontrollpanel **XXVII**

1. Einleitung

1.1. Motivation und Problemstellung

Durch die zunehmende Verbreitung der Elektromobilität hat entsprechend auch die Forschung auf diesem Themengebiet stark zugenommen. Das Hauptaugenmerk liegt hierbei auf dem Energiespeicher der Fahrzeuge, dem Akku. Im Fokus steht hier vor allem die Kapazität des Akku und die damit einhergehende Reichweite der Fahrzeuge. Um in der Automobilindustrie neue Technologien zu testen, ist seit jeher der Motorsport ein beliebtes Versuchsfeld, da durch den einhergehenden Wettkampf ein zusätzlicher Motivator zur schnellstmöglichen Weiterentwicklung und Verbesserung der Technologien entsteht. Um Studenten und junge Ingenieure auf dieses Berufsfeld zu führen, bietet die Formula Student eine entsprechende Plattform. Hierbei handelt es sich um einen internationalen Konstruktionswettbewerb, bei welchem von Studententeams gefertigte Fahrzeuge gegeneinander antreten und auf die Probe gestellt werden. Die Teams können grundsätzlich zwischen zwei Motorkonzepten wählen: einem Verbrennungsmotor oder einem Elektromotor. Eines dieser Teams ist Campus Tirol Motorsport (CTM), welches auf das Antriebskonzept des Elektromotors setzt. Die notwendige Entwicklung des Elektroantriebs des CTM Rennwagens bildet die Grundlage dieser Bachelorarbeit. Im Detail soll zur Vorbereitung auf den Wettbewerb eine Vorab-Validierung des Akkus erfolgen. Hierbei soll eine Möglichkeit geschaffen werden eine Lastsituation für den Akku zu simulieren, um daraus resultierende Daten auswerten zu können.

1.2. Stand der Technik

Aufgrund der im Verlauf der letzten Jahrzehnte zunehmenden Digitalisierung von Automobilen und deren internen Steuerelementen, bedarf es entsprechender interner Netzwerke zum Informationsaustausch. Zimmermann und Schmidgall stellen in [1] die für die Automobilindustrie etablierten Kommunikationssysteme dar. Borgeest in [2] und Hrach bzw. Cifrain in [3] erörtern verschiedenste Akkutechnologien und vergleichen diese miteinander. Kagermann beschreibt in [4] und Karle in [5] das sich in der Automobilindustrie aufgrund der dezentralen Installierbarkeit, der hohen Energiedichte und guten Skalierbarkeit vor allem die Lithium-Ionen Technologie bewährt hat. Bei der Validierung eines Akkus ist die Bestimmung dessen Ladezustands unabdingbar. Zhang bearbeitet in [6] und Pfeil in [7] diverse Methoden zur Ermittlung des Ladezustands von

Akkus für Elektrofahrzeuge. Paulweber und Lebert behandeln in [8] die Anforderungen an Akkuprüfstände und die notwendigen Hard-und Software Komponenten für die Validierung von Akkus. Liu et al. führen in [9] eine Akkuvalidierung mithilfe eines Akkutestsystems von ARBIN INSTRUMENTS durch. ARBIN INSTRUMENTS bietet entsprechende Komplettlösungen auf Soft- und Hardwareebene an. Im Gegensatz zu [9], soll in dieser Arbeit eigenständig eine vergleichsweise simple Alternative für eine Akkuvalidierung implementiert und angewandt werden.

1.3. Zielsetzung

Der Formula Student Wettbewerb setzt sich aus verschiedenen Disziplinen zusammen. Eine der Disziplinen ist das Endurance and Efficiency Event. Hierbei soll auf einem geschlossenem Rundkurs mit einer Länge von 1 km eine Gesamtdistanz von 22 km absolviert werden. Laut Regelwerk darf der Akku zu keinem Zeitpunkt des Events eine Temperatur von 60 °C überschreiten. Da der Akku keine aktive Kühlung besitzt, ist es notwendig vor dem Wettbewerb festzustellen, wie viel Leistung aus dem Akku extrahiert werden kann, ohne diesen zu überhitzen. Gleichzeitig gilt festzustellen, wie viel Leistung genutzt werden kann, um mit der verfügbaren Akkukapazität die Gesamtdistanz von 22 km absolvieren zu können.

Das Ziel dieser Bachelorarbeit ist daher die Implementierung eines Tools zur Validierung des Akkus. Hierbei soll mithilfe des Tools eine Lastsituation für den Akku simuliert werden, woraufhin entsprechende Sensordaten ausgelesen und interpretiert werden können. Der Akku besitzt zwar Sensoren zur Temperatur-, Strom- und Spannungsmessung, jedoch war es bisher nicht möglich diese Daten auch entsprechend auswerten zu können. Mithilfe der Simulation sollen folgende Daten visualisiert werden:

- Effektives Stromprofil
- Spannungsverlauf
- Temperaturverlauf
- Ladezustand

Letztendlich soll es möglich sein, über die Simulationen feststellen zu können, wie sich der Akku unter verschiedenen Lastsituationen verhält, um die im Akku gespeicherte Energie beim Endurance and Efficiency Event optimal nutzen zu können.

Hierbei ist zu erwähnen, dass sich diese Arbeit auf die Softwareebene fokussiert. Die Fertigung der notwendigen Hardware ist nur bedingt Teil dieser Arbeit.

2. Grundlagen

2.1. Bus (Datenverarbeitung)

Zur effektiven Gestaltung eines beliebigen Prozesses müssen die Einheiten, welche den Prozess überwachen und steuern, untereinander Informationen austauschen können [10]. Wie von Schnell und Wiedemann erläutert, entsteht bei einer Verknüpfung von mehreren Einheiten ein Netzwerk. Es wird dabei zwischen unterschiedlichen geometrischen Anordnungsmöglichkeiten unterschieden. Eine davon ist die sogenannte Bus-Struktur, auch Linienstruktur genannt. Dabei kommunizieren alle Einheiten bzw. Teilnehmer über eine gemeinsame Leitung, wobei die Anbindung der Teilnehmer an das Buskabel über kurze Stichleitungen erfolgt, siehe Abbildung 2.1 [10].

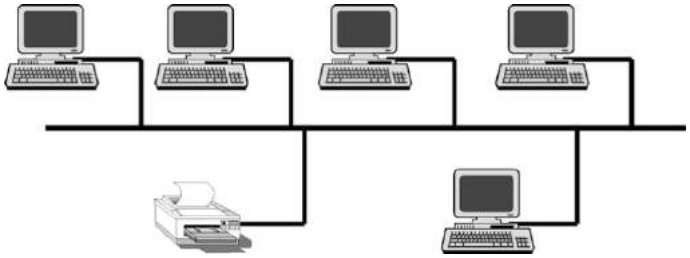


Abbildung 2.1.: Bus-Struktur [10]

Weiters beschreiben die Autoren, dass die Bus-Struktur den Vorteil eines vergleichsweise geringen Kabelaufwands verglichen zu anderen Netzwerktopologien aufweist. Jeder Teilnehmer benötigt nur eine Schnittstelle, um mit den anderen Teilnehmern kommunizieren zu können. Daraus entsteht jedoch das Problem, dass zu einem bestimmten Zeitpunkt immer nur ein Teilnehmer senden darf. Das Buszugriffsverfahren legt dieses Zugriffsrecht fest. Grundsätzlich unterscheidet man zwischen verschiedenen Buszugriffsverfahren. Für den in Kapitel 2.1.1 beschriebenen CAN-Bus ist das Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) Verfahren von Relevanz. Hierbei wird die Busleitung von einem sendewilligen Teilnehmer abgehört, welcher sendet, falls diese nicht belegt ist. Wenn die Busleitung belegt ist, wird die laufende Übertragung abgewartet und direkt danach mit der Sendung begonnen, wobei die Sendung andauernd überwacht wird. Für den Fall, dass zwei Teilnehmer zur selben Zeit senden wollen, sind Prioritäten vergeben. Der Teilnehmer mit der niedrigeren Priorität bricht in diesem Fall die Übertragung ab, wodurch Kollisionen vermieden werden und versucht seine Daten im Anschluss der laufenden Übertragung zu senden [10].

2.1.1. CAN-Bus

Der Controller Area Network (CAN)-Bus ist ein von Bosch im Jahre 1991 eingeführtes Bussystem, welches für Kraftfahrzeuge konzipiert wurde [11]. Wie von Reif beschrieben, hat sich der CAN-Bus mittlerweile als Standard in der Automobilindustrie etabliert und wird dort in verschiedenen Bereichen eingesetzt, wobei sich die Anforderungen an das Netzwerk in den Bereichen stark unterscheidet. Da aufgrund der schnellen Abläufe im Bereich des Motormanagements Informationen wesentlich schneller benötigt werden, als wie im Komfortbereich, werden Busse mit unterschiedlicher Datenrate eingesetzt. Hierbei wird zwischen Lowspeed- und Highspeed-CAN-Bussen unterschieden. Lowspeed-CAN (CAN-B) arbeitet mit einer Übertragungsrate von 5 bis 125 kbit/s, was für Anwendungsbereiche wie Sitzverstellung, Fensterheber oder ähnliches ausreichend ist. Highspeed-CAN (CAN-C) arbeitet mit einer Übertragungsrate von 125 kbit/s bis 1 Mbit/s, was den Datenübertragungsanforderungen des Antriebsstrangs gerecht wird. Des Weiteren findet der CAN-Bus in der Fahrzeugdiagnose Anwendung. Hierbei kann das Diagnosegerät direkt an den CAN-Bus angeschlossen werden, um für die Diagnose benötigte Informationen zu erhalten. Abbildung 2.2 stellt eine beispielhafte Vernetzung von Steuergeräten über CAN dar [11].

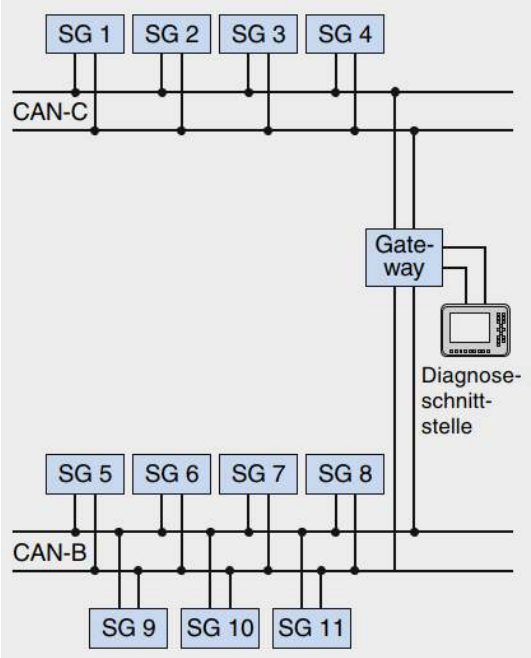


Abbildung 2.2.: Vernetzung von Steuergeräten im CAN [11]

In Abbildung 2.3 ist die Anbindung eines Teilnehmers an den Bus im CAN dargestellt. Reif beschreibt, dass ein derartiger Netzknoten aus einem CAN-Transceiver (Kombination aus Transmitter und Receiver), dem CAN-Controller und dem Mikrocontroller für die Anwendersoftware besteht. Der CAN-Controller erzeugt aus den zu übertragenden Daten den Bitstrom für die Datenkommunikation und leitet ihn über die TxD-Leitung an den Transceiver weiter. Der Transceiver verstärkt die Signale und erzeugt die notwendigen Spannungspegel für die differenzielle Datenübertragung und versendet den

aufbereiteten Bitstrom seriell auf der Busleitung. Umgekehrt werden eingehende Nachrichten vom Transceiver aufbereitet und über die RxD-Leitung an den CAN-Controller weitergeleitet. Der Mikrocontroller arbeitet das Anwenderprogramm ab und steuert den CAN-Controller. Des Weiteren stellt der Mikrocontroller die zu sendenden Daten bereit bzw. liest die empfangenen Daten aus. Die zwei Drähte des Bus werden mit CAN_High und CAN_Low bezeichnet, was eine symmetrische Datenübertragung ermöglicht. Dabei werden Bits über beide Leitungen unter Verwendung unterschiedlicher Spannungen übertragen, was die Empfindlichkeit gegen Gleichtaktstörungen verringert. Zur Kommunikation bzw. Übertragung der Informationsbits werden die Zustände dominant (übereinstimmend) und rezessiv (nachgebend) verwendet. Dabei stellt der dominante Zustand eine binäre „0“ und der rezessive eine binäre „1“ dar [11].

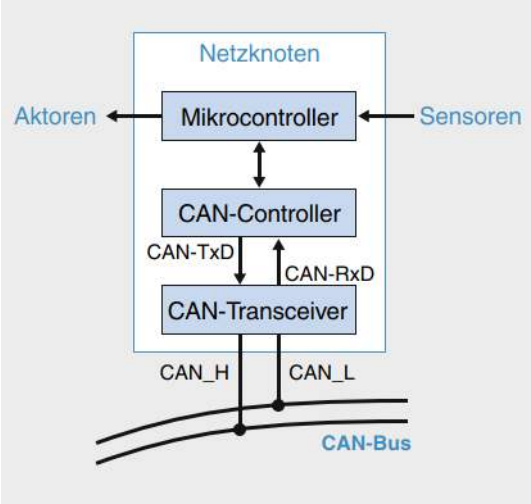


Abbildung 2.3.: Netzknoten im CAN [11]

Weiters beschreibt der Autor, dass im Gegensatz zu anderen Netzwerken von CAN nicht die einzelnen Netzknoten, sondern die übertragenen Nachrichten adressiert werden. Dabei verfügt jede Nachricht über eine eindeutige Kennung, dem Identifier, welcher den Inhalt der Botschaft kennzeichnet. Dadurch ist es einem Teilnehmer möglich, eine Botschaft an alle anderen Teilnehmer auszusenden. Die Teilnehmer verwerten daraufhin nur jene Daten, deren zugehöriger Identifier in der Liste der entgegenzunehmenden Botschaften gespeichert sind. Daher entscheidet jeder Teilnehmer selber, ob eine am Bus gesendete Nachricht benötigt wird oder nicht. Im Standardformat (CAN 2.0 A) besteht der Identifier aus 11 Bit und im erweiterten Format (CAN 2.0 B) aus 29 Bit. Somit können mit dem Standardformat 2048 und mit dem erweiterten Format 536000000 verschiedene CAN-Botschaften unterschieden werden. Zur Datenübertragung auf dem Bus wird ein Botschaftsrahmen, auch Frame genannt, aufgebaut. Dieser enthält Informationen zur Übertragung in einer festgelegten Reihenfolge [11].

2.2. Batterie/Akkumulator

2.2.1. Allgemein

Batterien sind elektrochemische Energiespeicher, bei denen zwischen Primär- und Sekundär Systemen unterschieden wird, wobei es sich bei einem Primär System um eine nicht wiederaufladbare und bei einem Sekundär System um eine wiederaufladbare Batterie handelt [12]. Wiederaufladbare Batterien werden auch Akkumulator genannt.

Die Kapazität eines Akkus wird in der Einheit [Ah] angegeben und beschreibt die Menge an elektrischer Ladung, welche von diesem unter spezifischen Entladebedingungen geliefert wird. Die Kapazität ist unter anderem abhängig vom Entladestrom, der Entladeschlussspannung und der Temperatur [13].

Leuthner beschreibt, dass abhängig von der Anwendung eine oder mehrere Akkuzellen verwendet werden, welche in Serie und/oder parallel verschaltet werden können. Bei einer Parallelschaltung addieren sich die einzelnen Ströme der Zellen bei gleichbleibender Spannung und bei einer Serienschaltung addieren sich die einzelnen Spannungen der Zellen bei gleichbleibendem Strom. Mehrere zusammengeschaltete Module ergeben ein Akkusystem, wie es unter anderem in der Automobilindustrie eingesetzt wird. Solche Akkusysteme verfügen über ein Akkumulatormanagementsystem (AMS), welches Sensorik zur Ermittlung von Strömen, Zellspannungen und Zelltemperaturen besitzt. Das AMS wird grundsätzlich zum An-, Abschalten und zum Thermomanagement des Akkusystems benötigt. Die Kommunikation zwischen dem Fahrzeug und dem Akku erfolgt meist über einen CAN-Bus [13].

2.2.2. Lithium-Ionen Akkumulator

Eine einzelne Lithium-Ionen Zelle liefert typischerweise eine Nominalspannung von 3,6 V, bei einer minimalen Entladeschlussspannung von 2,5 V und einer maximalen Ladeschlussspannung von 4,2 V [14].

Laut Zeyen und Wiebelt liegt die optimale Betriebstemperatur einer Lithium-Ionen Zelle zwischen 20 °C und 40 °C. In diesem Temperaturbereich weist eine Lithium-Ionen Zelle die höchste Leistugnsfähigkeit bei gleichzeitig tolerierbarem Alterungsverhalten auf [15]. Abbildung 2.4 zeigt das typische Entladeverhalten einer Lithium-Ionen Zelle bei unterschiedlichen Temperaturen, wobei ersichtlich ist, dass vor allem bei niedrigen Temperaturen die Kapazität stark limitiert ist.

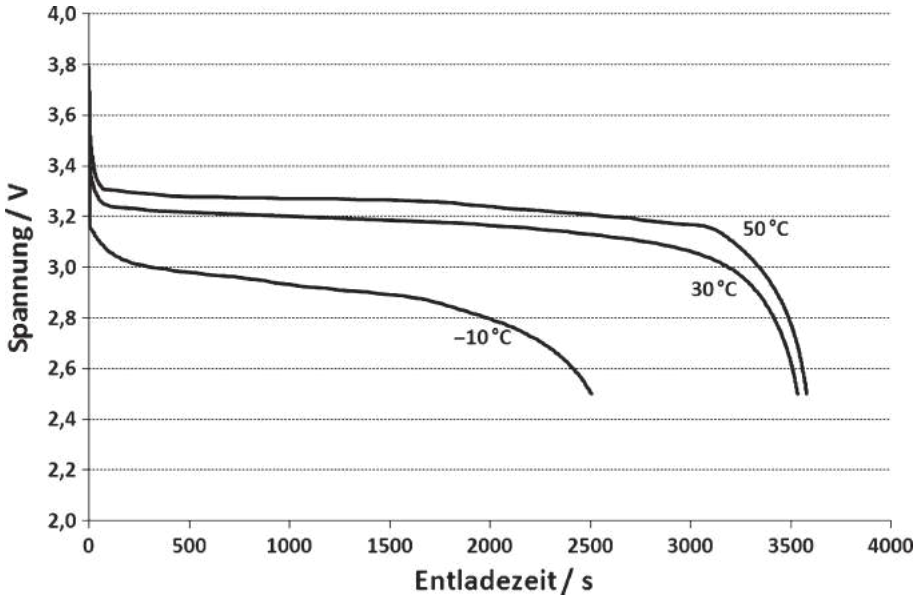


Abbildung 2.4.: Typische Entladekurve einer Lithium-Ionen Zelle bei unterschiedlichen Temperaturen und gleichem Entladestrom [3]

2.2.3. Ladezustandsermittlung

Der Ladezustand eines Akkus, besser bekannt als State Of Charge (SOC), beschreibt das Verhältnis zwischen der aktuell verbleibenden Kapazität $Q_{verbleibend}$ und der verfügbaren Kapazität $Q_{nominal}$ des Akkus welcher durch die Gleichung

$$SOC = \frac{Q_{verbleibend}}{Q_{nominal}} * 100\% \tag{2.1}$$

definiert ist [6]. Ein $SOC = 100\%$ entspricht dabei einem voll geladenen und ein $SOC = 0\%$ einem voll entladenem Akku.

Der SOC kann auf verschiedene Weisen ermittelt werden, wobei grundsätzlich zwischen direkten und modellbasierten Methoden unterschieden wird [7]. Pfeil erwähnt im Zusammenhang mit den direkten Methoden unter anderem den Restladungstest, die Ladungsintegration und die Spannungsauswertung, auf welche in [16] genauer eingegangen wird. Für die Ermittlung des SOC von Elektrofahrzeugen werden jedoch aus technischen und finanziellen Gründen modellbasierte Methoden bevorzugt. Laut Pfeil weisen diese vor allem hinsichtlich der Anwendbarkeit und Genauigkeit Vorteile gegenüber den direkten Methoden auf. Der Ansatz basiert hierbei auf Zustandsgleichungen, bei welchen die Messgrößen Strom und Spannung des Akkus kombiniert werden. Zu den bekanntesten modellbasierten Methoden zählt die Kalman-Filter (KF) Methode.

Wie von Murnane und Ghazel in [17] erwähnt, handelt es sich beim KF grundsätzlich um einen Algorithmus zur Abschätzung von inneren Zuständen eines beliebigen dynamischen Systems. Der KF bildet eine rekursive Lösung zur optimalen linearen Filterung von Zustandsbeobachtungs- und Vorhersageproblemen. Im Vergleich zu anderen

Schätzungsmethoden liefert der KF automatisch dynamische Fehlergrenzen für seine eigenen Zustandsschätzungen. Laut der Autoren ist der KF somit eine Zustandsschätzungsmehode mit integriertem Fehlerkorrekturmechanismus, über welchen Echtzeit-Vorhersagen für den SOC geliefert werden können.

Um eine exakte SOC-Schätzung auf Grundlage der KF Methode erhalten zu können, ist vorab ein passendes Akkumodell zu finden und eine Identifkation dessen Parameter notwendig. In [18] wird beschrieben, dass sich bei Elektrofahrzeugen aufgrund der simplen Struktur vor allem die Modellbildung mittels Ersatzschaltbildern bewährt hat. Das am häufigsten eingesetzte Ersatzschaltbild ist hierbei die RC-Schaltung. Diese kann aus mehreren RC-Gliedern ($i = 1, 2, 3, ...n$) bestehen, siehe Abbildung 2.5.

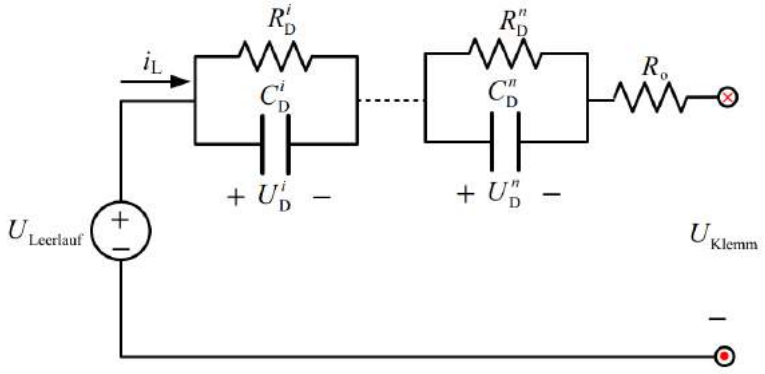


Abbildung 2.5.: Ersatzschaltbild eines Akkus mit n RC-Gliedern [18]

Sun et al. unterteilen das Ersatzschaltbild in drei Bereiche:

i $U_{Leerlauf}$ stellt die Spannungsversorgung durch den Akku dar

ii R_D beschreibt den Diffusionswiderstand und C_D die Diffusionskapazität, welche das dynamische Spannungsverhalten eines Akkus unter Last nachbilden

iii R_o ist ein ohmscher Widerstand welcher den Innenwiderstand der diversen Akkukomponenten darstellt

i_L bezeichnet den Strom, der unter Last aus dem Akku gezogen wird, welcher den Spannungsabfall U_D am RC-Glied zur Folge hat und in der Klemmspannung U_{Klemm} resultiert.

Die Anzahl der RC-Glieder im Ersatzschaltbild beeinflusst die Genauigkeit der SOC-Schätzung. Wie in [18] ermittelt, liefert ein Ersatzschaltbild mit zwei RC-Gliedern den besten Kompromiss aus Modellkomplexität und Genauigkeit.

Zur Identifikation der Parameter des Ersatzschaltbilds werden experimentelle Daten herangezogen. Hierbei werden meist durch wiederholte Pulsentladungen des Akkus bis zur Entladeschlussspannung die entsprechenden Messgrößen Strom und Spannung aufgezeichnet, aus welchen mittels der Methode der kleinsten Fehlerquadrate die gesuchten Systemparameter ermittelt werden können [17], [19].

Der KF nutzt letztlich die gewonnenen Systemparameter des Ersatzschaltbildes, um in Kombination mit den vom Akku erhaltenen Messgrößen Strom und Spannung den aktuellen SOC zu ermitteln.

2.3. Batteriesimulator

Batteriesimulatoren stellen eine leistungselektronische Nachbildung von elektrochemischen Speichereinheiten wie Lithium-Ionen Akkus dar [8]. Wie von Paulweber und Lebert beschrieben, dienen sie grundsätzlich zur Überprüfung und Kontrolle der Funktionsfähigkeit von elektrischen Geräten und kommen unter anderem für die Entwicklung und Validierung diverser Bestandteile von Elektroahrzeugen, wie dem Elektromotor oder dem Antriebsinverter, zum Einsatz [8]. Mithilfe eines Batteriesimulators kann je nach gegebener Hardwarekonfiguration ein Sollwert für Strom- oder Spannung eingestellt werden um eine entsprechende Last zu versorgen. Somit können beliebige Strom- bzw. Spannungsquellen simuliert werden. Die Autoren beschreiben weiters, dass ein Batteriesimulator mit Netzrückspeisefähigkeit auch zur Charakterisierung und Validierung von Fahrzeugakkus herangezogen werden kann. Daher kann ein Batteriesimulator als Quelle und Senke angesehen werden. Diese gegebene Bidirektionalität ermöglicht es daher auch für einen Fahrzeugakku eine Last zu simulieren.

3. Konzept

Die Implementierung des Validierungstools soll mithilfe der Programmierplattform MATLAB und dessen Blockdiagrammumgebung SIMULINK realisiert werden. Die Datenübertragung erfolgt über einen CAN-Bus. Die drei Teilnehmer des Bus sind der Rechner, auf welchem das Validierungstool ausgeführt wird, der Batteriesimulator und der Akku, siehe Abbildung 3.1.

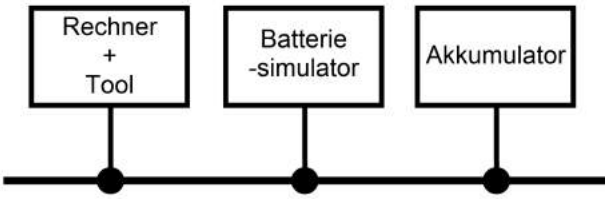


Abbildung 3.1.: Geplanter Aufbau des CAN-Bus

Für die Datenübertragung über den CAN-Bus werden die zu übermittelnden Daten beim Senden in Frames verpackt welche beim Empfangen durch einen Busteilnehmer wiederum zu entpacken sind. Um daher einerseits den Batteriesimulator bzw. den Akku ansprechen zu können und andererseits die vom Akku gesendeten Sensordaten interpretieren zu können, werden die notwendigen CAN-Nachrichten und deren Kodierung bzw. Dekodierung vorab definiert. Hierzu wird eine Data Base CAN (DBC)-Datei erstellt, welche die Informationen für die Übersetzung aller zu sendenden und empfangenden CAN-Nachrichten beinhaltet. Die DBC-Datei wird mithilfe des Programms BUSMASTER generiert.

Im nächsten Schritt werden die Stromprofile bestimmt, mit welchen der Akku in weitere Folge belastet wird. Einerseits wird ein Rechteckstromprofil definiert, welches zur Ermittlung der spezifischen Entladekurve des Akkus für die SOC-Berechnung benötigt wird. Und andererseits wird ein Stromprofil zur Fahrtsimulation für die spätere Akkuvalidierung festgelegt. Letzteres stellt den benötigten Strom vom Akku pro Zeiteinheit während einer Fahrt dar. Das Rechtecksignal wird direkt in SIMULINK generiert. Das Stromprofil für die Fahrtsimulation wird aus bestehenden Telemetriedaten von früheren Testfahrten des Rennwagens gewonnen und in SIMULINK eingelesen.

Um über den CAN-Bus kommunizieren zu können, wird die Vehicle Network Toolbox herangezogen, welche entsprechende MATLAB-Funktionen und SIMULINK-Blöcke beinhaltet. In Kombination mit der zuvor generierten DBC-Datei können Daten auf den Bus an die adressierten Teilnehmer gesendet und vom Bus empfangen und interpretiert werden. Das senden der Stromprofile und Empfangen der Sensordaten des Akkus erfolgt innerhalb des SIMULINK-Modells.

Zur Bestimmung des SOC wird die Unscented Kalman-Filter (UKF)-Methode herangezogen. Der UKF ist eine Erweiterung des KF für nicht lineare Probleme, wie jenes der Bestimmung des SOC eines Lithium-Ionen Akkus. Diese Methode zur SOC-Ermittlung eignet sich vor allem für Akkus von Elektrofahrzeugen, da im Vergleich zu anderen KF-Methoden trotz der hohen auftretenden Ströme eine ausreichende Genauigkeit erzielt wird [20]. Für die notwendige Modellierung des Akkus für den UKF wird ein RC-Ersatzschaltbild mit zwei RC-Gliedern gewählt. Die Modellierung des RC-Ersatzschaltbilds in Kombination mit der entsprechenden SOC-Berechnung erfolgt ebenfalls in SIMULINK.

Für eine einfachere Handhabung des Validierungstools wird ein Kontrollpanel in Form einer MATLAB App implementiert. Dieses soll Knöpfe zur Steuerung des Akkus, Batteriesimulators und SIMULINK-Modells beinhalten. Des weiteren soll zur Überwachung der Status des Akkus und des Batteriesimulators über das Kontrollpanel ausgegeben werden.

Die physische Verbindung zwischen dem Validierungstool und dem CAN-Bus erfolgt über einen CAN zu Universal Serial Bus (USB) Adapter. Die CAN-Bus Leitung mit entsprechenden Steckern zur Verbindung der drei Teilnehmer wird händisch in der Werkstatt des CTM gefertigt. Damit der Akku CAN-Nachrichten senden kann, muss dessen Steuereinheit von einer externen Spannungsquelle versorgt werden. Dies erfolgt über ein entsprechendes Netzteil.

Über den Batteriesimulator wird die Last für den Akku generiert. Dieser wird seitens ALPITRONIC zur Verfügung gestellt. Die Last wird dabei über die zuvor definierten Stromprofile vorgegeben. Der Batteriesimulator wird einerseits über ein Leistungskabel mit dem Akku und andererseits mit einem weiteren Leistungskabel mit dem Netz verbunden, um die im Akku gespeicherte Energie abführen zu können.

Mithilfe des vorliegenden Systemaufbaus können die Parameter des Ersatzschaltbildes für die SOC-Berechnung ermittelt werden. Hierbei wird das Rechteckstromprofil über das zuvor implementierte Validierungstool an den Batteriesimulator gesendet, welches eine gleichmäßige Pulsentladung des Akkus zur Folge hat. Aus den gewonnenen Messwerten für Strom und Spannung können daraufhin die Parameter des RC-Ersatzschaltbildes berechnet werden welche in die SOC-Berechnung des bestehenden SIMULINK-Modells implementiert werden können.

Durch Übermittlung eines der aus den Telemetriedaten gewonnen Stromprofile kann letztlich eine Fahrt für den Akku simuliert werden. Die aus der Simulation resultierenden Sensordaten des Akkus werden über das Validierungstool empfangen und in SIMULINK grafisch dargestellt. Dabei werden das effektive Stromprofil, die Zellspannungen pro Akkusegment und der Temperaturverlauf pro Akkusegment in Echtzeit grafisch dargestellt. Gleichzeitig wird durch die empfangenen Sensordaten der SOC über den zuvor implementierten UKF ermittelt, welcher ebenfalls in Echtzeit grafisch dargestellt wird. Um die empfangenen Sensordaten zu einem späteren Zeitpunkt analysieren zu können, besteht die Möglichkeit diese über den Data Inspector in SIMULINK in einer

MAT-Datei abzuspeichern. Dabei stellt das MAT-Format eine MATLAB spezifische Datei dar, in welcher in MATLAB generierte Werte gespeichert werden können.

Da MATLAB und SIMULINK als Desktopsprache Englisch verwenden, werden bei der Implementierung des Validierungstools aus Konsistenzgründen Variablennamen, Blockbezeichnungen etc. ebenfalls in englischer Sprache definiert. Somit wird beispielsweise das Stromprofil für die Fahrtsimulation in SIMULINK als “currentprofile" bezeichnet.

4. Umsetzung

4.1. Definiton der CAN-Nachrichten

Um Daten zwischen dem Validierungstool, dem Batteriesimulator und dem Akku über den CAN-Bus austauschen zu können, müssen diese vor der Übertragung in eine CAN-Nachricht verpackt werden. Die Form in welche die zu versendenden Daten verpackt werden, wird am Mikrocontroller der einzelnen Busteilnehmer bzw. dem Rechner definiert. Um die Definition der CAN-Nachrichten nicht mehrmals durchführen zu müssen, werden diese in einer Datenbank in Form einer DBC-Datei gebündelt. Diese enthält Informationen wie Nachrichten von einem Sender zu kodieren und einem Empfänger zu dekodieren sind. Im Falle des Validierungstools müssen am CAN-Bus Daten zu folgenden Informationen und Funktionen ausgetauscht werden:

- Steuerung des Batteriesimulators
- Status des Batteriesimulators
- Sensordaten des Batteriesimulators
- Steuerung des Akkus
- Status des Akkus
- Sensordaten des Akkus

Die Erstellung der Datenbank erfolgt mit dem Programm BUSMASTER. Hierbei ist zunächst über den Reiter “Tools" und der Funktion “CAN DBF Editor" eine DBF-Datei zu erstellen, welche eine BUSMASTER spezifische Datenbankdatei darstellt. Durch einen Rechtsklick auf das gelbe Datenbanksymbol kann eine Nachricht definiert werden, siehe Abbildung 4.1.

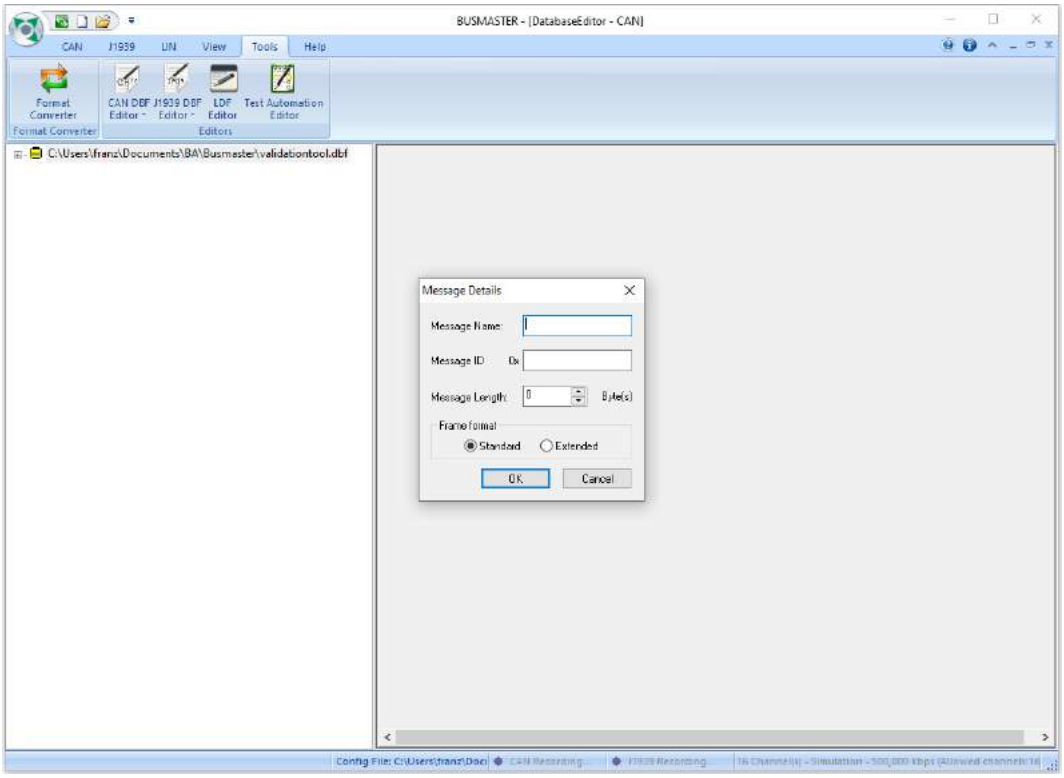


Abbildung 4.1.: Erstellen einer CAN Datenbank in BUSMASTER

Beim erstellen der Nachricht sind der Name, der Identifier, die Nachrichtenlänge und das Frame Format zu definieren. Der Identifier wird hierbei als Hexadezimalzahl vergeben und die Nachrichtenlänge kann maximal 8 Byte betragen. Beim Frame Format kann zwischen “Standard" und “Extended" gewählt werden wobei für das Validierungstool das Standard Frame Format verwendet wird. Innerhalb dieser Nachricht kann nun dessen Aufbau definiert werden, in dem einzelnen Signale hinzugefügt werden. Aus Effizienzgründen wird grundsätzlich angestrebt das 8 Byte lange Datenfeld einer CAN-Nachricht vollständig aufzufüllen. Daher kann beispielsweise eine einzelne Nachricht mehrere Informationen in Form von unterschiedlichen Signalen beinhalten, siehe Abbildung 4.2.

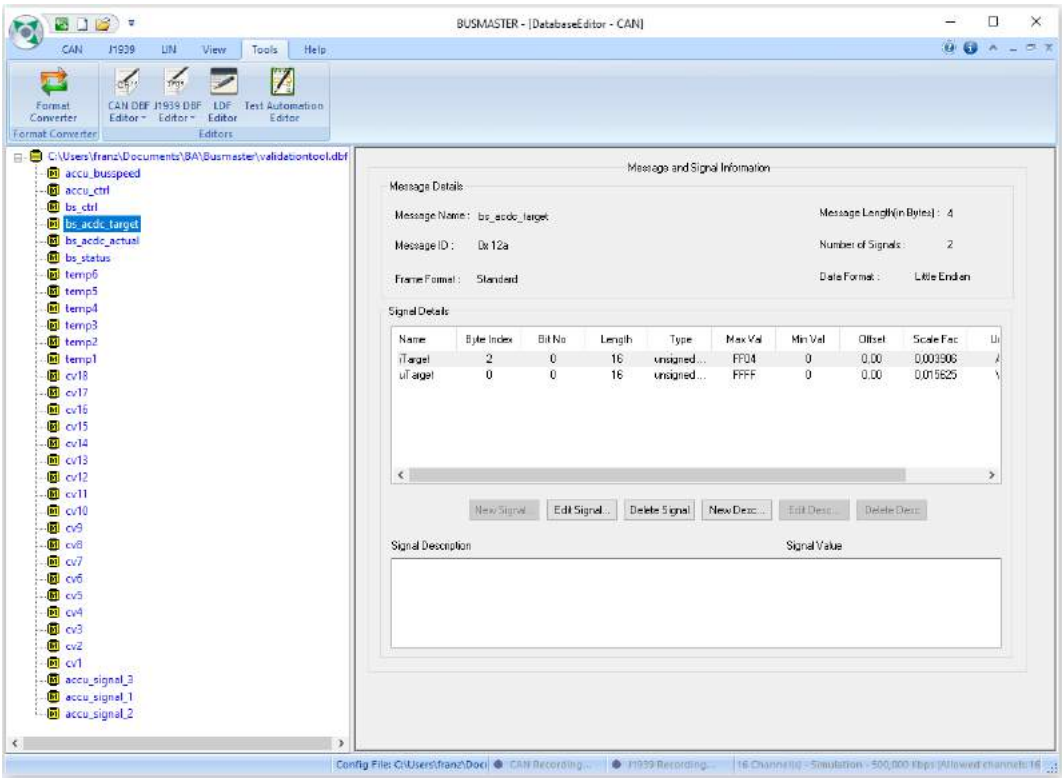


Abbildung 4.2.: Definition einer CAN-Nachricht in BUSMASTER

Bei mehreren Signalen innerhalb einer CAN-Nachricht muss deren Position innerhalb des Datenfelds definiert werden. Dies erfolgt über die Spalten “Byte Index" bzw. “Bit No". Die Länge des Signals ist unter “Length" zu definieren. In Spalte “Type" wird der Datentyp angegeben. Die Spalten “Max Val" und “Min Val" definieren den Bereich, über welchen das Signal dargestellt werden soll. “Offset" und “Scale Fac" bestimmen die Verschiebung und Schrittweite innerhalb des Datenfelds. Mit “Unit" kann eine Einheit vergeben werden und “Byte Order" definiert die Byte-Reihenfolge, wobei zwischen Little Endian (Intel) und Big Endian (Motorola) gewählt werden kann.

Die Identifier und der Aufbau der Nachrichten für die Kommunikation mit dem Akku werden einer bestehenden DBC-Datei entnommen, welche in der Vergangenheit für die CAN-Kommunikation des Vorjahresrennwagens erstellt wurde. Hierbei werden nur die notwendigen Definitionen für die Kommunikation mit dem Akku übernommen.

Die Identifier und der Aufbau der Nachrichten für die Kommunikation mit dem Batteriesimulator wird dem Programm entnommen, welches auf dessen Mikrokontroller läuft. Das Programm wurde seitens ALPITRONIC in Form einer C-Datei zur Verfügung gestellt. Tabelle 4.1 listet alle Signale auf, welche für die CAN-Kommunikation benötigt werden.

Tabelle 4.1.: Übersicht der notwendigen Signale für die CAN-Kommunikation

Identifier	Teilnehmer	Signal
0x10a	Batteriesimulator	enable
0x11a	Batteriesimulator	status
0x12a	Batteriesimulator	uTarget iTarget
0x13a	Batteriesimulator	uActual iActual
0x200- 0x211	Akku	cv1_1- cv_18_8
0x220- 0x225	Akku	ct1_1- ct6_8
0x240	Akku	max_cv min_cv
0x241	Akku	SDC_closed AMS_OK
0x242	Akku	i_low i_high
0x301	Akku	anti_timeout

Nachricht 0x10a besteht aus 1 Byte und dient zur Ansteuerung des Batteriesimulators, wobei dieser beim Empfang der Nachricht mit einer 02 im Datenfeld angeschaltet, mit einer 01 zurückgesetzt und einer 00 ausgeschaltet wird. Nachricht 0x11a besteht ebenfalls aus 1 Byte und gibt den aktuellen Status des Batteriesimulators aus. Das Empfangen des Werts 06 entspricht einem angeschalteten und 03 einem ausgeschalteten Zustand. Über Nachricht 0x12a erfolgt die Regelung des Batteriesimulators. An diese Adresse sind in weiterer Folge die Stromprofile zu senden. Die Nachricht besteht aus 4 Byte, wobei über die ersten beiden Byte die Spannung und über die letzten beiden der Strom geregelt werden kann. Da für das Validierungstool nur die Stromregelung von Relevanz ist, wird über die Bytes für die Spannungsvorgabe konstant eine 00 versendet. Nachricht 0x13a besteht ebenfalls aus 4 Byte und gibt die vom Strom- bzw. Spannungssensor des Batteriesimulators aktuell gemessenen Werte aus. Hierbei sind wiederum die ersten beiden Bytes für die Spannung und die letzten beiden für den Strom. Die Nachrichten 0x200 bis 0x211 geben die aktuellen Zellspannungen und 0x220 bis 0x225 die entsprechenden Zelltemperaturen des Akkus aus. 0x240 gibt jeweils die aktuell niedrigste und höchste Zellspannung aus. 0x241 gibt den aktuellen Status des Akkus aus, wobei sich AMS_OK auf den Zustand des Akkumulatormanagementsystems und SDC_closed auf den Zustand des Shutdown Circuit (SDC) bezieht. Der Shutdown Circuit ist ein Sicherheitskreis, welcher grundsätzlich eine über das ganze Fahrzeug verlaufende Aneinanderreihung von Relais darstellt und unter anderem über den Akku verläuft. Diese Relais sind grundsätzlich normal geschlossen und werden bei Störungen bzw. Gefahr durch ein auslösen entsprechender Logiken und Schalter geöffnet, was zu einer Abschaltung des Akkus führt. Das AMS überwacht

die Ströme, Spannungen und Temperaturen innerhalb des Akkus und öffnet den SDC, sobald eine Überschreitung von vordefinierten Grenzwerten erfolgt. Beide Signale können entweder den Zustand 0 oder 1 annehmen, wobei 1 einen betriebsbereiten Zustand impliziert und 0 in beiden Fällen eine Deaktivierung des Akkus zur Folge hat. Die Nachricht 0x242 gibt den aktuell gemessenen Strom des Akkus aus, wobei i_low einen Bereich von −75 A bis +75 A und i_high einen Bereich von −750 A bis +750 A abdeckt. Die Nachricht 0x301 dient zur Ansteuerung des Akkus und besteht aus 8 Byte. Hierbei ist nur das zweite Byte von Relevanz, über welches periodisch ein Hexadezimalwert von 80 übermittelt werden muss, um den Akku zu aktivieren und betriebsbereit zu halten. Falls diese Nachricht für eine Periode von 500 ms nicht an den Akku gesendet wird, öffnet das AMS den SDC.

Nach dem Einlesen aller benötigten Nachrichten in die Datenbank kann die vorliegende DBF-Datei über die Funktion “Format Converter" für die weitere Verwendung in MATLAB in eine DBC-Datei konvertiert werden, siehe Abbildung 4.3.

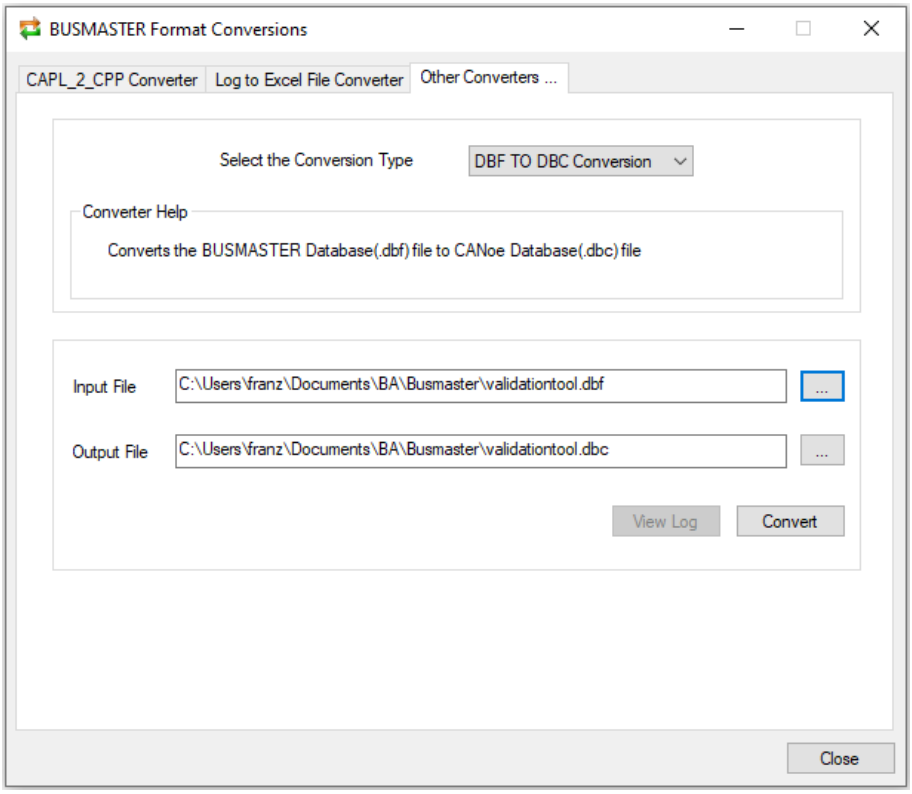


Abbildung 4.3.: Konvertierung vom DBF zum DBC Format

Die resultierende DBC-Datei ist Anhang A zu entnehmen.

4.2. Definition der Stromprofile

4.2.1. Entladekurve des Akkus

Für die Implementierung der SOC-Berechnung mittels des UKF sind für das dafür benötigte RC-Ersatzschaltbild vorab die jeweiligen Parameter zu bestimmen. Da das RC-Ersatzschaltbild das Entladeverhalten des realen Akkus nachbilden soll, muss die Charakteristik des realen Akkus ermittelt werden, um daraus die Parameter des RC-Ersatzschaltbildes berechnen zu können. Hierfür ist ein Entladetest am realen Akku durchzuführen. Dabei ist der Akku vom voll geladenen Zustand bis zur Entladeschlussspannung zu entladen. Für den Zweck dieser Arbeit wird eine Pulsentladung durchgeführt. Hierbei wird der Akku in 10% Schritten seiner Kapazität mit einem vorzugsweise geringen Strom entladen. Zwischen den Entladungen erfolgen entsprechende Ruhephasen. Im Vergleich zu einer konstanten Entladung ohne Ruhephasen ergibt sich der Vorteil einer geringeren Temperaturentwicklung innerhalb des Akkus. Durch die Wahl eines geringen Entladestroms im Verhältniss zur Akkukapazität wird auch der Spannungsabfall unter Last verringert. Unter Last fällt die Klemmspannung unter die tatsächlich vorherrschende Zellspannung, welche nach dem Entfernen der Last in Form der Leerlaufspannung sichtbar wird. In den Ruhephasen kann daher die Leerlaufspannung gemessen werden, was unter Last nicht möglich ist.

Der Akku welcher in weiterer Folge validiert werden soll, besitzt eine Kapazität von 12 Ah. Für den Entladestrom wird ein Wert von 6 A gewählt. 10% der Akkukapazität entsprechen 1,2 Ah. Um 10% der Kapazität zu entladen, muss der Akku somit über 12 min mit einem Strom von 6 A entladen werden. Die Ruhephasen werden mit dem selben Zeitintervall definiert. Das resultierende Rechteckstromprofil ist in Abbildung 4.4 dargestellt.

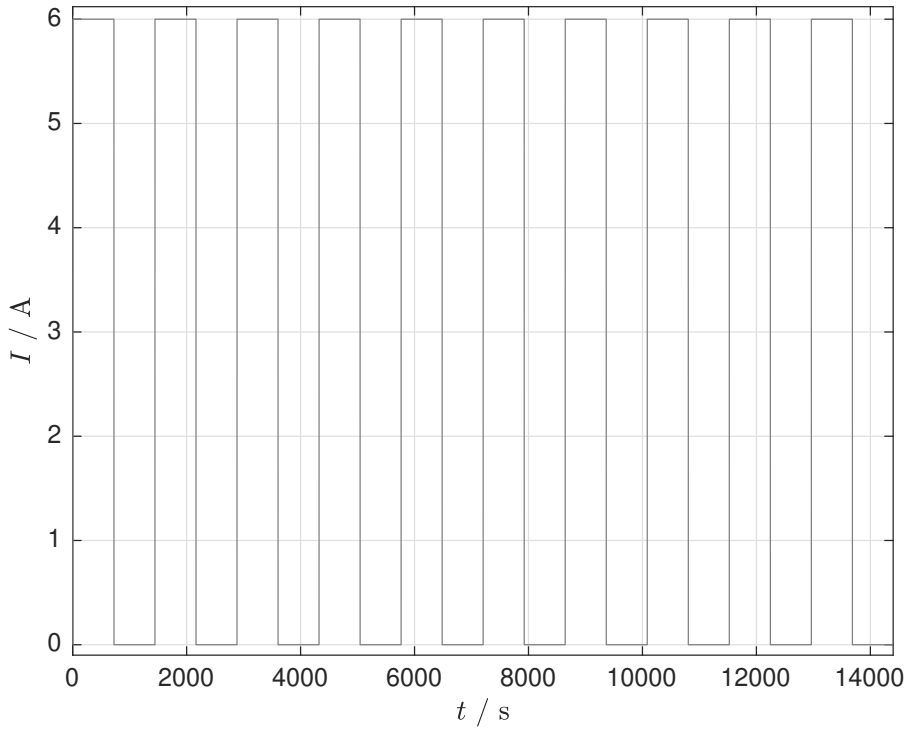


Abbildung 4.4.: Rechteckstromprofil zur Pulsentladung des Akkus

4.2.2. Fahrtsimulation

Um die erwünschte Validierung als Vorbereitung auf das Endurance and Efficiency Event durchführen zu können, ist ein Stromprofil zu wählen welches die Belastung des Akkus während einer Fahrt Wiederspiegelt. Das Team von CTM stellt hierfür eine Log-Datei im CSV-Format zur Verfügung [21]. Diese Datei wurde während einer Testfahrt mit dem Vorjahresrennwagen mithilfe eines Datenloggers generiert und enthält diverse Messdaten, wie den aktuellen Strom und die aktuelle Spannung des Akkus wie auch den zurückgelegten Weg. Da während der Testfahrt, in welcher die Messdaten aufgenommen wurden, das Fahrzeug des öfteren angehalten wurde und der Datenlogger in diesen Stehzeiten aktiv war, ist nur jener Teil der Messdaten brauchbar, welcher während des Fahrens aufgezeichnet wurde. Abbildung 4.5 zeigt den gewählten Ausschnitt der Messdaten.

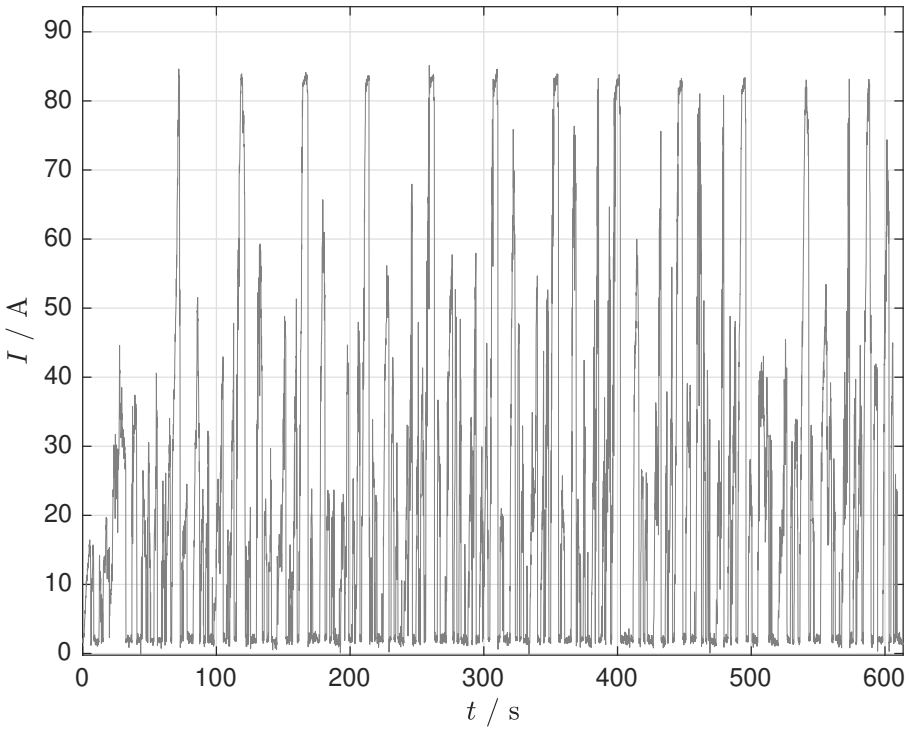


Abbildung 4.5.: Gewähltes Stromprofil aus Messdaten einer Testfahrt [21]

Laut Messdaten entspricht das dargestellte Stromprofil einer zurückgelegten Distanz von 7,3 km, was einem drittel der im Endurance and Efficieny Event zurückzulegenden Strecke entspricht. Durch ein dreifaches Aneinanderreihen der gewählten Messdaten erhält man ein für die gegebenen Erfordernisse angemessenes Fahrtprofil.

Bei der in der Log-Datei aufgenommenen Fahrt wurde die Leistung des Fahrzeuges auf 75% limitiert. Da davon auszugehen ist, dass beim Endurance und Efficiency Event eine Leistungsbegrenzung des Fahrzeuges notwendig sein wird, stellt das gewählte Stromprofil einen brauchbaren Ausgangspunkt für die Akkuvalidierung dar.

4.3. SIMULINK-Modell

4.3.1. Generierung der Stromprofile

Um die zuvor definierten Stromprofile in der Simulation verwenden zu können, sind diese in SIMULINK zu implementieren. Das Rechteckstromprofil zur Pulsentladung des Akkus wird über den Pulse Generator-Block generiert, siehe Abbildung 4.6.

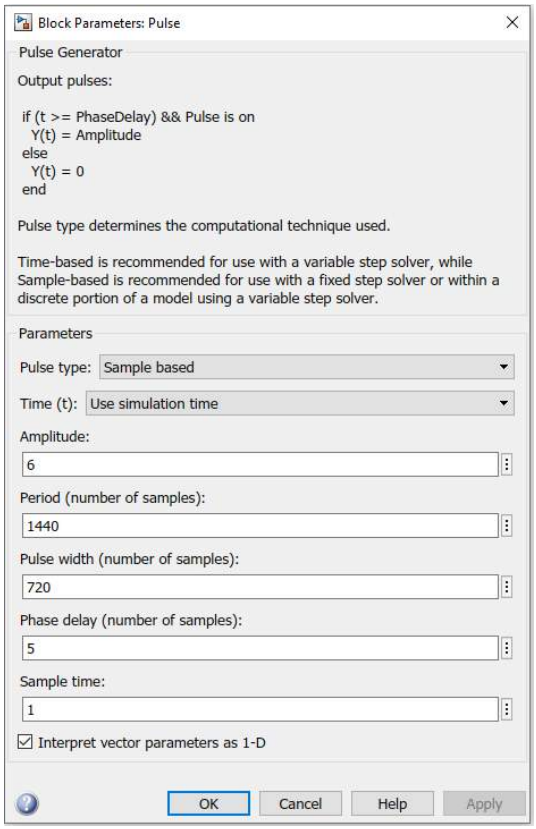


Abbildung 4.6.: Generierung des Rechteckstromprofils in SIMULINK

Das Einlesen des Stromprofils zur Fahrtsimulation erfolgt mit der Importfunktion von MATLAB. Hierbei werden die jeweiligen Spalten der Log-Daten in separaten Spaltenvektoren im MATLAB Worksapce gespeichert. Die Vektoren für die Zeit und für den Strom werden zu einer Matrix mit dem Namen “currentprofile" zusammengeführt, welche daraufhin in SIMULINK mit dem From Workspace-Block zu einem SIMULINK-Signal umgewandelt wird, siehe Abbildung 4.7.

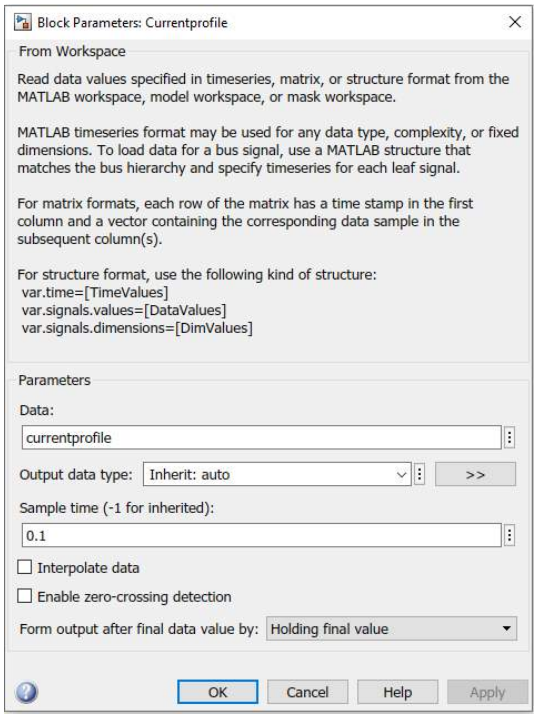


Abbildung 4.7.: Einlesen des Stromprofils der Fahrtsimulation in SIMULINK

Da das Zeitintervall zwischen zwei Messwerten in der Log-Datei 100 ms entspricht, wird für die Sample time 0,1 gewählt. Die Sample time definiert hierbei, in welchem Zeitintervall die eingelesene Matrix ausgelesen wird.

4.3.2. CAN-Kommunikation

Um eine Verbindung mit dem CAN-Bus aufbauen zu können, wird die Vehicle Network Toolbox verwendet. Die SIMULINK-Blöcke für die CAN-Kommunikation sind in Abbildung 4.8 dargestellt.

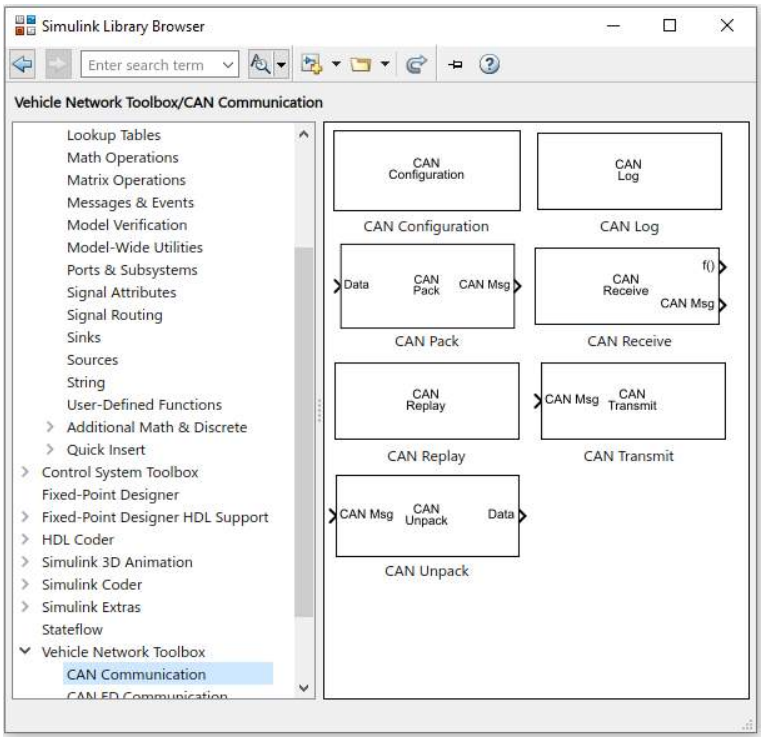


Abbildung 4.8.: SIMULINK-Blöcke für die CAN-Kommunikation

Zum Aufbau einer Verbindung zwischen dem Rechner und dem Bus bedarf es einer entsprechenden Schnittstelle. Diese Verbindung wird über einen CAN zu USB Adapter von Peak-System bewerkstelligt, siehe Abbildung 4.9.

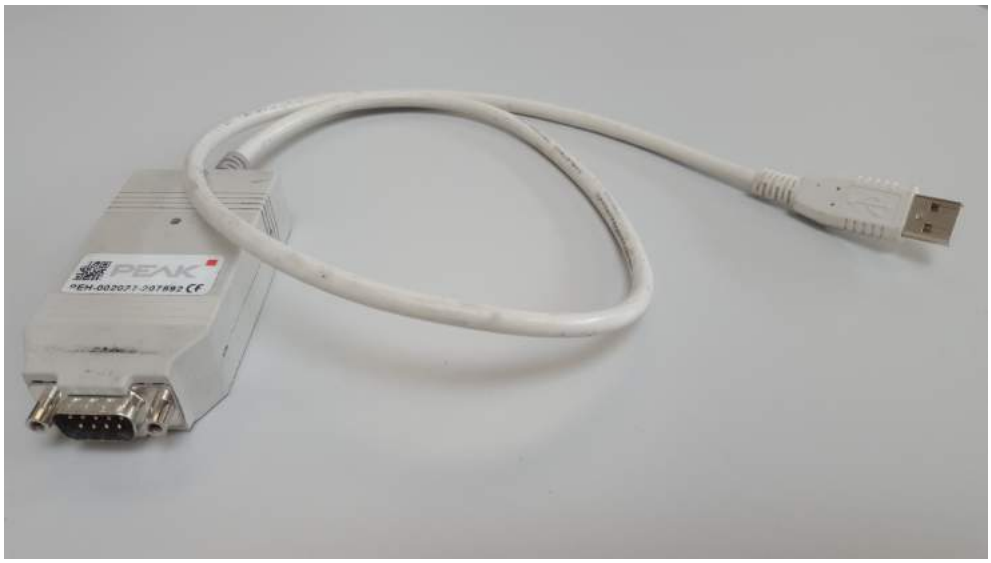


Abbildung 4.9.: Peak-System CAN zu USB Adapter

Die Definition der Schnittstelle in SIMULINK erfolgt dabei über den CAN Configuration-Block, siehe Abbildung 4.10. Bei Installation des Peak-System CAN interface support für MATLAB und der entsprechenden Treiber des CAN zu USB Adapters, scheint beim Verbinden des Adapters mit dem Rechner in diesem Block ein Peak-System Kanal auf, über welchen CAN-Nachrichten von SIMULINK auf den Bus gesendet und von diesem empfangen werden können. Des Weiteren kann an dieser Stelle auch die Bus-

geschwindigkeit definiert werden. Da der Batteriesimulator und der Akku mit einer Busgeschwindigkeit von 250 kbit/s arbeiten, wird an dieser Stelle der selbe Wert definiert.

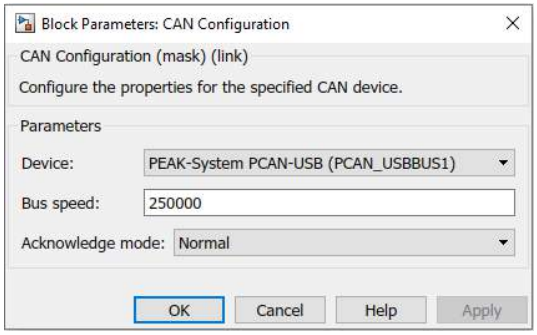


Abbildung 4.10.: CAN Configuration-Block

Die Übermittlung des Stromprofils an den Batteriesimulator soll Schrittweise erfolgen, d.h. es soll pro Zeitschritt ein einzelner Stromwert gesendet werden, mit welchem der Akku für das gegebene Zeitintervall belastet wird. Daher soll jede gesendete Nachricht einen einzelnen Stromwert enthalten. Die Generierung der Nachrichten erfolgt mit dem CAN Pack-Block, siehe Abbildung 4.11. In diesem-Block wird die zuvor generierte DBC-Datei eingelesen, wodurch aus den darin definierten Nachrichten gewählt werden kann. Zur Übermittlung der Stromwerte an den Batteriesimulator ist die Nachricht bs_acdc_target zu wählen. Das Stromprofil wird auf der Eingangsseite des Blocks mit dem iTarget Port verbunden. Der uTarget Port wird über einen Constant-Block mit konstant 0 V versorgt.

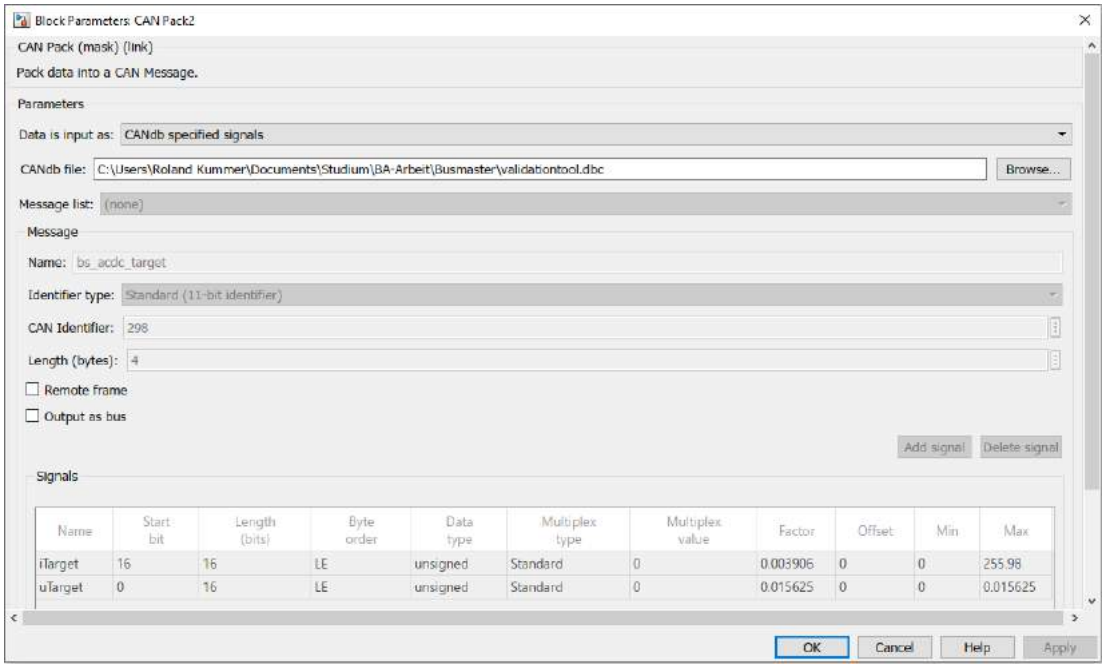


Abbildung 4.11.: CAN Pack-Block

Um die einzelnen Nachrichten auf den Bus senden zu können wird der CAN-Transmit-Block verwendet, in welchem die im CAN Configuration-Block definierte Schnittstelle und das Zeitintervall zwischen den einzelnen Nachrichten definiert wird, siehe Abbil-

dung 4.12. Das Zeitintervall wird mit 100 ms, festgelegt da dieses der Schrittweite des Stromprofils für die Fahrtsimulation entspricht.

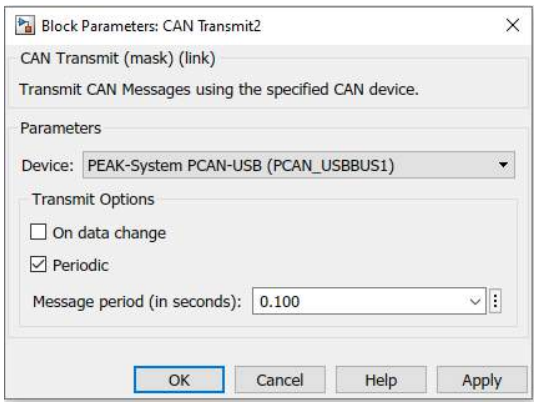


Abbildung 4.12.: CAN Transmit-Block

Um Sensordaten vom Akku über den Bus empfangen zu können, wird der CAN Receive-Block verwendet. In diesem wird definiert, über welche Schnittstelle Nachrichten empfangen werden sollen. Hier ist wiederum der Peak-System Kanal zu wählen. Des Weiteren können über diesen Block auch Nachrichten nach Identifier gefiltert und die Anzahl der zu empfangenden Nachrichten pro Zeitintervall definiert werden, siehe Abbildung 4.13. Im Filter werden die in der DBC-Datei definierten Identifier angegeben. Das Zeitintervall wird mit 100 ms definiert. Da im Feld für den Identifierfilter nur Dezimalzahlen eingegeben werden können, müssen die in Hexadezimal definierten Identifier ins Dezimal System umgerechnet werden.

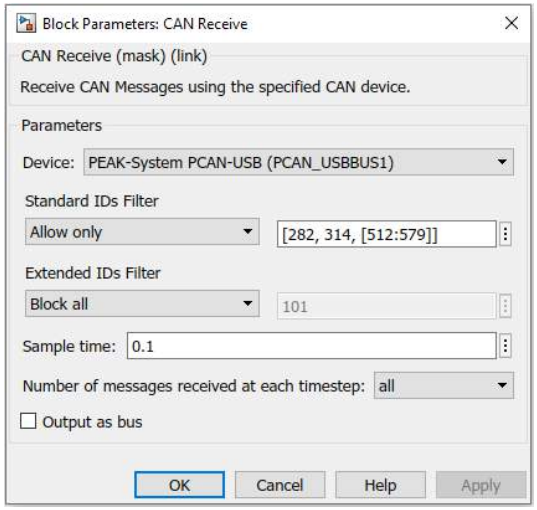


Abbildung 4.13.: CAN Receive-Block

Damit die empfangenen Nachrichten in ein SIMULINK-Signal übersetzten werden können, kommt der CAN Unpack-Block zum Einsatz. Wie im CAN Pack-Block wird auch hier die DBC-Datei definiert, um beim Empfangen von in der Datei definierten Nachrichten, diese in ein SIMULINK-Signal übersetzen zu können, siehe Abbildung 4.14.

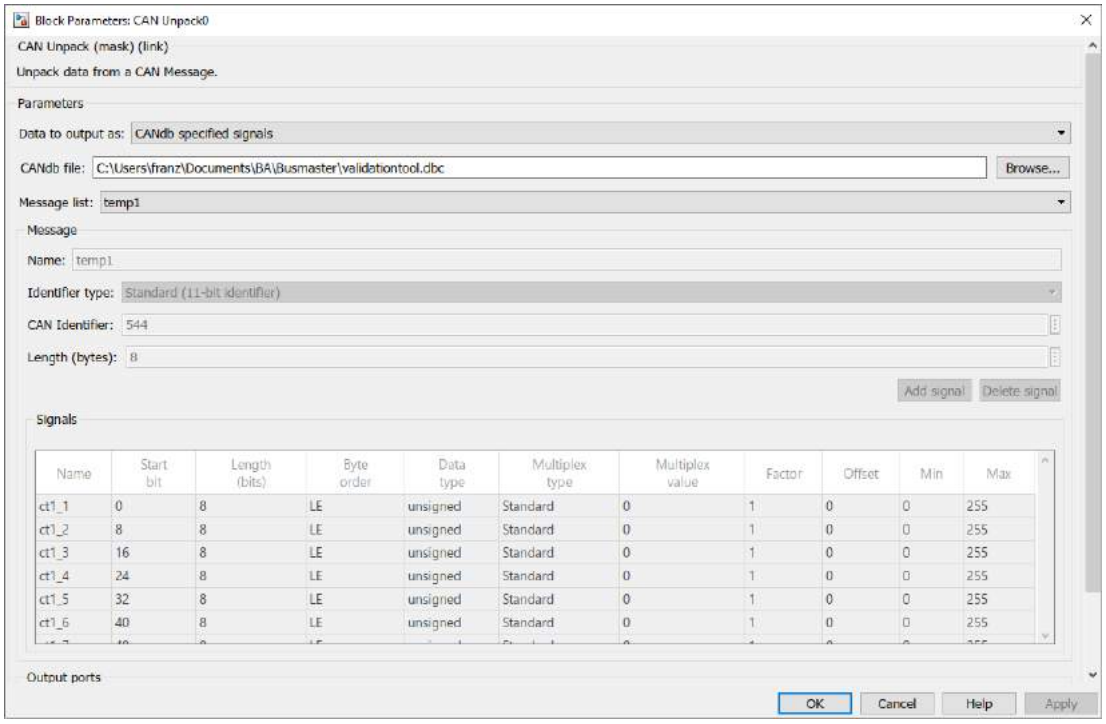


Abbildung 4.14.: CAN Unpack-Block

Da pro Block nur eine Nachricht empfangen werden kann und laut DBC-Datei in Summe 27 Nachrichten mit Sensorwerten bzw. Statuswerten empfangen werden sollen, muss für jede Nachricht ein eigener CAN Unpack-Block eingefügt werden. Um in dem vom CAN Receive-Block definierten Zeitintervall Nachrichten entpacken zu können sind die CAN Unpack-Blöcke in ein Function-Call-Subsystem zu integrieren. Das Subsystem erhält einen Funktionstrigger sobald vom CAN Transmit-Block Nachrichten empfangen wurden, woraufhin die jeweiligen CAN Unpack-Blöcke ausgeführt werden. Die Anwendung des Function-Call-Subsystems ist in Abbildung 4.15 dargestellt.

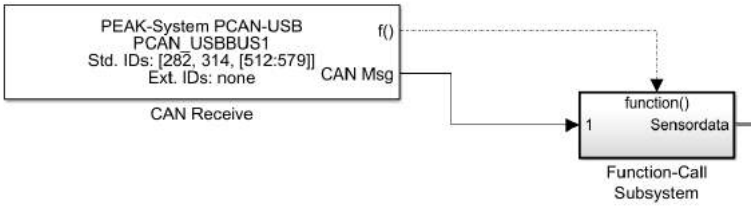


Abbildung 4.15.: Function-Call-Subsystem

Wie der DBC-Datei zu entnehmen ist, werden vom Akku 144 Spannungsmessungen und 48 Temperaturmessungen entgegengenommen. Der Akku wird intern in 6 Segmente unterteilt. Während der Simulation ist es ausreichend, jeweils die niedrigste Zellspannung und höchste Zelltemperatur pro Segment darzustellen. Hierfür wird nach den CAN Unpack-Blöcken ein MinMax-Block verwendet, welcher je nach Erfordernis von den ihm gegebenen Eingangssignalen jeweils den niedrigsten oder höchsten Wert ausgibt. Um die Sensordaten besser handhaben zu können, werden diese mithilfe des BusCreator-Blocks gebündelt und aus dem Function-Call-Subsystem geführt, siehe Abbildung 4.16. Der BusCreator ist in dem Fall wie ein Multiplexer (Mux) zu sehen,

wobei der BusCreator den Vorteil bietet in weiterer Folge mit dem BusSelector einzelne Signale wählen zu können, was mit einem Demux nicht möglich ist.

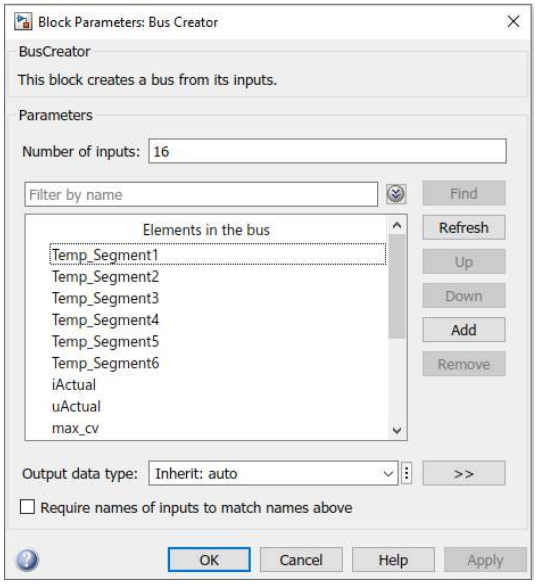


Abbildung 4.16.: BusCreator-Block

Um Daten in Echtzeit auf den Bus senden und vom Bus empfangen zu können, muss die Simulationsgeschwindigkeit eingestellt werden. Diese kann über den Reiter “Simulation" und durch Klicken auf den Pfeil unter dem “Run" Knopf mit der Option “Simulation Pacing" angepasst werden. Durch setzen des Schiebereglers auf 1, wird beim Start der Simulation diese in Echtzeit ausgeführt, siehe Abbildung 4.17.

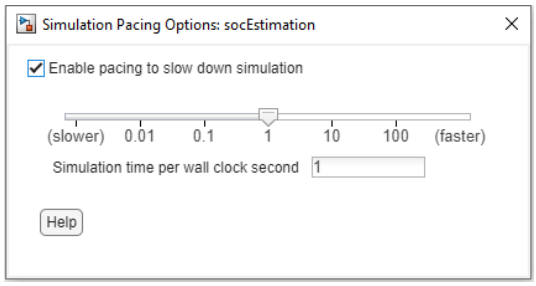


Abbildung 4.17.: Einstellen der Simultionsgeschwindigkeit

4.3.3. SOC-Berechnung

Die SOC-Berechnung erfolgt in einem eigenen Subsystem des SIMULINK-Modells. Dieses erhält die vom BusCreator-Block gebündelten Sensordaten, welche mit dem BusSelector innerhalb des SOC-Subsystems entpackt werden. Hierbei werden die Sensordaten iActual und min_cv gewählt. iActual liefert die Stromsensormesswerte vom Batteriesimulator. Diese Strommessung wird jener des Akkus vorgezogen, da diese mit einem Messfehler von 0,8 A im Vergleich zum Messfehler von 1 A des Akkustromsensors eine geringere Abweichung aufweist. min_cv gibt die aktuell niedrigste Zellspannung des Akkus aus.

Als Grundlage für die Implementierung der SOC-Berechnung dient ein von MathWorks zur Verfügung gestelltes Akkumodell, welches unter anderem eine SOC-Berechnung mittels einem UKF beinhaltet [22]. Die Implementierung des UKF erfolgt über den entsprechenden Block aus der Control System Toolbox.

Der UKF-Block benötigt zwei Funktionen als Argumente. Eine Zustandsübergangsfunktion und eine Messfunktion. Die Zustandsübergangsfunktion berechnet die Entwicklung der Zustände basierend auf dem Gegenwärtigen Eingangssignal und die Messfunktion berechnet das Ausgangssignal als Funktion der Zustände und des Eingangssignals. In Kombination mit dem UKF ermitteln beide Funktionen eine Zustandsschätzung für den gegenwärtigen Zeitschritt. Diese Funktionen werden in separaten Subsystemen definiert, jeweils mit dem vom BusSelector kommenden Stromsignal verbunden und im entsprechenden Feld des UKF-Block angegeben. Die Maske des UKF-Blocks ist in Abbildung 4.18 dargestellt.

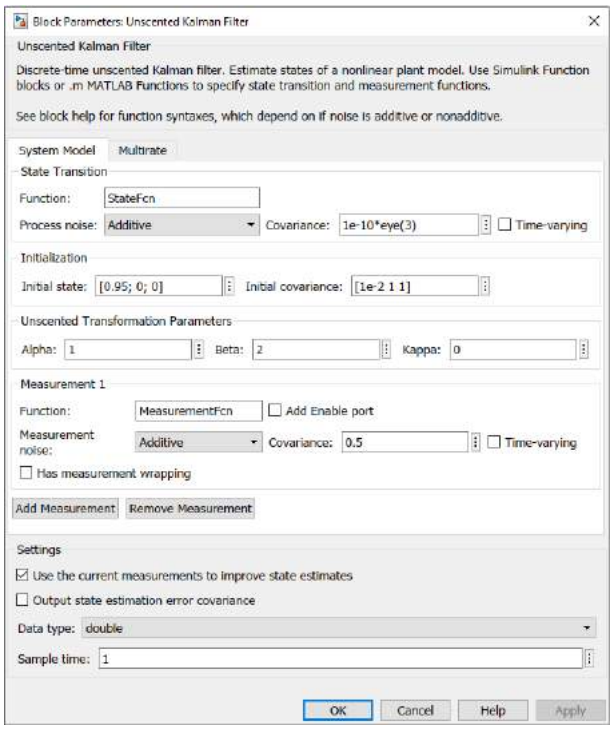


Abbildung 4.18.: UKF-Block

Im UKF-Block wird das verhalten des Prozess- und Messrauschens für die Zustandsübergangsfunktion bzw. Messfunktion, die Ausgangswerte der einzelnen Zustände der Schätzung und die Tranformationsparameter definiert. Für die Zustandsübergangsfunktion und Messfunktion werden ein additives Prozessrauschen gewählt. Die Kovarianz wird entsprechend der drei Zustände des Systems in einer 3x3 Matrix definiert. Als Ausgangswert für die SOC-Berechnung wird ein Wert von 95% angegeben. Die Transformationsparameter werden mit $Alpha = 1$, $Beta = 2$ und $Kappa = 0$ definiert. Der UKF-Block wird mit dem vom BusSelector kommenden Spannungssignal verbunden, wobei vor dem Eingang des UKF-Blocks ein Rate Transition-Block zu platzieren ist, um die Rechengeschwindigkeiten zwischen der Zustandsübergangsfunktion, der Messfunktion und dem UKF-Block anzugleichen.

Die Subsysteme der Zustandsübergangs- und Messfunktion enthalten die Parameter des RC-Ersatzschaltbilds in Form von Lookup-Tabellen, über welche der UKF den aktuellen SOC des Akkus schätzt. Die Lookup-Tabellen enthalten dabei jeweils 10 Werte, welche das Verhalten des Akkus bei den Ladezuständen von 100 bis 10% in 10% Schritten des SOC wiederspiegeln. Auf der Ausgangsseite des UKF-Block wird der SOC ausgegeben welcher aus dem Subsystem geführt wird.

4.3.4. Visualisierung

Um innerhalb des SIMULINK-Modells eine Übersichtlichkeit zu gewährleisten wird für die Visualisierung ein eigenes Subsystem erstellt, welches die jeweiligen Ausgangssignale des Function-Call-Subsytem und des SOC-Subsystem empfängt. Zur Darstellung der Sensordaten und des SOC während der Simulation wird die Viewer Funktion herangezogen. Viewer können durch markieren eines Signals im Modellbereich und Auswahl von “Add Viewer" unter dem Tab “Simulation" und dem Reiter “Prepare" hinzugefügt werden. Durch einen Rechtsklick auf weitere Signale und Auswahl der Option “Connect To Viewer" können diese zu dem bestehenden Viewer ergänzt werden. Mit Viewern verbundene Signale werden automatisch mit einem entsprechenden Symbol markiert. Im geöffneten Viewer kann dessen Layout angepasst werden. Dargestellt werden der effektive Strom, die niedrigsten Zellspannungen pro Akkusegment, die höchsten Zelltemperaturen pro Akkusegment und der SOC, siehe Abbildung 4.19.

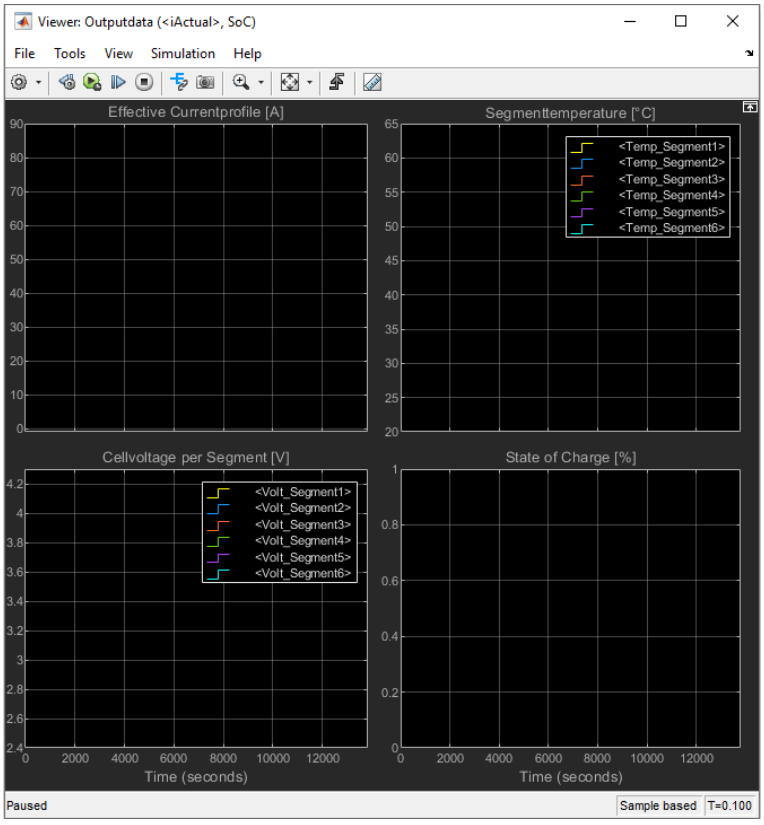


Abbildung 4.19.: Grafische Darstellung der Akkusensordaten und des SOC über einen Viewer

Über die Funktion “Log Signal" werden die im Viewer dargestellten Signale geloggt, welche über den Data Inspector als MAT-Datei für eine spätere Analyse abgespeichert werden können.

Das vollständige SIMULINK-Modell mit all seinen Subsystemen ist Anhang B zu entnehmen.

4.4. Kontrollpanel

Für die einfachere Handhabung des Validierungstools wird für dieses ein Kontrollpanel mithilfe der Design App Funktion in MATLAB implementiert. Design App bietet eine Bibliothek aus diversen Komponenten, wie beispielsweise Druckknöpfe oder Eingabefelder, welche per Drag and Drop in die “Design View" der App gezogen werden können, siehe Abbildung 4.20

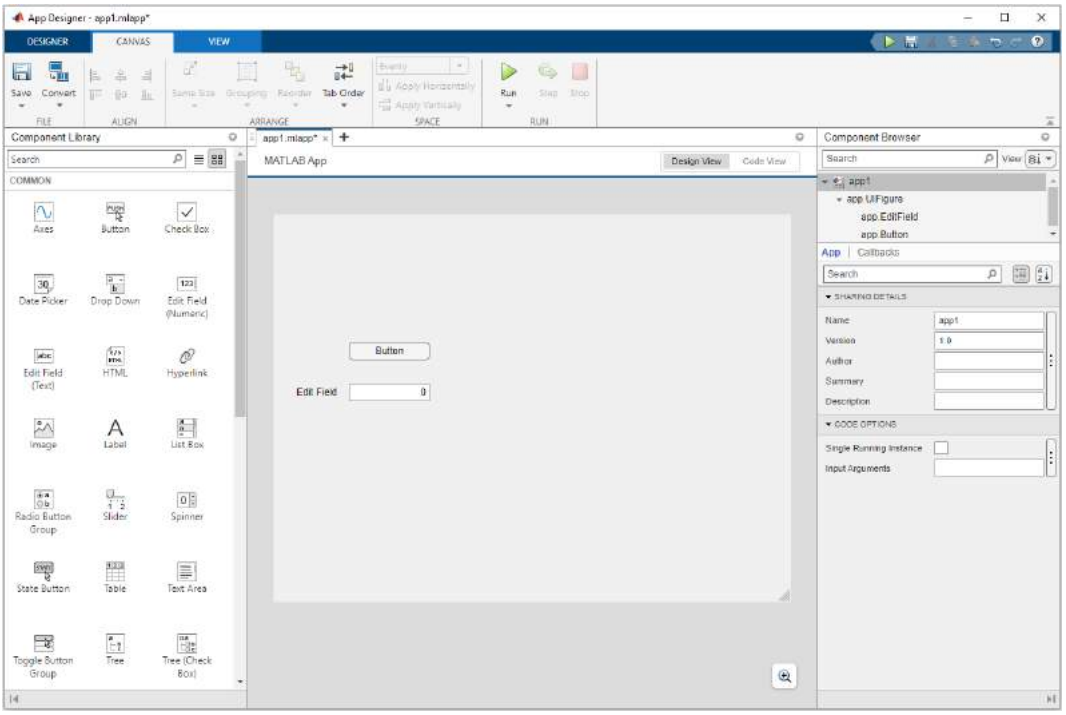


Abbildung 4.20.: Entwurfsbereich des App Designer

Für die in der “Design View" eingefügten Komponenten wird im Reiter “Code View" automatisch eine Funktion für deren Generierung zum Programmstart hinterlegt. Durch einen Rechtsklick auf eine Komponente, kann für diese mit der Option “Callback" eine benutzerdefinierte Funktion definiert werden, siehe Abbildung 4.21. Im “Code View" ist automatisch generierter Code grau und benutzerdefinierter Code weiß hinterlegt.

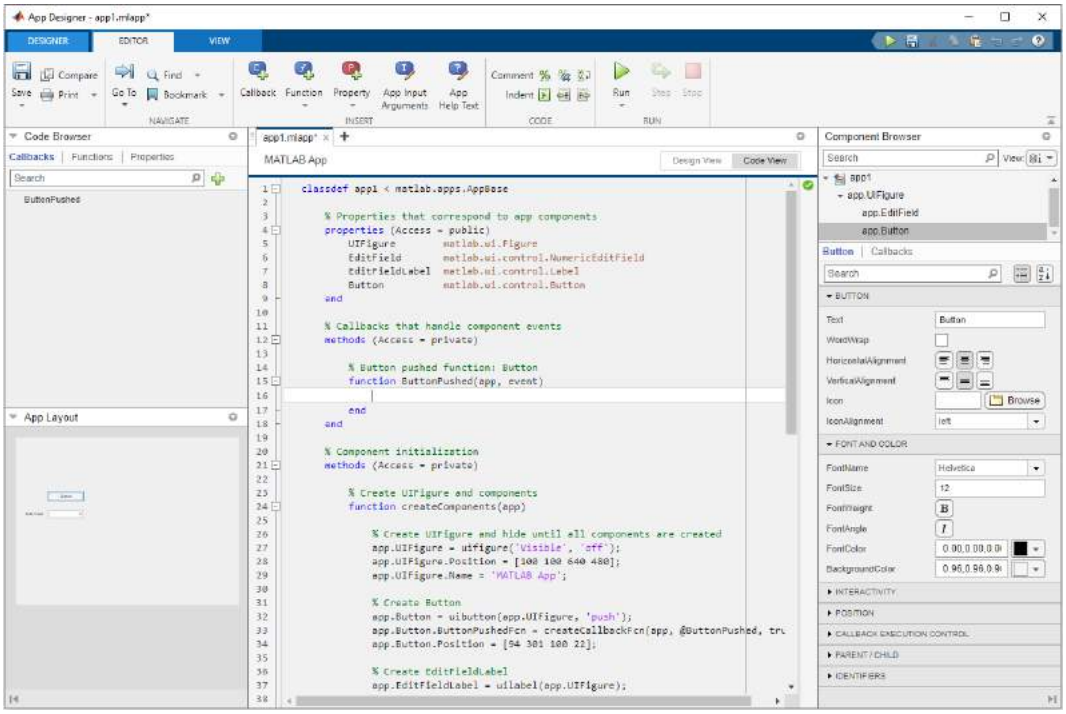


Abbildung 4.21.: Codebereich des App Designer

Über das Kontrollpanel sollen mittels Knöpfe der Akku, der Batteriesimulator und das SIMULINK-Modell gesteuert und mittels Kontrolllampen bzw. Zahlenfelder der aktuelle Status des Akkus und des Batteriesimulator ausgegeben werden. Das Kontrollpanel wird dabei entsprechend in drei Sektionen unterteilt. Für die Kommunikation mit dem Akku und dem Batteriesimulator wird wiederum die Vehicle Network Toolbox herangezogen, wobei nun dessen MATLAB Funktionen eingesetzt werden. Zunächst wird eine Startfunktion definiert, welche beim Öffnen des Kontrollpanels ausgeführt wird. Diese wird über den “Component Browser" durch einen Rechtsklick auf das oberste Element in der Hierarchie mit der Option “Callback" definiert. Beim Start des Kontrollpanels sollen die DBC-Datei eingelesen und der CAN-Kanal konfiguriert werden. Das Einlesen der DBC-Datei erfolgt über die Funktion canDatabase und das Konfigurieren des Kanals über die Funktion canChannel. Für den Kanal wird die Peak-System CAN zu USB Schnittstelle definiert und eine Busgeschwindigkeit von 250 kbit/s eingestellt. Die eingelesene DBC-Datei wird dem definierten Kanal zugeordnet. Die Definitionen der Nachrichten 0x301 und 0x10a für die Steuerung des Akkus und des Batteriesimulators werden über die Funktion canMessage aus der DBC-Datei ausgelesen. Des Weiteren werden die Farben der Kontrolllampen auf Rot gesetzt.

Für die Steuerung des Akkus werden drei Knöpfe implementiert. Um den Akku zu starten ist an die CAN Adresse 0x301 eine Flanke von Hexadezimal 00 auf 80 zu senden. Der erste Knopf wird mit “Initialize" bezeichnet und dient dem Start der periodischen Übertragung des Wert 00 mithilfe der transmitPeriodic Funktion. Der zweite Knopf wird mit “Start" bezeichnet und ändert den Wert im Datenfeld der periodischen Übertragung auf 80. Der dritte Knopf wird mit “Stop" bezeichnet und ändert den Wert im Datenfeld wiederum auf 00 um den Akku zu deaktivieren. Die Überwachung des Status erfolgt

über zwei Kontrolllampen, wobei eine den Status des AMS und die andere den Status des SDC widerspiegelt. Hierfür werden mit der receive Funktion, Daten vom Bus empfangen und daraufhin mit der groupfilter Funktion nach dem Identifier 0x241 gefiltert. Die aktuellen Werte für den Status des AMS und SDC werden über if-Anweisungen ausgewertet, welche entsprechend die Farben der Lampen verändern. Des Weiteren wird neben den Lampen der jeweilige Status als logische 0 oder 1 ausgegeben. Um den Status abfragen zu können, werden die receive Funktion, der Filter und die if-Anweisungen in einen Knopf mit dem Namen “Get Status" hinterlegt.

Die Steuerung des Batteriesimulators erfolgt ebenfalls über drei Knöpfe. Der erste Knopf wird mit “On" bezeichnet und sendet einmalig den Wert 02 zum Start des Batteriesimulators. Der zweite Knopf wird mit “Off" bezeichnet und sendet einmalig den Wert 00 zum Stoppen des Batteriesimulators. Der dritte Knopf wird mit “Reset" bezeichnet und sendet einmalig den Wert 01, um im Fehlerfall den Batteriesimulator zurücksetzen zu können. Die Überwachung des Status erfolgt über eine Kontrolllampe. Die Abfrage des Status erfolgt identisch zu jener des Akkus, wobei im Falle des Batteriesimulators nach dem Identifier 0x11a gefiltert wird. Der Status wird ebenfalls neben der Lampe als Zahlenwert ausgegeben.

Die Simulation wird über vier Knöpfe gesteuert. Der erste Knopf wird mit “Import Data" bezeichnet und lädt das für die Simulation notwendige Fahrtprofil. Hierfür wird die in Kapitel 4.3.1 erwähnte Importfunktion herangezogen. Der zweite Knopf wird mit “Initialize Sim" bezeichnet, welcher das SIMULINK-Modell öffnet, die Simulation startet und sofort pausiert. Da der Akku vor dem Start der Simulation gestartet werden muss und das Starten bzw. Laden der Simulation für einen kurzen Moment die CAN-Kommunikation und somit die periodische Übermittlung der Nachricht 0x301 unterbricht, kann die Simulation nicht direkt gestartet werden. Der dritte und vierte Knopf werden mit “Start Sim" und “Stop Sim" bezeichnet und führen die bezeichneten Aktionen aus.

Das Starten des Kontrollpanels erfolgt über den “Run" Knopf des App Designer. Die Benutzeroberfläche des Kontrollpanels ist in Abbildung 4.22 dargestellt.

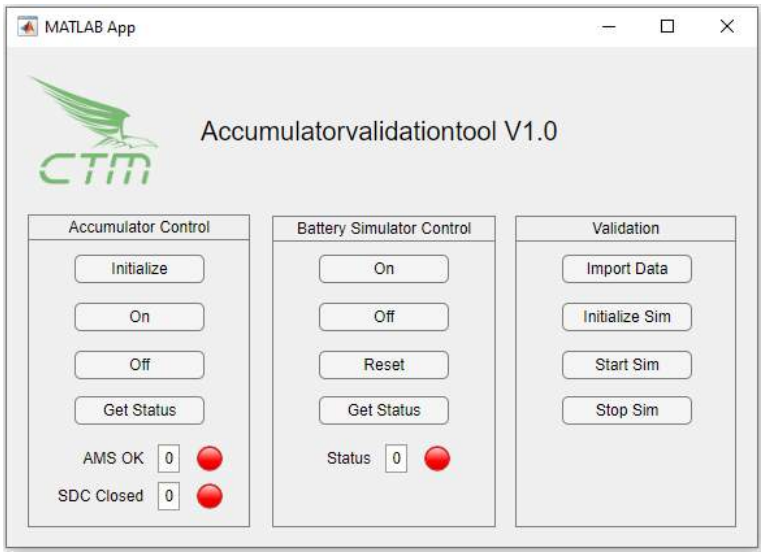


Abbildung 4.22.: Kontrollpanel des Validierungstools

Der vollständige Code für das Kontrollpanel ist Anhang C zu entnehmen.

4.5. Testequipment

Der Akku, welcher für den Test eingesetzt wird, ist jener des Vorjahresrennwagens. Der Akku besteht aus Sony VTC6 18650 Lithium-Ionen Zellen, wobei 4 Zellen jeweils parallel zu einem Modul von 4,2 V geschalten sind. Der gesamte Akku besteht aus 144 dieser Module, welche in Serie geschalten eine Maximalspannung von 600 V, eine Kapazität von 12 Ah und eine Gesamtleistung von zirka 80 kW liefern. Aus regeltechnischen Gründen muss der Akku in Segmente unterteilt werden, wobei ein Segment nicht mehr als 120 V DC besitzen darf. Der Aufbau des Akkkumulators ist in Abbildung 4.23 dargestellt.

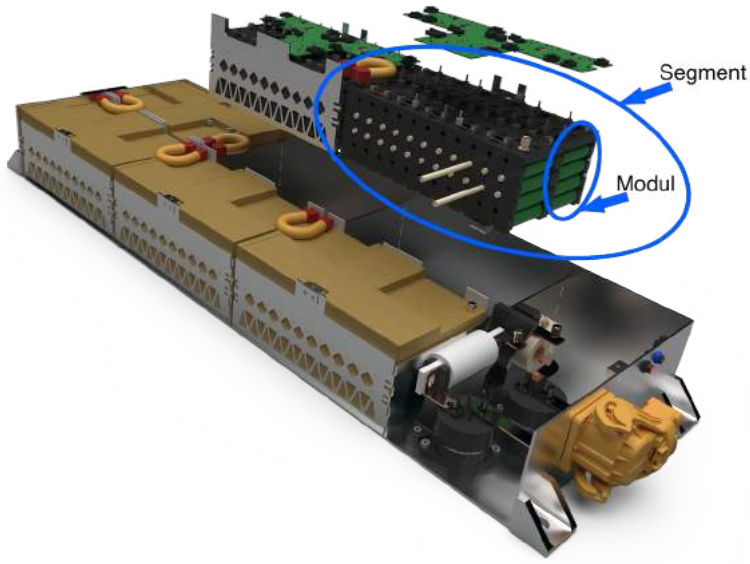


Abbildung 4.23.: CAD Modell des Akkus [23]

Der Batteriesimulator wird von der Firma ALPITRONIC zur Verfügung gestellt. Dabei handelt es sich um ein bidirektionales Modul, welches ursprünglich zur Validierung von PKW-Ladesäulen konzipiert und für die Zwecke dieser Arbeit umkonfiguriert wurde. Aus Sicherheitsgründen ist ein Transformator zwischen den Batteriesimulator und dem Netz einzubauen. Dieser trennt die Potentiale von der Erde, um bei einem Fehlerstrom und gleichzeitiger Berührung des Batteriesimulators keinen Stromschlag zu erhalten. Der Transformator wurde ebenfalls seitens ALPITRONIC zur Verfügung gestellt. Die beiden Komponenten sind in Abbildung 4.24 dargestellt.

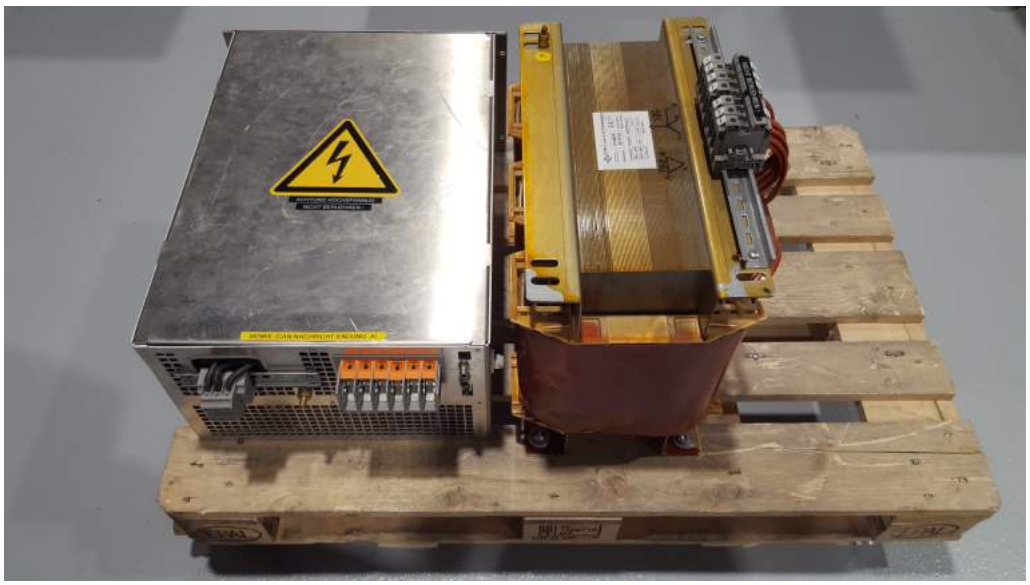


Abbildung 4.24.: Batteriesimulator und Transformator

4.6. Verbindung von Soft- und Hardware

Die Verbindung zwischen dem Batteriesimulator und dem Akku erfolgt über zwei 35 mm^2 Kabel per Schnellverschlussstecker auf der Akkuseite und Durchführungsklemmen auf der Batteriesimulatorseite. Die vom Akku abgeführte Energie wird über Durchgangsklemmen am Batteriesimulator in die Sekundärseite des Transformators geführt und über die Primärseite ins Netz zurückgespeist. Für das Kabel zwischen Batteriesimulator und Akku kann ein bereits bestehendes, mit passendem Schnellverschlussstecker verbundenes Kabel aus der CTM Werkstatt verwendet werden. Die Kabel für die Verbindung vom Batteriesimulator über den Transformator hin zur Rückspeisung ins Netz werden aus 10 mm^2 Kabel gefertigt, wobei jeweils ein Kabel für die drei Phasen und ein Kabel für den Schutzleiter vorzusehen sind. Der Nennquerschnitt mit 10 mm^2 wird hierbei laut [24] den gegebenen Anforderungen gerecht. Die Rückspeisung ins Netz erfolgt über eine 32 A Steckdose. Die Kabel zur Verbindung des Akkus mit dem Batteriesimulaotr und dem Transformator mit dem Netz sind in Abbildung 4.25 dargestellt.

Abbildung 4.25.: Kabel zur Verbindung von Akku-Batteriesimulator und Tranformator-Netz

Die Verbindung zwischen dem Batteriesimulator und dem Transformator ist in Abbildung 4.26 dargestellt. Im Vergleich zum Kabel für die Verbindung des Transformators mit dem Netz, wurde bei den Kabeln für die Verbindung zwischen dem Transformator und dem Batteriesimulator aus Gründen der Flexibilität, auf eine zusätzliche Ummantlung verzichtet.

Abbildung 4.26.: Verbindung des Batteriesimulators mit dem Transformator

Für die Kommunikation zwischen Rechner, Akku und Batteriesimulator wird ein Kabelbaum gefertigt. Die dafür notwendige Niedervolt Spannungsversorgung erfolgt über ein Labornetzteil, wobei eine Spannung von 24 V benötigt wird. Die Leitung des CAN-Bus bildet den Hauptbestandteil des Kabelbaums. Dieser wird in der klassischen 2-Draht Ausführung erstellt, wobei eine Leitung CAN_High und eine CAN_Low entspricht. Zur besseren Abschirmung gegen von außen einwirkende Störungen werden beide Leitungen miteinander verdrillt. Hierbei werden zwei gleich lange Kabel auf einer Seite in eine Schraubzwinge und auf der anderen Seite in einen Akkuschrauber eingespannt. Durch Betätigung des Akkuschraubers werden beide Kabel miteinander verdrillt. Da der Bus drei Teilnehmer besitzt, bedarf es auch 3 Kabel, welche zum jeweiligen Teilnehmer führen. An den Knotenpunkten werden diese über Wagoklemmen miteinander verbunden. Für den Anschluss am Rechner wird am Buskabel ein weiblicher D-Sub Stecker angelötet, um sich entsprechend mit dem Peak-System CAN zu USB Adapter mit männlichem D-Sub Stecker verbinden zu können. Dem Datenblatt vom CAN zu USB Adapter [25] ist dabei zu entnehmen, dass Pin 2 CAN_Low und Pin 7 CAN_High entspricht. Für den Anschluss am Batteriesimulator wird ein männlicher D-Sub Stecker am Buskabel angelötet. Hier liegt dieselbe Pinbelegung wie beim CAN zu USB Adapter vor. Der Anschluss zum Akku erfolgt über einen 12-poligen Signalstecker. Für die Verbindung zum Signalstecker werden Crimps am Ende von 0,20 mm^2 Leitungen angebracht, woraufhin die Leitungen in den Signalstecker geschoben werden können. Die Pinbelegung des Signalsteckers ist Tabelle 4.2 zu entnehmen, welche neben den Anschlüssen für den CAN-Bus und der Niedervolt Spannungsversorgung auch diverse sicherheitstechnisch relevante Anschlüsse auflistet.

Tabelle 4.2.: Pinbelegung des Signalsteckers vom Akku

Pin	Signal
1	24 V
2	GND (Masse)
3	CAN_High
4	CAN_Low
5	IMD_OK
6	Mainhoop_GND
7	Input_SDC
8	*Nicht relevant*
9	AMS_OK
10-12	*Nicht relevant*

Um bei der späteren Inbetriebnahme und Testdurchführung gefährliche Situationen zu vermeiden, wird von diesen bereits vorhandenen Sicherheitssystemen des Akkus Gebrauch gemacht. Das Signal AMS_OK wurde bereits im Zuge der Datenbankerstellung erläutert. IMD_OK gibt den Status des Insulation Monitoring Device (IMD) aus. Das IMD dient zur Überwachung des Isolationswiderstands und weist auf Kabelbrüche hin.

Zur Verarbeitung der Signale wird eine Latchingplatine (kurz Latching) verwendet, siehe Abbildung 4.27. Diese wird von CTM bereitgestellt, wobei jene des Vorjahresrennwagens verwendet wird.

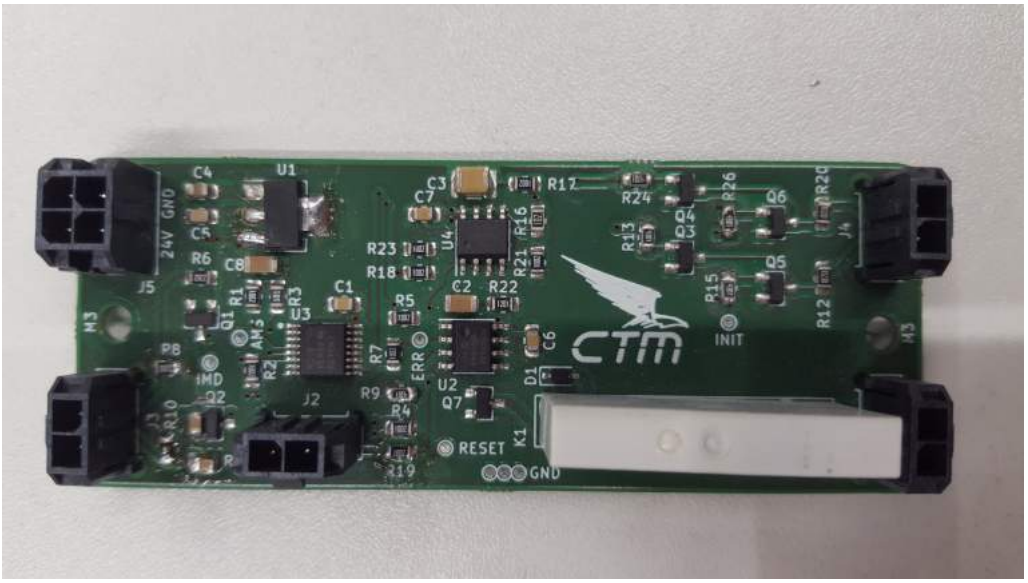


Abbildung 4.27.: Latchingplatine

Das Latching besteht aus einer Logik und einem Relais, welches normal geschlossen ist. Das Latching ist Teil des SDC. Der Pin Input_SDC des Akkus wird mit dem entsprechenden SDC Eingang am Latching verbunden.

Das Latching empfängt die Signale IMD_OK und AMS_OK vom Akku und überwacht diese. Sobald von IMD_OK und/oder AMS_OK eine Störung empfangen wird, öffnet das Relais des Latchings, welches wiederum den SDC öffnet. Um bei drohender Gefahr den SDC auch manuell auslösen zu können, wird am SDC Ausgang des Latching ein Not-Aus Schalter installiert. Die Kabel werden dabei auf die Anschlüsse des Not-Aus gelötet. Um das Relais des Latching händisch schließen zu können, wird ein Reset-Knopf implementiert. Dieser wird mit den entsprechenden Resetsteckern auf der Platine verbunden. Letztlich ist das Latching auch mit der 24 V Spannungsversorgung zu verbinden. Zum Schutz der Platine vor äußeren Einflüssen und zur Befestigung des Not-Aus bzw. des Reset-Knopfs wird eine Verteilerbox verwendet. Zur Durchführung der Kabel und Anbringung der Knöpfe werden Löcher in die Verteilerbox gebohrt. Um eine Zugentlastung am Latching zu gewährleisten, werden die Kabel durch einen Kabelverschraub geführt. Die Verbindung der jeweiligen Kabel mit der Latchingplatine erfolgt über Steckverbinder. Zur Verbindung der Kabel mit den Steckverbindern werden die Leitungsenden vercrimpt und in die Steckverbinder geschoben. Der Anschluss auf der Latchingplatine erfolgt über die entsprechenden Sockel. Abbildung 4.28 zeigt die fertige Verteilerbox.

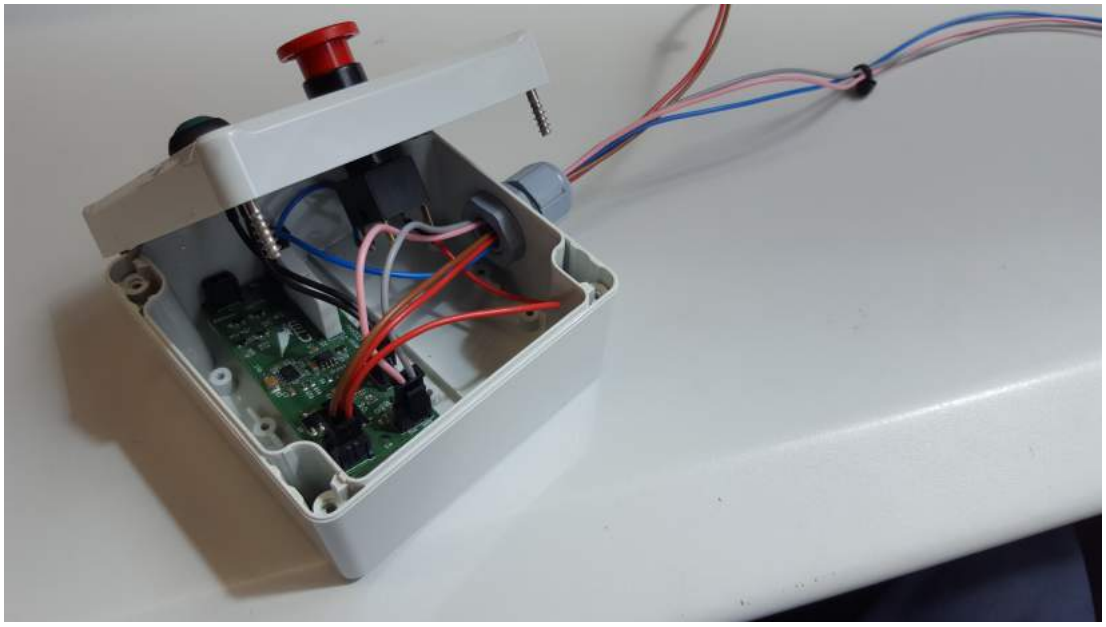


Abbildung 4.28.: Verteilerbox mit Latchingplatine, Not-Aus Schalter und Reset-Knopf

Um das vom Netzteil kommende 24 V- bzw. Massekabel mit dem Latching und dem Akkusignalstecker zu verbinden, werden wiederum Wagoklemmen verwendet.

Der Pin Mainhoop_GND stellt grundsätzlich die vom Regelwerk der Formula Student vorgeschriebene Verbindung zwischen dem IMD und Masse (Rahmen das Fahreugs) dar. Im Falle des Testaufbaus wird Mainhoop_GND mit dem Akkugehäuse per Kabelschuh verbunden.

Abbildung 4.29 zeigt den fertigen Kabelbaum mit einer Beschriftung seiner Bestandteile.

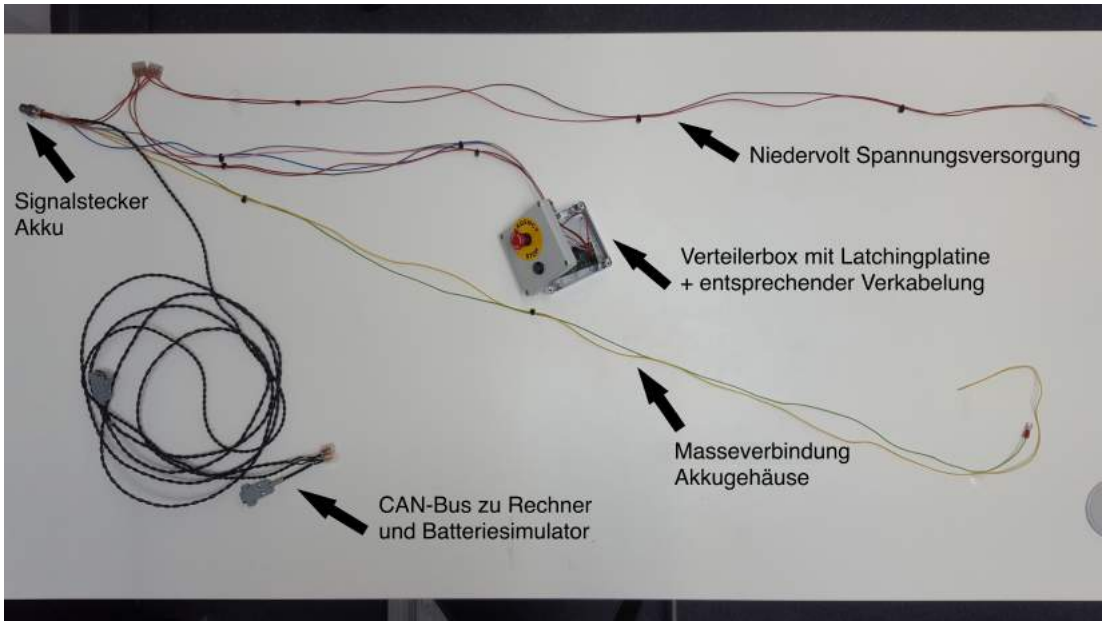


Abbildung 4.29.: Kabelbaum zur Kommunikation mit der Hardware

4.7. Inbetriebnahme

Das Testen des Validierungstools an der Hardware erfolgt in der Werkstatt der Firma MATTRO. Um den Batteriesimulator und den Transformator besser hantieren zu können, wird die Europalette auf welcher das Equipment steht mit vier Schwerlastrollen ausgestattet. Der Akku wird in einem Tischwagen neben der Europalette positioniert. Der Tischwagen wird von CTM zur Verfügung gestellt. Über das im Tischwagen verbaute Ladegerät kann der Akku in weiterer Folge nach einer Entladung wieder geladen werden. Der einsatzbereite Testaufbau ist in Abbildung 4.30 dargestellt.

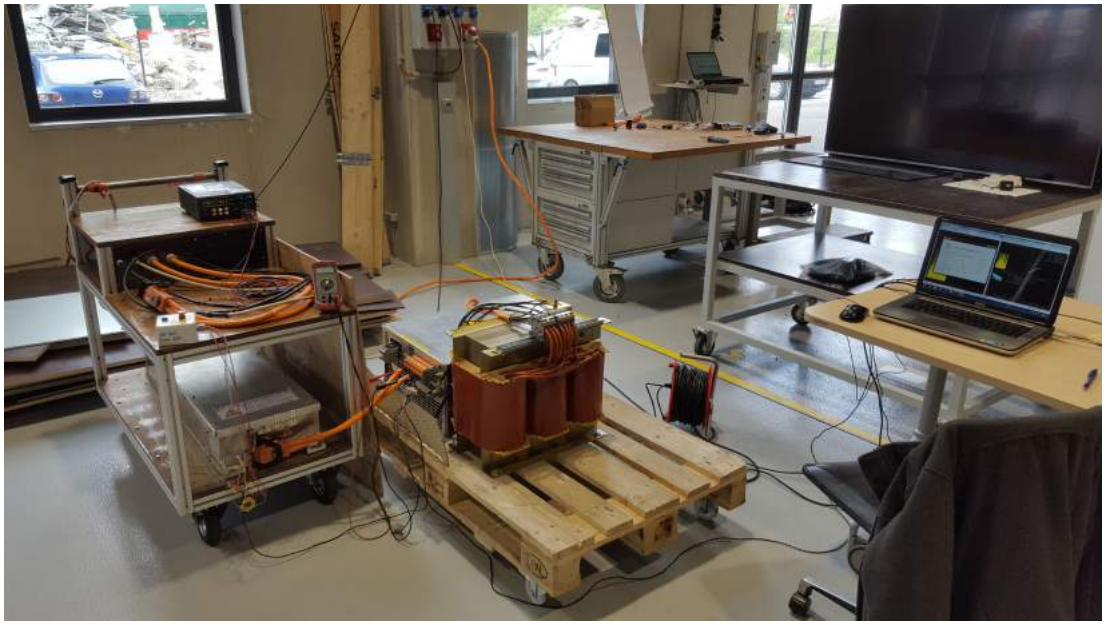


Abbildung 4.30.: Testaufbau

Das Startprozedere des Systems ist wie folgt:

1. Akku über Schnellverschlussstecker mit Batteriesimulator verbinden
2. Labornetzteil anschalten
3. Batteriesimulator über 32 A Stecker mit dem Netz verbinden
4. Kontrollpanel öffnen
5. Stromprofil importieren
6. Simulation initialisieren
7. Akku initialisieren
8. Reset-Knopf des Latching auf der Verteilerbox betätigen
9. Akku anschalten
10. Status des Akkus mit entsprechendem “Get Status" Knopf am Kontrollpanel abfragen
11. Wenn die Signale für AMS OK und SDC closed den Status 1 ausgeben, Batteriesimulator anschalten
12. Status des Batteriesimulators mit entsprechendem “Get Status" Knopf am Kontrollpanel abfragen
13. Wenn der Batteriesimulator den Status 6 ausgibt, kann die Simulation gestartet werden

Das manuelle schließen des Latching Relais ist notwendig, da das Relais vor dem Senden der Flanke von 00 auf 80 aufgrund des CAN Timeout des Akkus geöffnet ist.

4.8. Ermittlung der Ersatzschaltbildparameter für die SOC-Berechnung

Zur Ermittlung der Ersatzschaltbildparameter für die SOC-Berechnung wird der Akku mit dem in Kapitel 4.2.1 definierten Rechteckstromprofil mittels dem Testaufbau belastet. Hierbei ist der resultierende Verlauf der Zellspannungen und der gemessene Strom von Bedeutung. Das Verhalten der Zellspannungen pro Akkusegment ist in Abbildung 4.31 dargestellt.

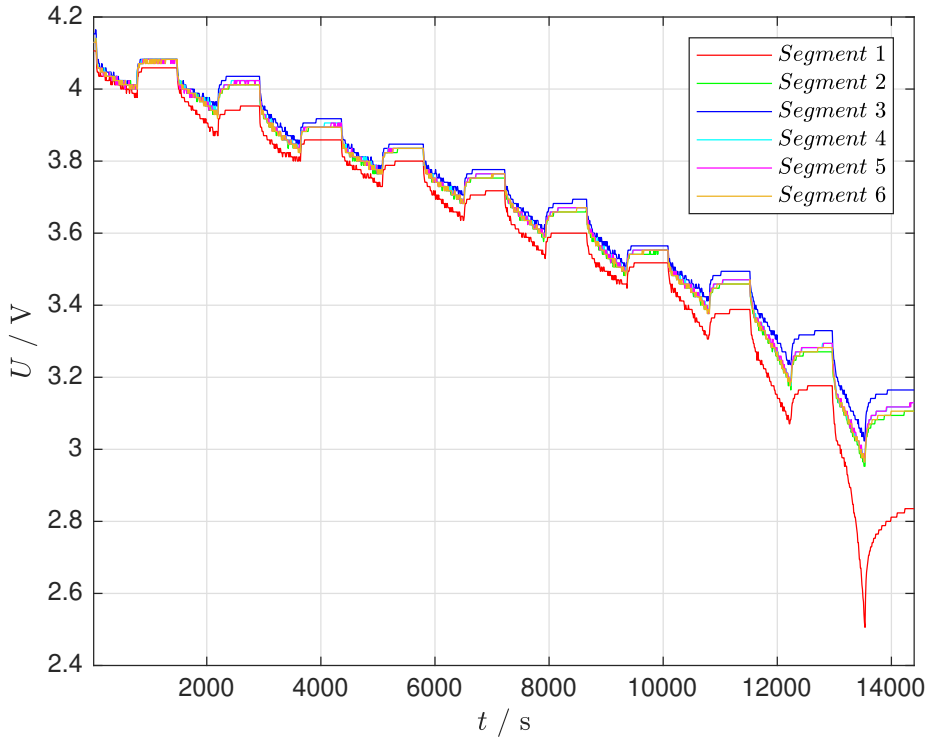


Abbildung 4.31.: Gemessener Zellspannungen pro Akkusegment bei der Pulsentladung

Obwohl der Akku zu Beginn der Entladung lediglich eine Spannungsabweichung von 50 mV zwischen den Zellen aufweist und somit als ausbalanciert angesehen werden kann, fällt über den Verlauf der Entladung das Segment 1 im Vergleich zu den anderen Segmenten deutlich ab. Da die niedrigste Zellspannung des ersten Segments während des letzten Pulszyklus die Entladeschlussspannung von 2,5 V erreicht, öffnet das AMS den SDC was den Pulsentladetest vorzeitig beendet. Segment 1 limitiert daher die Kapazität des gesamten Akkus. Dieses Verhalten ist möglicherweise auf eine Überbelastung und damit einhergehende irreversible Beschädigung der Zellen im Segment 1 in der Vergangenheit zurückzuführen. Da das Verhalten des Akkus über den Spannungsbereich von 4,2 V bis 2,5 V gut dargestellt wurde, ist das gewonnene Messergebnis zufriedenstellend und für die Parameterbestimmung hinreichend. Der gemessene Strom, welcher aus dem Akku extrahiert wurde ist in Abbildung 4.32 dargestellt.

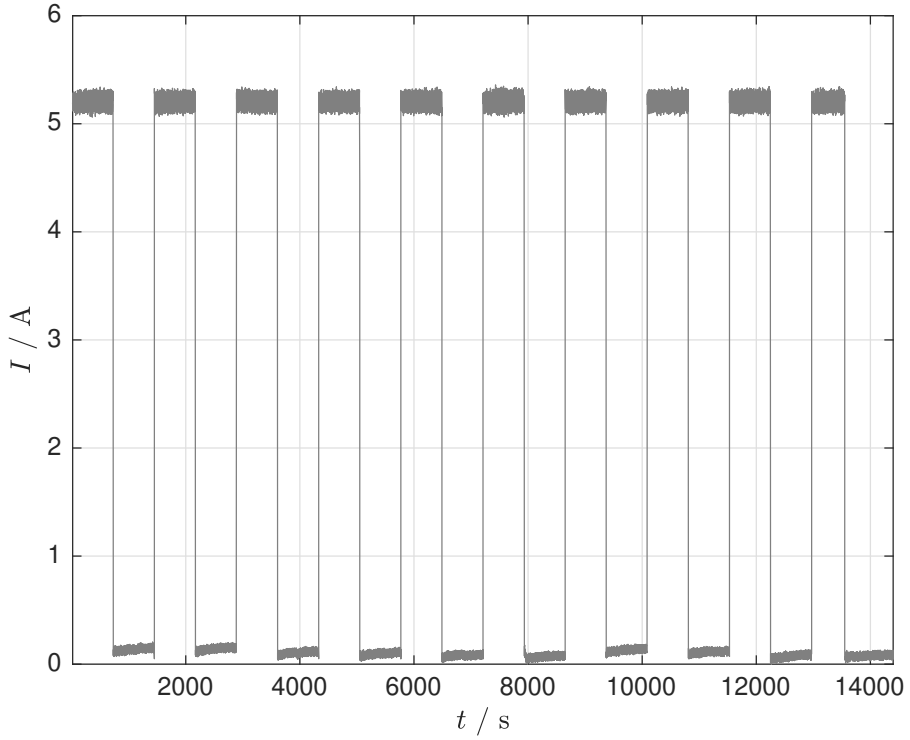


Abbildung 4.32.: Gemessener Strom bei der Pulsentladung

Es ist ein Messfehler des Stromsensors von 0,8 A und ein verhältnismäßig starkes Rauschen ersichtlich. Da die Stromregelung innerhalb des Batteriesimulators auf den Messwerten dieses Stromsensors beruht, weicht der vom Stromregler eingestellte Wert vom tatsächlich vorgegebenen Stromwert des Validierungstools ab. Eine Erhöhung der Amplitude des vorgegebenen Rechteckstromprofils zur Kompensation des Fehlers des Stromsensors führte zu einem höheren Spannungsabfall über den Verlauf der Pulsentladung, was ein früheres Erreichen der Entladeschlussspannung durch Segment 1 und eine damit einhergehende Abschaltung des Akkus zur Folge hatte. Da dies den Entladetest zu früh beendete, wurde auf diese Kompensation verzichtet. Aufgrund des Messfehlers des Stromsensors würde der Batteriesimulator bei einem Vorgabewert von 0 A den Akku mit 0,8 A laden, was das Ergebnis des Pulsentladetest unbrauchbar machen würde. Daher wurde im SIMULINK-Modell für die Pulsentladung zusätzlich zum Rechteckstromprofil eine if-Anweisung ergänzt, um den Batteriesimulator in den Ruhephasen auszuschalten.

Für die Parameterbestimmung des Ersatzschaltbild werden der gemessene Strom und die Zellspannungen der Segmente über den Data Inspector in eine MAT-Datei gespeichert. Die Parameterbestimmung erfolgt über den Parameter Estimator in SIMULINK. Hierbei werden mithilfe der in der MAT-Datei gespeicherten Messdaten die gesuchten RC-Ersatzschaltbildparameter ermittelt. Dem Parameter Estimator ist ein Modell vorzugeben, für welches dieser dessen Parameter bestimmen soll. Hierfür kommt ein von MathWorks zur Verfügung gestelltes Akkumodell in Form einer RC-Schaltung zum Einsatz [22]. Der Parameter Estimator passt die Werte der Komponenten des Akkumo-

dells über wiederholte Simulationen solange an, bis das vorgegebene Akkuverhalten hinreichend angenähert ist, siehe Abbildung 4.33. Am Ende der Parameterbestimmung werden die ermittelten Werte in einzelne Vektoren gespeichert, wobei für jeden Parameter jeweils ein Vektor für die Lookup-Tabelle der SOC-Berechnung generiert wird.

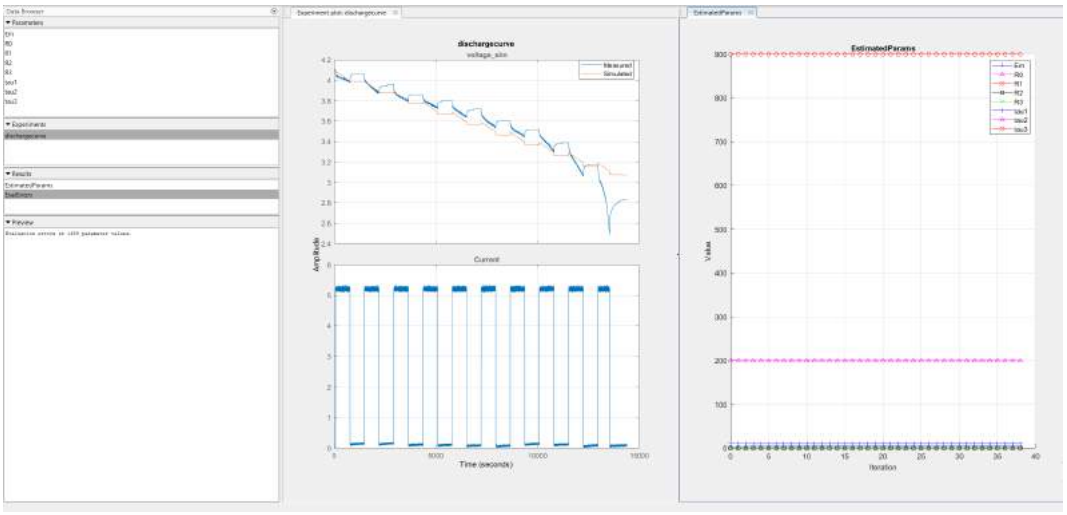


Abbildung 4.33.: Parameter Estimator in SIMULINK

Da die ermittelte Annäherung des Akkumodells den realen Akku nicht hinreichend genau beschreibt, werden die gewonnenen Parameter über die Parameter des Akkumodells der MathWorks Demo angepasst [22]. Die resultierenden Parameter sind Tabelle 4.3 zu entnehmen.

Tabelle 4.3.: RC-Ersatzschaltbildparameter [22]

$SOC/\%$	$U_{Leerlauf}/V$	R_0/Ω	R_1/Ω	R_2/Ω	C_1/F	C_2/F
10	3,19	0,0085	0,00041	0,00014	3,85	60,67
20	3,35	0,0086	0,00053	0,00029	9,35	192,23
30	3,48	0,0087	0,00045	0,00007	8,53	44,15
40	3,59	0,0084	0,00045	0,00041	8,67	54,95
50	3,69	0,0084	0,00053	0,00010	10,59	232,96
60	3,74	0,0082	0,00052	0,00043	9,07	96,09
70	3,84	0,0084	0,00052	0,00019	9,46	102,14
80	3,94	0,0088	0,00051	0,00027	10,02	92,34
90	4,06	0,0088	0,00049	0,00040	9,52	100,66
100	4,18	0,0096	0,00046	0,00016	10,32	113,37

Die ermittelten Ersatzschaltbildparameter werden letztlich in die entsprechenden Lookup-Tabellen der Zustandsübergangs- und Messfunktion im SIMULINK-Modell eingelesen. Da das zuweisen der Parameter bei jedem Start von SIMULINK manuell erfolgen müsste, werden im Kontrollpanel unter dem Knopf “Import Data" neben dem Stromprofil für die Fahrtsimulation auch die Ersatzschaltbildparameter eingelesen.

5. Resultate

Mithilfe des implementierten Validierungstool kann nun über das in Kapitel 4.2.2 definierte Stromprofil eine Fahrt für den Akku simuliert werden. Abbildung 5.1 zeigt die Benutzeroberfläche des implementierten Validierungstools während der Fahrtsimulation.

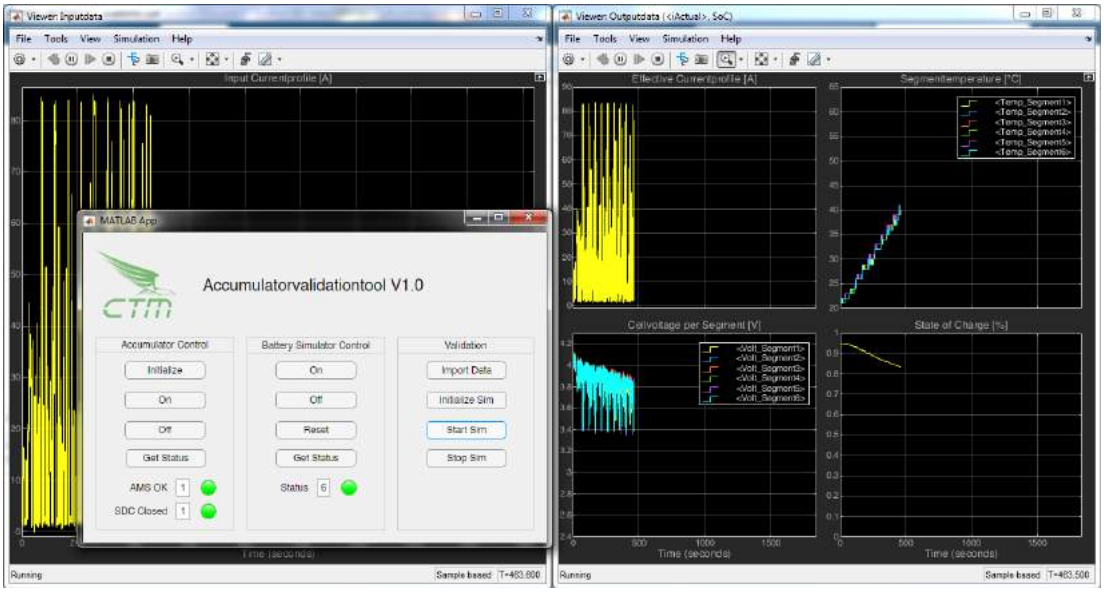


Abbildung 5.1.: Benutzeroberfläche des Validierungstool während der Fahrtsimulation

Die Simulation wurde bei einer Akkutemperatur von 22 °C gestartet. Dies entsprach der zu diesem Zeitpunkt vorherrschenden Umgebungstemperatur. Abbildung 5.2 zeigt den gemessenen Strom über den Verlauf der Simulation. Beim Vergleich mit dem an den Batteriesimulator gesendeten Fahrtprofil in Abbildung 4.5 wird ersichtlich, dass die vorgegebenen Lastsprünge von dessen Stromregler zufriedenstellend umgesetzt werden konnten. Der Messfehler des Stromsensors vom Batteriesimulator ist im Verhältnis zu den hohen Strömen des Fahrtprofils vernachlässigbar. Grundsätzlich war eine Simulationsdauer von 1850 s vorgesehen, wobei aus dem gemessenen Stromprofil hervorgeht, dass die Simulation frühzeitig beendet wurde.

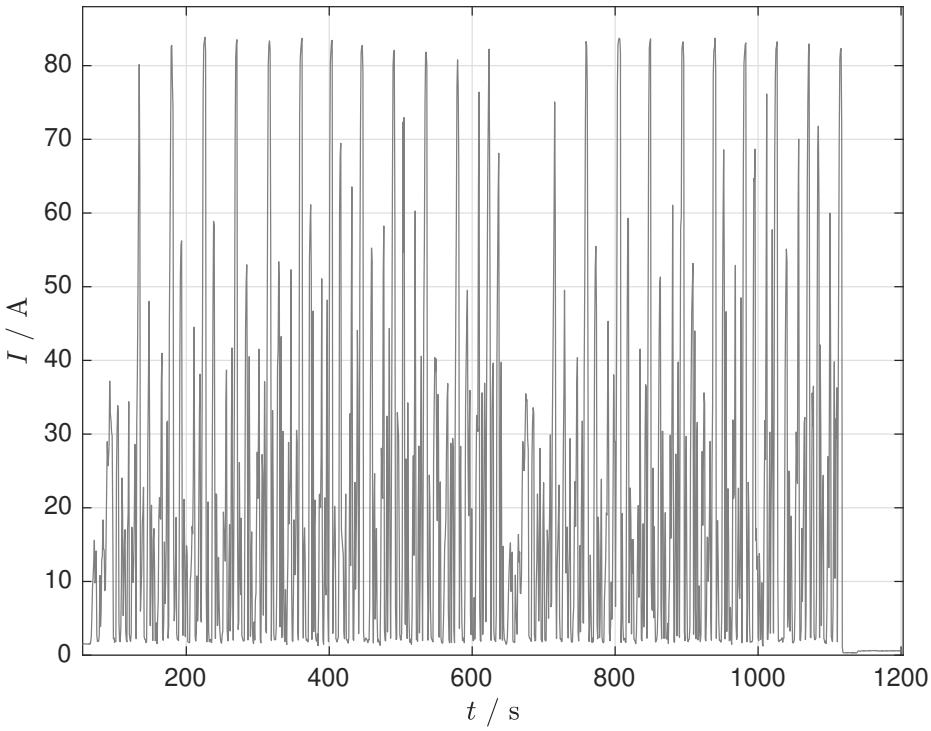


Abbildung 5.2.: Gemessener Strom bei einer Leistungsbegrenzung von 25%

Der Grund für den Abbruch der Simulation wird bei Betrachtung von Abbildung 5.3 deutlich. Hierbei ist zu erkennen, dass nach knapp unter zwei drittel der Simulationszeit durch Segment 5 die Temperaturobergrenze von 60 °C erreicht wurde, was zum automatischen öffnen des SDC durch das AMS führte. Die Temperatur stieg über die Segmente verteilt gleichmäßig an. Die kurzzeitige Stagnation der Temperaturentwicklung bei zirka 640 s ist auf den zu diesem Zeitpunkt geringeren Strombedarf zurückzuführen. Dieser macht sich auch in Abbildung 5.4 durch einen geringeren Spannungsabfall zum selben Zeitpunkt bemerkbar.

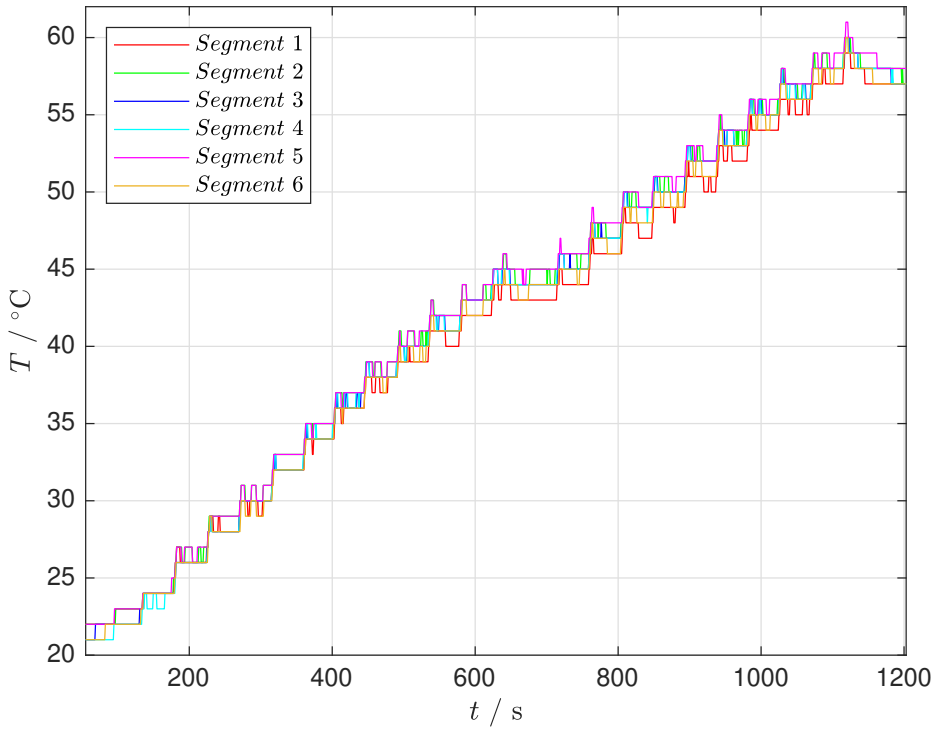


Abbildung 5.3.: Gemessene Segmenttemperaturen bei einer Leistungsbegrenzung von 25%

Aus dem Messergebnis der Zellspannungen ist zu sehen, dass die einzelnen Segmente untereinander gut ausbalanciert sind. Lediglich das Simulationsende lässt eine stärkere Entladung des Segment 1 erkennen, was nach dem Ergebnis der Pulsentladung in Kapitel 4.8 jedoch zu erwarten war. Die Anfangsspannung von 4,1 V ist auf das Ladegerät des Tischwagens zurückzuführen, welches nicht genügend Leistung besitzt, um den Akku auf 4,2 V Zellspannung zu laden.

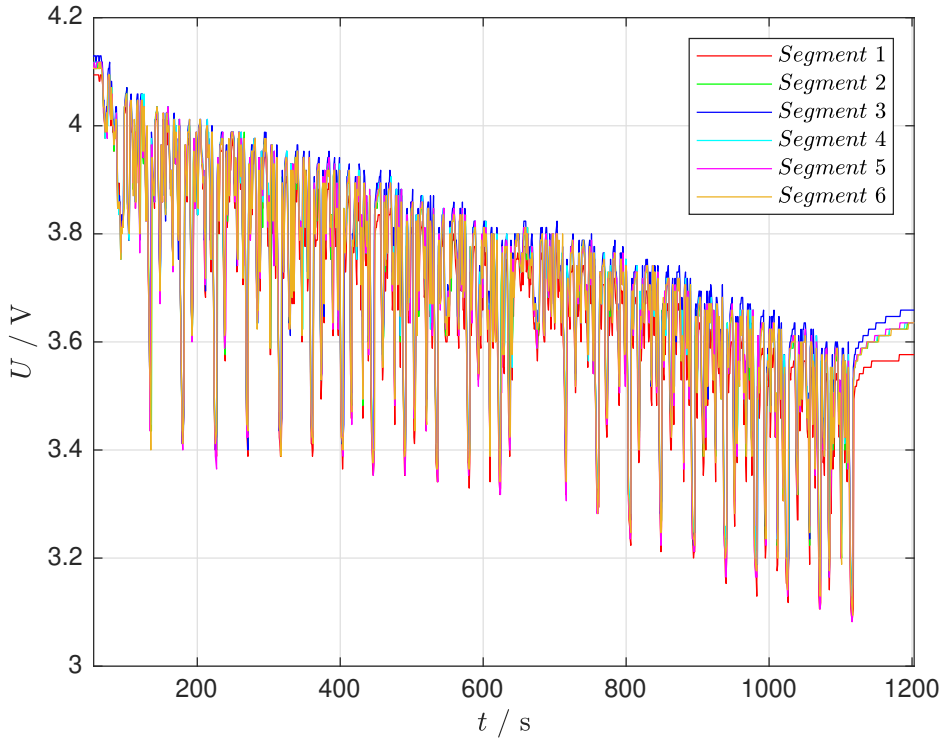


Abbildung 5.4.: Gemessene Zellspannungen pro Akkusegment bei einer Leistungsbegrenzung von 25%

Abbildung 5.5 zeigt die Entwicklung des SOC über den Verlauf der Fahrtsimulation, welche bei Vergleich mit der Entwicklung der Zellspannungen in Abbildung 5.4 auf eine zufriedenstellende Schätzung schließen lässt. Der Anfangswert mit 95% entspricht dem im UKF-Block definierten Ausgangswert für die Schätzung und der in Abbildung 5.4 dargestellten gemessenen Anfangsspannung.

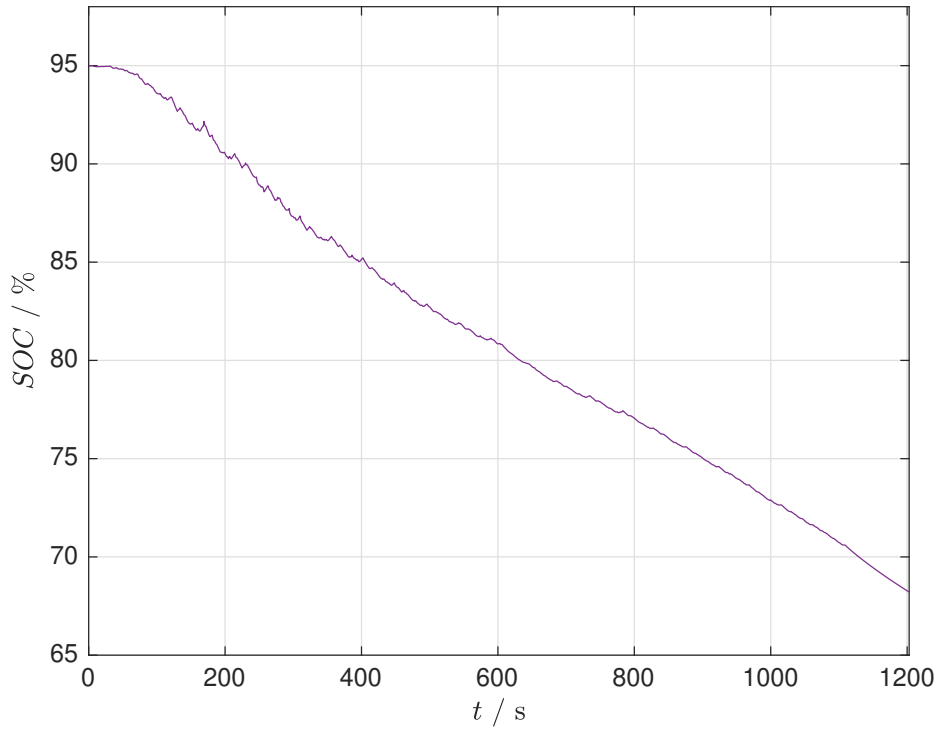


Abbildung 5.5.: Ermittelter SOC bei einer Leistungsbegrenzung von 25%

Da nun festgestellt wurde, dass eine Leistungsbegrenzung von 25% nicht ausreichend ist, um die Distanz von 22 km im Endurance and Efficiency Bewerb absolvieren zu können, wird eine weitere Fahrtsimulation mit einer höheren Leistungsbegrenzung simuliert. Hierbei soll eine Leistungsbegrenzung von 50% getestet werden. Um dies zu bewerkstelligen, wird das bestehende Fahrtprofil skaliert. Dabei werden die Stromwerte nach der Formel

$$I_{50} = \frac{0.5 * I_{25}}{0.75} \tag{5.1}$$

skaliert. Da eine Begrenzung der Leistung die durchschnittliche Geschwindigkeit reduziert, erhöht sich nach der Formel $s = v * t$ die Zeit, welche benötigt wird, um die selbe Strecke zu absolvieren. Für die Zwecke dieser Arbeit ist es ausreichend die Vereinfachung anzunehmen, dass eine Verringerung der Geschwindigkeit eine linear proportionale Verlängerung der zurückzulegenden Strecke zur Folge hat. Das skalierte Fahrtprofil mit einer Leistungsbegrenzung von 50% wird jenem mit einer Leistungsbegrenzung von 25% in Abbildung 5.6 gegenübergestellt.

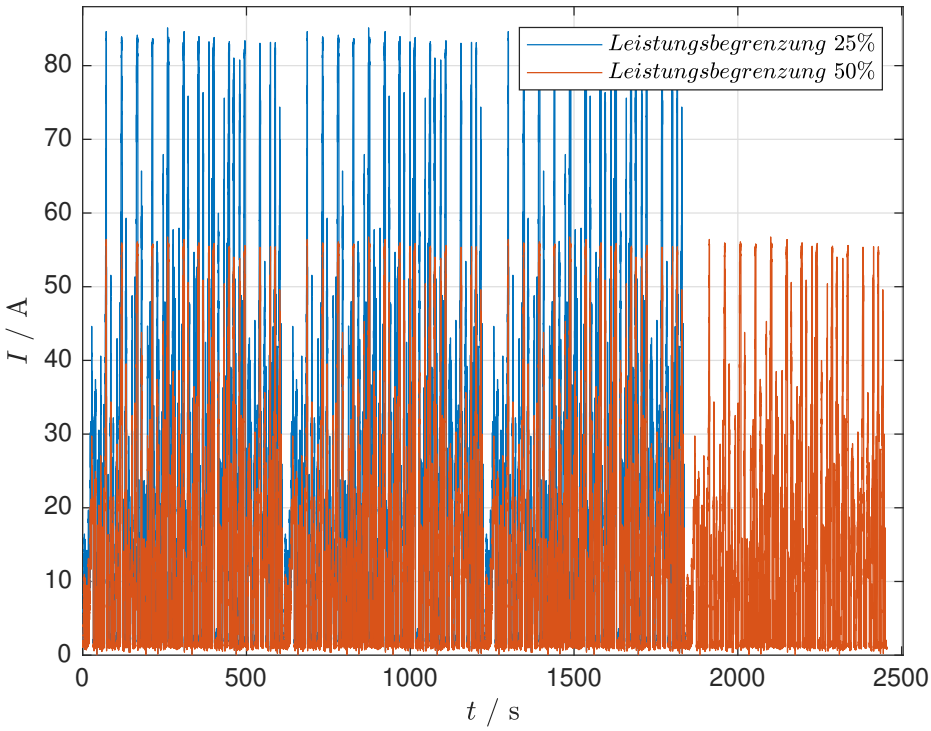


Abbildung 5.6.: Fahrtprofile im Vergleich: Leistungsbegrenzung 25% vs. 50%

Die Simulation des neu gewonnenen Fahrtprofils wird ebenfalls bei einer Akkutemperatur von 22 ℃ gestartet. Abbildung 5.7 zeigt den gemessenen Strom über den Verlauf der Simulation. Im Vergleich zum Fahrtprofil mit einer Leistungsbegrenzung von 25%, wurde beim Fahrtprofil mit einer Leistungsbegrenzung von 50% das Ende der Simulation erreicht.

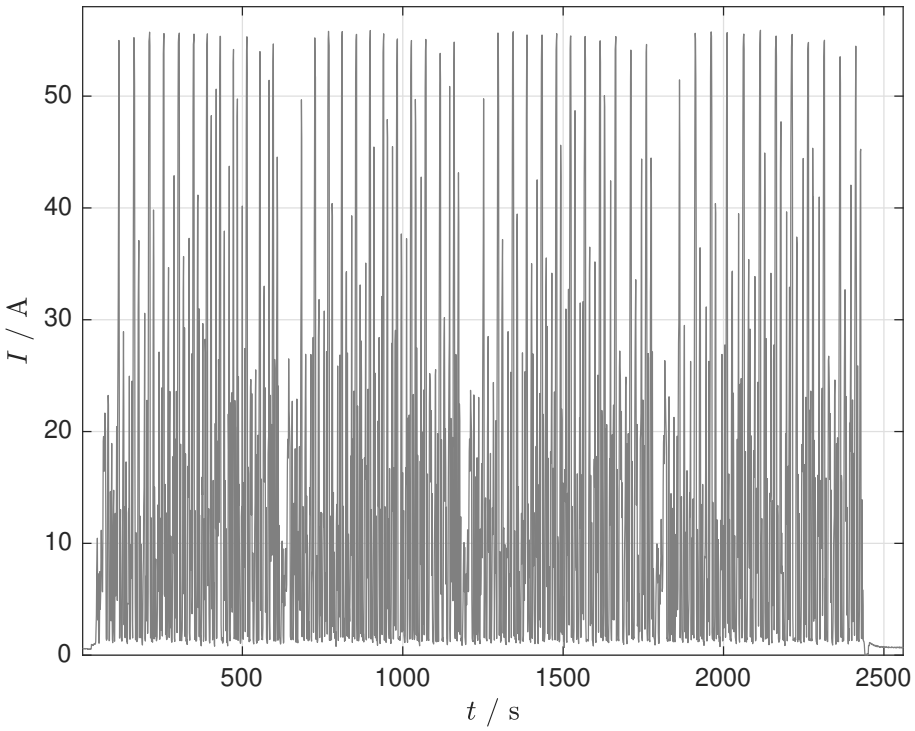


Abbildung 5.7.: Gemessener Strom bei einer Leistungsbegrenzung von 50%

Abbildung 5.8 zeigt den entsprechenden Verlauf der Segmenttemperaturen. Diese steigen im Vergleich zum Fahrtprofil mit einer Leistungsbegrenzung von 25% erwartungsgemäß flacher an. Dabei erreichen die Segmente 1, 2 und 5 kurz vor Ende der Simulation eine Temperatur von 59 °C.

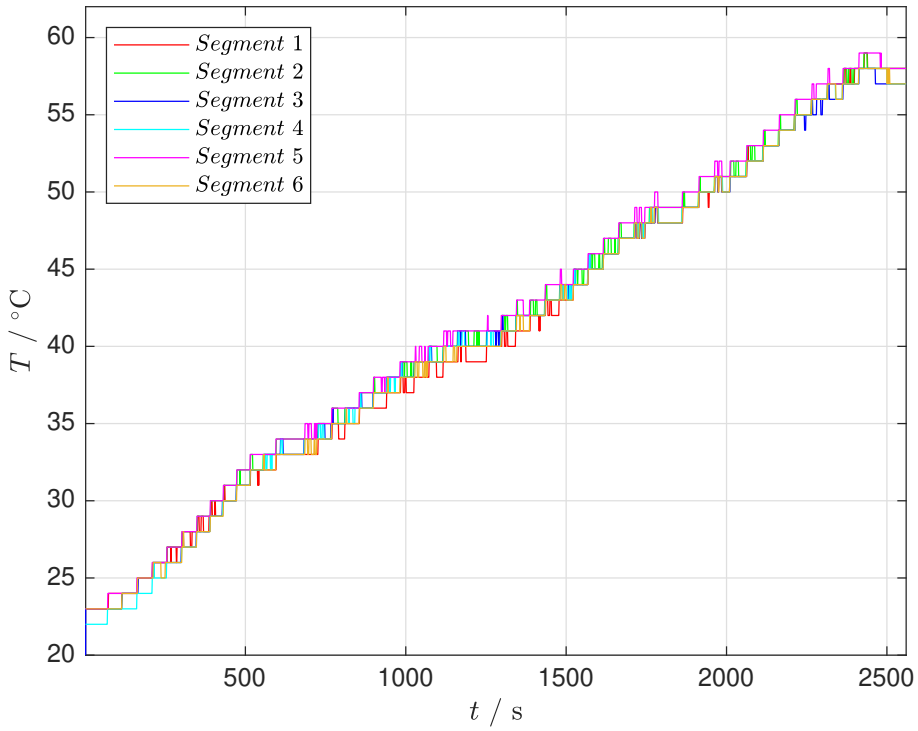


Abbildung 5.8.: Gemessene Segmenttemperaturen bei einer Leistungsbegrenzung von 50%

Der Verlauf der Zellspannungen pro Akkusegment ist in Abbildung 5.9 dargestellt. Im Vergleich zu Abbildung 5.4 fällt diese zwar flacher aus, weist aber am Ende der Simulation eine höhere Entladung der Zellen auf. Kurz vor Ende der Simulation ist verhältnismäßig zur restlichen Simulation ein stärkerer Spannungsabfall ersichtlich. Dieser entspricht dem üblichen Verhalten von Lithium-Ionen Zellen. Die höhere Entladung des Akkus resultierte wiederum in ein kontinuierliches Abdriften der Zellspannung des Segment 1.

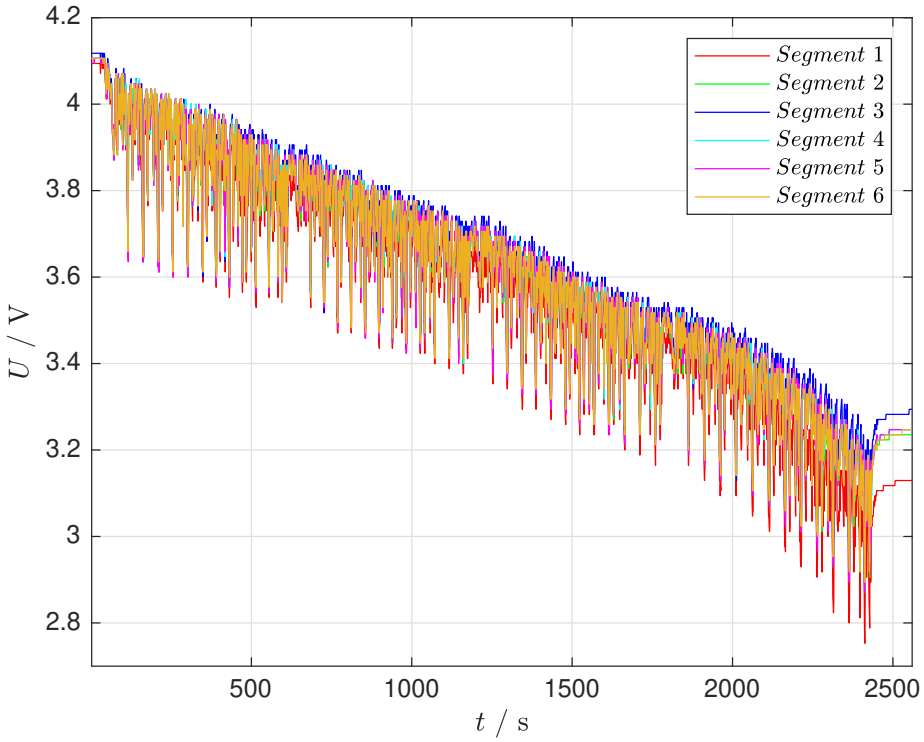


Abbildung 5.9.: Gemessene Zellspannungen pro Akkusegment bei einer Leistungsbegrenzung von 50%

Abbildung 5.10 zeigt die Entwicklung des SOC über den Verlauf der Fahrtsimulation mit einer Leistungsbegrenzung von 50%. Diese weist konsistent zu den Zellspannungen ebenfalls auf eine höhere Entladung im Vergleich zum Fahrtprofil mit einer Leistungsbegrenzung von 25% hin. Jedoch scheint sie nicht die steiler werdende Kurve der Zellspannungen gegen Ende der Simulation widerzuspiegeln. Da aber Verhältnismäßig die Temperatur und nicht die Akkukapazität den limitierenden Faktor darstellt, kann in diesem Fall die Abweichung in der SOC-Schätzung vernachlässigt werden.

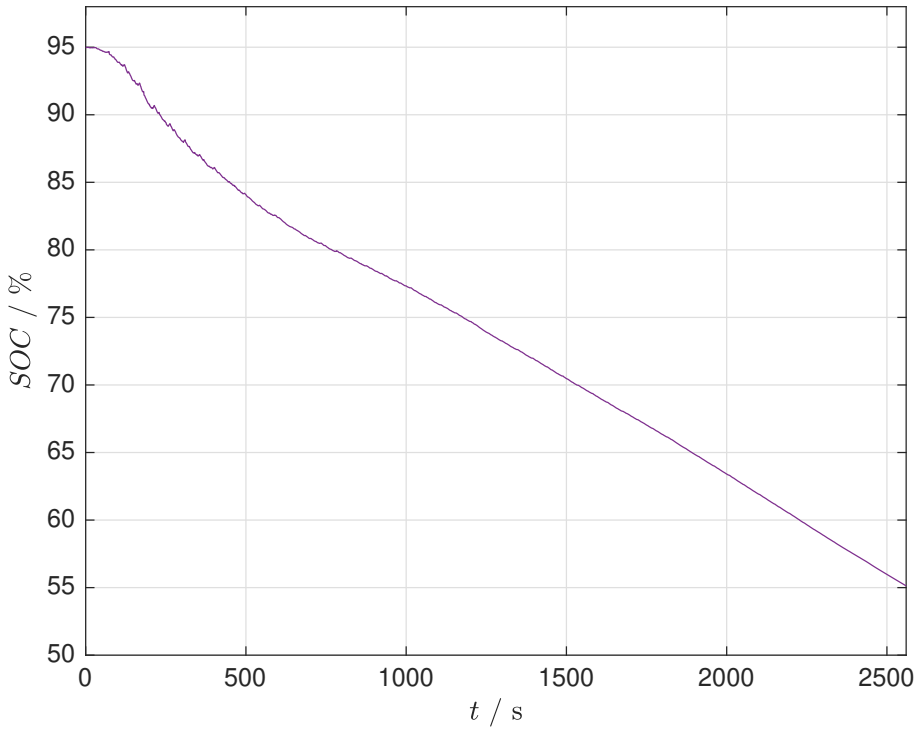


Abbildung 5.10.: Ermittelter SOC bei einer Leistungsbegrenzung von 50%

Aus den gewonnenen Daten kann geschlossen werden, dass bei einer Starttemperatur von 22 ℃ mit einer Leistungsbegrenzung von 50% eine Distanz von 22 km absolviert werden kann, ohne den Akku zu überhitzen oder diesen vorzeitig bis zur Entladeschlussspannung zu entladen. Für das Endurance and Efficiency Event ist somit die Akkutemperatur der limitierende Faktor. Die Akkukapazität spielt nur eine untergeordnete Rolle obwohl diese durch Segment 1 deutlich beeinträchtigt ist.

Durch die erfolgreiche Ermittlung eines Leistungsbegrenzungsrichtwertes für das Endurance and Efficiency Event wurde die Funktionsfähigkeit des implementierten Validierungstool bewiesen.

6. Zusammenfassung und Ausblick

6.1. Zusammenfassung

Mithilfe der Programmierplattform MATLAB und dessen Blockdiagrammumgebung SIMULINK gelang es, ein funktionsfähiges Validierungstool für den Akku des CTM Rennwagens zu implementieren, welches über ein simples Kontrollpanel bedient werden kann. Es wurde erfolgreich über einen CAN-Bus mit dem Akku und dem Batteriesimulator kommuniziert. Hierbei wurde ein Stromprofil in Echtzeit an den Batteriesimulator gesendet, welcher daraufhin eine Last für den Akku simulierte. Vom Akku empfangene Sensorwerte konnten in Echtzeit grafisch dargestellt werden. Des weiteren war es möglich, über einen UKF den SOC während der laufenden Simulation zu ermitteln und diesen ebenfalls in Echtzeit grafisch darzustellen. Über das Validierungstool konnte letztlich eine Aussage zur notwendigen Leistungsbegrenzung für das Endurance and Efficiency Event getroffen werden.

6.2. Reflexion und Ausblick

MATLAB stellt mit der Vehicle Network Toolbox eine potente Grundlage zur Kommunikation über einen CAN-Bus dar. In Kombination mit der eingängigen Handhabung von SIMULINK und dessen Funktionsblöcken sind die Möglichkeiten für Erweiterungen des implementierten Validierungstool vielfältig. Beispielsweise könnte die Genauigkeit der SOC-Ermittlung, über eine entsprechende Erweiterung der Lookup-Tabellen um die Dimension der Temperatur verbessert werden.

Die vom Datenlogger des Rennwagens ausgegebenen Log-Dateien sind grundsätzlich in einem ASC-Format vorzufinden. Eine Erweiterung des Validierungstools könnte darin bestehen, den Prozess des Einlesens und Verarbeitens der Log-Dateien weiter zu automatisieren, in dem die ASC-Dateien direkt eingelesen werden und nicht zuerst in ein CSV-Format übersetzt werden müssen.

Literaturverzeichnis

[1] W.Zimmermann und R.Schmidgall, *Bussysteme in der Fahrzeugtechnik - Protokolle, Standards und Softwarearchitektur*, 5. Aufl. Wiesbaden: Springer Vieweg, 2014.

[2] K.Borgeest, *Elektronik in der Fahrzeugtechnik - Hardware, Software, Systeme und Projektmanagement*, 2. Aufl. Wiesbaden: Vieweg+Teubner, GWV Fachverlag GmbH, 2010.

[3] D.Hrach und M.Cifrain, „Batterietechnik und -management im Elektrofahrzeug," *Elektrotechnik & Informationstechnik*, Vol. 128/1-2, S. 16–21, 2011.

[4] H.Kagermann, *Geleitwort, In: R. Korthauer (Hg.), Handbuch Lithium-Ionen- Batterien*, 1. Aufl. Berlin Heidelberg: Springer Vieweg, 2013, S.V-VI.

[5] A.Karle, *Elektromobilität - Grundlagen und Praxis*, 3. Aufl. München: Carl Hanser Verlag GmbH Co. KG, 2018.

[6] M.Zhang und X.Fan, „Review on the State of Charge Estimation Methods for Electric Vehicle Battery," *World Electric Vehicle Journal*, Vol. 11, S. Article number 23, 2020.

[7] R.Pfeil, *Methodischer Ansatz zur Optimierung von Energieladestrategien für elektrisch angetriebene Fahrzeuge*, 1. Aufl. Wiesbaden: Springer Vieweg, 2018.

[8] M.Paulweber und K.Lebert, *Mess- und Prüfstandstechnik; Antriebsstrangentwicklung, Hybridisierung, Elektrifizierung In: H. List (Hg.), Der Fahrzeugantrieb*, 1. Aufl. Wiesbaden: Springer Vieweg, 2014.

[9] Z.Liu, S.Onori, und A.Ivanco, „Synthesis and Experimental Validation of Battery Aging Test Profiles Based on Real-World Duty Cycles for 48-V Mild Hybrid Vehicles," *IEEE Transactions on Vehicular Technology*, Vol. 66, Nr. 10, S. 8702–8709, 2017.

[10] G.Schnell und B.Wiedemann, *Bussysteme in der Automatisierungs- und Prozesstechnik; Grundlagen, Systeme und Anwendungen der industriellen Kommunikation*, 9. Aufl. Wiesbaden: Springer Vieweg, 2018.

[11] K.Reif, *Bosch Autoelektrik und Autoelektronik - Bordnetze, Sensoren und elektronische Systeme*, 6. Aufl. Wiesbaden: Vieweg+Teubner, 2011.

[12] K.Möller, *Übersicht über die Speichersysteme/Batteriesysteme, In: R. Korthauer (Hg.), Handbuch Lithium-Ionen- Batterien*, 1. Aufl. Berlin Heidelberg: Springer Vieweg, 2013, S.3-9.

[13] S.Leuthner, *Übersicht zu Lithium-Ionen-Batterien, In: R. Korthauer (Hg.), Handbuch Lithium-Ionen- Batterien*, 1. Aufl. Berlin Heidelberg: Springer Vieweg, 2013, S.13-20.

[14] R.Dorn, R.Schwartz, und B.Steurich, *Batteriemanagementsystem, In: R. Korthauer (Hg.), Handbuch Lithium-Ionen- Batterien*, 1. Aufl. Berlin Heidelberg: Springer Vieweg, 2013, S.3-9.

[15] M.Zeyen und A.Wiebelt, *Thermisches Management der Batterie, In: R. Korthauer (Hg.), Handbuch Lithium-Ionen- Batterien*, 1. Aufl. Berlin Heidelberg: Springer Vieweg, 2013, S.165-175.

[16] M.Roscher, „Zustandserkennung von LiFePO4-Batterien für Hybrid- und Elektrofahrzeuge," *Zugl.: Techn. Hochschule Aachen, Diss., 2010.*, Bd. 54. Aachener Beiträge des ISEA. Shaker, Aachen, 2011.

[17] M.Murnane und A.Ghazel, „A Closer Look at State of Charge (SOC) and State of Health (SOH) Estimation Techniques for Batteries," *ANALOG DEVICES Inc., Technical Article*, 2017.

[18] F.Sun, R.Xiong, und H.He, „A systematic state-of-charge estimation framework for multi-cell battery pack in electric vehicles using bias correction technique," *Applied Energy*, Vol. 162, S. 1399–1409, 2016.

[19] W.Wand, X.Wang, C.Xiang, C.Wei, und Y.Zhao, „Unscented Kalman Filter-Based Battery SOC Estimation and Peak Power Prediction Method for Power Distribution of Hybrid Electric Vehicles," *IEEE Access*, Vol. 6, S. 35 957–35 965, 2018.

[20] S.Santhanagopalan und R.E.White, „State of charge estimation using an unscented filter for high power lithium ion cells," *International Journal Of Energy Research*, Vol. 34, S. 152–163, 2009.

[21] P.Stocker, „Sensordaten aus Testfahrt ÖAMTC Fahrtechnik Zentrum Innsbruck," Comma-separated values (CSV)-Datei (Nicht öffentlich), 2021, Campus Tirol Motorsport.

[22] J. Gazzarri, „Battery Modeling," 2012, Zugriff: 02.05.2022. [Online]. Verfügbar: https://www.mathworks.com/matlabcentral/fileexchange/36019-battery-modeling)

[23] L.Czarnecka, „High voltage accumulator, Technical Presentation," Presentation, 2020, Campus Tirol Motorsport.

[24] I.Kasikei, *Planung von Elektroanlagen - Theorie, Vorschriften, Praxis*, 3. Aufl. Berlin Heidelberg: Springer Vieweg, 2018.

[25] Peak-System, „PCAN-USB CAN Interface for USB Datenblatt," 2022, Zugriff: 01.05.2022. [Online]. Verfügbar: https://www.peak-system.com/produktcd/Pdf/Deutsch/PCAN-USB_UserMan_deu.pdf

Abbildungsverzeichnis

2.1. Bus-Struktur 3
2.2. Vernetzung von Steuergeräten im CAN 4
2.3. Netzknoten im CAN 5
2.4. Typische Entladekurve einer Lithium-Ionen Zelle bei unterschiedlichen Temperaturen und gleichem Entladestrom 7
2.5. Ersatzschaltbild eines Akkus mit n RC-Gliedern 8

3.1. Geplanter Aufbau des CAN-Bus 10

4.1. Erstellen einer CAN Datenbank in BUSMASTER 14
4.2. Definition einer CAN-Nachricht in BUSMASTER 15
4.3. Konvertierung vom DBF zum DBC Format 17
4.4. Rechteckstromprofil zur Pulsentladung des Akkus 19
4.5. Gewähltes Stromprofil aus Messdaten einer Testfahrt 20
4.6. Generierung des Rechteckstromprofils in SIMULINK 21
4.7. Einlesen des Stromprofils der Fahrtsimulation in SIMULINK 22
4.8. SIMULINK-Blöcke für die CAN-Kommunikation 23
4.9. Peak-System CAN zu USB Adapter 23
4.10. CAN Configuration-Block 24
4.11. CAN Pack-Block 24
4.12. CAN Transmit-Block 25
4.13. CAN Receive-Block 25
4.14. CAN Unpack-Block 26
4.15. Function-Call-Subsystem 26
4.16. BusCreator-Block 27
4.17. Einstellen der Simultionsgeschwindigkeit 27
4.18. UKF-Block 28
4.19. Grafische Darstellung der Akkusensordaten und des SOC über einen Viewer 29
4.20. Entwurfsbereich des App Designer 30
4.21. Codebereich des App Designer 31
4.22. Kontrollpanel des Validierungstools 33
4.23. CAD Modell des Akkus 34
4.24. Batteriesimulator und Transformator 34
4.25. Kabel zur Verbindung von Akku-Batteriesimulator und Tranformator-Netz 35
4.26. Verbindung des Batteriesimulators mit dem Transformator 36

4.27. Latchingplatine 37
4.28. Verteilerbox mit Latchingplatine, Not-Aus Schalter und Reset-Knopf . . 38
4.29. Kabelbaum zur Kommunikation mit der Hardware 39
4.30. Testaufbau 40
4.31. Gemessener Zellspannungen pro Akkusegment bei der Pulsentladung . 41
4.32. Gemessener Strom bei der Pulsentladung 42
4.33. Parameter Estimator in SIMULINK 43

5.1. Benutzeroberfläche des Validierungstool während der Fahrtsimulation . 44
5.2. Gemessener Strom bei einer Leistungsbegrenzung von 25% 45
5.3. Gemessene Segmenttemperaturen bei einer Leistungsbegrenzung von 25% 46
5.4. Gemessene Zellspannungen pro Akkusegment bei einer Leistungsbegrenzung von 25% 47
5.5. Ermittelter SOC bei einer Leistungsbegrenzung von 25% 48
5.6. Fahrtprofile im Vergleich: Leistungsbegrenzung 25% vs. 50% 49
5.7. Gemessener Strom bei einer Leistungsbegrenzung von 50% 50
5.8. Gemessene Segmenttemperaturen bei einer Leistungsbegrenzung von 50% 51
5.9. Gemessene Zellspannungen pro Akkusegment bei einer Leistungsbegrenzung von 50% 52
5.10. Ermittelter SOC bei einer Leistungsbegrenzung von 50% 53

Tabellenverzeichnis

4.1. Übersicht der notwendigen Signale für die CAN-Kommunikation 16
4.2. Pinbelegung des Signalsteckers vom Akku 37
4.3. RC-Ersatzschaltbildparameter 43

Abkürzungsverzeichnis

AMS Akkumulatormanagementsystem

CAN Controller Area Network

CSMA/CA Carrier Sense Multiple Access/Collision Avoidance

CSV Comma-separated values

CTM Campus Tirol Motorsport

DBC Data Base CAN

IMD Insulation Monitoring Device

KF Kalman-Filter

SDC Shutdown Circuit

SOC State Of Charge

UKF Unscented Kalman-Filter

USB Universal Serial Bus

A. DBC-Datei

```
VERSION ""


BS_:

BU_:


BO_ 577 accu_signal_2: 8 Vector__XXX
 SG_ AMS_OK : 45|1@1- (1,0) [0|1] "" Vector__XXX
 SG_ SDC_closed : 43|1@1- (1,0) [0|1] "" Vector__XXX

BO_ 576 accu_signal_1: 8 Vector__XXX
 SG_ min_cv : 23|16@0+ (0.0001,0) [0|6.5535] "V" Vector__XXX
 SG_ max_cv : 7|16@0+ (0.0001,0) [0|6.5535] "V" Vector__XXX

BO_ 578 accu_signal_3: 8 Vector__XXX
 SG_ i_high : 55|16@0- (0.04577,0) [-1499.79|1499.75] "A" Vector__XXX
 SG_ i_low : 39|16@0- (0.004577,0) [-149.979|149.975] "A" Vector__XXX

BO_ 512 cv1: 8 Vector__XXX
 SG_ cv1_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv1_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv1_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv1_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv1_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv1_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv1_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv1_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 513 cv2: 8 Vector__XXX
 SG_ cv2_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv2_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv2_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv2_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
```

```
 SG_ cv2_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv2_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv2_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv2_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 514 cv3: 8 Vector__XXX
 SG_ cv3_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv3_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv3_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv3_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv3_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv3_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv3_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv3_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 515 cv4: 8 Vector__XXX
 SG_ cv4_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv4_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv4_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv4_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv4_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv4_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv4_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv4_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 516 cv5: 8 Vector__XXX
 SG_ cv5_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv5_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv5_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv5_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv5_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv5_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv5_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv5_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 517 cv6: 8 Vector__XXX
 SG_ cv6_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv6_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv6_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv6_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv6_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv6_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
```

```
 SG_ cv6_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv6_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 518 cv7: 8 Vector__XXX
 SG_ cv7_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv7_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv7_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv7_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv7_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv7_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv7_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv7_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 519 cv8: 8 Vector__XXX
 SG_ cv8_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv8_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv8_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv8_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv8_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv8_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv8_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv8_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 520 cv9: 8 Vector__XXX
 SG_ cv9_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv9_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv9_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv9_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv9_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv9_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv9_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv9_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 521 cv10: 8 Vector__XXX
 SG_ cv10_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv10_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv10_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv10_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv10_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv10_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv10_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv10_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
```

```
BO_ 522 cv11: 8 Vector__XXX
 SG_ cv11_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv11_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv11_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv11_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv11_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv11_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv11_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv11_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 523 cv12: 8 Vector__XXX
 SG_ cv12_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv12_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv12_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv12_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv12_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv12_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv12_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv12_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 524 cv13: 8 Vector__XXX
 SG_ cv13_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv13_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv13_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv13_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv13_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv13_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv13_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv13_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 525 cv14: 8 Vector__XXX
 SG_ cv14_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv14_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv14_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv14_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv14_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv14_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv14_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv14_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 526 cv15: 8 Vector__XXX
```

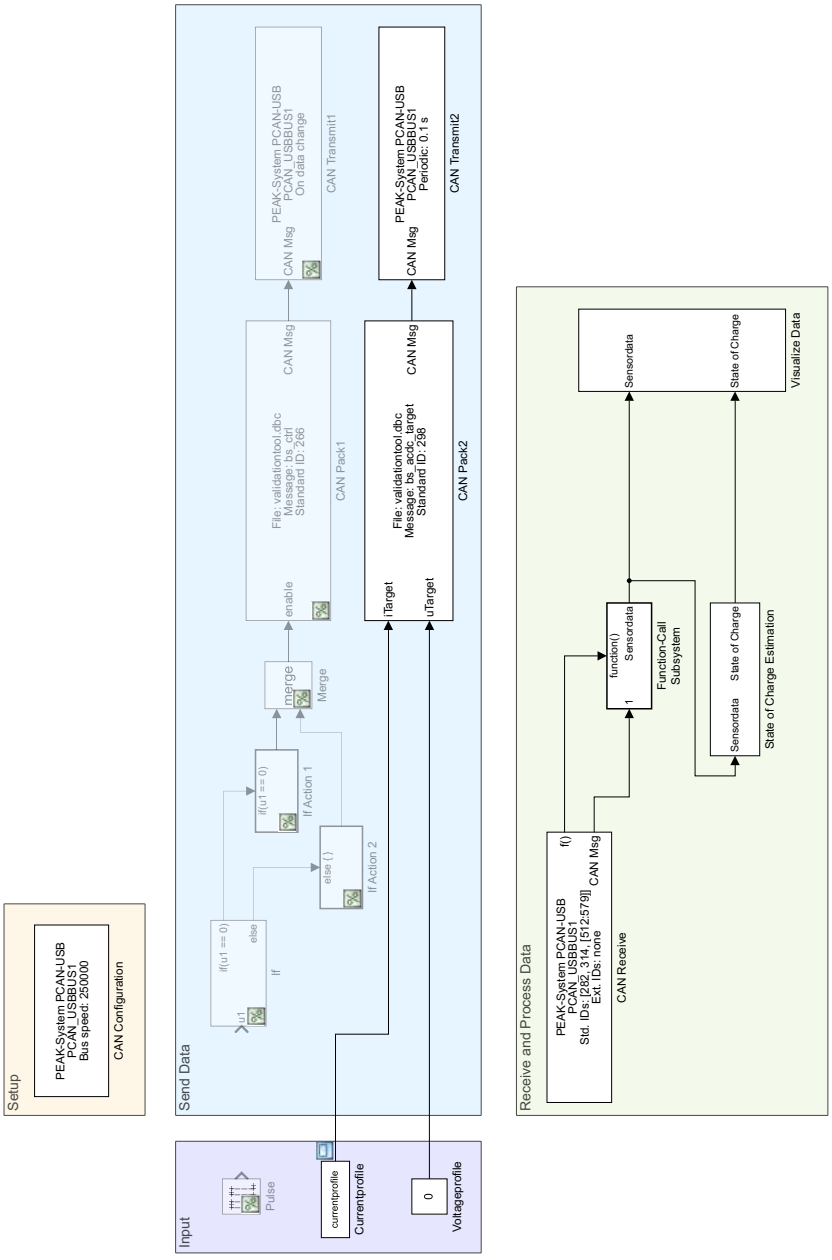
```
 SG_ cv15_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv15_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv15_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv15_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv15_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv15_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv15_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv15_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 527 cv16: 8 Vector__XXX
 SG_ cv16_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv16_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv16_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv16_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv16_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv16_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv16_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv16_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 528 cv17: 8 Vector__XXX
 SG_ cv17_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv17_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv17_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv17_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv17_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv17_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv17_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv17_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 529 cv18: 8 Vector__XXX
 SG_ cv18_8 : 56|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv18_7 : 48|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv18_6 : 40|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv18_5 : 32|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv18_4 : 24|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv18_3 : 16|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv18_2 : 8|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX
 SG_ cv18_1 : 0|8@1+ (0.011765,2) [2|5.00008] "V" Vector__XXX

BO_ 544 temp1: 8 Vector__XXX
 SG_ ct1_8 : 56|1@1- (1,0) [0|1] "C" Vector__XXX
 SG_ ct1_7 : 48|8@1+ (1,0) [0|255] "C" Vector__XXX
```

```
 SG_ ct1_6 : 40|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct1_5 : 32|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct1_4 : 24|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct1_3 : 16|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct1_2 : 8|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct1_1 : 0|8@1+ (1,0) [0|255] "C" Vector__XXX

BO_ 545 temp2: 8 Vector__XXX
 SG_ ct2_8 : 56|1@1- (1,0) [0|1] "C" Vector__XXX
 SG_ ct2_7 : 48|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct2_6 : 40|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct2_5 : 32|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct2_4 : 24|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct2_3 : 16|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct2_2 : 8|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct2_1 : 0|8@1+ (1,0) [0|255] "C" Vector__XXX

BO_ 546 temp3: 8 Vector__XXX
 SG_ ct3_8 : 56|1@1- (1,0) [0|1] "C" Vector__XXX
 SG_ ct3_7 : 48|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct3_6 : 40|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct3_5 : 32|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct3_4 : 24|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct3_3 : 16|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct3_2 : 8|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct3_1 : 0|8@1+ (1,0) [0|255] "C" Vector__XXX

BO_ 547 temp4: 8 Vector__XXX
 SG_ ct4_8 : 56|1@1- (1,0) [0|1] "C" Vector__XXX
 SG_ ct4_7 : 48|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct4_6 : 40|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct4_5 : 32|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct4_4 : 24|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct4_3 : 16|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct4_2 : 8|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct4_1 : 0|8@1+ (1,0) [0|255] "C" Vector__XXX

BO_ 548 temp5: 8 Vector__XXX
 SG_ ct5_8 : 56|1@1- (1,0) [0|1] "C" Vector__XXX
 SG_ ct5_7 : 48|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct5_6 : 40|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct5_5 : 32|8@1+ (1,0) [0|255] "C" Vector__XXX
```

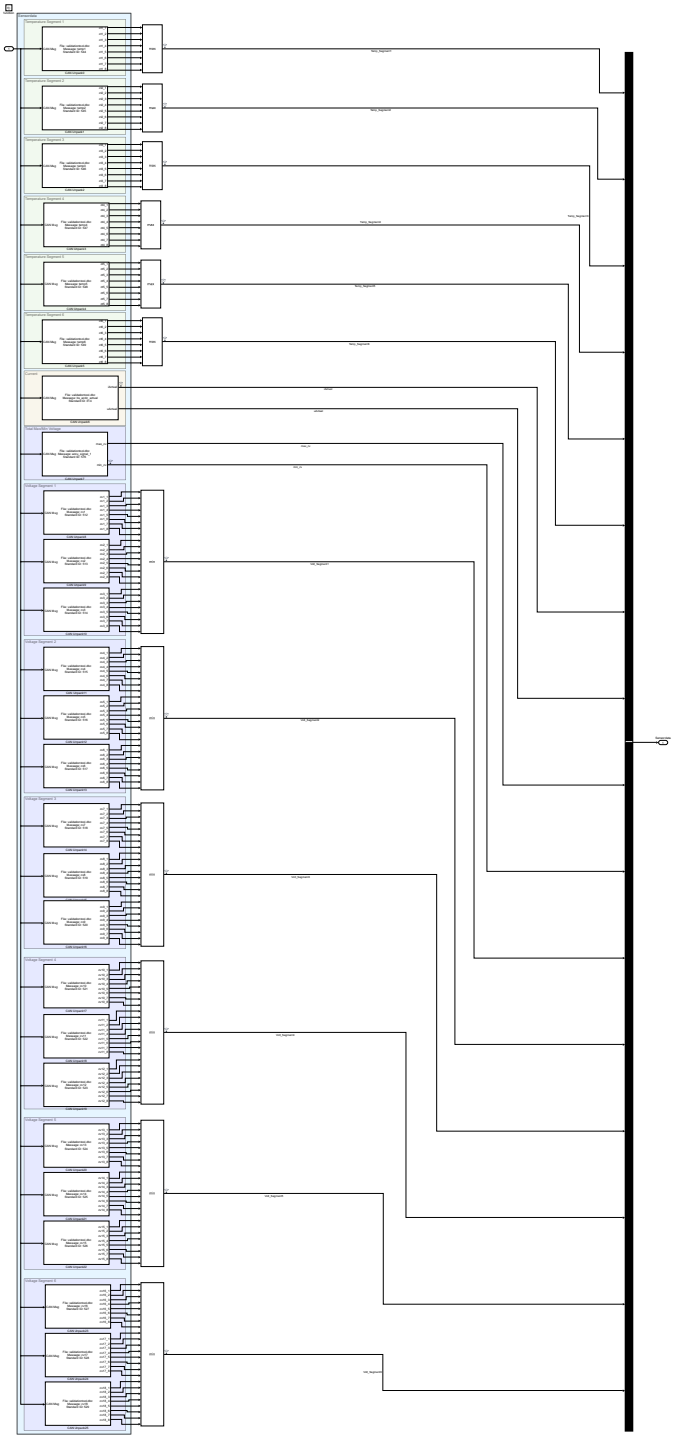
```
 SG_ ct5_4 : 24|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct5_3 : 16|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct5_2 : 8|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct5_1 : 0|8@1+ (1,0) [0|255] "C" Vector__XXX

BO_ 549 temp6: 8 Vector__XXX
 SG_ ct6_8 : 56|1@1- (1,0) [0|1] "C" Vector__XXX
 SG_ ct6_7 : 48|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct6_6 : 40|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct6_5 : 32|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct6_4 : 24|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct6_3 : 16|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct6_2 : 8|8@1+ (1,0) [0|255] "C" Vector__XXX
 SG_ ct6_1 : 0|8@1+ (1,0) [0|255] "C" Vector__XXX

BO_ 282 bs_status: 1 Vector__XXX
 SG_ status : 0|8@1+ (1,0) [0|255] "" Vector__XXX

BO_ 314 bs_acdc_actual: 4 Vector__XXX
 SG_ iActual : 16|16@1+ (0.003906,0) [0|255.98] "A" Vector__XXX
 SG_ uActual : 0|16@1+ (0.015625,0) [0|1023.98] "V" Vector__XXX

BO_ 298 bs_acdc_target: 4 Vector__XXX
 SG_ iTarget : 16|16@1+ (0.003906,0) [0|255.98] "A" Vector__XXX
 SG_ uTarget : 0|16@1+ (0.015625,0) [0|0.015625] "V" Vector__XXX

BO_ 266 bs_ctrl: 1 Vector__XXX
 SG_ enable : 0|8@1+ (1,0) [0|2] "" Vector__XXX

BO_ 769 accu_ctrl: 8 Vector__XXX
 SG_ anti_timeout : 8|8@1+ (1,0) [0|128] "" Vector__XXX
```

B. SIMULINK-Modell

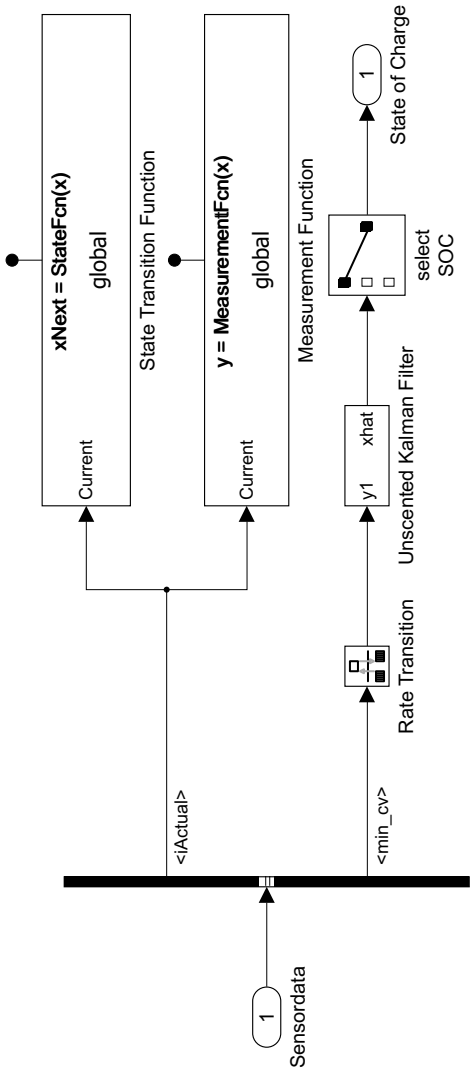
B.1. Oberste Ebene



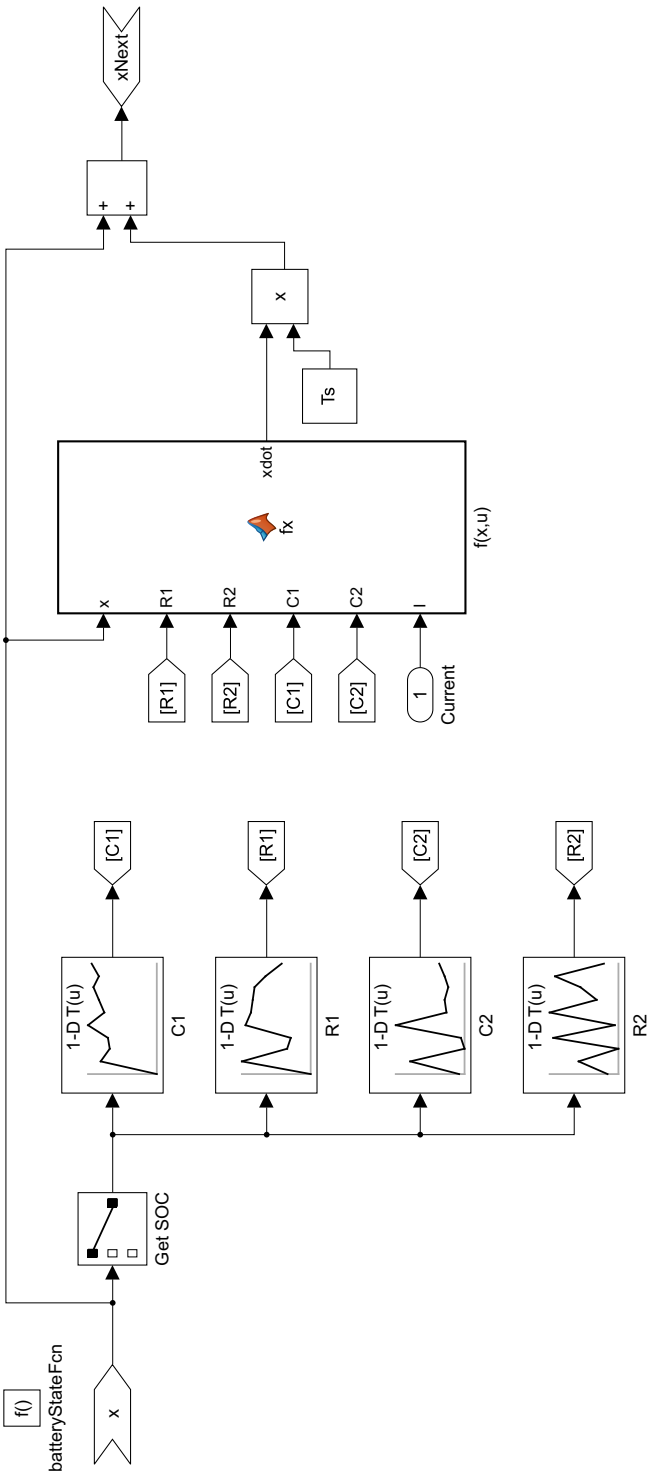
B.2. Function-Call-Subsystem



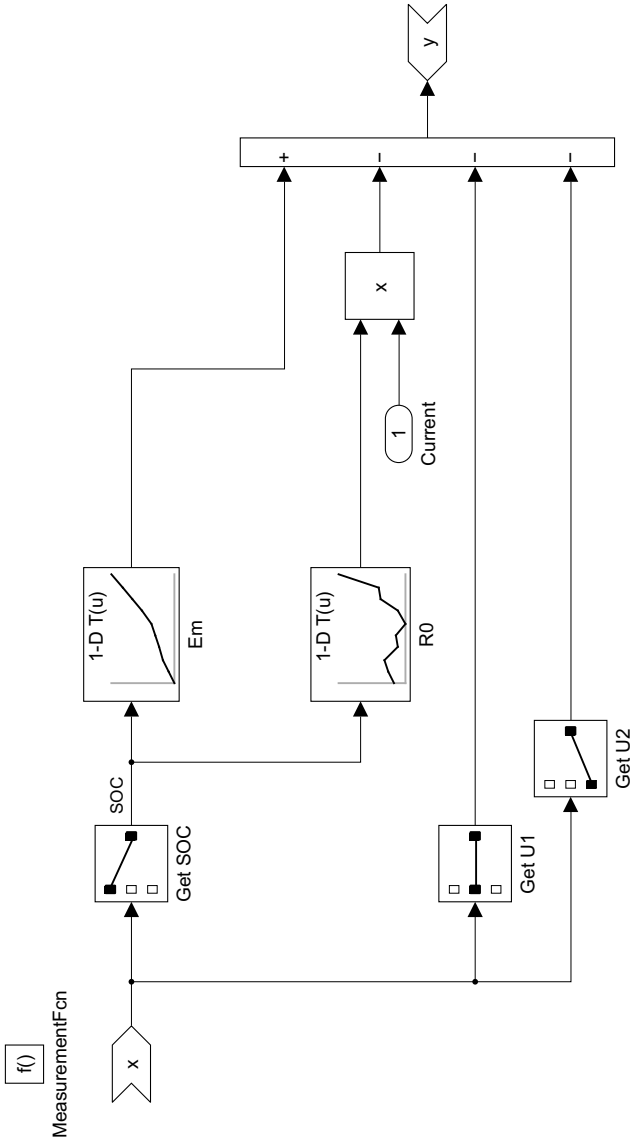
B.3. SOC-Subsystem



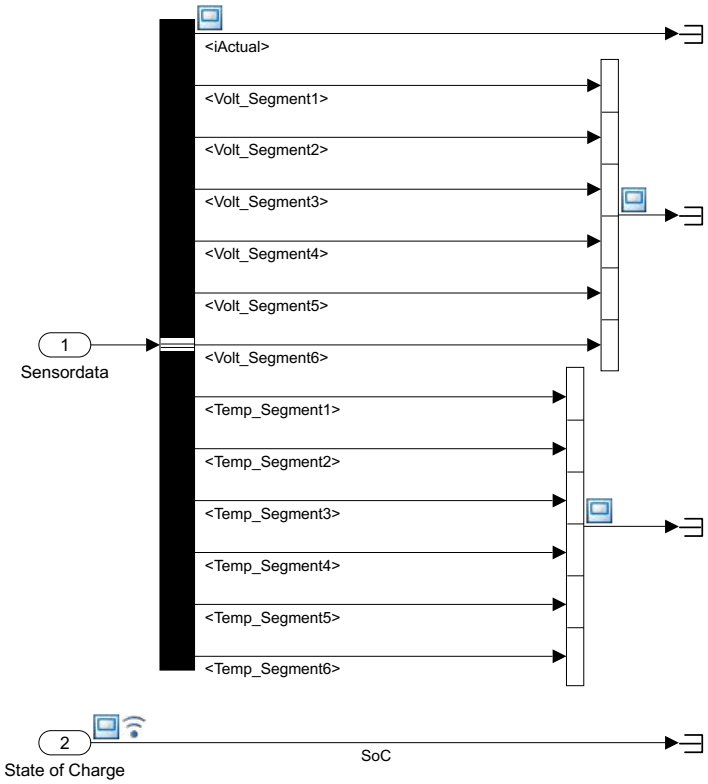
B.4. Zustandsübergangsfunktion



B.5. Messfunktion



B.6. Visualisierungs-Subsystem



C. MATLAB-Code für Kontrollpanel

```
classdef controlpanel < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        AccumulatorControlPanel     matlab.ui.container.Panel
        GetStatusButton             matlab.ui.control.Button
        SDCClosedEditField          matlab.ui.control.NumericEditField
        SDCClosedEditFieldLabel     matlab.ui.control.Label
        SDCClosedLamp               matlab.ui.control.Lamp
        AMSOKEditField              matlab.ui.control.NumericEditField
        AMSOKEditFieldLabel         matlab.ui.control.Label
        AMSOKLamp                   matlab.ui.control.Lamp
        OffButton_2                 matlab.ui.control.Button
        OnButton_2                  matlab.ui.control.Button
        InitializeButton            matlab.ui.control.Button
        AccumulatorvalidationtoolV10Label matlab.ui.control.Label
        ValidationPanel             matlab.ui.container.Panel
        StartSimButton              matlab.ui.control.Button
        StopSimButton               matlab.ui.control.Button
        InitializeSimButton         matlab.ui.control.Button
        ImportDataButton            matlab.ui.control.Button
        BatterySimulatorControlPanel matlab.ui.container.Panel
        GetStatusButton_2           matlab.ui.control.Button
        StatusEditField             matlab.ui.control.NumericEditField
        StatusEditFieldLabel        matlab.ui.control.Label
        StatusBSLamp                matlab.ui.control.Lamp
        ResetButton                 matlab.ui.control.Button
        OffButton                   matlab.ui.control.Button
        OnButton                    matlab.ui.control.Button
        Image                       matlab.ui.control.Image
    end


    properties (Access = private)
        channel
        db
        msg_acc_busspeed
        msg_acc_ctrl
        msg_acc_status
        msg_acc_status_filter
        msg_acc_status_AMS
        msg_acc_status_SDC
```

```
44          msg_bs_ctrl
45          msg_bs_status
46          msg_bs_status_filter
47          msg_bs_status_current
48      end
49
50
51      % Callbacks that handle component events
52      methods (Access = private)
53
54          % Code that executes after component creation
55          function startupFcn(app)
56              % Clear Workspace
57              evalin('base', 'clear all')
58              clc
59
60              % Setup CAN Channel
61              app.db = canDatabase('validationtool.dbc');
62              app.channel = canChannel('PEAK-System', 'PCAN_USBBUS1');
63              app.channel.Database = app.db;
64              configBusSpeed(app.channel, 250000);
65              start(app.channel);
66              assignin('base','channel',app.channel);
67
68              % Define CAN Messages for GUI Buttons
69              app.msg_acc_ctrl = canMessage(app.db, 'accu_ctrl');
70              app.msg_bs_ctrl = canMessage(app.db, 'bs_ctrl');
71
72              % Set Lamp Colors
73              app.AMSOKLamp.Color = 'r';
74              app.SDCClosedLamp.Color = 'r';
75              app.StatusBSLamp.Color = 'r';
76          end
77
78          % Button pushed function: InitializeButton
79          function InitializeButtonPushed(app, event)
80              %Initialize accumulator
81              transmitPeriodic(app.channel, app.msg_acc_ctrl , "On", 0.1);
82              app.msg_acc_ctrl.Data=[0 0 0 0 0 0 0 0];
83              assignin('base','msg_acc_ctrl',app.msg_acc_ctrl);
84          end
85
86          % Button pushed function: OnButton_2
87          function OnButton_2Pushed(app, event)
88              %Start accumulator
89              app.msg_acc_ctrl.Data=[0 128 0 0 0 0 0 0];
90              assignin('base','msg_acc_ctrl',app.msg_acc_ctrl);
91          end
92
93          % Button pushed function: OffButton_2
94          function OffButton_2Pushed(app, event)
95              %Stop accumulator
```

```
96          app.msg_acc_ctrl.Data=[0 0 0 0 0 0 0 0];
97          assignin('base','msg_acc_ctrl',app.msg_acc_ctrl);
98      end
99
100     % Button pushed function: GetStatusButton
101     function GetStatusButtonPushed(app, event)
102         % Get and display accumulator status
103         app.msg_acc_status = receive(app.channel, Inf, 'OutputFormat', 'timetable');
104         app.msg_acc_status_filter = groupfilter(app.msg_acc_status,'Time',@(x) x ==
                577, 'ID');
105         app.msg_acc_status_AMS = (app.msg_acc_status_filter.Signals{end}.AMS_OK);
106         app.msg_acc_status_SDC = (app.msg_acc_status_filter.Signals{end}.SDC_closed);
107
108         app.AMSOKEditField.Value = app.msg_acc_status_AMS;
109
110         if  app.AMSOKEditField.Value == -1
111             app.AMSOKEditField.Value=1;
112             app.AMSOKLamp.Color = 'g';
113         else
114             app.AMSOKEditField.Value=0;
115             app.AMSOKLamp.Color = 'r';
116         end
117
118         app.SDCClosedEditField.Value = app.msg_acc_status_SDC;
119
120         if  app.SDCClosedEditField.Value == -1
121             app.SDCClosedEditField.Value=1;
122             app.SDCClosedLamp.Color = 'g';
123         else
124             app.SDCClosedEditField.Value=0;
125             app.SDCClosedLamp.Color = 'r';
126         end
127     end
128
129     % Button pushed function: OnButton
130     function OnButtonPushed(app, event)
131         %Start batterysimulator
132         app.msg_bs_ctrl.Data=[2];
133         assignin('base','msg_bs_ctrl', app.msg_bs_ctrl);
134         transmit(app.channel, app.msg_bs_ctrl);
135     end
136
137     % Button pushed function: OffButton
138     function OffButtonPushed(app, event)
139         %Stop batterysimulator
140         stop(app.channel)
141         start(app.channel)
142         app.msg_bs_ctrl.Data=[0];
143         assignin('base','msg_bs_ctrl', app.msg_bs_ctrl);
144         transmit(app.channel, app.msg_bs_ctrl);
145     end
146
```

```
147         % Button pushed function: ResetButton
148         function ResetButtonPushed(app, event)
149             %Reset batterysimulator (needed when StatusBSLamp is orange)
150             app.msg_bs_ctrl.Data=[1];
151             assignin('base','msg_bs_ctrl', app.msg_bs_ctrl);
152             transmit(app.channel, app.msg_bs_ctrl);
153         end
154 
155         % Button pushed function: GetStatusButton_2
156         function GetStatusButton_2Pushed(app, event)
157             % Get and display batterysimulator status
158             app.msg_bs_status = receive(app.channel, Inf, 'OutputFormat', 'timetable');
159             app.msg_bs_status_filter=groupfilter(app.msg_bs_status,'Time',@(x) x == 282,
                    'ID');
160             app.msg_bs_status_current=app.msg_bs_status_filter.Signals{end}.status;
161 
162             app.StatusEditField.Value = app.msg_bs_status_current;
163 
164             if app.StatusEditField.Value == 6
165                 app.StatusBSLamp.Color = 'g';
166             elseif app.StatusEditField.Value == 3
167                 app.StatusBSLamp.Color = 'r';
168             else
169                 app.StatusBSLamp.Color = [0.93, 0.69, 0.13];
170             end
171         end
172 
173         % Button pushed function: ImportDataButton
174         function ImportDataButtonPushed(app, event)
175             %% Import currentprofile
176 
177             [Time, Distance, IDC, UDC] = importfile('210721_zenzenhof.csv', [4847,
                    10983]);
178 
179             % 75% Power
180 %           idc = transpose(IDC);
181 %           idc = transpose([idc idc idc]);
182 %           time75 = transpose(0:.1:1841.0);
183 %           signal = [time75, idc];
184 
185             % 50% Power
186             IDC75single = transpose(IDC);
187             IDC75 = [IDC75single IDC75single IDC75single];
188             IDC75 = transpose(IDC75);
189 
190             for i=1:length(IDC75single)
191             IDC50single(i)=(0.5*IDC75single(i))/0.75;
192             end
193 
194             IDC50=[IDC50single IDC50single IDC50single];
195             IDC50=transpose(IDC50);
196 
```

```
197             IDC50=[IDC50single IDC50single IDC50single IDC50single];
198             idc=transpose(IDC50);
199
200             time50 = transpose(0:.1:2454.7);
201
202             signal = [time50, idc];
203
204             assignin('base','currentprofile', signal);
205
206             clear Time
207             clear Distance
208             clear IDC
209             clear UDC
210
211             %% Import equivalent circuit data for SOC estimation
212
213             load('estimatedparameters.mat');
214
215             SOC_LUT = (0.1:.1:1)';
216
217             % Lookup Table Breakpoints
218             Battery.SOC_LUT = SOC_LUT;
219
220             % Em open-circuit voltage
221             Battery.Em_LUT = [estimatedparameters.Em];
222
223             % Terminal Resistance Properties
224             % R0 Resistance
225             Battery.R0_LUT = [estimatedparameters.R0];
226
227             % RC Properties
228             % R1 Resistance
229             Battery.R1_LUT = [estimatedparameters.R1];
230
231             % R2 Resistance
232             Battery.R2_LUT = [estimatedparameters.R2];
233
234             % C1 Capacitance
235             Battery.C1_LUT = [estimatedparameters.C1];
236
237             % C2 Capacitance
238             Battery.C2_LUT = [estimatedparameters.C2];
239
240             % UKF Sample time
241             Ts=1;
242
243             assignin('base','Battery',Battery);
244             assignin('base','Ts',Ts);
245
246             clear SOC_LUT
247         end
248
```

```
249         % Button pushed function: InitializeSimButton
250         function InitializeSimButtonPushed(app, event)
251             %Initialize simulation (starting the simulation AFTER
252             %starting the accumulator leads to a accumulator CAN Timeout.
253             %Thus the simulation shall be initialized before the
254             %accumulator is started)
255             open_system('validationtool');
256             set_param('validationtool','SimulationCommand','start');
257             set_param('validationtool','SimulationCommand','pause');
258         end
259 
260         % Button pushed function: StartSimButton
261         function StartSimButtonPushed(app, event)
262             %Start simulation
263             set_param('validationtool','SimulationCommand','continue');
264         end
265 
266         % Button pushed function: StopSimButton
267         function StopSimButtonPushed(app, event)
268             %Stop simulation
269             set_param('validationtool', 'SimulationCommand', 'stop');
270         end
271     end
272 
273     % Component initialization
274     methods (Access = private)
275 
276         % Create UIFigure and components
277         function createComponents(app)
278 
279             % Create UIFigure and hide until all components are created
280             app.UIFigure = uifigure('Visible', 'off');
281             app.UIFigure.Position = [100 100 577 385];
282             app.UIFigure.Name = 'MATLAB App';
283 
284             % Create Image
285             app.Image = uiimage(app.UIFigure);
286             app.Image.Position = [17 272 100 100];
287             app.Image.ImageSource = 'CTM_Logo_Final_4c_grun_ohne.png';
288 
289             % Create BatterySimulatorControlPanel
290             app.BatterySimulatorControlPanel = uipanel(app.UIFigure);
291             app.BatterySimulatorControlPanel.TitlePosition = 'centertop';
292             app.BatterySimulatorControlPanel.Title = 'Battery Simulator Control';
293             app.BatterySimulatorControlPanel.Position = [205 16 172 240];
294 
295             % Create OnButton
296             app.OnButton = uibutton(app.BatterySimulatorControlPanel, 'push');
297             app.OnButton.ButtonPushedFcn = createCallbackFcn(app, @OnButtonPushed, true);
298             app.OnButton.Position = [36 188 100 22];
299             app.OnButton.Text = 'On';
300 
```

```
301             % Create OffButton
302             app.OffButton = uibutton(app.BatterySimulatorControlPanel, 'push');
303             app.OffButton.ButtonPushedFcn = createCallbackFcn(app, @OffButtonPushed, true
                    );
304             app.OffButton.Position = [36 151 100 22];
305             app.OffButton.Text = 'Off';
306
307             % Create ResetButton
308             app.ResetButton = uibutton(app.BatterySimulatorControlPanel, 'push');
309             app.ResetButton.ButtonPushedFcn = createCallbackFcn(app, @ResetButtonPushed,
                    true);
310             app.ResetButton.Position = [36 114 100 22];
311             app.ResetButton.Text = 'Reset';
312
313             % Create StatusBSLamp
314             app.StatusBSLamp = uilamp(app.BatterySimulatorControlPanel);
315             app.StatusBSLamp.Position = [117 43 20 20];
316
317             % Create StatusEditFieldLabel
318             app.StatusEditFieldLabel = uilabel(app.BatterySimulatorControlPanel);
319             app.StatusEditFieldLabel.HorizontalAlignment = 'right';
320             app.StatusEditFieldLabel.Position = [36 42 40 22];
321             app.StatusEditFieldLabel.Text = 'Status';
322
323             % Create StatusEditField
324             app.StatusEditField = uieditfield(app.BatterySimulatorControlPanel, 'numeric'
                    );
325             app.StatusEditField.Position = [87 42 17 22];
326
327             % Create GetStatusButton_2
328             app.GetStatusButton_2 = uibutton(app.BatterySimulatorControlPanel, 'push');
329             app.GetStatusButton_2.ButtonPushedFcn = createCallbackFcn(app,
                    @GetStatusButton_2Pushed, true);
330             app.GetStatusButton_2.Position = [36 79 100 22];
331             app.GetStatusButton_2.Text = 'Get Status';
332
333             % Create ValidationPanel
334             app.ValidationPanel = uipanel(app.UIFigure);
335             app.ValidationPanel.TitlePosition = 'centertop';
336             app.ValidationPanel.Title = 'Validation';
337             app.ValidationPanel.Position = [392 16 170 240];
338
339             % Create ImportDataButton
340             app.ImportDataButton = uibutton(app.ValidationPanel, 'push');
341             app.ImportDataButton.ButtonPushedFcn = createCallbackFcn(app,
                    @ImportDataButtonPushed, true);
342             app.ImportDataButton.Position = [36 188 100 22];
343             app.ImportDataButton.Text = 'Import Data';
344
345             % Create InitializeSimButton
346             app.InitializeSimButton = uibutton(app.ValidationPanel, 'push');
```

```
347             app.InitializeSimButton.ButtonPushedFcn = createCallbackFcn(app,
                    @InitializeSimButtonPushed, true);
348             app.InitializeSimButton.Position = [36 151 100 22];
349             app.InitializeSimButton.Text = 'Initialize Sim';
350
351             % Create StopSimButton
352             app.StopSimButton = uibutton(app.ValidationPanel, 'push');
353             app.StopSimButton.ButtonPushedFcn = createCallbackFcn(app,
                    @StopSimButtonPushed, true);
354             app.StopSimButton.Position = [36 79 100 22];
355             app.StopSimButton.Text = 'Stop Sim';
356
357             % Create StartSimButton
358             app.StartSimButton = uibutton(app.ValidationPanel, 'push');
359             app.StartSimButton.ButtonPushedFcn = createCallbackFcn(app,
                    @StartSimButtonPushed, true);
360             app.StartSimButton.Position = [36 114 100 22];
361             app.StartSimButton.Text = 'Start Sim';
362
363             % Create AccumulatorvalidationtoolV10Label
364             app.AccumulatorvalidationtoolV10Label = uilabel(app.UIFigure);
365             app.AccumulatorvalidationtoolV10Label.FontSize = 20;
366             app.AccumulatorvalidationtoolV10Label.Position = [150 309 283 26];
367             app.AccumulatorvalidationtoolV10Label.Text = 'Accumulatorvalidationtool V1.0'
                    ;
368
369             % Create AccumulatorControlPanel
370             app.AccumulatorControlPanel = uipanel(app.UIFigure);
371             app.AccumulatorControlPanel.TitlePosition = 'centertop';
372             app.AccumulatorControlPanel.Title = 'Accumulator Control';
373             app.AccumulatorControlPanel.Position = [17 16 171 241];
374
375             % Create InitializeButton
376             app.InitializeButton = uibutton(app.AccumulatorControlPanel, 'push');
377             app.InitializeButton.ButtonPushedFcn = createCallbackFcn(app,
                    @InitializeButtonPushed, true);
378             app.InitializeButton.Position = [36 188 100 22];
379             app.InitializeButton.Text = 'Initialize';
380
381             % Create OnButton_2
382             app.OnButton_2 = uibutton(app.AccumulatorControlPanel, 'push');
383             app.OnButton_2.ButtonPushedFcn = createCallbackFcn(app, @OnButton_2Pushed,
                    true);
384             app.OnButton_2.Position = [36 151 100 22];
385             app.OnButton_2.Text = 'On';
386
387             % Create OffButton_2
388             app.OffButton_2 = uibutton(app.AccumulatorControlPanel, 'push');
389             app.OffButton_2.ButtonPushedFcn = createCallbackFcn(app, @OffButton_2Pushed,
                    true);
390             app.OffButton_2.Position = [36 114 100 22];
391             app.OffButton_2.Text = 'Off';
```

```
392
393             % Create AMSOKLamp
394             app.AMSOKLamp = uilamp(app.AccumulatorControlPanel);
395             app.AMSOKLamp.Position = [130 43 20 20];
396
397             % Create AMSOKEditFieldLabel
398             app.AMSOKEditFieldLabel = uilabel(app.AccumulatorControlPanel);
399             app.AMSOKEditFieldLabel.HorizontalAlignment = 'right';
400             app.AMSOKEditFieldLabel.Position = [37 42 52 22];
401             app.AMSOKEditFieldLabel.Text = 'AMS OK';
402
403             % Create AMSOKEditField
404             app.AMSOKEditField = uieditfield(app.AccumulatorControlPanel, 'numeric');
405             app.AMSOKEditField.Position = [100 42 17 22];
406
407             % Create SDCClosedLamp
408             app.SDCClosedLamp = uilamp(app.AccumulatorControlPanel);
409             app.SDCClosedLamp.Position = [130 14 20 20];
410
411             % Create SDCClosedEditFieldLabel
412             app.SDCClosedEditFieldLabel = uilabel(app.AccumulatorControlPanel);
413             app.SDCClosedEditFieldLabel.HorizontalAlignment = 'right';
414             app.SDCClosedEditFieldLabel.Position = [17 13 72 22];
415             app.SDCClosedEditFieldLabel.Text = 'SDC Closed';
416
417             % Create SDCClosedEditField
418             app.SDCClosedEditField = uieditfield(app.AccumulatorControlPanel, 'numeric');
419             app.SDCClosedEditField.Position = [100 13 17 22];
420
421             % Create GetStatusButton
422             app.GetStatusButton = uibutton(app.AccumulatorControlPanel, 'push');
423             app.GetStatusButton.ButtonPushedFcn = createCallbackFcn(app,
                    @GetStatusButtonPushed, true);
424             app.GetStatusButton.Position = [37 79 100 22];
425             app.GetStatusButton.Text = 'Get Status';
426
427             % Show the figure after all components are created
428             app.UIFigure.Visible = 'on';
429         end
430     end
431
432     % App creation and deletion
433     methods (Access = public)
434
435         % Construct app
436         function app = controlpanel
437
438             % Create UIFigure and components
439             createComponents(app)
440
441             % Register the app with App Designer
442             registerApp(app, app.UIFigure)
```

```
443
444             % Execute the startup function
445             runStartupFcn(app, @startupFcn)
446
447             if nargout == 0
448                 clear app
449             end
450         end
451
452         % Code that executes before app deletion
453         function delete(app)
454
455             % Delete UIFigure when app is deleted
456             delete(app.UIFigure)
457         end
458     end
459 end
```