Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelorarbeit

**Fabian Brandes**

## Implementierung einer Auswertungs-Software für einen Redox-Flow-Batterie-Prüfstand

*Fakultät Technik und Informatik*
*Department Maschinenbau und Produktion*

*Faculty of Engineering and Computer Science*
*Department of Mechanical Engineering and Production*

Fabian Brandes

# Implementierung einer Auswertungs-Software für einen Redox-Flow-Batterie-Prüfstand

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Maschinenbau / Entwicklung und Konstruktion
am Department Maschinenbau und Produktion
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstpr"ufer: Prof. Dr. Thorsten Struckmann
Zweitprüfer: Simon Ressel

Eingereicht am: 09.11.2017

**Fabian Brandes**

**Thema der Arbeit**
Implementierung einer Auswertungs-Software für einen Redox-Flow-Batterie-Prüfstand

**Stichworte**
Redox-Flow-Batterie, Auswerte-Software, Polarisationskurve

**Kurzzusammenfassung**
In dieser Arbeit wird die Erstellung einer Auswerte-Software für einen Redox-Flow-Batterie-Prüfstand gezeigt und deren Anwendung demonstriert. Die Umsetzung erfolgt in Matlab und die Beispielauswertung thematisiert Polarisationskurven einer Vanadium-Luft RFB. Die Software erfüllt die Anforderungen und die ausgewerteten Daten lassen sich mit Literaturwerten vergleichen. Die Qualität und das Tempo zukünftiger RFB-Auswertungen an der HAW konnte signifikant gesteigert werden.

**Fabian Brandes**

**Title of the paper**
Implementation of an analysis software for a redox flox battery test rig

**Keywords**
redox flow battery, analysis software, polarization curve

**Abstract**
This thesis shows the implementation of an analysis software for a redox flow battery test rig and demonstrates its use. The implementation is carried out in Matlab and an example analysis shows polarization curves for a vanadium/air redox flow battery. The software meets the requirements and the analyzed data is comparable to literature. Quality and rate of future RFB analysis at the HAW was improved significantly.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

| **Symbole** | **Einheit** | **Beschreibung** |
|---|---|---|
| $A$ | $[cm^2]$ | Fläche |
| $c_i$ | $[mol\ l^{-1}]$ | molare Konzentration von $i$ |
| $E$ | $[V]$ | Gleichgewichtspotential |
| $E^0$ | $[V]$ | Standardpotential |
| $E^0_{PHZ}$ | $[V]$ | positives Halbzellenstandardpotential |
| $E^0_{NHZ}$ | $[V]$ | negatives Halbzellenstandardpotential |
| $E'^0$ | $[V]$ | Formalpotential |
| $E^+$ | $[V]$ | positives Halbzellenpotential |
| $E^-$ | $[V]$ | negatives Halbzellenpotential |
| $E_{cell}$ | $[V]$ | Zellspannung |
| $U_{loss}$ | $[V]$ | Verlustspannung |
| $F$ | $[C\ mol^{-1}]$ | Faraday-Konstante |
| $I$ | $[A]$ | Stromstärke |
| $i$ | $[A\ cm^-2]$ | Stromdichte |
| $n$ | [-] | Anzahl ausgetauschter Elektronen |
| $p$ | $[bar,Pa]$ | Druck |
| $Q$ | $[C]$ | Ladung |
| $R$ | $[J\ K^{-1}\ mol^{-1}]$ | universelle Gaskonstante |
| $t$ | $[s]$ | Zeit |
| $T$ | $[K]$ | Temperatur |
| $V$ | $[m^3]$ | Volumen |
| $\eta_{act}$ | $[V]$ | Aktivierungsüberspannung |
| $\eta_{conc}$ | $[V]$ | Konzentrationsüberspannung |
| $\eta_{ohm}$ | $[V]$ | ohm'sche Überspannung |
| $\eta_{res}$ | $[V]$ | Restüberspannungen (residual overpotentials) |
| $\eta_{pos}$ | $[V]$ | Überspannung der positiven Halbzelle |

| | | |
|---|---|---|
| $\eta_{neg}$ | $[V]$ | Überspannung der negativen Halbzelle |

### b) Abkürzungen

| **Abkürzung** | **Bedeutung** |
|---|---|
| ASR | flächenspezifischer Widerstand (areal specific resistance) |
| ch | laden (charge) |
| CI | Stromunterbrechung (current interrupt) |
| dch | entladen (discharge) |
| EIS | elektrochemische Impedanzspektroskopie |
| GfE | Gesellschaft für Elektrometallurgie |
| NHE/SHE | Normalwasserstoffelektrode |
| NHZ | negative Halbzelle |
| OCV | Leerlaufspannung (open circuit voltage) |
| OER | Sauerstoffentwicklungsreaktion (oxidation evolution reaction) |
| ORR | Sauerstoffreduktionsreaktion (oxidation reduction reaction) |
| PHZ | positive Halbzelle |
| pot | Potentiostat |
| RFB | Redox Flow Batterie |
| SoC | Ladezustand (State Of Charge) |
| V | Vanadium |
| VARFB/VLRFB | Vanadium-Luft-Redox-Flow-Batterie |
| VI | Virtual instruments |
| VRFB | Vanadium Redox Flow Batterie |

# Kapitel 1

# Einführung

## 1.1 Kontext

Durch ein Zusammenspiel aus der steigenden Verknappung der fossilen Energieträgervorräte und dem globalen Bevölkerungswachstum sehen sich weltweit viele Regierungen nach erneuerbaren Energien um. Gerade Deutschland hatte in dieser Sparte jahrelang eine Vorreiterrolle inne, doch holen andere europäische Länder oder auch China massiv auf. Bis die großen Volkswirtschaften ihren Strombedarf komplett aus erneuerbaren Energien decken können, werden wohl noch viele Jahre ins Land gehen. Das Engagement für erneuerbare Energien und auch der Rückhalt in der Bevölkerung steigt jedoch spürbar an.

Das Wachstum der erneuerbaren Energien bringt aber auch neue Herausforderungen mit sich. Photovoltaik- und Windkraft-Anlagen sind im hohen Maße durch zeitliche und klimatische Verfügbarkeit gekennzeichnet. Da eine Anpassung des Stromverbrauchs an diese schwankende Produktion völlig unpraktikabel ist, müssen adäquate Energiespeichermaßnahmen geschaffen werden. Für verschiedene Stromerzeugungsarten existieren verschieden gut geeignete Speichermaßnahmen. Für den Anwendungsfall der Photovoltaik und der Windkraft haben elektrochemische Lösungen großes Potential. [7]

### 1.1.1 Redox-Flow-Batterien

Einen solchen elektrochemischen Energiespeicher stellt die Redox-Flow-Batterie dar. Sie besteht aus zwei Elektrolytkreisläufen und einer elektrochemischen Zelle. Die beiden Elektrolyte, die in Tanks gelagert und über Pumpen bei Bedarf in die Zelle gefördert werden, speichern die Energie, während in der elektrochemischen Zelle die Energieumwandlung stattfindet. Die Zelle wiederum ist durch eine Membran in zwei Halbzellen aufgeteilt: Jede dieser Halbzellen verfügt über eine Elektrode, die den Austausch von Elektronen mit dem Elektrolyt erlaubt. [8]

Wie der Name vermuten lässt, besteht die Energieumwandlung aus elektrochemischer Reduktion und Oxidation. Die Halb-Reaktion an der Elektrode auf Reduktionsseite entnimmt Elektrolyt A Elektronen und Ionen. Die andere Halb-Reaktion an der Elektrode auf Oxidationsseite führt ebendiese Elektronen und Ionen Elektrolyt B zu. Die Ionen wandern dabei durch die Ionenaustauschmembran. Da diese Membran für Elektronen undurchdringbar ist, nehmen diese den Weg über den Leiter, der die beiden Elektroden miteinander verbindet.

Ein großer Vorteil der Technologie springt direkt ins Auge: Die Trennung von Energiespeicherung und -umwandlung. Daraus ergibt sich eine theoretisch lediglich durch die Tankgröße limitierte Kapazität. Bei Verwendung der korrekten Elektrolytpaarung ist der Prozess darüber hinaus komplett reversibel. Weitere Vorteile der Batterie beinhalten geringe Wartungskosten und die Möglichkeit auch große Entladungen zu tätigen. [7]

Die noch relativ geringe Energiedichte macht die Batterie unpassend für mobile Anwendungen. Außerdem kann die Batterie recht große Dimensionen annehmen. Am wichtigsten ist jedoch die Notwendigkeit einer ständigen Temperaturüberwachung auf annähernd Raumtemperatur um die Funktionalität der Elektrolyte zu gewährleisten. [7]

### 1.1.2 Redox-Flow-Batterie-Projekt an der HAW

Die Hochschule für Angewandte Wissenschaften Hamburg beschäftigt sich mit Redox-Flow-Batterien im Rahmen des Projektes tubulair±. In diesem Projekt soll eine konkurrenzfähige Batterie entwickelt werden, die in den Punkten Energiedichte, Leistungsdichte und auch Herstellungskosten bestimmte Anforderungen erfüllt. Das Besondere an dem Projekt ist, dass bei der Zellenarchitektur nicht auf die übliche planare Zelle gebaut wird, sondern versucht wird, eine tubuläre Lösung zu finden. Ferner soll die Batterie nicht mit einer Vanadium-Elektrolyt-Paarung operieren, sondern eine Halbzelle mit Luft, oder genauer gesagt der darin enthaltenen Luftfeuchte, betrieben werden. Durch die dadurch eingesparte Elektrolytmenge erhofft man sich eine verbesserte Kosteneffizienz. [9]

Die Hochschule für Angewandte Wissenschaften Hamburg betreibt im Rahmen dieses Projektes zwei Prüfstände: Einen für All-Vanadium-Betrieb und einen spezifisch für das Projekt Tubulair. Mit beiden Prüfständen lassen sich automatisiert verschiedene Messzyklen fahren, wie beispielsweise Lade-/Entladezyklen. Während des Messzyklus werden dabei verschiedene Parameter aufgezeichnet, die von Temperaturen und Drücken bis Stromstärken und Spannungen ein weites Feld abdecken. Aufgenommen werden die Messdaten dabei von zwei verschiedenen Instanzen, zum Einen dem sogenannten VI - einem Computer, der eine Software betreibt auf dem die namensgebenden *virtual instruments* laufen - und zum Anderem einem Potentiostaten.

## 1.2 Zielstellung der Arbeit

### 1.2.1 Implementierung und Konsolidierung einer Auswertungssoftware

In diesem Zusammenhang soll eine Auswertungssoftware in Matlab entwickelt werden, welche eine möglichst einfache Aufbereitung der Messdaten ermöglicht.

Die Software soll in der Lage sein, die Messdateien beider Eingangsgeräte einlesen zu können. Der Einlesevorgang läuft dabei möglichst automatisiert ab. Eine Umbenennung der Größen auf einen einheitlichen Standard erfolgt über eine externe Steuerdatei. Der wichtigste Teil der Software soll die Messdaten beider Quellen auf einen gemeinsamen Timer bringen um so eine Vergleichbarkeit zu schaffen. Die so zusammengeführten Daten liegen in einer Matrix vor, die einen zugänglichen Überblick und eine einfache Weiterverarbeitung der aufbereiteten Dateien ermöglicht. Am Ende soll es möglich sein, die Daten der Matrix zu schneiden, um bei der Analyse kleinerer Teile der Messroutine nicht unnötig große Datenpakete verwalten zu müssen. Die Implementierung von spezifischen Auswerteprogrammen für die unterschiedlichen Bestandteile einer Messreihe wird hingegen nur ermöglicht, aber im Rahmen dieser Arbeit nicht für alle Messmethoden umgesetzt.

### 1.2.2 Dokumentation der Software

Neben der reinen Implementierung der Software soll ebenfalls eine Dokumentation der derselbigen entstehen. Diese soll zum Einen dem Anwender die Bedienung erleichtern und zum Anderen die Unterfunktionen im Detail erläutern, um eine mögliche zukünftige Umprogrammierung oder Erweiterung zu erleichtern. Neben ausführlichen Beschreibungen der Funktionen soll diese Arbeit ebenfalls leicht nachzuschlagende Instruktion für den Bediener bieten. Neben der Beschreibung der einzelnen Funktionen der Software soll auch das Zusammenspiel selbiger behandelt werden und anhand von Schaubildern, Beispielen und eines exemplarischen Programmdurchlaufes verdeutlicht werden.

### 1.2.3 Auswertung von Vanadium-Luft-Daten

Um zu demonstrieren, wofür die Software den Grundstein legt, folgt am Ende dieser Arbeit die Auswertung zweier Messläufe einer Vanadium-Luft-Batterie. Nach einer Einführung in die naturwissenschaftlichen Grundlagen der Thematik sollen Polarisationskurven diskutiert werden. Die Ergebnisse sollen mit der Literatur verglichen werden. Außerdem erfolgt eine Untersuchung des Einflusses von verschiedenen Stickstoff-Anteilen auf die positive Halbzelle.

# Kapitel 2

# Grundlagen

In diesem Kapitel wird die Funktionsweise der Vanadium-Luft-Redox-Flow-Batterie (kurz: VLRFB oder VARFB) erläutert. Dazu werden im ersten Teil des Kapitels die grundlegenden elektrochemischen Zusammenhänge erklärt, während im zweiten Teil des Kapitels der Prüfstand und damit das Messverfahren an der HAW näher beleuchtet wird.

## 2.1 Elektrochemische Grundlagen

### 2.1.1 Redoxreaktionen in der VLRFB

Wie bereits in der Einleitung kurz umrissen, handelt es sich bei einer RFB um einen elektrochemischen Energiespeicher, der durch eine Membran getrennt aus zwei Halbzellen besteht. Man unterscheidet dabei die *Negative* und die *Positive Halbzelle*. In jeder Halbzelle befindet sich eine Elektrode, an deren Oberfläche die namensgebenden Redoxreaktionen ablaufen. Die Elektrode selbst nimmt jedoch nicht an der Reaktion teil. Das Prinzip einer VLRFB ist in Abbildung 2.1 dargestellt.

Eine Redoxreaktion ist eine Kombination aus einer Reduktion und einer Oxidation. Bei einer Oxidation gibt ein Reduktionsmittel $A_{Red}$ Elektronen ab und erhöht seine Oxidationsstufe, siehe Formel 2.1.1. Parallel nimmt bei einer allgemeinen Reduktion ein Oxidationsmittel $B_{Ox}$ Elektronen auf und verringert somit seine Oxidationsstufe, wie in Formel 2.1.2 gezeigt. Die Zahl der pro elektrochemisch aktivem Teilchen ausgetauschten Elektronen wird über die Ladungszahl $z$ angegeben. [10, S.16f] [11, S.185f]

$$\mathrm{A_{Red}} \rightleftarrows \mathrm{A_{Ox}} + z \cdot \mathrm{e^-} \tag{2.1.1}$$

$$\mathrm{B_{Ox}} + z \cdot \mathrm{e^-} \rightleftarrows \mathrm{B_{Red}} \tag{2.1.2}$$

Verbindet man nun beide Teilreaktionen, erhält man unter Berücksichtigung des direkten Elektronenaustausches die sogenannte Redoxreaktion. Zur besseren Übersicht über die Reaktionen wurden die beiden Zähler $n$ und $m$ in

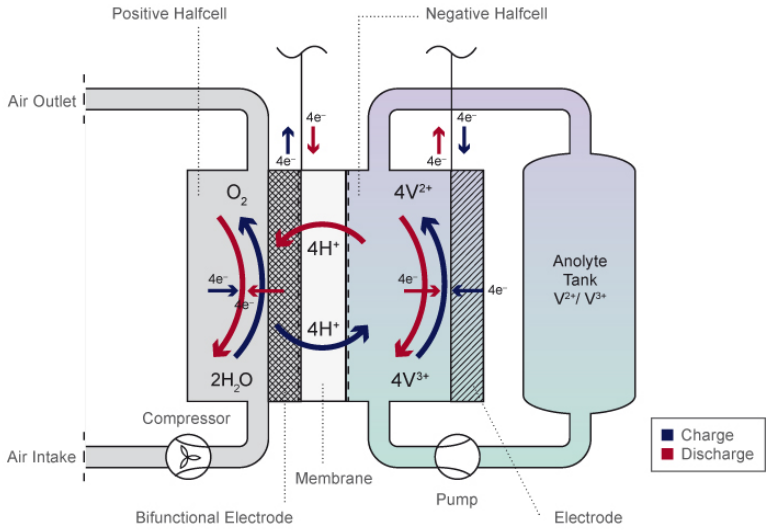


Abbildung 2.1: Schaubild einer VLRFB [1]

die Formel 2.1.3 eingearbeitet.

$$n \cdot \mathrm{A_{Red}} + m \cdot \mathrm{B_{Ox}} \underset{\mathrm{z \cdot e^-}}{\overset{\mathrm{z \cdot e^-}}{\rightleftharpoons}} n \cdot \mathrm{A_{Ox}} + m \cdot \mathrm{B_{Red}} \tag{2.1.3}$$

Betrachten wir nun eingehender die VLRFB. Beide Halbzellen sind bei einem Ladevorgang elektrisch über eine Stromquelle und bei einem Entladevorgang über eine Last miteinander verbunden. In der negativen Halbzelle wird Vanadium, das über eine Pumpe zur Elektrode gefördert wird, nach Formel 2.1.4 umgesetzt. Bei einem Ladevorgang nimmt im Elektrolyt gelöstes Vanadium der Oxidationsstufe $V^{3+}$ Elektronen auf und wird zu $V^{2+}$ reduziert. Bei einem Entladevorgang läuft die Reaktion entsprechend umgekehrt als Oxidation von $V^{2+}$ zu $V^{3+}$ ab. [12]

$$\mathrm{4\,V^{3+} + 4\,e^-} \underset{\text{Entladen}}{\overset{\text{Laden}}{\rightleftharpoons}} \mathrm{4\,V^{2+}} \tag{2.1.4}$$

Die positive Halbzelle ist bei der VLRFB die *Luft-Seite*. In dieser Halbzelle findet bei einem Ladevorgang die Sauerstoffentwicklungsreaktion (eng: oxidation evelution reaction - kurz: OER) statt: An der Elektrodenoberfläche der Halbzelle wird feuchte Luft, die über einen Kompressor zugeführt wird, in seine Bestandteile Wasserstoff und Sauerstoff zerlegt. Der resultierende Wasserstoff ist einfach positiv geladen und diffundiert durch die Ionenaustauschmembran in die negative Halbzelle, also in den flüssigen Vanadium-Elektrolyten. Der Sauerstoff wird wieder an die Umgebungsluft übergeben.

Entsprechend findet bei einem Entladevorgang die Umkehrreaktion, also die Sauerstoffreduktionsreaktion (eng: oxidation reduction reaction - kurz: ORR) statt. Dessen Endprodukt ist Wasser. Damit diese Reaktion ablaufen kann, müssen Sauerstoff und Wasserstoff als Edukte vorhanden sein: der Sauerstoff wird aus der Umgebungsluft entnommen, der Wasserstoff wieder in Form von Ionen aus dem flüssigen Vanadium-Elektrolyten. Die entsprechende Reaktionsgleichung ist in Formel 2.1.5 dargestellt. [12]

$$2\,H_2O \underset{\text{Entladen}}{\overset{\text{Laden}}{\rightleftharpoons}} 4\,H^+ + O_2 + 4\,e^- \tag{2.1.5}$$

Die kombinierte Reaktionsgleichung lautet also wie in Formel 2.1.6 dargestellt.

$$4\,V^{3+} + 2\,H_2O \underset{\text{Entladen}}{\overset{\text{Laden}}{\rightleftharpoons}} 4\,V^{2+} + O_2 + 4\,H^+ \tag{2.1.6}$$

### 2.1.2 Potentiale und Nernst-Gleichung

Eine der wichtigsten Größen, die eine RFB im Betrieb ausmachen, ist das Potential der Zelle $E_{Zelle}$. Theoretisch ergibt sich dieses Potential aus der Differenz der absoluten Einzelpotentiale der Elektroden in der positiven und negativen Halbzelle. [10]

$$E_{Zelle} = \phi_{PHZ} - \phi_{NHZ} \tag{2.1.7}$$

Absolute Elektrodenpotentiale sind jedoch nicht messbar. Daher greift man auf Potentialdifferenzen zu einer definierten Referenzelektrode zurück. Eingebürgert hat sich der Abgleich mit einer Normalwasserstoffelektrode (bezeichnet als NHE oder auch SHE). Das Potential der NHE wird als $0V$ definiert. [11] Gegenüber der NHE weisen Referenzelektroden ein Standardredoxpotential $E^0$ auf, dieses wird in $V\ vs\ NHE$ angegeben. Standardredoxpotentiale vieler Elektroden sind in der elektrochemischen Spannungsreihe verzeichnet. Dieses Potential gilt allerdings nur bei folgenden definierten Standardbedingungen: Einer Temperatur von $\theta = 298,15K$, einem Umgebungsdruck von $p = 1bar$ und einer Konzentration beider Redoxpaare von $c_{Ox} = c_{Red} = 1M/l$. Das Standardredoxpotential der negativen Halbzelle ist in Formel 2.1.8 und das der positiven Halbzelle in Formel 2.1.9 gezeigt. [12]

$$E^0_{NHZ} = -0,26V\ vs\ NHE \tag{2.1.8}$$

$$E^0_{PHZ} = 1,23V\ vs\ NHE \tag{2.1.9}$$

Für das Standardredoxpotential der Vanadium-Luft-Zelle gilt also Gleichung 2.1.10.

$$E^0_{Zelle} = E^0_{PHZ} - E^0_{NHZ} = 1,49V\ vs\ NHE \tag{2.1.10}$$

### 2.1.3 Das Nernst-Potential

Weichen die Betriebsparameter von den oben aufgeführten Standardbedingungen ab, so lässt sich über den Nernst-Term das Standardredoxpotential korrigieren, siehe Formel 2.1.11, oder auch Formel 2.1.12. [13]

$$E = E^+ - E^- = E^0 + \frac{RT}{nF} ln \left[ \frac{c_{Ox}}{c_{Red}} \cdot \frac{\gamma_{Ox}}{\gamma_{Red}} \right] \tag{2.1.11}$$

$$E = E^0 + \frac{RT}{nF} ln \left[ \frac{c_{Ox}}{c_{Red}} \right] + \frac{RT}{nF} ln \left[ \frac{\gamma_{Ox}}{\gamma_{Red}} \right] \tag{2.1.12}$$

Das Potential bei abweichenden Bedingungen wird über die allgemeine Gaskonstante $R$, die absolute Temperatur $T$, die Anzahl der Elektronen $n$, die Faraday-Konstante $F$ sowie die Konzentrationen $c$ und die Aktivitäten $\gamma$ der beteiligten Stoffe. Um die weiteren Schritte zu vereinfachen wird ein Formalpotential $E'^0$ eingeführt, siehe Formel 2.1.13.

$$E'^0 = E^0 + \frac{RT}{nF} ln \left[ \frac{\gamma_{Ox}}{\gamma_{Red}} \right] \tag{2.1.13}$$

Da die Aktivitäten der Stoffe nicht experimentell bestimmbar sind, werden sie oft als 1 angenommen. In Gleichung 2.1.13 würde also nur das Standardredoxpotential $E^0$ übrig bleiben. Ungeachtet dessen, ob die Aktivitäten als 1 angenommen werden oder nicht, vereinfacht sich die Bestimmung des Potentials der Zelle unter Berücksichtigung des Formalpotentials zu Gleichung 2.1.14. [14]

$$E = E^+ - E^- = E'^0 + \frac{RT}{nF} ln \left[ \frac{c_{Ox}}{c_{Red}} \right] \tag{2.1.14}$$

Solange der Stromkreis offen ist, stellt sich bei den gegebenen Bedingungen die sogenannte *Open Circuit Voltage* OCV oder zu deutsch auch Leerlaufspannung oder offene Zellspannung ein.

### 2.1.4 Verluste und Überspannungen

Die bisherigen Ausführungen liefen alle unter der Annahme eines offenen Stromkreises. Wird der Stromkreis nun geschlossen, also ein Verbraucher oder eine Spannungsquelle angeschlossen, treten verschiedene interne Verluste in der RFB auf. Die Verluste werden oft auch als Überspannungen $\eta$ bezeichnet. Überspannungen lassen die Zellspannung $E_{cell}$ von der OCV abweichen und können in verschiedene Kategorien eingeteilt werden: ohmsche Verluste $\eta_{ohm}$, Aktivierungsüberspannungen $\eta_{act}$ und Konzentrationsüberspannungen $\eta_{con}$. [15]

Die Überspannungen steigen mit erhöhter Stromdichte $i$. Diese errechnet sich aus der Stromstärke $I$ bezogen auf die projizierte Membranfläche $A$,

siehe Formel 2.1.15. Üblich ist eine Angabe in $\left[\frac{mA}{cm^2}\right]$.

$$i = \frac{I}{A}\left[\frac{mA}{cm^2}\right] \tag{2.1.15}$$

Aktivierungsüberspannungen treten bei einer Behinderung des Ladungsaustausches an der Phasengrenze von Elektrolyt zu Elektrode auf. Ohmsche Überspannungen resultieren aus den ohmschen Widerständen aller beteiligten Strom leitenden Komponenten der Zelle. Konzentrationsüberspannungen treten auf, wenn die Reaktion durch einen Mangel an Reaktionspartnern begrenzt wird. [8, 16, 2]

$$E_{cell} = OCV - U_{loss} \tag{2.1.16}$$

$$U_{loss} = \eta_{ohm} + \eta_{act} + \eta_{conc} \tag{2.1.17}$$

$$U_{loss} = \eta_{ohm} + \eta_{res} \tag{2.1.18}$$

$$U_{loss} = \eta_{ohm} + \eta_{pos} + \eta_{neg} \tag{2.1.19}$$

Die resultierende Zellspannung setzt sich aus der OCV modifiziert mit der Summe aller Überspannungen zusammen. Aktivierungs- und Konzentrationsüberspannungen werden oft auch zu $\eta_{res}$ zusammengefasst. Da diese beiden Überspannungen an den Elektroden der jeweiligen Halbzellen auftreten, wird $\eta_{res}$ oft auf die beiden Halbzellen aufgeteilt.

Für die weiteren Ausführungen wird allgemein von folgender Vorzeichenkonvention für die Stromdichte ausgegangen: eine negative Stromdichte impliziert einen Ladevorgang, eine positive einen Entladevorgang. Diese Konvention entstammt der Brennstoffzellentechnik, mit der sich Redox-Flow-Batterien viele Eigenschaften teilen. Für die gewünschte Zellreaktion müssen die Überspannungen also beim Beladen zusätzlich aufgebracht werden; wird die Batterie entladen verringern die Überspannungen die nutzbare Zellspannung.

### 2.1.5 Polarisationskurven

Um sich die Verluste möglichst anschaulich darstellen zu lassen, haben sich in der Brennstoffzellentechnik sogenannte Polarisationskurven durchgesetzt. Da eine RFB grob vereinfacht eine Brennstoffzelle mit Möglichkeit der Aufladung darstellt, lässt sich diese Messmethode auf die RFB übertragen. [2] Abbildung 2.2 zeigt eine beispielhafte Polarisationskurve. In ihr wird das Zellpotential über der Stromdichte aufgezeichnet. Je nach erreichter Stromdichte dominiert ein anderer Überspannungsanteil. In einem ersten exponentiellen Teil des Verlaufs dominieren die Aktivierungsüberspannungen, es folgt ein linearer Anteil mit dominierenden ohmschen Überspannungen abgeschlossen von einem exponentiellen Spannungsabfall, in dem Transportüberspannungen dominieren. Es lässt sich daher schon aus dem Verlauf einer Polarisationskurve eine Aussage über die Verlustquellen aufstellen. [2]

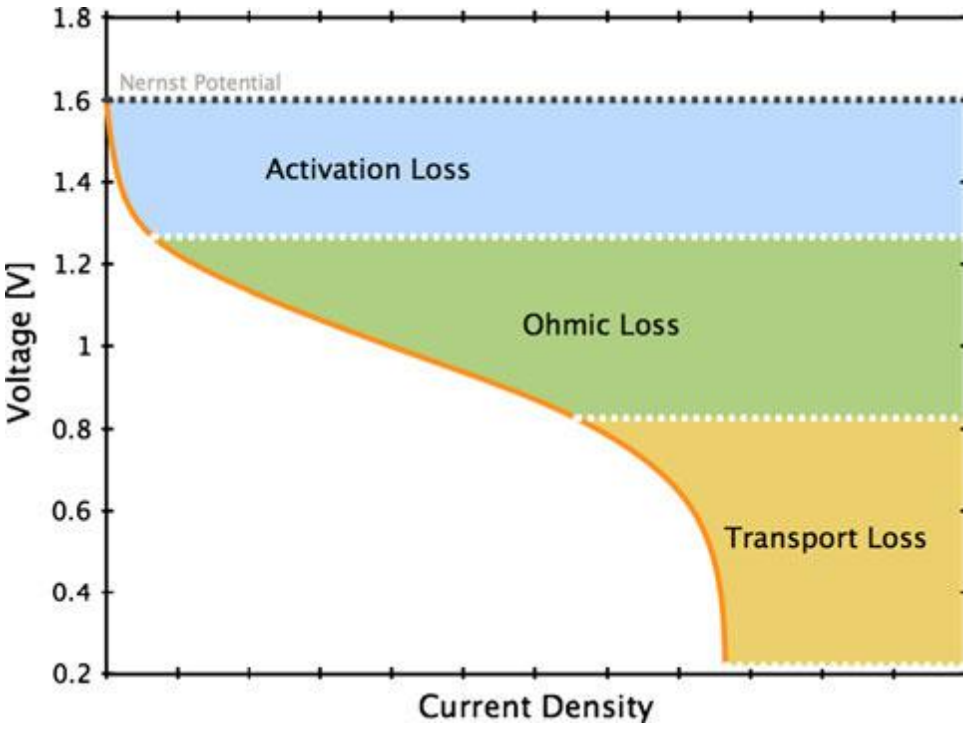


Abbildung 2.2: Dominante Überspannungsquellen in einer generalisierten Polarisationskurve [2]

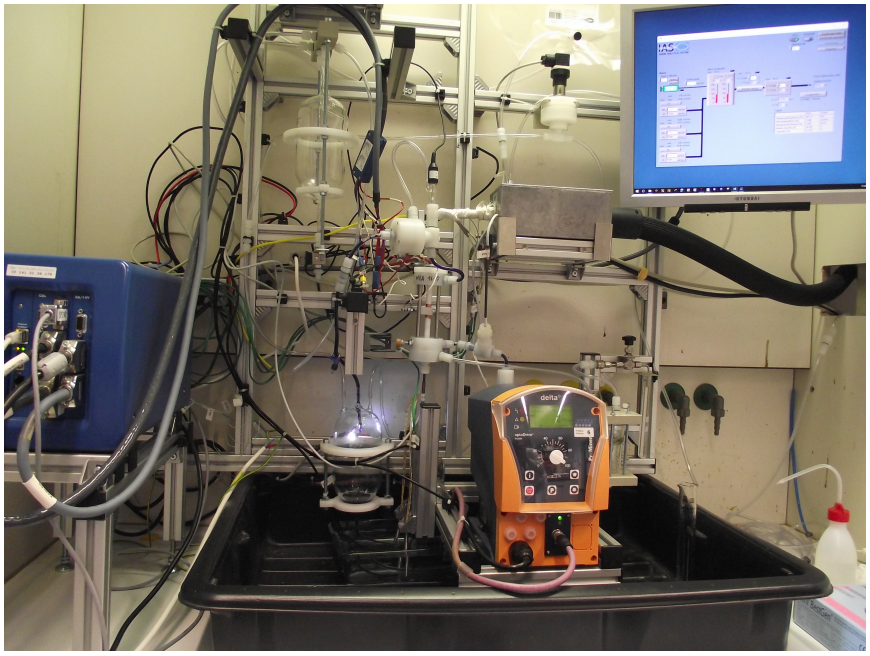

Abbildung 2.3: Der Vanadium-Luft-Prüfstand der HAW [3]

## 2.2 Prüfstand

Diese Sektion beschäftigt sich mit den praktischen Umständen der Messung. Es werden die verwendeten Geräte sowie deren Aufbau und Verknüpfungen erläutert. Als Abschluss dient eine kurze Beschreibung der verwendeten Zelle.

### 2.2.1 Aufbau des Prüfstandes

Abbildung 2.3 zeigt ein Foto des Prüfstandes mit dem die Vanadium-Luft-Messungen, die in dieser Arbeit hauptsächlich behandelt werden, aufgezeichnet wurden. Links im Bild lässt sich das Potentiostat (*BioLogic, SP-240 / 4A / 14V*) erkennen, rechts oben der Luftbefeuchter, der das Luft-Wasserdampf-Gemisch bereitstellt. Das orange Gerät im unteren Bildbereich ist eine Magnetmembran-Dosierpumpe (*Delta®, ProMinent®*, Deutschland), über die das Vanadium-Elektrolyt vom unteren Tank durch die Zelle in den höher gelegenen Tank gefördert werden kann. Der höher gelegene Tank besitzt eine Rückführung zum niedrig gelegenen Tank, die über ein Ventil geschlossen werden kann. Ist das Ventil geschlossen können auf diese Weise etappenweise Messungen mit einem Elektrolyt gleichbleibendem Ladezustands gefahren werden. Die tubuläre Zelle ist in der Mitte des Bildes zu erkennen.

In Abbildung 2.4 ist der Versuchsaufbau außerdem in einer abstrakten Übersicht dargestellt. Auf der linken Seite des Bildes ist der Vanadium-Kreislauf dargestellt, inklusive Tank, Leitungen, Pumpen und Messstellen.

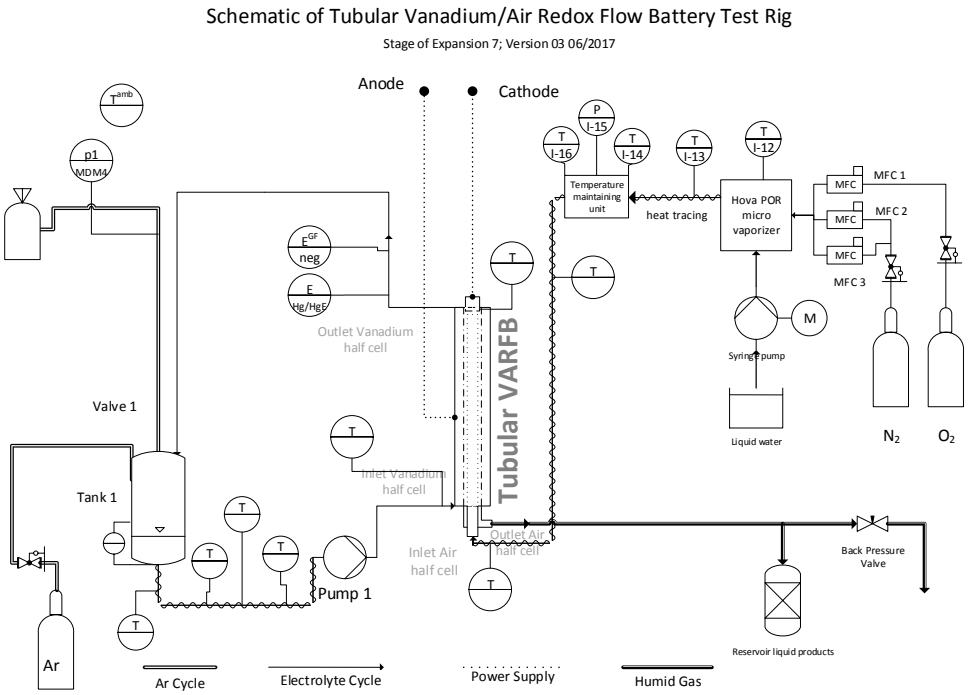


Abbildung 2.4: Schaltbild des Vanadium-Luft-Prüfstandes [3]

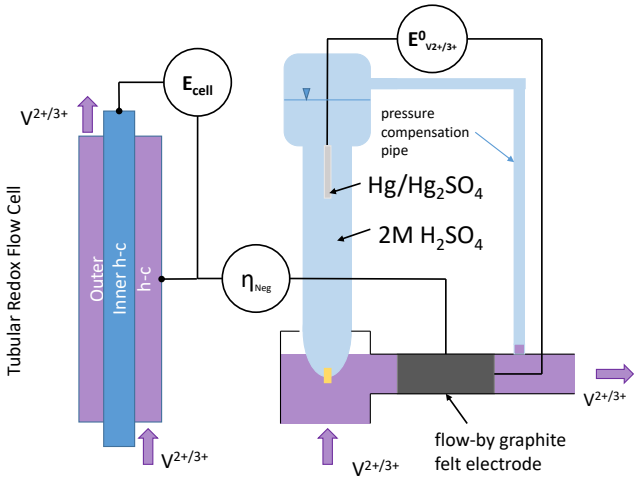


Abbildung 2.5: Prinzip der Potential- und Überspannungsaufnahme der negativen Halbzelle in einer VARFB [3]

Man beachte außerdem die angeschlossene Argon-Flasche, die für die Herstellung einer Schutzgas-Atmosphäre vor dem Messbetrieb benötigt wird.

Auf der rechten Seite der Abbildung ist die Luftseite des Versuchsaufbaus zu sehen. Zu sehen sind die Reservoirs für Stickstoff, Sauerstoff sowie Wasser, aus denen über den Luftbefeuchter die feuchte Luft gewonnen wird. Ebenfalls zu sehen ist der Behälter, in dem die entstehende Flüssigkeit aufgefangen wird. Auch auf dieser Seite sind wieder die verschiedenen Messstellen, Ventile und Leitungen abgebildet.

Die Zelle schließlich befindet sich im Zentrum der Abbildung. Der tubuläre Aufbau bestimmt die Aufteilung in eine innere und eine äußere Halbzelle. Beide Halbzellen sind mit einer Elektrode versehen.

Aus Abbildung 2.5 lässt sich erkennen, wie die unterschiedlichen Potentiale gemessen werden. Das Potential der Zelle $E_{cell}$ und die Überspannung der negativen Halbzelle werden über einen Abgleich der jeweiligen Zelle mit dem Graphitfilz aufgenommen. Mit diesen Werten kann ein Abgleich mit der Bezugselektrode $Hg/Hg_2SO_4$ erfolgen, die ein bekanntes Potential von $657mV$ gegenüber der SHE besitzt.

### 2.2.2 Potentiostat

Zentrales Element des Messaufbaus ist das Potentiostat. Dabei handelt es sich in erster Linie um eine sehr genaue Gleichspannungsquelle, die besonders bei elektrochemischen Anwendungsfällen Verwendung findet. Das Potentio-

stat arbeitet über drei Elektroden: die zu untersuchende Arbeitselektrode, eine Referenzelektrode und eine Gegenelektrode. Das Potential der Referenzelektrode ist dabei stets stromlos und über die elektrochemische Spannungsreihe bekannt; daher kann ein Abgleich mit der Arbeitselektrode erfolgen. Das gewünschte Potential der Arbeitselektrode wird über einen Stromfluss zwischen ihr und der Gegenelektrode eingestellt. Dadurch können hohe Präzisionsgrade und geringe Reaktionszeiten erreicht werden.

Im Projekt wird das Potentiostat dazu genutzt, die Zellspannung, wichtige Überspannungen und den ohmschen Widerstand auf einen internen Timer bezogen zu messen. Darüber hinaus werden auch die Größen aus den periodischen Messzyklen mit diesem Gerät aufgezeichnet. Die Größen werden normalerweise mit einer Abtastrate von $1Hz$ abgefragt. Im Gegensatz zum VI ist das Potentiostat in der Lage auch mit höheren Abtastraten als $1Hz$ zu arbeiten. Daher ermöglicht die Software es auch, verschiedene Messmethoden der Elektrochemie durchzuführen, z.B. eine *current interrupt*-Messung.

### 2.2.3 VI

Ergänzt wird das Potentiostat durch das sogenannte VI, den *virtual instruments*. Über einen Computer werden mit Hilfe der Software *LabView* verschiedene Daten aufgenommen, zum Beispiel über einen SPS-Controller, Multimeter und Waagen. Ebenfalls über die Software können die entsprechenden Daten in eine Textdatei exportiert werden. Diese Textdatei ermöglicht die Weiterverarbeitung der Messwerte in Matlab. Per VI werden relativ viele Messkanäle aufgezeichnet; in dem im nächsten Kapitel folgenden Beispiel einer Vanadium-Luft-Messung sind es zum Beispiel 57 verschiedene Kanäle. Vor allem Temperaturen, Drücke oder Massen werden mit dem VI aufgenommen. Für die elektrischen Größen ist das Potentiostat allerdings besser geeignet.

Als Ergänzung zum Potentiostat ist das VI sehr gut geeignet. Für eine Auswertung ergeben sich durch die Aufteilung der Messwerte auf zwei Geräte jedoch verschiedene Probleme. Da das Potentiostat im Gegensatz zum VI nicht durchgehend in Betrieb ist, sind die Messwerte aus dem Potentiostat auf mehrere Dateien aufgeteilt, die den entsprechenden VI-Daten zugeordnet werden müssen. Da jedoch auch mit jeder neuen Potentiostat-Datei ein neuer Timer gestartet wird, und das Potetiostat darüber hinaus auch in der Lage ist mit anderen Abtastraten zu arbeiten als das VI, ist ein Zusammenführen der Daten aus beiden Quellen über den originalen Zeitstempel nicht möglich.

### 2.2.4 Zellaufbau

Redox-Flow-Zellen bieten meist einen planaren Aufbau, wie in Abbildung 2.6 A dargestellt. Eine solche Zelle teilt sich in zwei Halbzellen auf, die durch einen Ionenaustauschmembran getrennt werden. Als Elektrode für

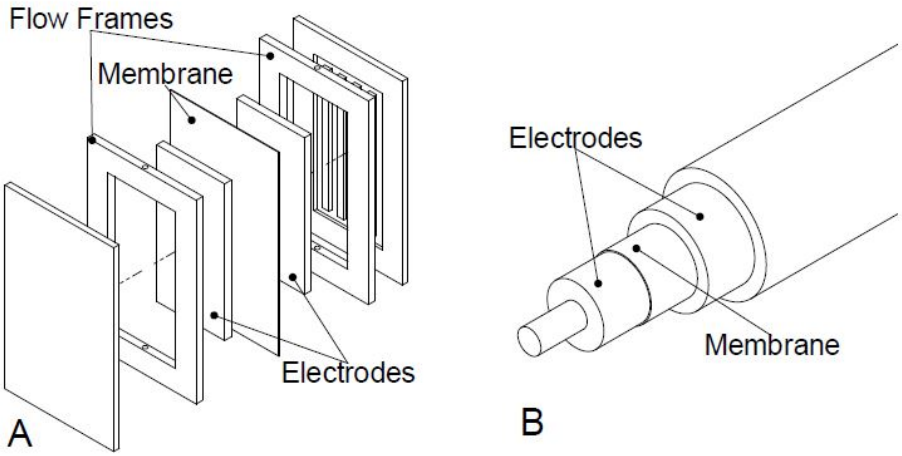


Abbildung 2.6: A: planare Zellbauweise. B: tubuläre Zellbauweise [1]

die Halbzelle dient eine Graphitfilzelektrode oder ein dünnes Kohlenstoffpapier. Eingeschlossen wird die Elektrode in den sogenannten *Flow Frame*, also einen Rahmen, der über Anschlüsse für die Elektrolytleitungen verfügt. Abgeschlossen wird die Zelle über Endplatten auf beiden Seiten. Die Endplatten pressen über Schrauben die Komponenten zusammen. Über chemisch beständiges Dichtungsmaterial zwischen den einzelnen Komponenten werden Lecks verhindert. [16]

Im Rahmen des Projektes tubulair± arbeitet die HAW mit einem anderen Zellaufbau: der tubulären, also röhrenförmigen Zelle, siehe Abbildung 2.6 B. Die Membran ist wie jedes Element der Zelle röhrenförmig aufgebaut und trennt die beiden Halbzellen voneinander. Die äußere Halbzelle wird um diese Membran gewickelt und wird nach außen von einem Stromsammler umgeben. Diesem Aufbau folgend wird die innere Halbzelle um den im Zentrum der Zelle liegenden Stromsammler gewickelt. Beide Halbzellen enthalten eine Graphitfilzelektrode. Die Unterschiede sind nicht Gegenstand dieser Arbeit und können zum Beispiel der Veröffentlichung von Ressel [16] entnommen werden.

In den Ausführungen dieser Arbeit wird vermehrt auf Messungen mit der tubulären Zelle eingegangen.

### 2.2.5 Aufbau eines Messablaufs

Ein Messablauf besteht aus mehreren Messmethoden, die sinnvoll aneinander gehängt werden. In Abbildung 3.4 im nächsten Kapitel ist ein beispielhafter Messablauf dargestellt und näher beschrieben.

An dieser Stelle soll es ausreichen zu erwähnen, dass Messungen oft mit einer Elektrolyt-Formierung und -Ladung beginnen. An diese wird oft ei-

ne lange OCV-Messung angeschlossen, um einen Vergleichswert der offenen Zellspannung zu einem bestimmten Ladezustand herzustellen. Für die Bestimmung der Verluste lassen sich anschließend eine oder mehrere Polarisationskurven aufnehmen. Im Mittelteil der Messung stehen oft Lade- und Entladezyklen um einen Realbetrieb zu simulieren. Diese werden wieder mit einer OCV-Messung abgeschlossen, an die sich wieder Polarisationskurven anschließen. Somit lassen sich die Eigenschaften der Zelle zu bestimmten Betriebspunkten bestimmen. In dieser Arbeit nicht behandelt und mit der Software auch nicht abgedeckt wird die sogenannte elektrochemischen Impedanzspektroskopie. In einer solchen EIS wird ein kleines Wechselstromsignal auf die Zelle gegeben und die resultierende Größe und Phase der Strom- und Spannungsantwort gemessen. Dieses Vorgehen wird mit vielen Frequenzen des Wechselstroms wiederholt. Mit den Ergebnissen lassen sich Material und Systemeigenschaften beschreiben. Genauere Informationen zu Messzyklen und deren Durchführungen finden sich in [17].

# Kapitel 3

# Software

## 3.1 Einführung

### 3.1.1 Anforderungen an die Software

Durch die Trennung der Messdaten in mehrere VI- und Potentiostat-Daten ist es notwendig, eine Einleseprozedur zu schaffen, welche mit beiden Datentypen zurechtkommt. Da der Aufbau der txt-Daten voneinander abweicht und einen Header beinhaltet, welcher diverse Informationen an verschiedenen Stellen bereithält, muss das Programm in der Lage sein, je nach eingelesenem Datentyp die richtigen Informationen und Messgrößen aus der Datei in den Workspace einzulesen.

Die nächste Anforderung an die Software ist die Schaffung einer externen Steuerdatei. In dieser Steuerdatei soll es hauptsächlich möglich sein, zentral die Kanäle der Messgrößen umzubenennen. Dies minimiert im weiteren Verlauf der Auswertung eine Verwechslung von Eingangsgrößen. Nötig ist dies, da die Ports der Messgeräte verschieden belegt sein können.

Nach dem Einlesevorgang steht die wichtigste Funktion der Software: dem Zusammenführen der beiden getrennt laufenden Messungen. Anstatt mehrerer Dateien mit unterschiedlichen Zeitstempeln soll das Ergebnis eine Datei mit einem gemeinsamen Zeitstempel sein. Diese Datenmenge soll in Form einer Matrix vorliegen, die alle Messgrößen umfasst.

Im Anschluss soll es möglich sein, die Datenmenge dieser Matrix graphisch aufzubereiten. Um die Betrachtung bestimmter Messroutinen zu erleichtern und die dafür benötigte Datenmenge zu reduzieren soll die Software außerdem die Möglichkeit bieten, aus der Matrix bestimmte Intervalle und Punkte herauszuschneiden. Die Programmierung von Auswerte-Tools für ausgewählte Messroutinen soll hingegen nicht Gegenstand dieser Arbeit sein, sondern lediglich die Schaffung einer Grundlage für solche Zusatzprogramme.

### 3.1.2 Ausgangszustand

Die Software musste nicht von Grund auf neu programmiert werden. Ein sehr früher Bestand der Software ist bereits durch L. Holtz beschrieben. [18] An dieser Stelle soll kurz der Stand der Software zum Zeitpunkt der Übergabe erläutert werden.

Die Einleseprozedur war bereits umgesetzt worden, allerdings fehlte ein automatisierter Umgang mit den Header-Zeilen in den Text-Dateien der Messungen. Die Einbindung einer Steuerdatei war noch nicht erfolgt. Das Matching war bereits umgesetzt worden, funktionierte allerdings bei den neueren Messungen nicht. Nach dem Matchen der Dateien gab es die Möglichkeit verschiedene Plots zu zeichnen. Aufbauend auf dieser Software waren bereits verschiedene Add-Ons entstanden, die die genauere Auswertung verschiedener Messprozeduren zum Ziel hatten.

Die Software hatte allerdings mit mehreren Problemen zu kämpfen. Das Matching war noch fehlerhaft und führte zu einem Versatz der Messdaten. In den Plots wurde nicht auf die fehlerhaft gematchten Daten zugegriffen, sondern versucht, über die umständliche Auswahl der jeweiligen Potentiostat-File lokal den Versatz auszugleichen. Eine Weiterverarbeitung der Daten war somit außerhalb der Plotting-Funktion der Software unmöglich. Das Programm war zudem fehleranfällig. Wenn Kanalnamen nicht genau dem entsprachen, was das Programm erwartete, führte dies zu einem Abbruch. Dieser Umstand wog umso schwerer, da das komplette Programm bis zum Plotting ohne Zwischenspeicherungen geschrieben wurde. Ein Abbruch durch einen Bug oder einen Bedienfehler seitens des Users zwang somit jedes Mal zum kompletten Neustart des Programms. Zudem war die Verwaltung des Workspace ineffizient, da die Daten intern im struct nach jedem Berechnungsschritt neu abgespeichert wurden und veraltete Datenbestände nicht aus dem Workspace gelöscht wurden.

Für einige dieser Probleme war das Festhalten an der Matlab-Datenstruktur *struct* zur Verwaltung der Messwerte und das strikte Teilen der verschiedenen Pot- und VI-Files verantwortlich. Andere entstanden wahrscheinlich durch die Vielzahl an Beteiligten, die die Software stückweise um kleine Teile erweitert hatten.

### 3.1.3 Überprüfung der Funktionen

Die in den kommenden Kapiteln ausgeführten Funktionen wurden je nach ihrem Zweck verschieden getestet. Der Test erfolgte im Generellen in mehreren Stufen. In einem ersten Test wurde ein stark vereinfachtes Beispiel verwendet, ähnlich denen, die auch in dieser Arbeit zur Erklärung der Funktionen benutzt werden. Die nächste Stufe bestand darin, die Funktionen isoliert mit konkreten Messdaten laufen zu lassen. Die Ergebnisse wurden auf Plausibilität geprüft und die Funktionen gegebenenfalls angepasst. Der letzte Schritt

war das Testen der kompletten Software mit der neuen Funktion, um auch die Kompatibilität der Funktionen untereinander zu gewährleisten.

Das Validieren der Funktionen, die beide Messgeräte aufeinander abstimmen, also das Matching und die Matrix-Erstellung gestaltete sich schwierig, da diese nicht beide die selben Messgrößen aufnehmen. Lediglich in früheren All-Vanadium-Messungen wurde mit dem Potentiostat die Zellspannung und mit dem VI die OCV gemessen, diese haben jedoch natürlich ein Spannungs-Offset zueinander. Es wurde also auch in großem Maße darauf gebaut, dass eventuelle Fehler in den Weiterverarbeitungen auffallen würden, die zeitgleich zur Erstellung der Arbeit abliefen. Die Software wurde also im Realbetrieb getestet.

## 3.2 Überblick

### 3.2.1 Aufbau

In Abbildung 3.1 sind die verschiedenen Zwischenschritte des Programmes in grob vereinfachter Form gezeigt. Sämtliche Funktionen werden über eine übergeordnete Main-Funktion aufgerufen; dies ermöglicht einen gewissen Grad der Modularität. Ein solcher beispielhafter Funktionsaufruf ist in Quellcode 3.1 dargestellt.

```
1 Ausgabeparameter = beispielFunktion(Eingabeparameter)
```

Quellcode 3.1: Beispiel eines Funktionsaufrufes

Der Startdialog lässt die Wahl zwischen 6 verschiedenen Punkten zum Einstieg in das Programm. Auf dem Schaubild sind zwar nur zwei Pfeile abgebildet, allerdings ist der Einstieg an jedem der orangen Prozesse möglich. Die Prozesse sind mit den entsprechenden Kapiteln in dieser Arbeit versehen, um schnell zu der entsprechenden Beschreibung des Prozesses zu finden.

Die Software beginnt mit dem Import der Dateien in einer Schleife bis der Anwender dem Programm zu verstehen gibt, dass er alle benötigten Dateien eingelesen hat. Die eingelesenen Dateien sind im Schaubild als importierte Rohdaten gekennzeichnet. Ist der Import erfolgt, beschreibt die Software anschließend die Steuerdatei, eine Excel-Tabelle. Außerhalb von Matlab soll der User nun die Messkanäle in der Steuerdatei umbenennen. Das Programm schließt sich daher an dieser Stelle. Wenn die Umbenennung erfolgt ist, wird das Programm erneut aufgerufen. Der User wählt im Startdialog die Möglichkeit nach Umbenennung der Steuerdatei mit der Software fortzufahren und liest sowohl die vorher abgespeicherten importierten Rohdaten als auch die beschriebene Steuerdatei ein. Nun erfolgt in Matlab die Umbenennung der Kanalnamen, Ergebnis sind die umbenannten Rohdaten. Darauf folgt das Matching der Zeitstempel anhand des Triggersignales. Hierzu wird der User zur eindeutigen Bestimmung des jeweilig gültigen Triggersignales dazu aufgerufen, einen Bereich anzugeben, in dem sich das Signal befindet.

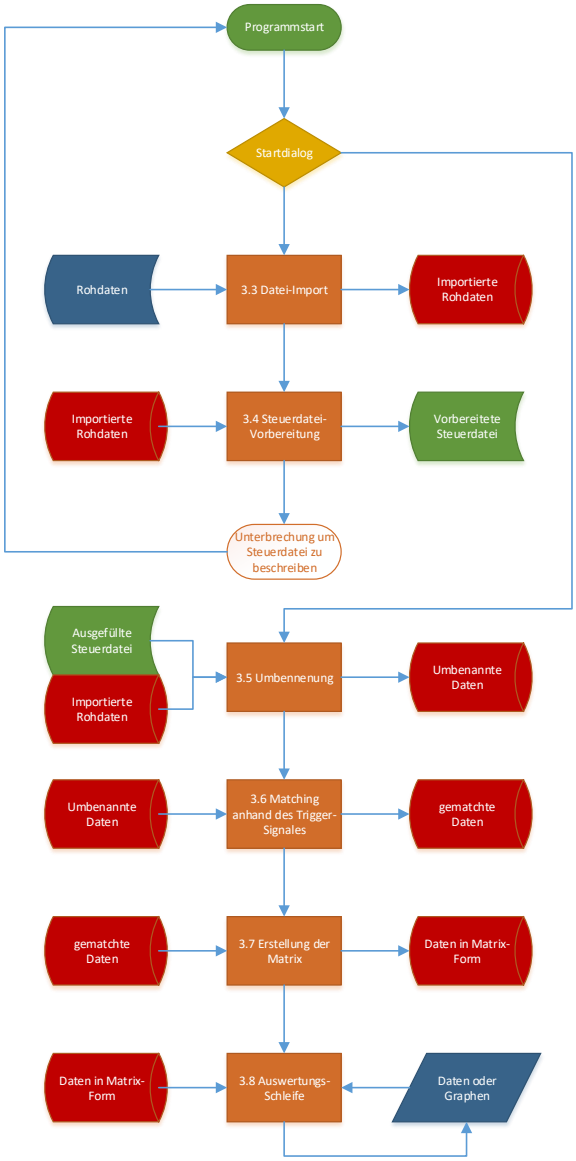


Abbildung 3.1: Darstellung des Programmablaufes

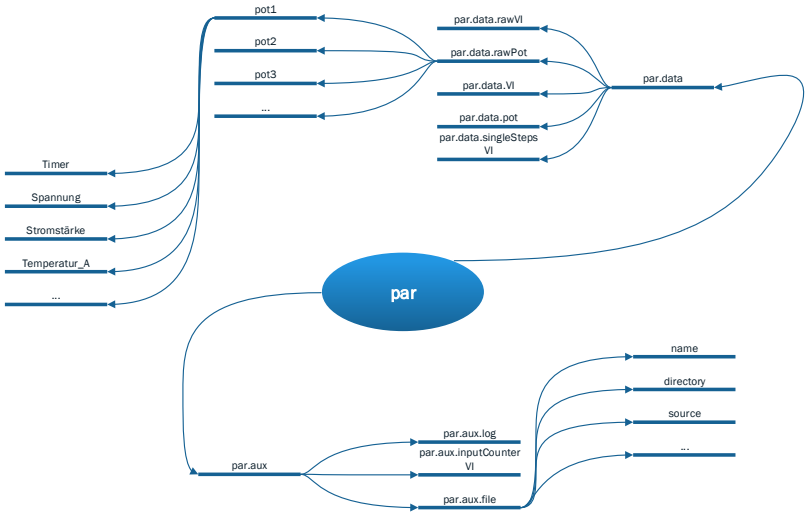


Abbildung 3.2: Prinzip der verwendeten Datenstruktur

Mit diesen nun also gematchten Daten kann die Erstellung der Matrix erfolgen. Mit der Matrix ist es nun im letzten Schritt der Software möglich, sich die verschiedenen Größen plotten zu lassen. Auch das Ausschneiden von bestimmten Bereichen und Größen der Matrix ist in diesem Programmteil möglich. Im Anschluss können nun eingehendere Auswertungen bestimmter Teile des Messdurchlaufes durchgeführt werden.

### 3.2.2 Datenstruktur

Die Software nutzt die Matlab-Datenstruktur des *structure arrays*. Daten werden in Feldern gespeichert. Mehrere Felder bilden dabei eine Ebene. Dies bietet sich besonders für eine variable Anzahl an zu verarbeitenden Daten an. Solange man zum Beispiel eingelesene Messdaten fortlaufend nummeriert, ist eine Eindeutigkeit gegeben. Das Prinzip ähnelt einer Mindmap, wie Abbildung 3.2 zeigt.

Die Informationen werden in der ersten Ebene in Daten und hilfreiche Informationen aufgeteilt. Letztere enthalten zum Beispiel Speicherorte, Dateinamen oder eine Zählervariable, die Aufschluss darüber gibt, wie viele Potentiostat-Dateien insgesamt eingelesen wurden. Im Daten-Teil werden sämtliche eingelesenen Messdateien eingelagert. Diese werden wiederum unterteilt nach Herkunft, Bearbeitungsschritt oder Dateinummer. In der letzten Ebene gibt es für jeden Kanal ein eigenes Feld.

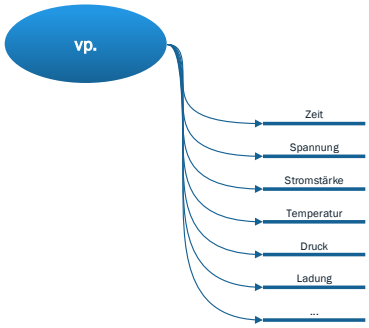


Abbildung 3.3: Die reduzierte vp-Struktur

Was allerdings für die Organisation Sinn ergibt, verkompliziert die Auswertung der Dateien. Um in einem Plot Daten aus VI und Potentiostat darzustellen, müssen in einem enormen Aufwand erst alle benötigten Kanäle aus mitunter vielen Unterebenen gesammelt werden. Der nächste Schritt ist das Aneinanderhängen der Informationen, um schlussendlich einen Plot erzeugen zu können. Diese Prozedur ist stark fehleranfällig. Zudem ist es durch die strikte Trennung der Größen durch diese Struktur nicht möglich, die gespeicherten Daten einfach in den Workspace zu laden und mit Standard-Befehlen zu untersuchen.

Abhilfe schafft das Beziehen sämtlicher Daten auf einen gemeinsamen Zeitvektor. Auf die Problematiken und die genaue Umsetzung wird in einem späteren Kapitel genauer eingegangen. Die Vorteile liegen dabei auf der Hand: Jeder Messwert ist eindeutig einem Zeitpunkt zugeordnet und jeder Messwert liegt in einer gemeinsamen Ebene vor. Die Aufbereitung der Werte gleicht damit einer Matrix. Um die vorhandenen Funktionen weiterhin benutzen zu können, wurde zwar nicht komplett auf die Organisation über das *structure array* verzichtet, trotzdem ist die Zugänglichkeit durch das Wegfallen der vielen verschiedenen Ebenen gestiegen. Wird in einem ersten Schritt diese gemeinsame Matrix noch unter der *data*-Ebene als Feld *vp* gespeichert, so reduziert sich die Struktur schlussendlich auf den in Abbildung 3.3 gezeigten Inhalt.

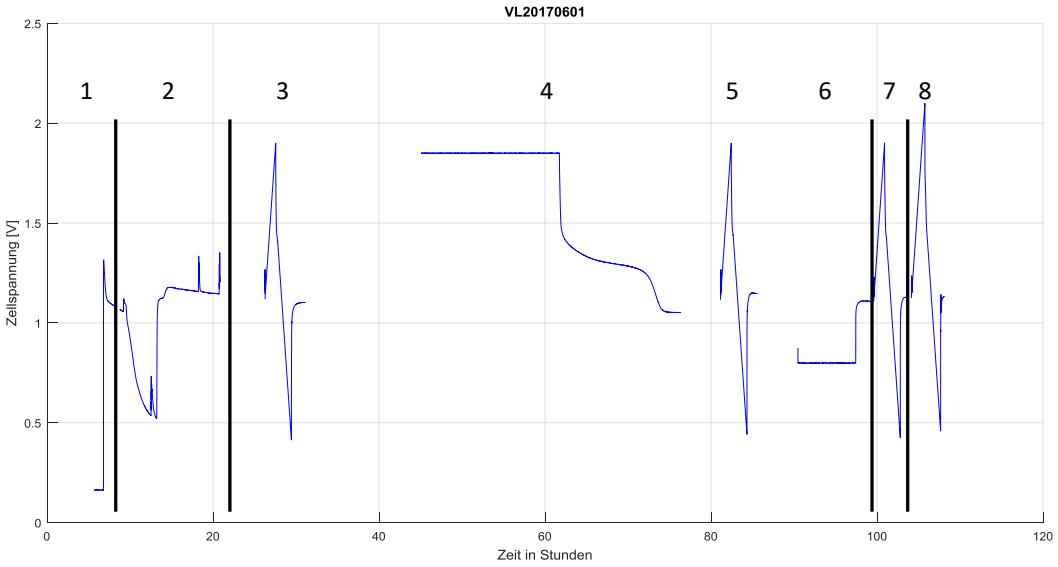


Abbildung 3.4: Messablauf der Vanadium-Luft-Messung vom 01.06.2017

### 3.2.3 Zum Aufbau der Beschreibungen

In diesem Abschnitt soll der Ablauf der Software an einem kontinuierlichen Beispiel erläutert werden. Als Beispiel wird hier die Messung einer VLRFB gewählt. Die Messung stammt vom 1. Juni 2017 und umfasst das Starten des Prüfstandes (1), die Aufnahme von ASR- und OCV-Werten bei verschiedenen Parametern (2), eine Polarisationskurve (3), dann einen Ladevorgang (4), eine Polarisationskurve (5), gefolgt von einem Entladevorgang (6), abgeschlossen mit zwei weiteren Polarisationskurven (7-8), siehe dazu auch Abbildung 3.4. In dieser Abbildung ist zu erkennen, dass das Potentiostat nicht über den kompletten Messlauf Daten aufnimmt, die VI-Messung wäre durchgehend. Die Unterfunktionen der Software werden nacheinander erläutert, mit Möglichkeit am Beispiel der Messung. Zur Verdeutlichung einzelner Programmzeilen wird aber auch auf vereinfachte Beispiele zurückgegriffen werden. Besonderer Wert wird dabei auf Teile der Software gelegt, die im Rahmen dieser Arbeit programmiert oder angepasst wurden. Unterfunktionen, die bereits vorhanden waren, werden kürzer erläutert.

Im Allgemeinen gilt: Matlab-Befehle wurden nach den Informationen der Mathworks-Online-Dokumentation genutzt, diese ist unter [19] zu finden.

Ist eine Funktion nicht der Dokumentation entnommen, wird eine Quelle angegeben und eine Beschreibung geliefert.

### 3.2.4 Der Startdialog

Der Aufbau der Software sieht vor, an verschiedenen Stellen in das Programm einzusteigen. Daher wird beim Aufruf der Main-Funktion als Erstes ein Dialog gestartet, in dem der Anwender gefragt wird an welchem Punkt er diesen Einstieg vornehmen möchte. Jeder Programmpunkt, der in Abbildung 3.1 mit einer Zahl versehen ist, kann als Einstieg genutzt werden. Die Wahl des Users wird über einen simplen input-Aufruf in Matlab abgefragt und als Zahl in den Workspace eingespeichert. Der Aufruf ist in Quellcode 3.2 zu sehen. Ein Standardwert ist bei ausbleibender Eingabe ebenfalls zugeordnet.

```
% ask for import option
importOption = input(['Wie soll vorgegangen werden? [0]\n0: Eine neue'...
  ' Messung liegt vor, .txt-Files einlesen.\n1: Nach File-Input ein'...
  'steigen; Steuerdatei beschreiben lassen.\n2: Nach Bearbeitung der'...
  ' Steuerdatei fortfahren.\n3: Nach Umbenennung vor dem Matching ein'...
  'steigen.\n4: Nach Matching, aber vor der Matrix-Erstellung einsteigen'...
  '.\n5: Mit fertiger Datei plotten oder cutten.\n']);

% set default
if isempty(importOption)
  importOption = 0;
end
```

Quellcode 3.2: Der Startdialog

Die Ausführungen in dieser Arbeit gehen davon aus, dass ein kompletter Programmaufruf durchgeführt wird; dies entspricht einer $importOption = 0$. Der Einstieg in andere Programmteile ist über eine if-else-Abfrage realisiert worden. Soll das Programm weiter hinten im Ablauf starten, wird in einen Fall gesprungen, der die Ausführung der Unterprogramme bis zu diesem Punkt nicht vorsieht. Durch dieses Vorgehen ist eine einfache Erweiterung des Programmablaufes nach hinten gesichert, also zum Beispiel die Einführung eines neuen sechsten Einsprungpunktes. Die Modularität der Unterfunktionen ermöglicht dieses Vorgehen, ohne dabei den Quellcode zu sehr zu überfüllen. Das Prinzip ist in Quellcode 3.3 dargestellt.

```
importOption = input('Abfrage:\n');
if importOption == 0
  % start with 0
```

```
4   % carry on with 1
5   % finally do 2
6  elseif importOption == 1
7   % carry on with 1
8   % finally do 2
9  elseif importOption == 2
10  % finally do 2
11 end
```

Quellcode 3.3: Prinzip des Startdialoges

## 3.3 Der Dateien-Import

Der erste Schritt der Software sieht einen Import der Messdaten vor. Nach der Auswahl der Messdatei muss das Programm die Datei in Hinsicht auf Trennzeichen richtig interpretieren und die Informationen korrekt und eindeutig in einer geeigneten Datenstruktur im Workspace abspeichern. Der grobe Ablauf des Imports wird in Abbildung 3.5 dargestellt.

Aus den im .txt-Format oder im Falle von Potentiostat-Dateien im .mpt-Format vorliegenden Daten werden Einträge in der *structure array*-Datenstruktur im Workspace generiert.

Die Bedienung dieses Abschnittes der Software erfordert Folgendes: Die Auswahl einer Messdatei über den Dateibrowser, dargestellt in Abbildung 3.6 und die Bestätigung der Quelle über das Kommandofenster von Matlab. Diese beiden Schritte wiederholen sich so lange, bis der Anwender in der Kommandofenster-Abfrage angibt, dass keine Dateien mehr einlesen werden sollen. Die Reihenfolge der eingelesenen Dateien **muss** dabei der Abfolge in der Messung entsprechen. EIS- und PEIS-Messungen sind noch nicht in der Software vorgesehen und sollten übersprungen werden. Zur Vereinfachung der Prozedur bietet es sich an, vor dem Starten der Software eine kleine Übersicht in Form von Tabelle 3.1 anzulegen. Die Informationen, die für eine solche Übersicht benötigt werden, lassen sich aus der *measurement routine* ablesen. Dort findet sich unter dem Reiter *measurement run* eine Abfolge aller Messzyklen. Der Name der VI-Messdatei befindet sich ganz oben in der Datei. Nacheinander kann der Anwender nun die Messzyklen durchgehen, und jeden Schritt, der nicht eine EIS- oder PEIS-Messung darstellt, als Eintrag in die Übersicht übernehmen. Dazu bietet es sich an, sich auf die in Tabelle 3.1 aufgeführten Kategorien zu beschränken. Der Name der Potentiostat-Messdatei befindet sich in den Kopfzeilen jedes Messzyklus. Am Wichtigsten ist es, sich den VI-Schritt zu notieren, in dem das Trigger-Signal auftritt. In Tabelle 3.2 ist ein solcher Eintrag in der *measurement routine* gezeigt. Falls gewünscht, lassen sich diese Ergebnisse am Ende des Schrittes über eine Bestätigung im Kommandofenster und eine Verzeichniswahl im Datei-Browser speichern.

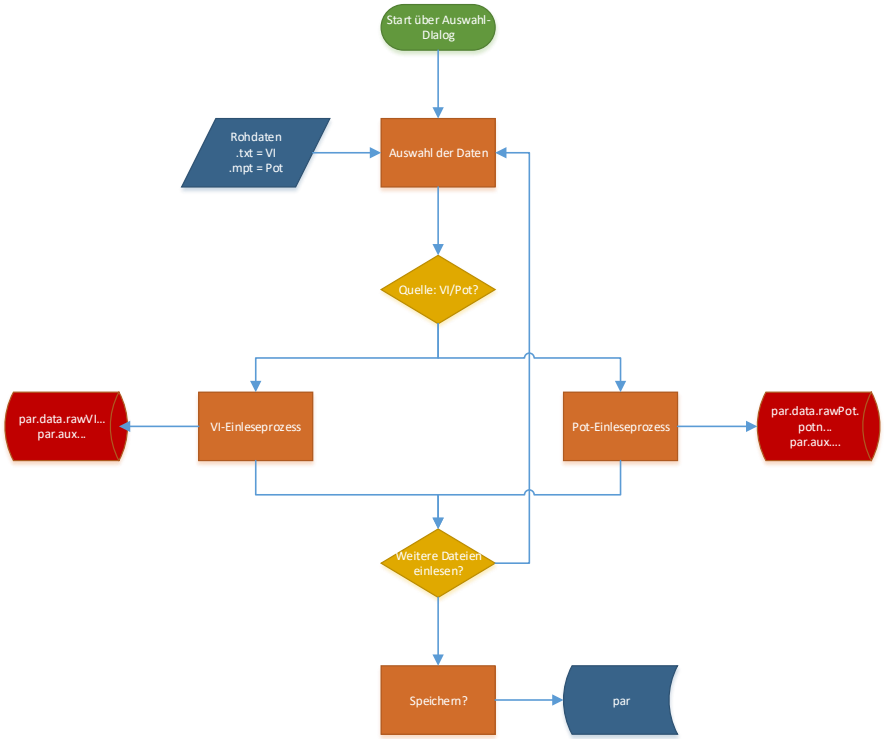


Abbildung 3.5: Die Einleseprozedur im Detail

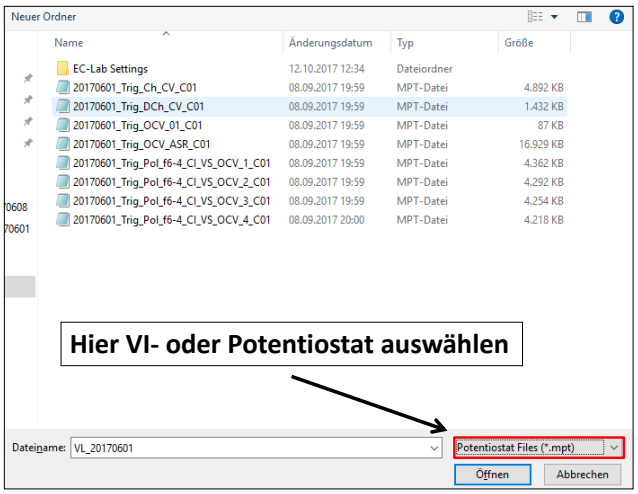


Abbildung 3.6: Auswahl der Messdateien im Import

Tabelle 3.1: Hilfstabelle; Beispiel für VL20170601

| VI | VL_170601_12.07_data.txt |
|---|---|

| pot-nr | Schritt | Trig | Dateiname |
|---|---|---|---|
| 1 | 0-5 | 3 | 20170601_Trig_OCV_01.mpr |
| 2 | 6-16 | 6 | 20170601_Trig_OCV_ASR.mpr |
| 3 | 20 | 20 | 20170601_Trig_Pol_f6-4_CI_VS_OCV_1.mpr () |
| 4 | 25-35 | 25 | 20170601_Trig_Ch_CV.mpr |
| 5 | 38 | 38 | 20170601_Trig_Pol_f6-4_CI_VS_OCV_2.mpr () |
| 6 | 40-48 | 40 | 20170601_Trig_DCh_CV.mpr |
| 7 | 49 | 49 | 20170601_Trig_Pol_f6-4_CI_VS_OCV_3.mpr () |
| 8 | 50 | 50 | 20170601_Trig_Pol_f6-4_CI_VS_OCV_4.mpr () |

### 3.3.1 Der Einlesevorgang im Detail

Mess-Dateien liegen entsprechend der zwei verschiedenen Geräte in zwei verschiedenen Formen vor.

VI-Dateien sind Textdateien, die aus einer einzelnen Kopfzeile mit den Namen der Messgrößen und unzähligen Zeilen mit den entsprechenden Werten bestehen. Sowohl die Namen der Messgrößen in der Kopfzeile als auch die Messwerte an sich im Rest der Datei sind Tabulator-getrennt. Matlab interpretiert dies in Textdateien richtig als Trennzeichen von Werten. Der Import besteht daher nur aus dem *genvarname*-Befehl und dem Erstellen von Vektoren, die die Messdaten enthalten. Nachteil dieser Methode ist der hohe Anspruch an den Arbeitsspeicher des Systems. Da VI-Dateien durchaus große Ausmaße von mehreren Hundert $MB$ annehmen können, ist dies ein nicht zu verachtender Faktor. Die eingelesenen Dateien werden in die struct-Datenstruktur eingebaut und liegen dann in dem Muster *par.data.rawData.VI1* vor. Im Normalfall gibt es pro Messlauf nur eine einzige VI-File; bei außerplanmäßigen Neustarts des Prüfstandes kann jedoch auch eine zweite Mess-Datei existieren. Die Importfunktion für VI-Dateien lag bereits vor und wurde auch unverändert übernommen.

Der zweite Typ Mess-Datei ist die Potentiostat-Datei. Diese Dateien werden aus einem anderen Programm generiert: EC-Lab. Ursprünglich liegen die Daten im Format *.mpr* vor. Über eine Exportfunktion in EC-Lab kann jedoch ein Export in das Format *.mpt* vorgenommen werden. *.mpt*-Dateien können über einen gängigen Texteditor geöffnet werden, und damit auch in Matlab eingelesen werden. Allerdings ist der Import in Matlab dieses Mal im Gegensatz zum Import der VI-Dateien nicht so trivial. Grund dafür ist ein umfangreicher Header, der verschiedene Informationen enthält, die jedoch zum großen Teil für die Auswertung völlig unwichtig sind. Unter diesem Header liegen die Messgrößen allerdings in der aus dem VI bekannten Form vor: Einer Zeile mit den Namen der Messgrößen und vielen weiteren mit den tatsächlichen Messwerten. Matlab muss daher den Großteil des Headers ignorieren, um dann mit dem Import der Messgrößen beginnen zu können.

Glücklicherweise gibt der Header in einer bestimmten Zeile aus, um wie viele Zeilen es sich bei dem kompletten Header handelt. Allerdings enthält der Header auch Leerzeilen. Diese machten bisher beim automatischen Auslesen der Informationen Probleme und die Anzahl dieser Zeilen musste vom Anwender per Hand eingegeben werden. Auch eine zweite Eingabe war erforderlich: Die Anzahl der aufgenommenen Größen. Da pro Messlauf durchschnittlich mit mehr als zehn Potentiostat-Dateien gerechnet werden kann, ergibt dies einen unnötigen Mehraufwand für den Anwender.

Die Importfunktion *importfilePot* ist dabei zweigeteilt. In einem ersten Vorgang, der von der Unterfunktion *getNHeaderLines* übernommen wird, wird die Anzahl der Zeilen im Header ermittelt, und wie viele davon Leerzeilen sind. Die Angabe über die Zeilenanzahl im Header liegt in den Text-

dateien in der im Code 3.4 gezeigten Form vor.

```
EC–Lab ASCII FILE
Nb header lines : 106
```

Quellcode 3.4: Anzahl der Header-Zeilen

Matlab muss also nun erfolgreich die 106 auslesen und als Information speichern. Dazu wird die Matlab-Funktion *textscan* verwendet. Der entsprechende Aufruf ist in Quellcode 3.5 gezeigt.

```
dataRaw      = textscan(fid, '%s', 8, 'Delimiter', ' ');
```

Quellcode 3.5: textscan

Der Aufruf liest die ersten acht Zeichenketten, die durch Leerzeichen getrennt sind, der momentan geöffneten Datei *fid* in die Variable *dataRaw* ein. Nun muss Matlab nur noch die Information der achten Zeichenkette in eine Zahlenvariable umwandeln. Theoretisch müsste Matlab jetzt nur Zeilen in Höhe der Anzahl der so ermittelten Headerzeilen überspringen; für die schon vorhandene übergeordnete Funktion wurde jedoch ein anderes Delimiter-Verhalten gewählt. Um diese Funktion nicht umschreiben zu müssen, wurde lediglich die Ermittlung der Leerzeilen automatisiert. Dies geschieht über den folgenden Quellcode.

```
frewind(fid);
headerRaw     = textscan(fid, '%s', nHeaderLinesRaw, '
    Delimiter', '\n');

% get number of empty lines
headerRawlength = length(headerRaw{1,1});
nEmptyLines     = 0;
for headerCount = 1:headerRawlength
  if cellfun(@isempty, headerRaw{1,1}(headerCount,1))
    nEmptyLines = nEmptyLines + 1;
  end
end


% subtract number of empty lines
nHeaderLines  = nHeaderLinesRaw - nEmptyLines;
```

Quellcode 3.6: Ermittlung der Leerzeilen

Zuerst wird über den Befehl *frewind* der Zeiger an den Anfang der Datei zurückgesetzt. Mit dem nächsten Befehl werden in ähnlicher Manier zu 3.5 sämtliche Zeilen des Headers in eine Variable eingelesen. Das Trennzeichen ist hier aber der Zeilenumbruch statt dem Leerzeichen. In einer anschließenden Schleife wird jede so eingelesene Zeile über eine Zellfunktion auf ihren

Inhalt untersucht: Ist dieser Inhalt leer, so wird die Zählvariable *nEmptyLines*, die gleichzeitig das Übergabeargument der Unterfunktion darstellt, um eins erhöht.

Mit dieser Information kann die Funktion *importfilePot* ebenfalls über die *textscan*-Operation die Messdaten aus der Textdatei in die Datenstruktur einlesen.

Ist eine Datei eingelesen worden, wird der Anwender gefragt, ob er weitere Dateien einlesen möchte. Diese *while*-Schleife ermöglicht eine variable Anzahl an einzulesenden Dateien. Sobald der User keine weitere Datei mehr einlesen möchte, wird das Abbruchkriterium gesetzt, und die Schleife wird verlassen. Nach dem Import der Dateien hat der Anwender die Möglichkeit zu speichern. Darüber wird der oben erwähnte Einstieg an verschiedenen Programmstellen ermöglicht. Im Startdialog gibt es die Möglichkeit, genau an dieser Stelle das Programm zu beginnen: Dazu benötigt der Anwender die soeben gespeicherte Datei, die er über einen Ladedialog auswählen kann.

### 3.3.2 Datenverwaltung über structure array

Wie bereits erwähnt, wird die Verwaltung der Daten über ein sogenanntes *strucure array* geregelt. Dieses Unterkapitel befasst sich mit den Ansteuern der Dateien in Matlab.

Das Abrufen von Informationen aus den Unterebenen wird durch Punkte geregelt. In Quellcode 3.7 werden die Felder aus dem Rohimport der ersten eingelesenen VI-Messdatei abgerufen. Der Übersichtlichkeit wegen wurde die abgebildete Antwort des Programms gekürzt.

```
1 par.data.rawVI.VI1
2 ans =
3  struct with fields:
4      Timer_s: [37325x1 double]
5      Schritt: [37325x1 double]
6      ...
```

Quellcode 3.7: struct-Aufruf

Möchte man auf eine spezifische Information aus dem struct zugreifen, so muss das Datenformat beachtet werden, in dem diese Information vorliegt. Verdeutlicht wird dies im Quellcode 3.8, in dem einmal eine Zahl (double) mit den runden Klammern und einmal eine Zeichenfolge im char-Format mit geschwungenen Klammern ausgelesen wird.

```
1 par.data.rawVI.VI1.Timer_s(1:5)
2 ans =
3  178.3000
4  179.3000
5  180.3000
6  181.5000
```

```
 182.5000

par.aux.file.name{1}
ans =
    'VL_170601_12.07_data.txt'
```

Quellcode 3.8: Spezifischer struct-Aufruf

## 3.4 Die Vorbereitung der Steuerdatei: *steuerdatei-Writer*

Eine der Hauptanforderungen an die Software stellte die Implementierung einer Steuerdatei dar. Da die Ports der Messkanäle in VI und Potentiostat variabel belegbar sind und auch durch verschiedene Gründe variabel belegt wurden, waren bisherige Datensätze ohne ständigen Abgleich der Belegung der Messkanäle nicht vergleichbar. In der Steuerdatei sollen alle existierenden Kanäle aufgelistet werden und dem Anwender die Möglichkeit gegeben werden, diese umzubenennen. Dies geschieht zur Minimierung von Komplikationen direkt nach dem Einlesen der Dateien in Matlab. Das Umbenennen der Kanäle garantiert bei konsequenter Durchführung zum Beispiel, dass die Zellspannung - unabhängig davon aus welcher Messung diese stammt - immer als *E_ cell* bezeichnet wird.

Im ersten Schritt des Arbeitens mit der Steuerdatei werden aus Matlab sämtliche Kanalnamen in eine externe Excel-Tabelle übertragen.

Der Schritt benötigt die eingelesenen Rohdaten, also das Ergebnis des letzten Kapitels und eine vorhandene Excel-Datei. Diese kann leer sein oder bereits andere Daten umfassen, da ein neuer Reiter angelegt wird. Ergebnis des Schrittes ist eine vorbereitete Steuerdatei mit sämtlichen Kanalnamen aus den Rohdaten, sortiert nach VI und Potentiostat.

Um den Schritt ausführen zu können benötigt der Anwender eine außerhalb von Matlab erstellte Excel-Tabelle. Der einzige Bedienungsschritt in Matlab ist das Auswählen der entsprechenden Tabelle über den Datei-Browser. Nach dem Vorbereiten der Steuerdatei beendet sich das Programm, um das Ausfüllen der Steuerdatei zu ermöglichen.

### 3.4.1 Umsetzung

Matlab bietet eingebaute Funktionen, um Excel-Dateien zu manipulieren. An dieser Stelle ist vor allem der *xlswrite*-Befehl interessant. Das Beschreiben des Headers über xlswrite gestaltet sich relativ einfach, wie der Quellcode 3.9 am Beispiel der Erstellung des Headers für die VI-Kanäle zeigt.

```
% create the header for the VI names
xlswrite(filename, {'VI-Originalnamen'}, 'Steuerdatei', 'A1'
    );
```

```
3 xlswrite(filename, {'VI - sprechender Variablenname'}, '
    Steuerdatei', 'B1');
```

Quellcode 3.9: xlswrite für den Header

Die vier Bestandteile des Funktionsaufrufes sind der Name der entsprechenden Datei, der gewünschte Text, der Reiter, in den geschrieben werden soll, sowie die Koordinate der Zelle, in die geschrieben werden soll.

Über die Unterfunktion *steuerdateiWriter* sollen nun die oben genannten Anforderungen erreicht werden. Der Funktionen wird das *struct*, der Dateiname der Excel sowie deren Speicherort übergeben. Die Funktion erstellt den Header der Steuerdatei nach Quellcode 3.9. Dann sammelt die Funktion die Kanalnamen aus den VI-Dateien; dies wird über den *fieldnames*-Befehl realisiert. Da es selten mehr als eine VI-File gibt, wird bei mehreren Dateien lediglich verglichen ob die VI-Kanalnamen gleich sind. Stimmen diese nicht überein, bekommt der Anwender eine entsprechende Fehlermeldung und das Programm wird geschlossen.

Die Daten aus dem Potentiostaten sind interessanter: Da es auf jeden Fall mehrere Dateien gibt, muss bei der Sammlung von Kanalnamen für jede neu eingelesene Datei ein Abgleich der Kanalnamen erfolgen. Von der ersten Potentiostat-Datei kann bedenkenlos jeder Kanalname in den Übergabeparameter übertragen werden. Ab der zweiten Datei wird über eine for-Schleife jeder Kanalname der zweiten Datei über eine weitere for-Schleife mit jedem Kanal des Übergabeparameter verglichen. An dieser Stelle gibt es lediglich zwei Fälle: Entweder der Name stimmt überein oder eben nicht. Wenn der Kanalname übereinstimmt, wird die erste for-Schleife verlassen und mit dem nächsten Kanalnamen der aktuellen Potentiostat-Datei fortgefahren. Ist der Name nicht gleich, so wird mit dem nächsten Eintrag des Übergabeparameters verglichen. Neu ist ein Kanalname genau dann, wenn er mit jedem Eintrag des Übergabeparameters verglichen wurde und keine Übereinstimmung festgestellt wurde. In diesem Fall wird der Name in den Übergabeparameter übertragen.

An einem kleinen Fallbeispiel kann dies verdeutlicht werden. Man nehme an, es existieren die zwei Potentiostat-Dateien **Primus** und **Secundus**. Primus beinhaltet die Kanalnamen Alpha, Bravo und Charlie. Secundus beinhaltet Alpha, Charlie und Delta. Führt man die oben ausgeführten Operationen durch, passiert folgendes: Die Kanalnamen von Primus werden in den Übergabeparameter Exitum übertragen. Exitum beinhaltet nun Alpha, Bravo und Charlie. Nun wird der erste Eintrag von Secundus mit dem Übergabeparameter verglichen. Der Vergleich ergibt direkt einen Treffer, es wird also zum nächsten Eintrag in Secundus übergegangen. Charlie wird erfolglos mit Alpha und Beta verglichen, dann erfolgt ein Treffer und es wird zum nächsten Eintrag in Secundus gesprungen. Delta wird mit allen drei Einträgen erfolglos verglichen. Er ist also noch nicht vorhanden und wird in Exitum aufgenommen. Exitum besteht nun erfolgreich aus den ersten vier Einträgen

des Nato-Alphabets und könnte nun entweder mit einer Datei Tertius verglichen werden oder per xlswrite in eine Excel geschrieben werden.

Durch dieses Vorgehen werden auf jeden Fall alle Kanalnamen der Potentiostat-Dateien erfasst. Die Reihenfolge, in der diese auftauchen, spielt keine Rolle. Probleme könnten sich höchstens dadurch ergeben, wenn die Schreibweise variiert, dies ist aber bei den Potentiostat-Kanalnamen bisher nicht der Fall gewesen. In Quellcode 3.10 findet sich der Quellcode für die Erstellung der Sammlung von Potentiostat-Kanalnamen.

```
% determine the number of pot files
n_pot = par.aux.inputCounterPot;

for idxPot = 1:n_pot
  if idxPot == 1
    % if this is the first pot just collect all the channel names into
    % a variable
    fieldnames_pot = fieldnames(par.data.rawPot.(sprintf('pot%d', idxPot)));

  else
    % if this is not the first pot create another fieldnames variable
    fieldnames_add = fieldnames(par.data.rawPot.(sprintf('pot%d', idxPot)));

    % determine the number of the new fieldnames
    n_pot_names = length(fieldnames_add);

    for idxPotNames = 1:n_pot_names
      % check every channel name and save if for comparison
      tmpName = fieldnames_add{idxPotNames};

      % update the number of channel names
      laenge_pot_names_neu = length(fieldnames_pot);

      % do this for every channel name already existing
      for idxOldNames = 1:laenge_pot_names_neu
        % get the channel name
        tmpNameOld = fieldnames_pot{idxOldNames};

        % if the "new" channel name already exists exit the for
        % loop and do this all over again for the next potentional
        % new channel name
        if strcmp(tmpName, tmpNameOld)
          break
        end
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | VI-Originalnamen | VI - sprechender Variablenname | | pot-Originalnamen | pot - sprechender Variablenname |
| 2 | Timer_s | | | mode | |
| 3 | Schritt | | | oxred | |
| 4 | Typ | | | error | |
| 5 | U_AG_V | | | controlchanges | |
| 6 | I_AG_A | | | Nschanges | |
| 7 | I_EL_A | | | counterinc | |
| 8 | I_PS_A | | | IRange | |
| 9 | Ladung_Ws | | | EweV | |
| 10 | Entladung_Ws | | | EceV | |
| 11 | Ladung_As | | | EweEceV | |
| 12 | Entladung_As | | | Schritt | |
| 13 | V1_I_0 | | | Timer_s | |
| 14 | Waage3_g | | | controlmA | |
| 15 | rho1_g_ml | | | ImA | |
| 16 | T1_C | | | dqmAh | |
| 17 | Pumpe6_min1 | | | QQomAh | |
| 18 | Level | | | EnergyWh | |
| 19 | UTH6_1_C | | | RappOhm | |
| 20 | UTH6_2_C | | | freqHz | |
| 21 | UTH7_1_C | | | ZOhm | |
| 22 | UTH7_2_C | | | PhaseZdeg | |
| 23 | RTD1_C | | | QchargedischargemAh | |
| 24 | RTD2_C | | | halfcycle | |
| 25 | RTD3_C | | | PW | |

Abbildung 3.7: Ausschnitt einer noch nicht ausgefüllten Steuerdatei

```
35
36          % if this is the last channel name
37          if idxOldNames == laenge_pot_names_neu
38            % if the current channel name is not included so
    far
39              if strcmp(tmpName, tmpNameOld) ~= 1
40                % update the number of channel names
41                neue_laenge = laenge_pot_names_neu + 1;
42                % put the new channel name in the fieldnames
43                % variable
44                fieldnames_pot{neue_laenge, 1} = tmpName;
45              end
46            end
47          end
48        end
49      end
50  end
```

Quellcode 3.10: Schleife in steuerdateiWriter

### 3.4.2 Das Füllen der Steuerdatei

Nach dem Ausführen von *steuerdateiSchreiber* beendet sich die Software, um dem Anwender Zeit zu geben, die Steuerdatei mit den erwünschten Kanalnamen zu füllen. Die Steuerdatei befindet sich in der in Abbildung 3.7 gezeigten rohen Form.

Alle auftretenden Kanalnamen sind unter dem Messgerät, in dem sie

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | VI-Originalnamen | VI - sprechender Variablenname | | pot-Originalnamen | pot - sprechender Variablenname |
| 2 | Timer_s | Timer_s | | mode | mode |
| 3 | Schritt | Schritt_VI | | oxred | oxred |
| 4 | Typ | Typ | | error | error |
| 5 | U_AG_V | OCV | | controlchanges | controlchanges |
| 6 | I_AG_A | I_AG_A | | Nschanges | Nschanges |
| 7 | I_EL_A | I_EL_A | | counterinc | counterinc |
| 8 | I_PS_A | I_PS_A | | IRange | IRange |
| 9 | Ladung_Ws | Ladung_Ws | | controlmA | controlVmA |
| 10 | Entladung_Ws | Entladung_Ws | | EweV | E_cell |
| 11 | Ladung_As | Ladung_As | | ImA | ImA |
| 12 | Entladung_As | Entladung_As | | dqmAh | dqmAh |
| 13 | V1_1_0 | V_neg | | QQomAh | QQomAh |
| 14 | V2_1_0 | V_pos | | EnergyWh | EnergyWh |
| 15 | Waage1_g | mass_neg | | RappOhm | RappOhm |
| 16 | Waage2_g | mass_pos | | QchargedischargemAh | QchargedischargemAh |
| 17 | T1_C | T_neg | | halfcycle | halfcycle |
| 18 | rho1_g_ml | rho_neg | | EnergychargeWh | EnergychargeWh |
| 19 | T2_C | T_pos | | EnergydischargeWh | EnergydischargeWh |
| 20 | rho2_g_ml | rho_pos | | CapacitancechargmyF | CapacitancechargmyF |
| 21 | f1_min1 | pump_freq_neg | | CapacitancedischargmyF | CapacitancedischargmyF |
| 22 | f2_min1 | pump_freq_pos | | QdischargemAh | QdischargemAh |
| 23 | Level1 | Level_neg | | QchargemAh | QchargemAh |
| 24 | Level2 | Level_pos | | CapacitymAh | CapacitymAh |
| 25 | p1_raw | p_neg_raw | | Efficiency | Efficiency |

Abbildung 3.8: Ausschnitt einer ausgefüllten Steuerdatei

auftreten, aufgelistet. Zum Umbenennen der Kanäle muss nun lediglich der gewünschte Kanalname in die Zelle rechts vom Originalnamen geschrieben werden. Jedem Kanal muss ein gewünschter Name zugeordnet sein, auch wenn dieser sich nicht vom originalen Namen unterscheidet. Der Timer muss *Timer_s*, der Kanal, der das Triggersignal beinhaltet, *Trig_01*, der Kanal für die Schritte vom VI *Schritt_VI* und der Kanal für die Schritte vom pot *Schritt_pot* heißen. Ansonsten können die Kanäle mit allen von Matlab interpretierbaren Zeichen umbenannt werden. Dabei sollte sich aber an den Konventionen orientiert werden, die sich im Team durchgesetzt haben. Aufschluss über die Benennung der Kanäle kann in der *measurement routine* der Reiter *Connections* liefern. Abbildungen 3.9 und 3.10 zeigen die Belegung der Kanäle in einer All-Vanadium-Messung vom 15. März 2017, da der *Connections*-Reiter in der Vanadium-Luft-Beispielmessung nicht beispieltauglich dokumentiert war. Ein Ausschnitt der befüllten gezeigten Steuerdatei von oben findet sich in Abbildung 3.8.

## 3.5 Das Interpretieren der Steuerdatei: *steuerdatei-Renamer*

Dieser Unterpunkt des Programms benennt die Kanäle der eingelesenen Rohdaten nach den Angaben der Steuerdatei um. Anschließend erfolgt im Falle von mehreren VI-Dateien ein Zusammenlegen der Rohdaten zu einem Eintrag. Als Vorbereitung für das Matchen der Zeitvektoren im nächsten Schritt der Software wird der nun komplette VI-Eintrag im *structure array* nach VI-

| Cycle | | | | |
|---|---|---|---|---|
| | pos/neg | Sensors | | |
| Left 1 | Pos | p1 | V1 | Level1 |
| | | Pump 3 | p1 | |
| Right 2 | Neg | p2 | V2 | Level2 |
| | | Pump 4 | p2 | |

Abbildung 3.9: Ausschnitt aus dem Reiter *Connections* der *measurement routine* mit Informationen für die Steuerdatei: Drücke, Temperaturen usw.

| DMM | | | |
|---|---|---|---|
| | HI | LO | Measuring |
| U_AG | Ref_pos | Ref_neg | OCV |
| Pot1 (MY50510103) | Ref_neg | Hg/HgSO4_neg | E_Nernst_neg |
| Pot2 (TW4810098) | Ref_pos | Hg/HgSO4_pos | E_Nernst_pos |
| Pot3 (TW48100171) | S_neg | Ref_neg | Eta_neg |
| Pot4 (TW48100154) | S_pos | Ref_pos | Eta_pos |

Abbildung 3.10: Ausschnitt aus dem Reiter *Connections* der *measurement routine* mit Informationen für die Steuerdatei: Spannungen und Überspannungen

Schritten aufgeteilt.

Der Unterpunkt benötigt die ausgefüllte Steuerdatei und die eingelesenen Rohdaten, um zu funktionieren. Als Ergebnis der Operation wird das *structure array* die beiden neuen Felder *VI* und *singleStepsVI* umfassen.

Der Anwender muss nach der Umbenennung der Kanäle in der Steuerdatei die *main*-Funktion starten. Als Einstiegspunkt dient dieses Mal die Option, nach der Umbenennung der Kanäle zu beginnen. Der Anwender wird aufgefordert das abgespeicherte *struct* sowie die Steuerdatei über den Datei-Browser zu laden. Der Rest der Operationen erfolgt automatisch. Um dem Benutzer zu melden, dass auch tatsächlich etwas passiert, füllt sich ein Ladebalken abhängig davon, wie viele Messdateien bereits umbenannt worden sind.

### 3.5.1 Funktionsweise

Nachdem der Anwender die Rohdaten sowie die Steuerdatei ausgewählt hat, folgt der Aufruf von *steuerdateiRenamer*. Dieser Aufruf enthält dabei mehrere Eingangsparameter, wie Quellcode 3.11 zeigt.

```
1 function [par, headerNames] = steuerdateiRenamer_1_0(par,
      headerNames, dataType, filename, pathname, idxData)
2 % Beispiel Aufruf
3 [par, ~] = steuerdateiRenamer_1_0(par, NaN, 1, par.aux.
      Steuerdatei_filename, par.aux.Steuerdatei_pathname,
      idxVI);
```

Quellcode 3.11: Aufruf von *steuerdateiRenamer*

Das Wichtigste ist, das par-struct zu übergeben. Die Variable *headerNames* zu übertragen ist ein Überbleibsel aus Debug-Vorgängen und wird in der Software selbst ignoriert. Der *dataType* wird entweder mit einer 1 für VI-Daten oder einer 2 für Potentiostat-Daten belegt. Die letzten drei Eingangsparameter geben den Speicherort und Dateinamen der Steuerdatei sowie die Information, um die wievielte Datei es sich respektive handelt, an die Unterfunktion weiter. Als Ausgabeparameter dient wieder das par-struct. Die headerNames-Ausgabe wird unterdrückt, da sie, wie bereits beschrieben, nur dem Debugen der Funktion diente.

Die Unterfunktion selbst folgt je nach eingelesenem dataType einem von zwei leicht unterschiedlichen Abläufen. Die Informationen aus der Steuerdatei werden über den xlsread-Befehl in den Matlab-Workspace geladen. Als Erstes wird überprüft, ob das Feld, in dem der neue Name stehen sollte, leer ist. Ist dies der Fall, bekommt der Anwender eine entsprechende Warnung vom Programm. Als Nächstes wird überprüft ob der neue und der alte Name identisch sind, also keine Umbenennung erwünscht ist. In diesem Fall springt die Funktion direkt zum nächsten Kanalnamen. Ist eine Umbenennung gewünscht, wird über den *isfield*-Befehl abgeglichen, ob der

alte originale Kanalname sich in der aktuell untersuchten VI- oder Pot-Datei befindet.

Für das Umbenennen eines Feldes in einem structure array in Matlab gibt es leider keine Funktion. Daher muss eine Notlösung bemüht werden: In der gleichen Feldebene wird eine Kopie des alten Kanals erstellt, allerdings benannt mit dem neuen Namen. Nun wird über den rmfield-Befehl der alte Kanal gelöscht. Im Quellcode 3.12 findet sich ein Beispiel für die Umbenennung. Man beachte auch den sprintf-Befehl, der es ermöglicht, Felder anzusteuern, die eine variable Zahl in ihrem Namen haben.

```
% create a copy of the old struct field with the new name
[par.data.rawVI.(sprintf('VI%d',idxData)).(name_desired)]
    ...
  = par.data.rawVI.(sprintf('VI%d',idxData)).(name_orig_VI);
% remove the struct field with the old name
par.data.rawVI.(sprintf('VI%d',idxData)) ...
  = rmfield(par.data.rawVI.(sprintf('VI%d',idxData)),
      name_orig_VI);
```

Quellcode 3.12: Umbenennen eines struct-fields

Der Anwender wird außerdem gewarnt, wenn der in der Steuerdatei angegebene alte Kanalname in den Workspace-Kanalnamen nicht existiert; das Programm bricht jedoch in einem solchen Fall nicht ab. Nach der Umbenennung der Kanäle folgt wieder ein Aufruf der Speicher-Funktion.

### 3.5.2 Weitere Schritte vor dem Matching

Bevor die Potentiostat-Messdaten an die VI-Zeit angeglichen werden können, müssen noch zwei vorbereitende Unterfunktionen ausgeführt werden. Falls mehr als eine VI-Messdatei eingelesen wurde, zum Beispiel durch eine Unterbrechung der Messung, so sind momentan im par-struct mehrere rawVI-Felder vorhanden. Da mehrere VI-Dateien die Ausnahme darstellen und in den folgenden Programmpunkten von einer einzelnen VI-Datei ausgegangen wird, muss in diesem Fall aus diesen Feldern ein einzelner Unterpunkt im struct entstehen. Die Funktion war bereits bei der Übergabe vorhanden und funktionstüchtig, daher folgt an dieser Stelle nur ein kleiner Überblick.

Die Unterfunktion assembly erstellt aus den Messdateien unter par.data.rawVI ein neues Feld par.data.VI. Die Daten aus data.rawVI unterscheiden sich von denen in data.VI nur, wenn es mehrere Einträge unter data.rawVI gibt. Die Funktion kombiniert die VI-Dateien, indem es die Kanäle hintereinander hängt. Das funktioniert bei allen Größen außer dem Timer und der Aufnahme der Schritte. Diese werden bei Unterbrechung der Messung neu gestartet, sollen aber kontinuierlich gezählt werden. Die Unterfunktion nimmt an dieser Stelle entsprechende Modifikationen an den Kanälen vor. Wurden die Kanäle von Hand getrennt um den Import der VI-Messdateien auch auf

schwächeren Systemen zu ermöglichen, entfällt diese Modifikation und die Kanäle werden lediglich kombiniert. Da aber mehr Rechenleistung auch die folgenden Unterfunktionen beschleunigt, sollte in Erwägung gezogen werden, das Auftrennen der VI-Dateien zu unterlassen und alle Schritte bis zur Auswertung allgemein auf einem schnellen Rechner durchzuführen.

Als Vorbereitung für das Matching im nächsten Schritt wird das soeben kombinierte VI nun wieder aufgeteilt. Diese Maßnahme verringert den Rechenaufwand bei der Suche des Trigger-Signales. Die Funktion *splitSteps* war bereits vorhanden und wurde im alten Programmablauf wesentlich häufiger eingesetzt. Sie unterteilt das VI-Feld par.data.VI wieder in mehrere Felder unter par.data.singleStepsVI, in Abhängigkeit der Variable *Schritt_ VI*. Dazu wird eine weitere bereits vorhandene Funktion genutzt: *dissemble*. Danach kann der Zwischenstand über die Speicher-Funktion vor dem Matching gesichert werden.

## 3.6 Das Matching der Daten

An nächster Stelle steht das Matching der VI- und Potentiostat-Dateien. Zur Erinnerung: Während die VI-Daten mittlerweile im Workspace aus einer fortlaufenden Messung mit einem kontinuierlichen Timer bestehen, liegen die Potentiostat-Messdaten immer noch als einzelne Dateien vor. Der Timer der Potentiostat-Dateien startet außerdem bei jeder Datei wieder von 0. Das Matching soll anhand des Trigger-Signales den Timer der einzelnen Potentiostat-Dateien in erster Näherung dem VI-Timer angleichen.

In der Datenstruktur werden als neue Felder *par.data.pot.potn*-Felder erstellt. Das $n$ gibt Auskunft darüber, um die wievielte Potentiostat-Datei es sich handelt.

Der Anwender muss für jede eingelesene Potentiostat-Datei angeben, in welchem VI-Schritt sich das zugehörige Trigger-Signal befindet. Dies geschieht über einen Auswahldialog, wie er in Abbildung 3.11 dargestellt ist. Die Information über das Signal findet sich entweder in der *measurement routine* oder idealerweise in einer Übersicht wie in Tabelle 3.1 bereits dargestellt.

### 3.6.1 Ausgangslage vor dem Matching und Ergebnis nach dem Matching

Für die Verdeutlichung der Datenlage vor dem Ausgleich des Versatzes durch das Matching dient Abbildung 3.12. Auf ihr ist der OCV-Verlauf aus dem VI und der Zellspannungsverlauf aus dem Potentiostat einer All-Vanadium-Messung (15. März 2017) abgebildet, da in den Vanadium-Luft-Messungen über das VI keinerlei Spannungen aufgenommen wurde, und es somit keine übereinstimmende Größe gibt. Man erkennt, dass beide Messungen bei

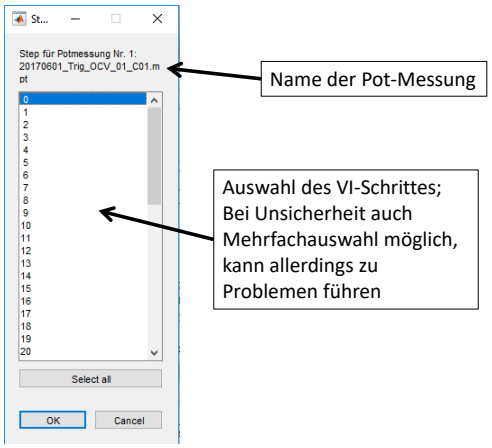


Abbildung 3.11: Auswahl des Schrittes, der das Trigger-Signal beinhaltet

einer Zeit von 0 Sekunden beginnen. Ebenfalls erkennen lassen sich in beiden Graphen die charakteristischen Formen des Entlade-/Lade-Messzyklus. Auch der zeitliche Versatz ist eindeutig zu erkennen.

Über das Bestimmen des Timers zum Beginn des Triggersignales, lässt sich der Versatz bestimmen, der auf die Potentiostat-Zeit aufaddiert werden muss, siehe Abbildung 3.13.

Nach dem Ausgleichen des Versatzes lassen wir uns den oben gezeigten Plot noch einmal ausgeben. Abbildung 3.14 zeigt den kompletten Messlauf, während in Abbildung 3.15 auf Sekundenebene in den Lade-/Entladezyklus herangezoomt wird, um zu demonstrieren, dass der Versatz bei Abtastraten beider Geräte von $1Hz$ eine Sekunde Unterschied nicht überschreitet.

### 3.6.2 Funktionsweise

Das Trigger-Signal ist eine binäre Größe im VI, die immer dann auf 1 springt, wenn sie durch das Potentiostat aktiviert wird. Die Aktivierung geschieht immer dann, wenn eine neue Potentiostat-Messung gestartet wird. Die Idee dieses Programmschrittes ist, den VI-Timer zum Zeitpunkt des Triggersignales auszulesen und auf den immer von 0 startenden Pot-Timer zu addieren. Diese erste Näherung ergibt eine Genauigkeit von $0 < \Delta t < 1s$, da die Periodendauer beider Signale für die größte Dauer der Messung mit einer Abtastrate von $1Hz$ aufgenommen wird. Ob die Potentiostat-Sekunde nun aber genau zum Zeitpunkt der VI-Sekunde losläuft oder im ungünstigen Fall

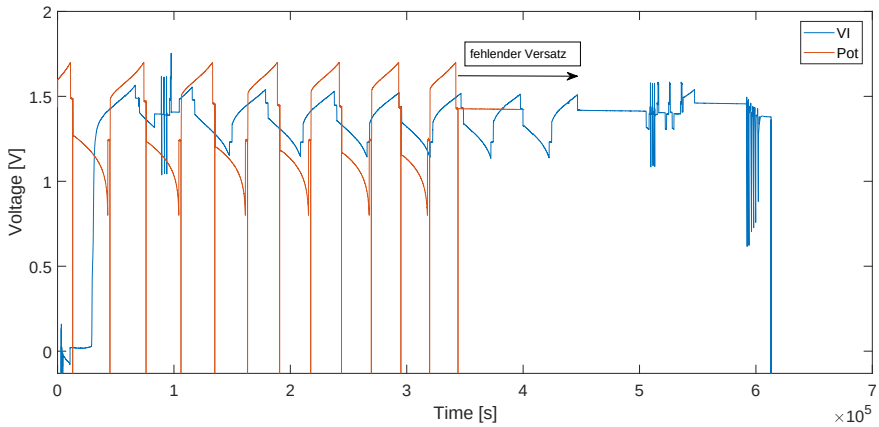


Abbildung 3.12: Zeitlicher Versatz der Entlade-/Ladezyklen von Potentiostat und VI vor dem Matching

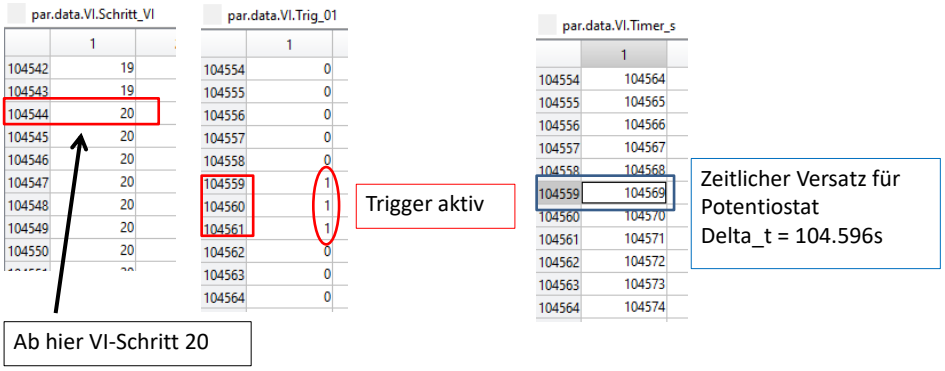


Abbildung 3.13: Bestimmung des Versatzes über das Trigger-Signal

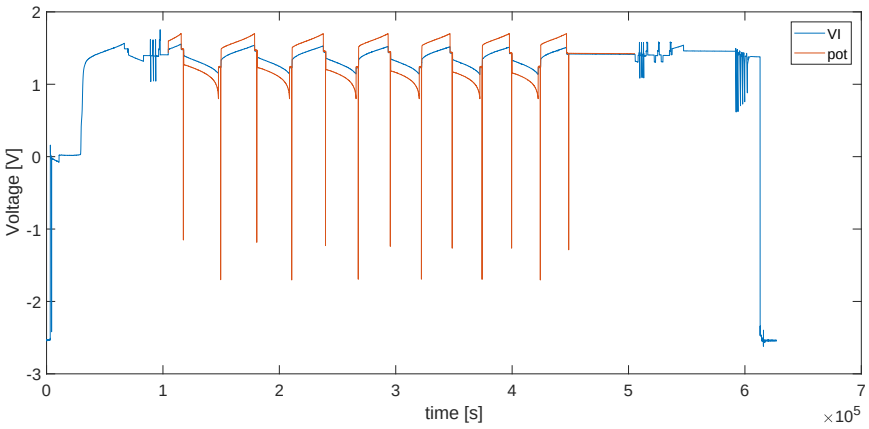


Abbildung 3.14: Vergleich der Zellspannungen nach Matching

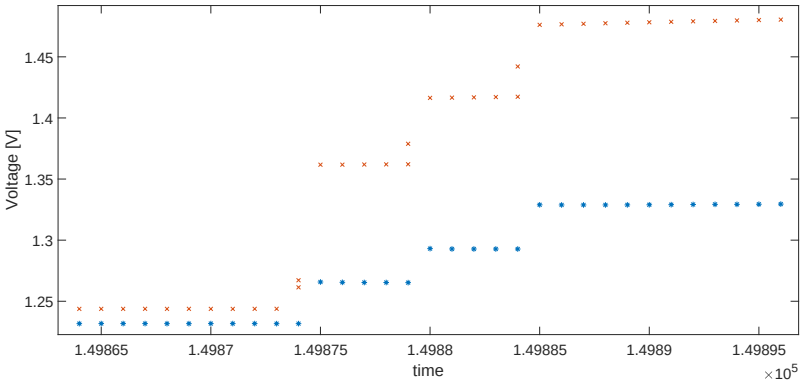


Abbildung 3.15: Vergleich der Zellspannungen nach Matching; vergrößert

$0,95s$ später kann in diesem Schritt nicht berücksichtigt werden, woraus die maximale Ungenauigkeit von bis zu einer Sekunde resultiert.

Die Matching-Funktion bildet eine Schleife über alle vorhandenen Potentiostat-Dateien. Für jede wird dann eine Abfolge von Funktionen durchgeführt. An erster Stelle steht die Auswahl des VI-Schrittes in dem sich das zugehörige Trigger-Signal befindet. Über die sogenannte *measurement routine*, die für jeden Messlauf angelegt wird, lässt sich dieser Schritt herausfinden. In Tabelle 3.2 ist ein Ausschnitt aus einer solchen *measurement routine* zu sehen.

Tabelle 3.2: Measurement-Routine: Trigger-Signal

| Number | STEP# | Set Parameters / Goal | Remark |
|---|---|---|---|
| 0 | 20 | Trigger out for 2 s rising | |

Der Eintrag zeigt, dass das Triggersignal im VI-Schritt 20 zu finden ist. Im Programmablauf folgt der Aufruf der Unterfunktion *getCharacteristicData*. Diese Funktion war bereits vorhanden und durch den neuen Ablauf sind weite Teile der Funktion überflüssig. Trotzdem erfüllt sie die Aufgabe, die an dieser Stelle an sie gestellt wird. Die Unterfunktion gibt dem Anwender ein Fenster mit einer Liste aller verfügbaren VI-Schritte aus. Im Kopf des Fensters wird der Dateiname der aktuellen Potentiostat-Datei angezeigt. Mit Hilfe der *measurement routine* kann wie oben beschrieben der oder die Schritte selektiert werden, in dem sich das Signal befindet. Theoretisch sollte es immer nur ein einzelner Schritt sein, falls die measurement-file jedoch nicht gründlich geführt wurde, können auch mehrere Schritte angegeben sein. Dies führt mitunter noch im weiteren Verlauf der Funktion zu einem Problem. Die Funktion lädt dann aus par.data.singleStepsVI, welches im letzten Unterkapitel erstellt wurde, die selektierten Schritte und baut diese ähnlich wie in der ebenfalls schon beschriebenen Unterfunktion *assembly* zu einem Feld zusammen.

Es folgt der Aufruf der Unterfunktion *brutMatchTrig*. Diese Funktion soll den zeitlichen Versatz in die Potentiostat-Datei einbauen. Dazu wird als Erstes der vorher festgelegte Ausschnitt aus den VI-Dateien über eine weitere Unterfunktion - *getChange* - nach Änderungen des Trigger-Signales durchsucht. Das Signal kann entweder den Wert 0 oder 1 annehmen. Die Funktion unterscheidet dabei drei Fälle: es werden genau zwei Änderungen gefunden, es wird keine Änderung gefunden oder es wird eine andere Anzahl an Änderungen gefunden. Letztere Fälle führen beide zu einer Fehlermeldung, da es bei keiner Änderung offensichtlich kein Trigger-Signal gibt und bei mehr als zwei Änderungen keine Eindeutigkeit mehr gegeben ist. Der einzig valide Fall ist also logischerweise die zweifache Änderung des Signals, da es für einen kurzen ununterbrochenen Zeitraum von 0 auf 1 springen muss. Über den Index der entsprechenden Variable *trigChange*, lässt sich der VI-Timer

an dieser Stelle auslesen. Addiert man diesen Wert nun auf jeden Timer-Wert der aktuellen Potentiostat-Datei, sind die Dateien in erster Näherung gematcht.

Der Rest der matching-Funktion war dazu konzipiert worden, das Angleichen des Versatzes über einen Plot dem User visuell zu zeigen. Dabei wurde die Spannung von VI und Potentiostat vor und nach dem erfolgten Matching geplottet. In den neueren Vanadium-Luft-Messungen existiert jedoch keine Größe mehr, die sowohl vom VI als auch vom Potentiostaten aufgenommen wird, daher ist diese Funktion je nach eingelesener Messung obsolet. Als letzter Punkt des Matchings wird ein neues Feld im par-struct erstellt. An dieser Stelle erfolgt wieder der Aufruf der Speicherfunktion.

## 3.7 Die Erstellung der Matrix

Als größte Neuerung der Software sollte die Erstellung einer Matrix erfolgen, die sämtliche eingelesenen Daten auf einen gemeinsamen Timer bezogen darstellt. Für jeden Wert im VI-Timer wird also höchstens ein Index aus den Potentiostat-Daten bestimmt. Neben der Matrix-Erstellung wird auch eine Schritt-Variable über den kompletten Messverlauf generiert. Den Abschluss der Software vor der Auswertung bildet die Berechnung von alternativen Timern, Stromdichte und Ladung.

In diesem Schritt wird aus den Feldern *pot* und *VI* unter dem *data*-Feld des *par*-structs ein weiteres Feld generiert, namentlich *vp*. Das *vp*-Feld beinhaltet sämtliche Messgrößen auf die VI-Zeit bezogen. Dem *vp*-Feld werden die oben beschriebenen zusätzlichen Größen hinzugefügt. Der Workspace wird als zwei verschiedene Dateien gespeichert; einmal komplett und einmal nur das *vp*-Feld als zweites *structure array*.

Der Nutzer muss in diesem Schritt lediglich über das Matlab-Kommandofenster den später thematisierten Rundungs-Threshold angeben, er sollte jedoch auf dem Standard-Wert belassen werden. Die projizierte Fläche der Membran und die bevorzugte Vorzeichenkonvention für Strom und Ladung wird ebenfalls über das Kommandofenster angeben. Bisher wurde sich an die Konvention aus der Brennstoffzellentechnik gehalten, eine Entladung wurde also als positiv angenommen. Das Abspeichern der Dateien erfolgt über den Datei-Browser. Neben dem kompletten Workspace wird dabei automatisch eine zweite Datei mit dem Namenszusatz _ *vp* im gleichen Zielordner erzeugt.

### 3.7.1 Erster Lösungsansatz

Problematisch beim Angleichen der Werte auf einen Zeitvektor ist die Variabilität der Abtastrate des Potentiostaten: bei *current interrupts* steigt die Samplerate zum Beispiel von den üblichen $1Hz$ auf $2500Hz$ an. Bei der Programmierung der Software fiel außerdem auf, dass auch die Samplerate des VIs nicht konstant war. Diese nahm in unregelmäßigen Abständen erst nach

$1,2s$ statt nach $1s$ den nächsten Wert auf. Ein reines Orientieren an der Abtastrate fällt also als Lösung heraus.

Es wurde beschlossen, auf ein Upsampling der VI-Messwerte, also dem Interpolieren von Daten entsprechend der zu diesem Zeitpunkt aktuellen Potentiostat-Abtastrate zu verzichten und stattdessen nur einen Potentiostat-Wert zu übernehmen, der ausreichend nah am entsprechenden VI-Wert liegt. Da ein genaues Übereinstimmen der Zeiten bei den gegebenen Rahmenbedingungen nicht gegeben ist, liegt die Idee nahe, die Potentiostat-Zeiten auf die VI-Zeit zu runden.

Es wird also eine Funktion benötigt, die jede Potentiostat-Zeit mit allen VI-Zeiten vergleicht und auf die nächstgelegene VI-Zeit rundet. Genau diese Anforderung erfüllt eine Funktion, die vom Nutzer „Tom R "über den MathWorks File Exchange geteilt wurde. [20] Die Funktion benötigt als Eingabe den zu rundenden Vektor A und den Vektor B, auf dessen Werte gerundet werden soll. Als drittes Eingabeargument dient eine Spezifikation des Rundungsverhaltens: Möglich ist das kaufmännische Runden, das Runden gegen 0 oder von 0 weg sowie das Runden gegen $\pm\infty$. Wird kein Typ spezifiziert, wird auf das kaufmännische Runden als Standardfall zurückgegriffen.

Für das Ermitteln der zueinander passenden Indizes werden zwei Befehle genutzt, gezeigt in Quellcode 3.13.

```
% round every Timer_s-value of pot towards a Timer_s-value
    included in VI
time_pot_rounded = roundtowardvec(time_pot, timeVI);

% get indices of the coinciding Timer_s-values
[~, idxPot_temp, idxVI_temp] = intersect(time_pot_rounded,
    timeVI);
```

Quellcode 3.13: roundtowardvec-Aufruf

Im ersten Schritt wird wie oben beschrieben der Potentiostat-Timer auf die VI-Zeit gerundet. Der Befehl *intersect* ermittelt die Indizes, an denen sowohl in der gerundeten Potentiostat-Zeit als auch in der VI-Zeit der gleiche Wert zum ersten Mal auftaucht. Dies stellt eine gewisse Ungenauigkeit dar, da nicht der Index aus dem Potentiostat genommen wird, der am wenigsten von der VI-Zeit abweicht, sondern der Index, der sich als Erstes auf die VI-Zeit runden lässt. Bei großen Datenmengen steigt die Berechnungsdauer für das Runden drastisch an. Für eine erste lauffähige Version der Software wurden diese Nachteile jedoch in Kauf genommen. Die Beschreibung der verbesserten Version erfolgt im nächsten Kapitel. An einem Beispiel in Tabelle 3.3 soll verdeutlicht werden, wie das Zusammenspiel der beiden Funktionen funktioniert.

Sind die entsprechenden Indizes auf diese Weise ermittelt worden, werden diese im *aux*-Teil des *structs* abgespeichert. Ebenso wird die Länge des aktuellen Schrittes auf diese Weise abgespeichert, diese dient später der Vor-

Tabelle 3.3: Beispiel für das Verhalten der Rundung mit dem ersten Lösungsansatz

| Index | VI-Zeit | Pot-Zeit | gerundet | Index VI | Index Pot |
|---|---|---|---|---|---|
| 1 | 20 | 21 | 21 | 2 | 1 |
| 2 | 21 | 22 | 22 | 3 | 2 |
| 3 | 22 | 22,1 | 22 | 4 | 8 |
| 4 | 23,2 | 22,2 | 22 | 5 | 14 |
| 5 | 24,2 | 22,3 | 22 | 6 | 15 |
| 6 | 25,2 | 22,4 | 22 | 7 | 16 |
| 7 | 26,4 | 22,5 | 22 | 8 | 17 |
| 8 | 27,4 | 22,6 | 23,2 | 9 | 19 |
| 9 | 28,4 | 22,7 | 23,2 | | |
| 10 | 29,4 | 22,8 | 23,2 | | |
| 11 | 30,6 | 22,9 | 23,2 | | |
| 12 | 31,8 | 23 | 23,2 | | |
| 13 | 33 | 23,1 | 23,2 | | |
| 14 | 34 | 24 | 24,2 | | |
| 15 | 35 | 25 | 25,2 | | |
| 16 | 36 | 26 | 26,4 | | |
| 17 | | 27 | 27,4 | | |
| 18 | | 27,5 | 27,4 | | |
| 19 | | 28 | 28,4 | | |
| 20 | | 28,5 | 28,4 | | |

bereitung der Matrix. Diese Operationen werden in einer Schleife für jede Potentiostat-Datei durchgeführt.

Das Übertragen der Werte vom VI gestaltet sich simpel: Hier wird ausnahmslos jeder Wert in die Matrix kopiert. Die Matrix wird der Einfachheit halber ebenfalls im par-struct gespeichert und zwar unter der Adresse par.data.vp. Über eine Schleife wird jeder Messkanal unter seinem Namen in die neue Struktur kopiert. Die Schleife ist unter Quellcode 3.14 gezeigt.

```
% get VI-fieldnames + number of them
VI_names          = fieldnames(par.data.VI);
number_VI_names   = length(VI_names);

for VI_counter = 1:number_VI_names
  VI_fieldname     = char(VI_names(VI_counter,:));
  VI_transporter   = par.data.VI.(sprintf('%s', VI_fieldname));
  par.data.vp.(sprintf('%s', VI_fieldname)) = VI_transporter;
end
```

Quellcode 3.14: Kopieren der VI-Kanäle in die Matrix

Zu sehen ist, dass für jeden Kanalnamen des VIs der entsprechende Kanalname als *character*-Variable zwischengespeichert wird, um sie mit dem *sprintf*-Befehl verwenden zu können. Anschließend wird der Kanal komplett kopiert und an der neuen Stelle im struct eingefügt; die Transporter-Variable verbessert lediglich die Lesbarkeit des Quellcodes.

Das Einfügen der Potentiostat-Kanäle gestaltet sich wie gewohnt ein wenig komplizierter. Über eine erste Schleife wird sichergestellt, dass jede Potentiostat-Datei bearbeitet wird. Eine zweite Schleife erstreckt sich über jeden Kanalnamen der Datei. Als Erstes wird der Kanalname darauf geprüft, ob er eine Timer-Variable oder *stepNew* ist. Der Abgleich mit stepNew ist nur noch für alte Dateien in der Funktion enthalten, wird das Programm mit der aktuellen Version durchgeführt ist diese Variable zu diesem Zeitpunkt im Programm noch nicht vorhanden. Die Kontrolle, ob es sich um eine Timer-Variable handelt, dient dem Zweck, keine veralteten und damit obsoleten Timer in die Matrix zu inkludieren.

Sind diese Möglichkeiten ausgeschlossen, wird überprüft, ob sich der Kanalname bereits in der Matrix befindet. Diese Operation ist besonders wichtig, damit sämtliche Kanäle der Matrix am Ende eine einheitliche Größe besitzen, obwohl einige Kanäle nur in manchen Potentiostat-Dateien enthalten sein können. Um diesen Fall abzufangen, wird jedes Mal, wenn ein Kanalname gefunden wird, der noch nicht in der Matrix vertreten ist, eine neues Feld mit dem Kanalnamen in der Matrix erstellt. Dieses Feld besitzt die maximale Länge, also die Länge der VI-Kanäle, und ist mit *NaN* gefüllt. Die Belegung mit *NaN* garantiert, dass diese Werte in der Auswertung keine

Ergebnisse verzerren (z.B. bei einer Integration) und dient außerdem dazu, das Programm zu beschleunigen, da eine Vorbelegung einer kompletten Variable mit anschließender Befüllung immer schneller ist als das sukzessive Erweitern. Das Verteilen der Werte in die Matrix geschieht analog zum Vorgehen bei VI-Daten, allerdings muss zusätzlich die Nummer der Pot-Messdatei, sowie die Indizes, an denen die Werte einsortiert werden sollen beachtet werden. Der in Quellcode 3.15 abgebildete Code ist an dieser Stelle ein wenig gekürzt, um nicht zu viel Platz zu verbrauchen.

```
1  for current_pot = 1:n_pot_existing
2    % how many pot-values have to be distributed?
3    index_length     = par.aux.indices(current_pot).idxLength;
4
5    % get pot-fieldnames + number of them
6    pot_names        = fieldnames(par.data.pot.(sprintf('pot%d',
       current_pot)));
7    number_pot_names    = length(pot_names);
8
9    % loop for every field in the current pot-data
10   for pot_counter = 1:number_pot_names
11
12     pot_fieldname    = char(pot_names(pot_counter,:));
13
14     if strcmp(pot_fieldname, 'Timer_s') == 1
15       continue
16       % hier wuerden noch andere Faelle abgedeckt werden
17     end
18
19     if isfield (par.data.vp, sprintf('%s', pot_fieldname))
       == 0
20       par.data.vp.(sprintf('%s', pot_fieldname))(1:
       max_length_VI, :) = NaN;
21     end
22
23     for pot_transporter_marker = 1:index_length
24       pot_transporter = par.data.pot.(sprintf('pot%d',
       current_pot)).(sprintf('%s', pot_fieldname))(par.aux.
       indices(current_pot).idxPot(pot_transporter_marker));
25       par.data.vp.(sprintf('%s', pot_fieldname))(par.aux.
       indices(current_pot).idxVI(pot_transporter_marker)) =
       pot_transporter;
26     end
27   end
28 end
```

Quellcode 3.15: Kopieren der Pot-Kanäle in die Matrix

### 3.7.2 Verbesserte Umsetzung

Für eine erste Version der Matrix-Erstellung ist dieses Vorgehen ausreichend. Größter Nachteil dieser Ausführung der Matrix-Erstellung ist, dass nicht der Potentiostat-Index übernommen wird, der die geringste Entfernung von der VI-Zeit aufweist, sondern der Potentiostat-Index, der als Erstes auf die entsprechende VI-Zeit gerundet werden kann. Das Runden jeder einzelnen Potentiostat-Zeit auf die VI-Zeit resultiert außerdem in einer enormen Berechnungsdauer. Da allerdings schon das Matching anhand des Trigger-Signales mit einer Ungenauigkeit behaftet ist, bringt eine erhöhte Genauigkeit bei der Matrix-Erstellung nur ein geringfügig besseres Ergebnis. Die Kombination aus Ungenauigkeit und Rechenzeit dieser Version veranlasste dazu, die Matrix-Erstellung zu verfeinern.

```
% create waitbar
waitB = waitbar(0,'Find equivalent times');

% do this for every pot-file
for current_pot_initial = 1:n_pot_existing

  % refresh waitbar
  waitbar((current_pot_initial-1)/(n_pot_existing-1));

  % get pot time of current step
  time_pot = par.data.pot.(sprintf('pot%d',
    current_pot_initial)).Timer_s;

  % start time
  startTime = time_pot(1);
  % end time
  endTime = time_pot(end);

  % find corresponding time in VI
  startTimeVI = roundtowardvec(startTime, timeVI);
  endTimeVI = roundtowardvec(endTime, timeVI);

  % find their indices
  [startTimeVIIdx, ~] = find(timeVI == startTimeVI);
  [endTimeVIIdx, ~] = find(timeVI == endTimeVI);

  % counter to determine the new length of the pot-data
  numberOfMatches = 0;

  % find the closest Pot-time for every VI-Time
  for loopCounter = startTimeVIIdx:1:endTimeVIIdx
    % calculate time difference
    potDiff = timeVI(loopCounter) - time_pot;
    % get absolute values
    potDiffAbs = abs(potDiff);
```

```
35
36      % find the minimum and its index
37      [actDiff, idxDiff] = min(potDiffAbs);
38
39      if potDiffAbs > corrThresh
40        continue
41      else
42        numberOfMatches = numberOfMatches + 1;
43      end
44
45      %   % the actual pot time
46      %   time_pot_min_diff = time_pot(idxDiff);
47
48      % save both indices
49      idxPot_temp(numberOfMatches) = idxDiff;
50
51      idxVI_temp(numberOfMatches) = loopCounter;
52
53      % save time-diff
54      timeDiffPot(numberOfMatches) = actDiff;
55
56    end
57    % get the number of pot-values to distribute for the
        current step
58    index_length_current = length(idxPot_temp);
59    % write information regarding indices in output file
60    par.aux.indices(current_pot_initial).idxPot = idxPot_temp;
61    par.aux.indices(current_pot_initial).idxVI = idxVI_temp;
62    par.aux.indices(current_pot_initial).idxLength =
        index_length_current;
63    par.aux.indices(current_pot_initial).timeDiff =
        timeDiffPot;
64
65    clear idxPot_temp
66    clear idxVI_temp
67    clear timeDiffPot
68  end
69  close(waitB)]
```

Quellcode 3.16: Neuer Ansatz zum Ermitteln der Indizes

Die überarbeitete Version der Index-Bestimmung ist in Quellcode 3.16 dargestellt. Dieser Ansatz verzichtet auf eine Rundung zugunsten einer absoluten Differenzberechnung. Vorteilhaft an diesem Vorgehen ist, dass auf jeden Fall die Potentiostat-Zeiten mit der geringsten Differenz zur VI-Zeit übernommen werden. Um zu verhindern, dass für alle VI-Zeiten eine Entsprechung im Potentiostat genutzt wird, gibt es eine Zusatzbedingung: Sie besagt, dass Werte nur als übereinstimmend gelten, wenn diese sich innerhalb einer halben Sekunde zueinander befinden. Dieser Threshold ist variabel

ausgelegt, sollte aber aufgrund der gegebenen Abtastraten vom Standardfall einer halben Sekunde nicht signifikant abweichen. Da immer nur eine konkrete VI-Zeit mit einer Potentiostat-Messdatei verglichen wird, verringert sich außerdem die Berechnungsdauer um ein Vielfaches. Zu beachten ist an dieser Stelle, dass es bei der Wahl eines ungünstigen Thresholds theoretisch zur Doppelbelegung eines Potentiostat-Indizes kommen kann. Das Zuordnen der Werte hingegen wurde nicht verändert und funktioniert genau so wie oben beschrieben. In Tabelle 3.4 ist an einem einfachen Beispiel das Verhalten dieser Funktion dargestellt.

Tabelle 3.4: Beispiel des Verhaltens bei der Index-Bestimmung mit dem neuen Ansatz

| Index | VI-Zeit | Pot-Zeit | VI-Index | Pot-Index |
|---|---|---|---|---|
| 1 | 20 | 21 | 2 | 1 |
| 2 | 21 | 22 | 3 | 2 |
| 3 | 22 | 22,1 | 4 | 13 |
| 4 | 23,2 | 22,2 | 5 | 14 |
| 5 | 24,2 | 22,3 | 6 | 15 |
| 6 | 25,2 | 22,4 | 7 | 16 |
| 7 | 26,4 | 22,5 | 8 | 18 |
| 8 | 27,4 | 22,6 | 9 | 20 |
| 9 | 28,4 | 22,7 | | |
| 10 | 29,4 | 22,8 | | |
| 11 | 30,6 | 22,9 | | |
| 12 | 31,8 | 23 | | |
| 13 | 33 | 23,1 | | |
| 14 | 34 | 24 | | |
| 15 | 35 | 25 | | |
| 16 | 36 | 26 | | |
| 17 | | 27 | | |
| 18 | | 27,5 | | |
| 19 | | 28 | | |
| 20 | | 28,5 | | |

Im Vergleich zum ersten Lösungsansatz ist in Tabelle 3.5 zu erkennen, dass bei diesem Vorgehen tatsächlich die Indizes des Potentiostat übernommen werden, die am nächsten an der VI-Zeit liegen.

Die neue Möglichkeit die übereinstimmenden Indizes zu ermitteln stützt sich auf zwei for-Schleifen. In der äußeren Schleife wird sichergestellt, dass jede Pontentiostat-Datei erfasst wird. In ihr wird jeweils die Start- und End-Zeit der aktuellen Potentiostat-Datei erfasst und auf die VI-Zeit gerundet. Die Indizes dieser Zeiten im VI-Timer dienen als Rahmenbedingung für die zweite for-Schleife. In ihr werden alle VI-Indizes vom Start der Potentiostat-

Tabelle 3.5: Vergleich zwischen den übernommenen Zeiten bei altem und neuem Ansatz

| | altes Vorgehen | neues Vorgehen |
|---|---|---|
| VI-Zeit | Pot-Zeit | Pot-Zeit |
| 21 | 21 | 21 |
| 22 | 22 | 22 |
| 23,2 | 22,6 | 23,1 |
| 24,2 | 24 | 24 |
| 25,2 | 25 | 25 |
| 26,4 | 26 | 26 |
| 27,4 | 27 | 27,5 |
| 28,4 | 28 | 28,5 |

Messung bis zum Ende evaluiert. Zuerst wird die Zeitdifferenz zwischen dem aktuellen VI-Index und jedem Zeitstempel der aktuellen Potentiostat-Messung als Betrag erfasst. Liegt der minimale Wert der Beträge über dem festgelegten Threshold von einer halben Sekunde, wird zum nächsten VI-Index gesprungen. Liegt der Betrag jedoch darunter, so wird der entsprechende Potentiostat-Index erfasst und gespeichert.

Die Korrektheit der Matrix-Erstellung lässt sich über die All-Vanadium-Messungen nachweisen. In diesen wurde die OCV mit dem VI und die Zellspannung mit dem Potentiostaten aufgenommen. Die Werte haben ein Spannungsoffset zueinander. Trotzdem müssen charakteristische Abfälle und Anstiege mit diesem Vorgehen an den gleichen Stellen erfolgen: Gleiche Stelle bedeutet in diesem Fall ein Abweichen von maximal einer Abtastrate des VIs. In Abbildung 3.16 ist der Spannungsabfall in einer Polarisationskurve als Beispiel dargestellt. Es lässt sich erkennen, wie jeweils der Wert übernommen wird, der den VI-Zeiten am nächsten liegt.

Abbildung 3.17 zeigt eine *current interrupt*-Messung der gleichen Polarisationskurve. Dieses Beispiel soll verdeutlichen, dasss bei erhöhter Abtastrate vom Potentiostat viele Werte wegfallen.

### 3.7.3 Erstellung von stepNew

Neben den beiden Schritt-Variablen *Schritt_ VI* und *Schritt_ pot*, die jeweils eine Variable nur für das entsprechende Gerät darstellen, hat es sich für die Auswertung der Messzyklen als nützlich erwiesen, eine globale Schritt-Variable einzuführen. Bisher wurde diese aufgrund der Zerstückelung der Dateien in einem komplizierten Verfahren zusammengesetzt. Mit der neuen Aufbereitung der kompletten Messdaten in einer Matrix, vereinfacht sich die Erstellung dieser Variable.

Die Unterfunktion *stepNewCreation* soll die beiden Schritt-Messkanäle

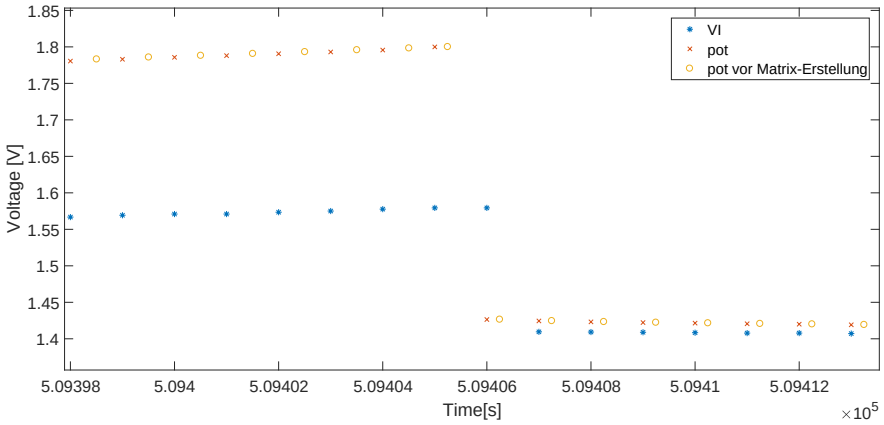


Abbildung 3.16: Spannungsabfall einer Polarisationskurve mit Spannung aus VI und Potentiostat

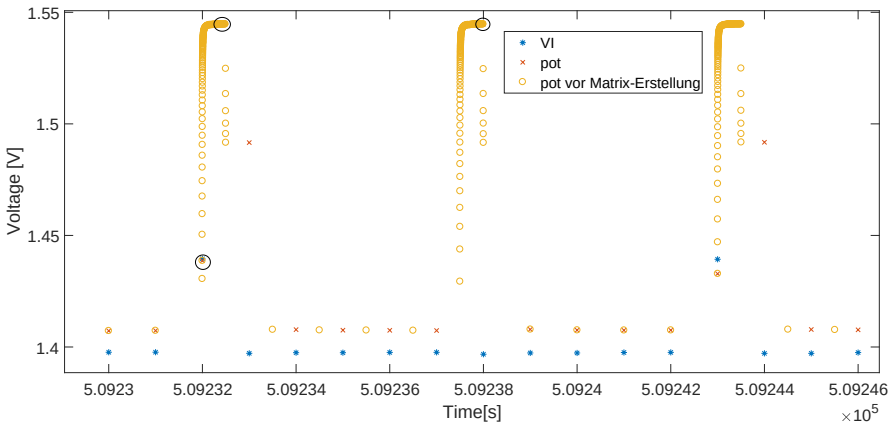


Abbildung 3.17: *current interrupt* mit Original- und übernommenen Zeiten

Index für Index auf Veränderung untersuchen. Wird bei einer Variable ein Wechsel festgestellt, so wird an dieser Stelle ein neuer *stepNew* gezählt. Für den Fall, dass beide Schritte an derselben Stelle einen Wechsel aufweisen, wird die *stepNew*-Variable logischerweise ebenfalls nur um 1 erhöht.

In einem ersten Schritt überprüft die Unterfunktion über eine weitere Unterfunktion *getChangeNaNCheck* an welchen Indizes die Änderungen in den beiden Schritt-Variablen auftreten und wie viele Änderungen es gibt. Die Funktion interpretiert zwei aufeinanderfolgende NaN nicht als Wechsel, eine Änderung von beispielsweise 3 auf NaN allerdings schon. Im Anschluss folgt die Vorbelegung von *stepNew* mit Nullen zu Geschwindigkeitszwecken. Eine for-Schleife zählt den Index der Matrix herunter. Ist der Index Teil der oben ermittelten Änderungen, wird vom aktuellen Index bis zum Ende der Matrix *stepNew* um Eins erhöht. Wie im gekürzten Quellcode 3.17 gezeigt, fängt die Funktion auch den Fall ab, dass sich die Schritte zeitgleich verändern.

```
1  [stepChange_pot, stepChangeCounter_pot] =
       getChangeNaNCheck_1_0(vp.Schritt_pot);
2  [stepChange_VI, stepChangeCounter_VI] =
       getChangeNaNCheck_1_0(vp.Schritt_VI);
3
4  newTimer_length = length(vp.Timer_s);
5  vp.stepNew = zeros(newTimer_length,1);
6
7  % top-down-for-loop
8  for timerIdx = newTimer_length:-1:1
9    % check if the index is part of the pot-stepChanges
10   if ismember(timerIdx, stepChange_pot)
11     % add a step
12     vp.stepNew(timerIdx:end) = vp.stepNew(timerIdx:end) + 1;
13     % move on to the next index
14     continue
15   end
16
17   % check if the index is part of the VI-stepchanges
18   if ismember(timerIdx, stepChange_VI)
19     % add a step
20     vp.stepNew(timerIdx:end) = vp.stepNew(timerIdx:end) + 1;
21   end
22 end
```

Quellcode 3.17: Erstellung von stepNew

In 3.6 wird die Logik zur Erstellung von stepNew an einem Beispiel verdeutlicht.

### 3.7.4 Erste Berechnungen

Als abschließende Operation des Einlesevorgangs werden noch einige Größen für die Auswertung vorberechnet. Der Timer wird von Sekunden für bessere

Tabelle 3.6: Beispiel zur stepNew-Erstellung

| Timer_s | Schritt_VI | Schritt_pot | stepNew |
|---:|---:|---:|---:|
| 1 | 1 | NaN | 0 |
| 2 | 1 | NaN | 0 |
| 3 | 1 | NaN | 0 |
| 4 | 1 | 1 | 1 |
| 5 | 1 | 1 | 1 |
| 6 | 2 | 2 | 2 |
| 7 | 2 | 2 | 2 |
| 8 | 2 | 2 | 2 |
| 9 | 2 | NaN | 3 |
| 10 | 3 | 1 | 4 |
| 11 | 3 | 2 | 5 |
| 12 | 3 | 3 | 6 |
| 13 | 3 | 4 | 7 |
| 14 | 3 | 4 | 7 |
| 15 | 3 | 4 | 7 |
| 16 | 4 | NaN | 8 |
| 17 | 4 | 1 | 9 |

Übersichtlichkeit auch in Minuten und Stunden umgerechnet. Es folgt die Auswahl der gewünschten Vorzeichenkonvention: Folgt man der Konvention, die sich in der Brennstoffzellentechnik durchgesetzt hat, bildet eine positive Stromstärke einen Entladevorgang ab. Nach der Eingabe der Membranfläche durch den Anwender wird die Stromdichte entsprechend der gewählten Vorzeichenkonvention berechnet. Zuletzt wird über den *cumtrapz*-Befehl, also über eine kumulative Anwendung der Trapezregel, das Integral der Stromstärke über die Zeit gebildet, also die Ladung. Dafür werden fehlende Werte im Stromstärke-Vektor im Rahmen dieser Funktion als Nullen interpretiert. An dieser Stelle ist jedoch Achtung geboten: Falls im Einlesevorgang nicht zu verarbeitende EIS- oder PEIS-Messungen übersprungen wurden, könnten diese einen Einfluss auf die tatsächliche Ladung haben. Da in dieser Operation aber rein über die vorhandenen Stromstärke integriert wird, kann dieser Einfluss nicht berücksichtigt werden.

Nach diesem Punkt ist eine Messung erfolgreich eingelesen. Die Daten liegen in Kanalform, korrekt umbenannt und alle auf einen gemeinsamen Timer bezogen in einer Matrix vor. Alle Operationen bis hierher müssen genau einmal korrekt ablaufen, danach kann mit der Datenmatrix ausgewertet werden. Natürlich wird an dieser Stelle noch ein letztes Mal abgespeichert, die Speicherfunktion erkennt außerdem automatisch, dass bereits die vp-Matrix existiert und speichert nur diese als separate Datei mit einer Endung *_vp*.

## 3.8 Auswerte-Software im Detail

Eine zweite Anforderung an die Software war es, dem Anwender einige einfache Auswerte-Tools zur Verfügung zu stellen. Das Programm soll die Möglichkeit geben, die Kanäle zu plotten, die Matrix zur einfacheren Detailauswertung zu schneiden und Datenpunkte auszuwählen. Die Daten sollen außerdem als Vektorsammlung abgespeichert werden können, um eine Weiterverarbeitung über die gängigen Matlab-Befehle zu ermöglichen.

Dafür muss der Anwender die abgespeicherte vp-Matrix über den Daten-Browser laden. Anschließend folgt der Nutzer den Aufforderungen im Kommandofenster von Matlab oder gegebenenfalls in den generierten Fenstern. Die folgenden Unterkapitel geben einen Überblick über die jeweiligen Funktionalitäten der Operationen.

### 3.8.1 Die Auswertungs-Schleife

Um dem Anwender eine möglichst große Freiheit darüber zu bieten, in welcher Reihenfolge er die verschiedenen Tools zur Auswertung bedient, wurden sämtliche Optionen in eine while-Schleife eingelesen. Der Anwender kann in einem Auswahldialog (vgl. Quellcode 3.18) die Operation, die er durchführen möchte, auswählen, die dann über eine *case*-Abfrage bearbeitet wird. Da das Schneiden von Messungen ein zentraler Bestandteil der Auswerteroutine

sein soll, sind die Optionen auch darauf abgestimmt, sowohl mit der vollen Matrix als auch einem Ausschnitt kompatibel zu sein.

```
 Momentan äausgewhlt: vp
Bitte Aktion äwhlen:
1: switch between vp and vp_cut
2: Preview-Plot
3: plot a single variable
4: plot a single variable with datapoint-selection
5: cut data
6: save vp_cut
7: save vp_cut als vp-matrix
8: save datapoints
```

Quellcode 3.18: Die Optionen der Auswertungs-Schleife

Um diese Anforderung zu erfüllen, hat der Anwender zwei Speicher-Slots. Der erste Slot ist für die komplette Matrix reserviert und wird hier als **vp** bezeichnet. Sollte ein Ausschnitt erfolgt sein, wird der zweite Slot belegt: **vp_cut**, dieser ist überschreibbar. Der Anwender kann zu jedem Zeitpunkt zwischen den beiden Slots wechseln. Die Zwischenspeicherung von Datenpunkten erfolgt ebenfalls über einen überschreibbaren Slot.

### 3.8.2 Der Preview-Plot

Der Preview-Plot ist vor Allem dazu gedacht, sich die komplette Matrix anzuschauen. Der Anwender wird gefragt, welche zwei Größen er abhängig von der Zeit geplottet haben möchte. Die Größen werden dann in einem vordefiniertem Fenster gezeichnet; das Programm wählt automatische Achsen-Skalierung. Als zusätzliche Übersicht werden sowohl die Schritte als auch die stepNew geplottet und deren Übergänge als gestrichelte Linien in die anderen Plots übertragen. Ein Beispiel ist in Abbildung 3.18 gezeigt.

### 3.8.3 Der Single-Plot

Der Anwender kann sich auch dazu entscheiden, eine einzelne Größe abhängig von der Zeit plotten zu lassen. Nach der Auswahl der Messgröße über eine Liste öffnet sich ein Fenster, in dem der Graph aufbereitet wird. Die Achsen werden automatisch skaliert und beschriftet.

Wird die Option ausgewählt, Datenpunkte zu selektieren, wird das Programm nach der Erstellung des Graphen pausiert. Der Anwender kann nun über den Datacursor-Mode in Matlab Datenpunkte auf dem Graphen selektieren. Dabei kann der Graph auch über die üblichen Werkzeuge im Plot-Fenster vergrößert oder verkleinert werden. Über die Eingabetaste wird die Auswahl bestätigt und die Koordinaten der Datenpunkte sowie deren Indizes in der Matrix zwischengespeichert. In Abbildung 3.19 sind beispielsweise zwei Extremwerte der Zellspannung einer Polarisationskurve selektiert. Ein

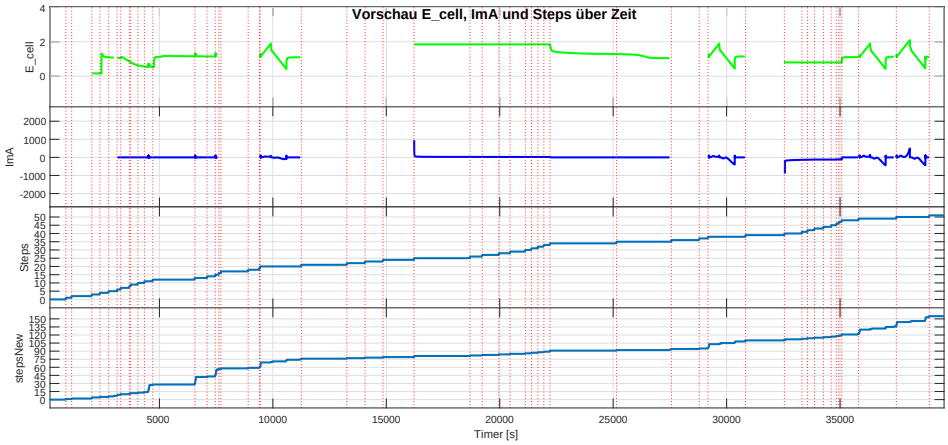


Abbildung 3.18: Der Preview-Plot

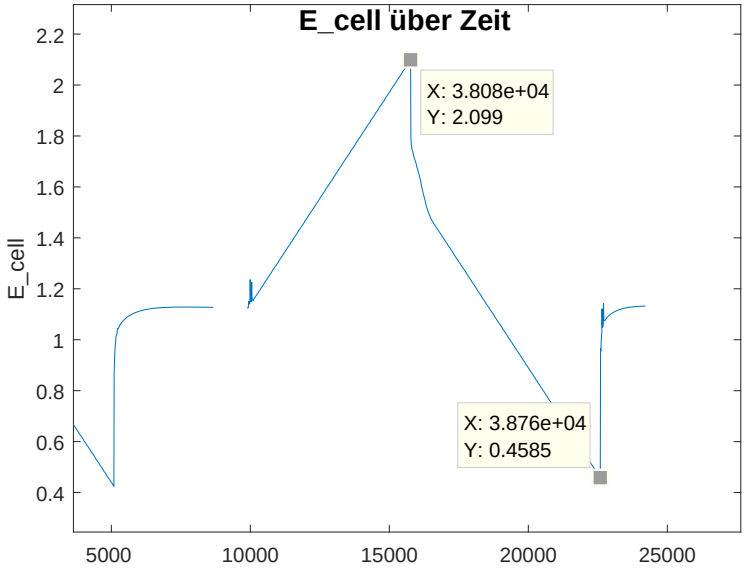


Abbildung 3.19: Single-Plot mit ausgewählten Datenpunkten

Problem der Bestätigung über die Eingabetaste war, dass das Programm den Tastendruck bereits als Eingabe für die Auswahl des nächsten Durchgangs interpretiert hat. Da jedoch keine Zahl in das Kommandofenster geschrieben worden war und das Programm diesen Fall nicht abdeckte, brach das Programm mit einem Fehler ab. Eine andere Taste als Bestätigung der Pause zu deklarieren war aufgrund der verschiedenen Shortcuts, die in einem Plot aktiv sind, ebenfalls nicht praktikabel. Nach der Datenpunkt-Auswahl wurde daher eine weitere Pause eingebaut. Diese reagiert nur auf die Betätigung der *rightArrow*-Taste. Danach kann der Anwender gewohnt den Auswertungs-Loop fortsetzen. Der zu Grunde liegende Quellcode ist in Quellcode 3.19 dargestellt und die Ausgabe im Kommandofenster in Quellcode 3.20.

```
while true
  w = waitforbuttonpress;
    switch w
      case 1 % (keyboard press)
        key = get(gcf,'currentcharacter');
        switch key
          case 29 % 29 is the rightArrow key
            disp('User pressed the rightArrow key. Quitting
                the loop.')
            break
          otherwise
            % Wait for a different command.
        end
    end
end
```

Quellcode 3.19: Zusatzpause

```
 Achtung, Testpause! Bitte rightArrow-Taste üdrcken!
User pressed the rightArrow key. Quitting the loop.

 Momentan ausgewaehlt: vp
Bitte Aktion waehlen:
1: switch between vp and vp_cut
2: Preview-Plot
3: plot a single variable
4: plot a single variable with datapoint-selection
5: cut data
6: save vp_cut
7: save vp_cut als vp-matrix
8: save datapoints
```

Quellcode 3.20: Command-Window nach Datenpunkt-Auswahl

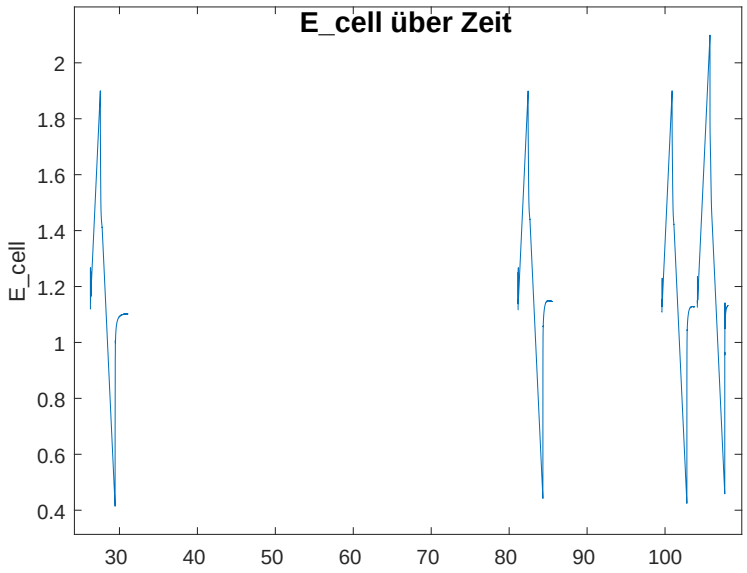


Abbildung 3.20: Ausschnitt anhand von Schritten

### 3.8.4 Schneiden der Messdaten

Wie bereits erwähnt, besteht die Auswertung oft daraus, charakteristische Messzyklen auszuwerten. Dabei ist es unnötig kompliziert über die komplette Matrix im Workspace zu verfügen. Eine zentrale Forderung an die Software bestand also darin, dem Benutzer eine Möglichkeit zur Verfügung zu stellen, aus der Matrix bestimmte Bereiche herauszuschneiden.

Die Unterfunktion dafür gibt dem Anwender drei verschiedene Arten an die Hand, nach denen er schneiden kann. Die erste Möglichkeit stützt sich auf die Schritt-Variable des VIs. In der Measurement-Routine lässt sich ablesen, in welchem Schritt welche Messprozedur gefahren wird. Möchte der Anwender also eine bestimmte Polarisationskurve ausschneiden, so gibt er in einer Auswahlliste an, welche Schritte er behalten möchte. Nach dem Schneiden besteht die Variable *vp_ cut* dann nur noch aus der erwähnten Polarisationskurve. Der Anwender kann sich auch dazu entscheiden, einfach einen bestimmten Zeitbereich auszuschneiden. Darauf baut auch die dritte Möglichkeit zu schneiden auf: Anhand von den nach der oben beschriebenen Methode ausgewählten Datenpunkten. In Abbildung 3.20 wurde ein Schnitt generiert, indem aus der Messung die vier Schritte ausgewählt wurden, die die Polarisationskurven beinhalten. Abbildung 3.21 wurde hingegen erzeugt, indem mit Hilfe der in Abbildung 3.19 ausgewählten Datenpunkte geschnitten wurde.

Unabhängig von der gewählten Methode besteht ebenfalls die Möglich-

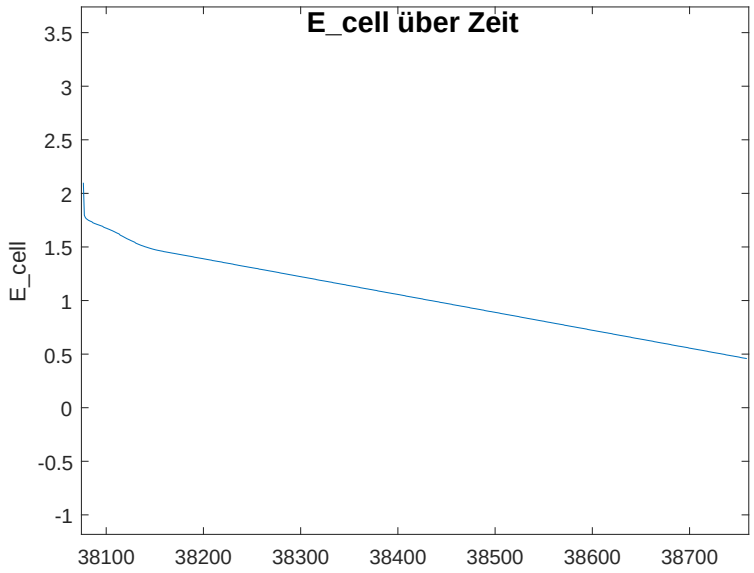


Abbildung 3.21: Ausschnitt anhand von Datenpunkten

keit die Größen auszuwählen, die behalten werden sollen. Interessieren den Anwender zum Beispiel nur die Zell- und gemessenen Überspannungen so selektiert er in der Auswahlliste nur diese.

Das tatsächliche Schneiden des vp-structs geschieht über die Funktion *subsetstruct*. Diese Funktion ist keine vorgegebene in Matlab sondern ist Teil einer Sammlung von Funktionen für Logistiker und wurde im Rahmen eines Kurses an der North Carolina State University erstellt und wurde in dieser Arbeit auch in keiner Weise verändert. [21] Die Funktion hat zwei Eingabeargumente: das zu bearbeitende struct und einen logischen Ausdruck, der definiert, was ausgeschnitten werden soll. Ein entsprechendes Minimalbeispiel, angelehnt am tatsächlichen Funktionsaufruf, findet sich in Quellcode 3.21.

```
1  vp.Schritt_VI = [1  1  1  2  2  2  3  3  3  4];
2  vp.einWert = [1  2  3  4  5  6  7  8  9  10];
3  stepsSelected = [1  3];
4  vp_cut = subsetstruct(vp, ismember(vp.Schritt_VI,
       stepsSelected));
5
6  vp_cut.Schritt_VI
7
8  ans =
9
10    1    1    1    3    3    3
```

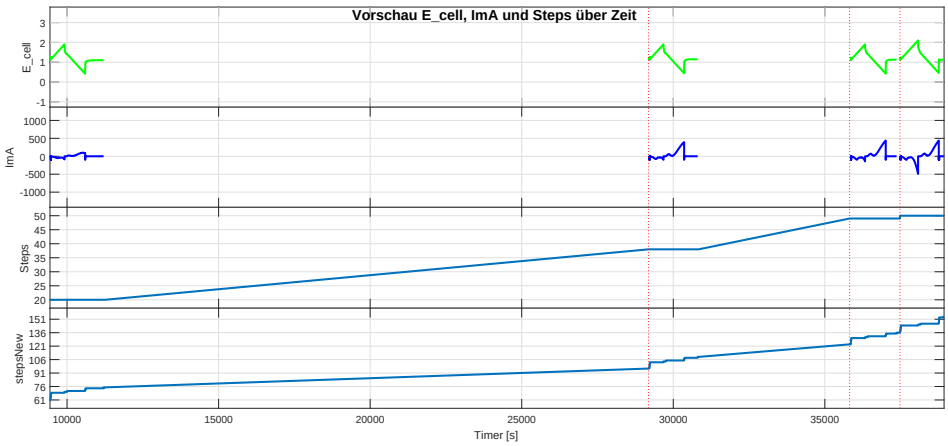


Abbildung 3.22: Vier Polarisationskurven im neuen Funktionsaufruf

```
11
12 vp_cut.einWert
13
14 ans =
15
16     1     2     3     7     8     9
```

Quellcode 3.21: Minimalbeispiel von subsetstruct

Je nachdem, ob nach Schritten oder Zeit geschnitten werden soll, ändert sich entsprechend der logische Ausdruck im Aufruf der Funktion. Das neue struct wird als *vp_cut* zwischengespeichert. Im Anschluss werden über den bereits beschriebenen *rmfield*-Befehl alle Felder entfernt, die der Anwender nicht behalten will.

### 3.8.5 Speichern der Ergebnisse

Die Schleife sieht drei verschiedene Optionen für das Speichern von Daten vor. Das geschnittene struct *vp_cut* lässt sich entweder erneut als *struct* abspeichern oder als simple Sammlung der enthaltenen Vektoren in einer *.mat*-Datei. Erstere Option benutzt die gleiche Speicherfunktion, die bereits an den verschiedenen Speicherpunkten in der *main*-Funktion benutzt wurde. Dies hat den Vorteil, das abgespeicherte *struct* erneut mit Hilfe der Auswertungs-Schleife untersuchen zu können, wie in Abbildung 3.22 die oben ausgeschnittenen vier Polarisationskurven. Einen flexibleren Ansatz bietet die Möglichkeit, den Ausschnitt als Vektorensammlung zu speichern. Auf diese Weise lässt sich der Ausschnitt vom Anwender über die gewohnten Matlab-Tools ohne Rücksicht auf eine spezielle Datenstruktur auswerten.

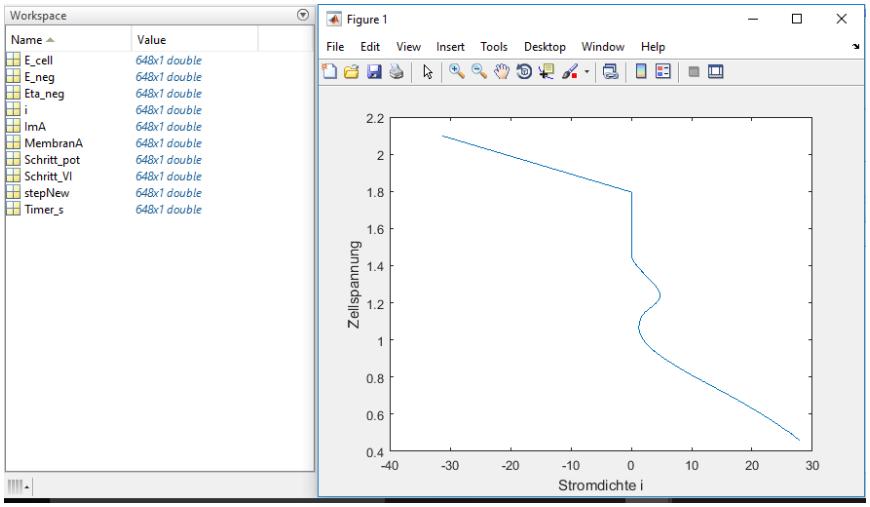


Abbildung 3.23: Anwendung der Vektorensammlung

Gerade für Messzyklen, die noch nicht standardisiert ausgewertet werden, stellt dies einen Vorteil dar. Der Aufruf dafür benutzt die *save*-Funktion in Matlab mit einem Eingabeargument, welches die struct-Ordnung der Variable aufhebt, dargestellt in Quellcode 3.22.

```
1 save(completeName, '-struct', 'vp_cut')
```

Quellcode 3.22: Speichern als Vektorensammlung

In Abbildung 3.23 wurde ein Plot anhand der in Abbildung 3.21 ausgeschnittenen Daten erzeugt. Die Abbildung des Workspace soll verdeutlichen, dass beim Schneiden nur bestimmte Variablen übernommen wurden.

Eine andere Möglichkeit ist das Abspeichern von ausgewählten Datenpunkten. Diese müssen zuerst, wie oben beschrieben, über die single-Plot-Funktion ausgewählt werden. Das Programm erstellt dann eine Kopie des momentan aktiven structs und speichert davon eine Kopie unter dem Namen *vp_ datapoints*. Aus diesem werden über *subsetstruct* alle Werte gelöscht, deren Index nicht dem eines der selektierten Datenpunkte entspricht. Da bei einigen wenigen Punkten eine Strukturierung über ein *structure array* sinnlos ist, werden die Punkte ebenfalls als Vektorschar abgespeichert, analog zu Quellcode 3.22. In Abbildung 3.24 wurden auf einer Polarisationskurve sechs Punkte markiert, abgespeichert und dann geplottet.

## 3.9 Ausblick

Dies fasst die aktuellen Möglichkeiten der Software zusammen. In diesem Kapitel sollen erste Verbesserungsvorschläge geliefert werden, die eine Weiterentwicklung der Software vereinfachen sollen.

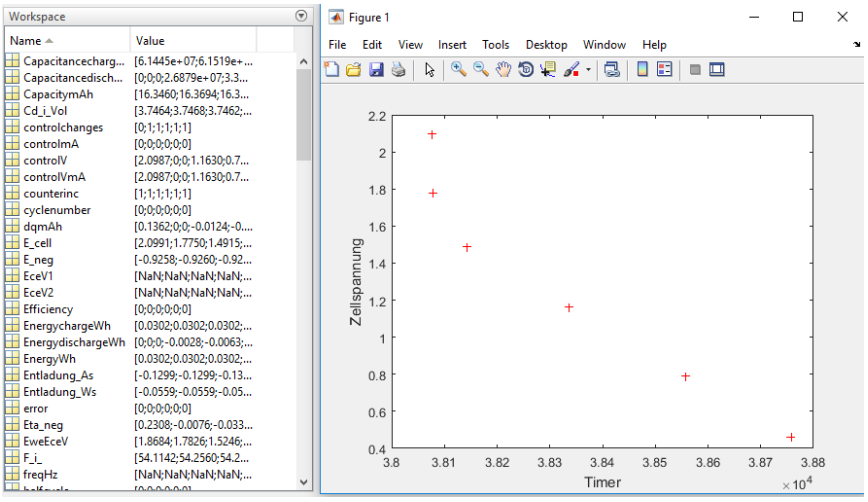


Abbildung 3.24: abgespeicherte Datenpunkte

Durch die Vielzahl an Personen, die bereits an der Erweiterung der Software gearbeitet haben, ist die Software stellenweise recht unübersichtlich. Um an einigen Stellen den Zeitaufwand für das Testen auf Kompatibilität zu verringern, wurde darauf verzichtet, aus Funktionen, die nur noch teilweise für die Funktionalität der Software essentiell sind, unnötige Teile herauszuschneiden.

Ebenfalls könnte ein Upsampling der VI-Daten auf das Potentiostat-Niveau in Erwägung gezogen werden. Über Interpolation lässt sich dieses Ziel erreichen. Das Streichen des Großteils der Potentiostat-Werte in *current interrupt*-Zyklen würde dementsprechend entfallen, allerdings steht auch diesem Ansatz die mangelhafte Genauigkeit des Matching anhand des Trigger-Signals im Weg.

Ein weiterer Schritt, der in größerer Benutzerfreundlichkeit resultieren würde, wäre das Integrieren der vorgestellten Auswertungs-Operationen in eine graphische Benutzeroberfläche, wie in Abbildung 3.25.

Die beiden Plot-Fenster könnten beliebige Kanäle nach Wahl des Users plotten. Über Buttons am unteren Ende der Oberfläche könnten bestimmte Operationen wie das Zuschneiden der Daten oder das interaktive Auswählen von Datenpunkten in den Graphen gesteuert werden.

Zusammenfassend ist zu sagen, dass ohne eine Verbesserung der Aufnahme des Startzeitpunktes des Potentiostaten die Genauigkeit der Matrix-Erstellung nicht weiter zu verbessern ist. Die Auswertungsmöglichkeiten dieser Matrix können jedoch in alle erdenklichen Richtungen erweitert werden. Eine naheliegende Erweiterung ist das Schreiben von automatisierten Auswertungen für bestimmte Messläufe. Wie eine solche Auswertung aussehen könnte, zeigt das nächste Kapitel. Polarisationskurven sind aber natürlich nur eine von vielen denkbaren Anwendungen.

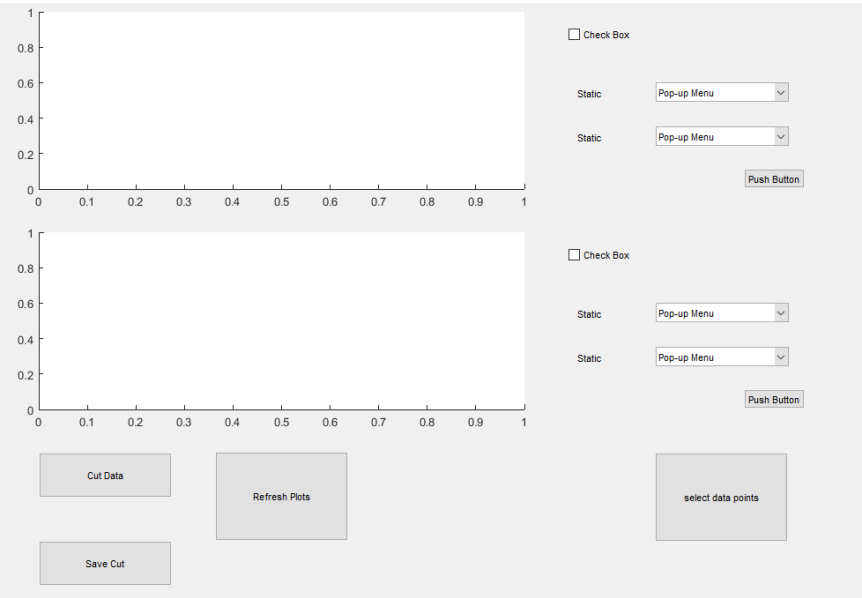


Abbildung 3.25: Beispielhafte graphische Benutzeroberfläche

# Kapitel 4

# Auswertung der Vanadium-Luft-Daten

Anhand der Diskussion von Polarisationskurven einer Vanadium-Luft-RFB soll an dieser Stelle gezeigt werden, dass die überarbeitete Software alle Voraussetzungen schafft, um auch detailliertere Auswertungen mit den Messdaten durchzuführen.

## 4.1 Daten-Voraussetzungen

Für die Auswertung einer Polarisationskurve bietet es sich an, den entsprechenden VI-Schritt zu isolieren, also über die Software auszuschneiden. Innerhalb der Polarisationskurve herrscht eine Eindeutigkeit der Variable für die Potentiostat-Schritte. So umfasst der Schritt 5 den kompletten Ladevorgang, während in Schritt 7 der Entladevorgang enthalten ist. Es wurde sich außerdem dazu entschieden, die *structure array* Struktur beizubehalten, um die beschriebene Unterfunktion *subsetstruct* verwenden zu können.

## 4.2 Betrachtung einer Polarisationskurve

An dieser Stelle soll eine einzelne Polarisationskurve betrachtet werden und deren Anteile im Detail aufgeführt werden. Die verwendete Messung ist eine Vanadium-Luft-Messung vom 01. Juni 2017 mit einer tubulären Zelle. Aus dieser Messung ist es die letzte Polarisationskurve, vergleiche auch [1]. Abbildung 4.1 zeigt die Polarisationskurve inklusive aller bestimmbaren Überspannungen.

Die Messung erfolgte bei einem SOC von $0,5$. Nach [17] entspricht das in dem untersuchten Versuchsaufbau grob einem Potential der NHZ von $E_{NHZ} = 0,95V$. Als Referenzelektrode wurde eine $Hg/Hg_2SO_4$ genutzt, die gegenüber einer SHE ein Potential von $657mV$ aufweist. Die Durchflussraten betragen $6,4ml/min$ für das Elektrolyt ($1,6M$ $V3,5+$, *GfE-Metalle*,

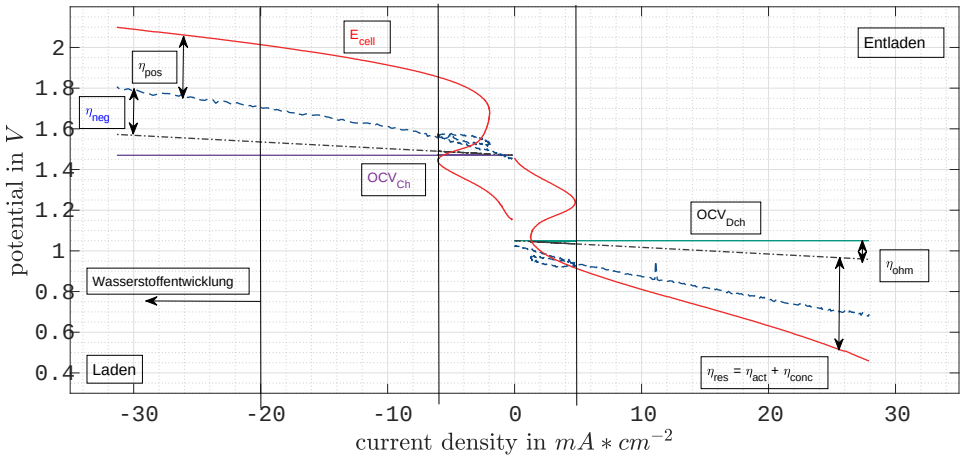


Abbildung 4.1: letzte Polarisationskurve der Vanadium-Luft-Messung vom 01.06.2017, vergleiche auch [1]

Germany) (genauere Angaben zur Formierung dieses Elektrolyts in [17]) und für den Sauerstoff $80ml/min$.

Für die Auswertung der Polarisationskurve wird von einem für die komplette Kurve konstantem flächenspezifischen Widerstand $ASR$ ausgegangen. Die aktive Membranoberfläche der Zelle beträgt $A = 15,7cm^2$. Der Widerstand wird punktweise in den *current interrupts* der Messung aufgenommen und kann über *EC-Lab* ausgelesen werden. Der mittlere ASR für die Messung errechnete sich zu $(3,27 \pm 0,09)\Omega cm^2$. Für die Graphen wird die Vorzeichenkonvention der Brennstoffzellentechnik verwendet, Entladungen sind also mit einer positiven Stromdichte versehen. Die maximale Stromdichte für das Beladen beläuft sich auf $i = -31mA/cm^2$ und auf $i = 28mA/cm^2$ für das Entladen.

### 4.2.1 Abweichung von $\eta_{neg}$

Die Masterprojekt-Arbeit von P.Kuhn [22] konnte einen systematischen Fehler bei der Aufnahme von $\eta_{neg}$ und $\eta_{pos}$ nachweisen. Um die für diese Messung gültige Abweichung von $\eta_{neg}$ zu bestimmen, wurde die Langzeit-OCV-Messung genauer untersucht. Die Überspannung der negativen Halbzelle sollte an dieser Stelle 0 betragen, weicht jedoch ab. Der offensichtliche abweichende Messpunkt in Abbildung 4.2 wird durch eine lineare Interpolation anhand der beiden benachbarten Punkte ersetzt, resultierend in Abbildung 4.3. Matlab berechnet einen Mittelwert von $-0,004V$. Für die Polarisationskurven-Auswertung wird $\eta_{neg}$ also um $+0,004V$ korrigiert. $\eta_{pos}$ wurde nicht direkt aufgezeichnet, sondern wird über die in Kapitel 2 angegebenen Zusammenhänge ermittelt.

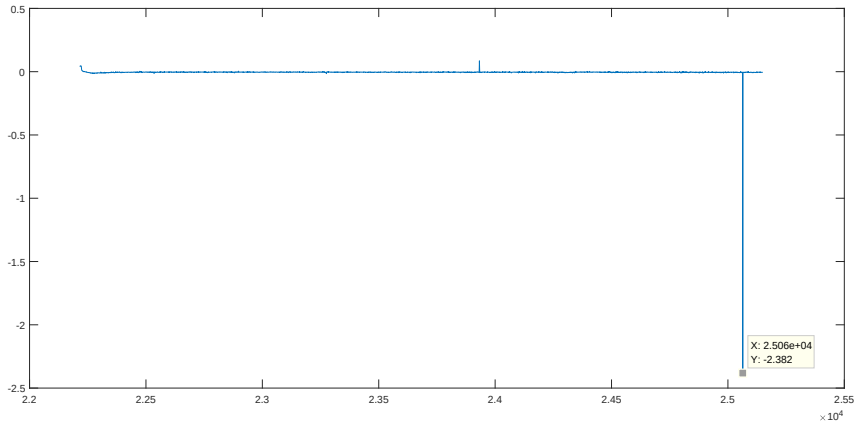


Abbildung 4.2: $\eta_{neg}$ in der Langzeit-OCV mit fehlerhaftem Punkt

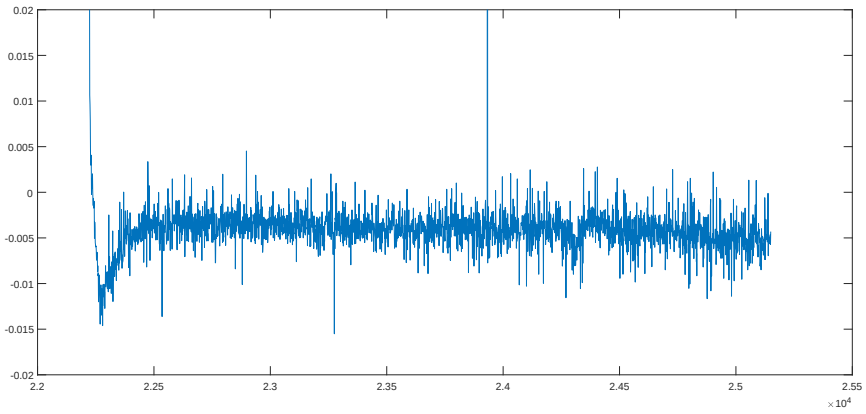


Abbildung 4.3: $\eta_{neg}$ in der Langzeit-OCV bereinigt

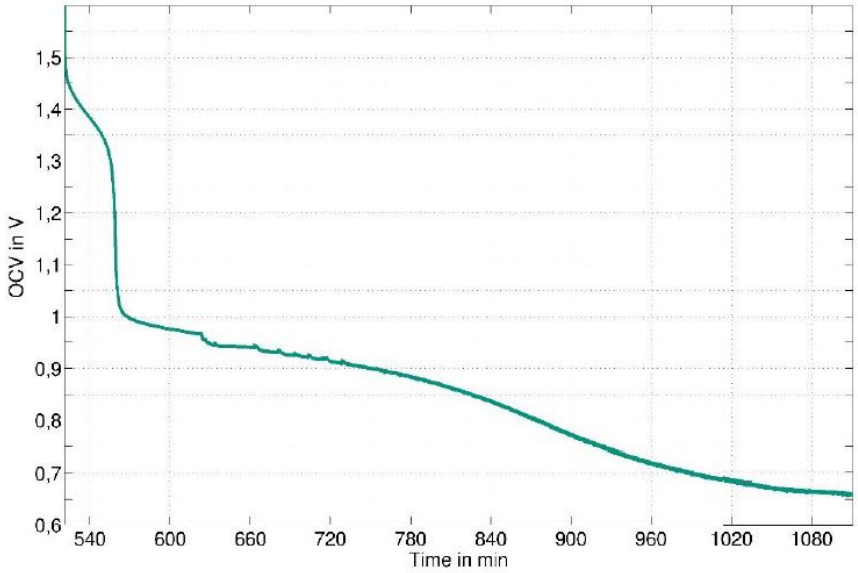


Abbildung 4.4: OCV-Verhalten bei Vanadium-Luft-RFBs [1]

### 4.2.2 OCV

Die OCV zeigt bei der Vanadium-Luft-Paarung ein auffälliges Verhalten, zu sehen in Abbildung 4.4. Die OCV für einen bestimmten SOC bleibt nicht konstant über die Zeit, sondern fällt nach dem gezeigten Muster ab. Dieses Besonderheit ist schon länger bekannt, zum Beispiel nach Ressel [1] oder Hosseiny [12]. Allerdings ist das Verhalten bisher noch nicht vollständig begründet. Schon nach geringer Zeit ist ein Abfall von grob einem halben Volt zu verzeichnen.

Für das Entladen und Laden der VARFB wird daher jeweils eine eigene OCV bestimmt. Für die betrachtete Polarisationskurve wurde die OCV unter Berücksichtigung von Abbildung 4.4 beim Laden zu $1.47V$ und beim Entladen zu $1,05V$ bestimmt.

### 4.2.3 Überspannungen

Die Polarisationskurve gibt die Möglichkeit, in Abhängigkeit der aktuellen Stromdichte die dominanten Überspannungsquellen zu bestimmten. Für All-Vanadium-RFBs teilen sich die dominanten Überspannungen wie in Abbildung 2.2 gezeigt auf. Der weitaus größte Teil der Stromdichten wird von ohmschen Verlusten dominiert, für kleine Stromdichten sind die Aktivierungs-Überspannungen bestimmend und bei sehr großen Stromdichten treten sehr große Konzentrations-Überspannungen auf.

Statt Aktivierungs- und Transport-Überspannungen wurden in dieser Messung die Überspannungen der PHZ und der NHZ gemessen. Diese wür-

den sich theoretisch wieder in Aktivierungs- und Transport-Überspannungen aufteilen lassen, dafür liegen jedoch keine Messwerte vor. Trotzdem lassen sich aus den Graphen interessante Rückschlüsse ziehen.

Im betrachteten Bereich für das Laden der Batterie zeigt sich über die gesamte Stromdichten-Breite ein dominantes Verhalten der Verluste, die durch die positive Halbzelle erzeugt werden. Die Überspannungen der NHZ sind im Mittel etwa halb so groß, während die ohmschen Überspannungen klein sind. $\eta_{ohm}$ weist sogar mit steigender Stromdichte einen noch geringeren Einfluss auf die Zellspannung auf, während der Einfluss der NHZ steigt. Der Einfluss von $\eta_{pos}$ ändert sich im betrachteten Bereich kaum. Zu beachten ist außerdem, dass mit wachsender Stromdichte beim Laden eine Wasserstoffbildung einsetzen sollte, deren Einfluss allerdings nicht so stark ausfiel, wie erwartet.

Ein anderes Verhalten zeigt die Entlade-Seite. Hier stellen die Überspannungen der NHZ den dominierenden Anteil an den Verlusten, gefolgt von $\eta_{pos}$. Die ohm'schen Verluste stellen auch hier den geringsten Anteil. Bei steigenden Stromdichten steigt der Einfluss der beiden Halbzellen noch weiter an. Ließen sich noch höhere Entlade-Stromdichten untersuchen könnte $\eta_{pos}$ der dominante Verlust werden.

Im Gegensatz zu den Polarisationskurven der All-Vanadium-RFBs entwickelt sich diese Polarisationskurve nicht annähernd symmetrisch um eine Stromdichte von 0. Die Verluste beim Entladen der Batterie sind wesentlich geringer als beim Laden.

### 4.2.4 Vergleich mit anderen Veröffentlichungen

Für eine Einordnung der Zahlenwerte bietet sich ein Vergleich zu ähnlichen Versuchen in der Literatur an. Hierbei interessieren in absteigender Wichtigkeit andere VARFB, Vanadium-Sauerstoff-Brennstoffzellen und All-Vanadium-RFB.

#### Vergleich mit einer anderen VARFB: Austing

Die Veröffentlichung von Austing [4] behandelt ebenfalls eine VARFB. Die Zelle von Austing bietet einen interessanten Aufbau mit einer zweilagigen Kathode. Die aktive Membranoberfläche der planaren Zelle beträgt $A = 4cm^2$. In einer ersten Messung stellt Austing die Zellspannung über der Kapazität mit Stromdichten von 15, 20 und $40mAcm^{-2}$ bei Raumtemperatur dar. Verwendet wurde ein Anolyt mit $1,2M\ V^{3+}$. Die Veröffentlichung führt an einer Stelle aus, dass die Ladevorgänge bis zu einem SOC von 93% ausgeführt wurden. Nimmt man den Wert bei einer Kapazität von 0 als OCV für den SOC von 0% an, so lassen sich grob folgende Werte ablesen: $1,6V$ für das Beladen und knapp unter $1V$ für das Entladen. Während die Entlade-OCV mit $0,1V$ Differenz leicht niedriger liegt, ist die Belade-OCV von Austing knapp $0,15V$ höher. Man beachte jedoch, dass die Werte der HAW bei ei-

nem SOC von 50% aufgenommen wurden. Austing zeigt außerdem für einen Lade-/Entladevorgang bei $40 mA/cm^2$ die Zellspannung über der Zeit. Betrachtet man jeweils den Wert auf halber Strecke des Lade- und Entladevorgangs, so kann man von einem SOC von grob 50% ausgehen. Ein Vergleich der Zellspannungen findet sich in Tabelle 4.1.

Tabelle 4.1: Vergleich von OCV bei SOC=0,5 (HAW) und etwa 0 (Austing) und Zellspannung mit Austing [4]

| | | OCV [V] | i [mA/cm$^2$] | Zellspannung [V] |
|---|---|---|---|---|
| HAW | Charge | 1,47 | 31 | 2,1 |
| | Discharge | 1,05 | 28 | 0,45 |
| Austing | Charge | 1,65 | 40 | 1,95 |
| | Discharge | 0,95 | 40 | 0,85 |

Bei $10 mA/cm^2$ Differenz weist die HAW-Batterie eine leicht höheres Lade-Zellspannung auf, bei einer deutlich geringerer Entlade-Zellspannung. Zu einzelnen Überspannungsanteilen macht der Autor leider keine genauen Angaben.

### Vergleich mit Vanadium-Sauerstoff-Brennstoffzellen: Noack und Menictas

Vanadium-Sauerstoff-Brennstoffzellen benutzen die gleichen Elektrolyte wie die VARFB, sehen jedoch - wie der Name bereits vermuten lässt - keinen Beladebetrieb vor. Für einen Vergleich der Leistungsfähigkeit bei der Entladung ist ein Vergleich jedoch trotzdem interessant.

In der Veröffentlichung von Menictas und Skyllas-Kazacos [5] wurden Entladungen mit $50 ml$ $1,8 M$ $V2+$ durchgeführt. Für die Vergleichbarkeit der beschriebenen OCV-Werte wird sich auf die Einzelzelle statt dem ebenfalls beschriebenen Zellen-Stack beschränkt. Da keine genauen Angaben zum SOC beschrieben sind, wird von einem Start-SOC von knapp unter 100% ausgegangen.

Es wird eine OCV von $1,33 V$, beziehungsweise $1,41 V$ mit frischem Elektrolyt angegeben. Dieser Wert liegt recht deutlich über dem OCV-Wert von $1,05$ bei der HAW-VARFB im Entlade-Betrieb bei einem SOC von 50%. Die Brennstoffzelle von Menictas wurde außerdem mit einer maximalen Entlade-Stromdichte von $40,5 mA/cm^2$ getestet und liegt damit etwas mehr als $10 mA/cm^2$ über dem Maximum der HAW-RFB. In Tabelle 4.2 sind Entlade-Zellspannungen bei verschiedenen Stromdichten und Raumtemperatur gegenübergestellt.

Die Zellspannungen der VARFB und der Brennstoffzelle liegen bei Stromdichten bis $20 mA/cm^2$ dicht beieinander, danach liegen die Werte der Brenn-

Tabelle 4.2: Vergleich von Zellspannungen mit Menictas [5]

| Stromdichte [mA/cm$^2$] | Zellspannung Menictas [V] | Zellspannung HAW [V] |
|---:|---:|---:|
| 5,4 | 0,842 | 0,902 |
| 10,1 | 0,763 | 0,792 |
| 20,8 | 0,613 | 0,617 |
| 27,9 | | 0,459 |
| 30,3 | 0,526 | |
| 40,5 | 0,452 | |

stoffzelle höher. Beide Energieträger decken jedoch einen ähnlichen Wertebereich an Spannungen ab. Allgemein ist aus diesen Werten keine signifikant schlechtere Leistungsfähigkeit der VARFB im Entlade-Betrieb gegenüber der Brennstoffzelle nachzuweisen, es sollten jedoch die leicht unterschiedlichen und im Falle des SOC sogar unbekannten Betriebsparameter bei dem Vergleich beachtet werden.

Die Veröffentlichung von Noack et al. [6] beschreibt ebenfalls eine Vanadium-Sauerstoff-Brennstoffzelle. Als Elektrolyt wurde $1,6M\ V^{2+}$ verwendet, das ebenfalls aus dem Elektrolyt von GfE-Metalle formiert wurde, welches auch in der HAW-RFB Verwendung findet. Die $300s$ lange OCV-Messung von Noack ergibt im Mittel ungefähr einen Wert von $1,25V$, allerdings bei einem angegebenen SOC von knapp 100%. Damit liegt sie über den $1,05V$ der HAW-VARFB bei einem SOC von 50% im Entlade-Betrieb. Die Brennstoffzelle von Noack erreicht allerdings mit Werten von knapp $60mA/cm^2$ eine doppelt so hohe Entlade-Stromdichte.

Tabelle 4.3: Vergleich von Zellspannungen mit Noack [6]

| Stromdichte [mA/cm$^2$] | Zellspannung HAW [V] | Zellspannung Noack 2nd Batch [V] | Zellspannung Noack 47th Batch [V] |
|---:|---:|---:|---:|
| 5,4 | 0,902 | 0,900 | 1,100 |
| 10,1 | 0,792 | 0,800 | 0,950 |
| 20,8 | 0,617 | 0,600 | 0,750 |
| 27,9 | 0,459 | 0,450 | 0,600 |
| 30,3 | | 0,400 | 0,550 |

In Tabelle 4.3 sind Zellspannungen bei verschiedenen Entlade-Stromdichten mit zwei Batches der Brennstoffzelle von Noack verglichen. Die Unterschiede

zwischen den verschiedenen Batches werden in [6] im Detail erläutert. Trotz des SOC von knapp 100% bei Noack sind die Zellspannungen der HAW-Batterie fast identisch mit dem zweiten Batch von Noack und im Mittel etwa $0,15V$ kleiner als die Ergebnisse des späten Noack-Batch. Tabelle 4.4 zeigt bei den selben Stromdichten die Spannungen der negativen Halbzelle. Bei allen drei untersuchten Messungen ist die Anodenspannung relativ konstant über die verschiedenen Stromdichten, die Spannungen von Noack sind jedoch ein wenig höher als die der HAW-Zelle.

Tabelle 4.4: Vergleich von Anodenspannungen mit Noack [6]

| Stromdichte [mA/cm$^2$] | Anodenspannung HAW [V] | Anodenspannung Noack 2nd Batch [V] | Anodenspannung Noack 47th Batch [V] |
|---:|---:|---:|---:|
| 5,4 | -0,268 | -0,320 | -0,380 |
| 10,1 | -0,268 | -0,300 | -0,380 |
| 20,8 | -0,266 | -0,290 | -0,360 |
| 27,9 | -0,265 | -0,290 | -0,350 |
| 30,3 | | -0,280 | -0,350 |

### Vergleich mit All-Vanadium-RFB

Für den Vergleich mit einer All-Vanadium-RFB kann auf die Messungen der HAW zurückgegriffen werden. In Abbildung 4.5 ist eine Polarisationskurve aus der Messung einer tubulären All-Vanadium-Zelle vom 23. November 2016 nach Lade-/Entladezyklen inklusive aller mitgemessenen Überspannungsanteilen abgebildet. In der Masterprojekt-Arbeit von Kuhn [22] wurde ebenfalls schon mit dieser Messung gearbeitet. Sie ist sehr gut mit der Vanadium-Luft-Messung vergleichbar. Es wurde das gleiche Elektrolyt verwendet, das gleiche Zelldesign und es wurde auf den selben SOC geladen.

Als erstes fällt die konstante OCV ins Auge, die bei der All-Vanadium-Messung mit $1,37V$ etwas unterhalb der Lade-OCV der Vanadium-Luft-Messung liegt, aber weit über der Entlade-OCV von $1,05V$. In Tabelle 4.5 sind Zellspannungen der beiden Versuche gegenübergestellt. Im direkten Vergleich zeigt sich, dass die Vanadium-Luft-Paarung beim Beladen erkennbar höhere Zellspannungen, dafür aber im Entlade-Betrieb ebenso erkennbar niedrigere Zellspannungen erreicht.

In der All-Vanadium-Messung wurde genau wie in der Vanadium-Luft-Messung eine Flow-By-Zelle mit acht Kanälen benutzt. Die unterschiedlichen Bauweisen werden von Ressel in [16] näher erläutert. Im Nachhinein wurde jedoch festgestellt, dass in der All-Vanadium-Messung von diesen acht Kanälen sieben Stück teilweise verstopft waren. Daraus ergeben sich die sehr hohen

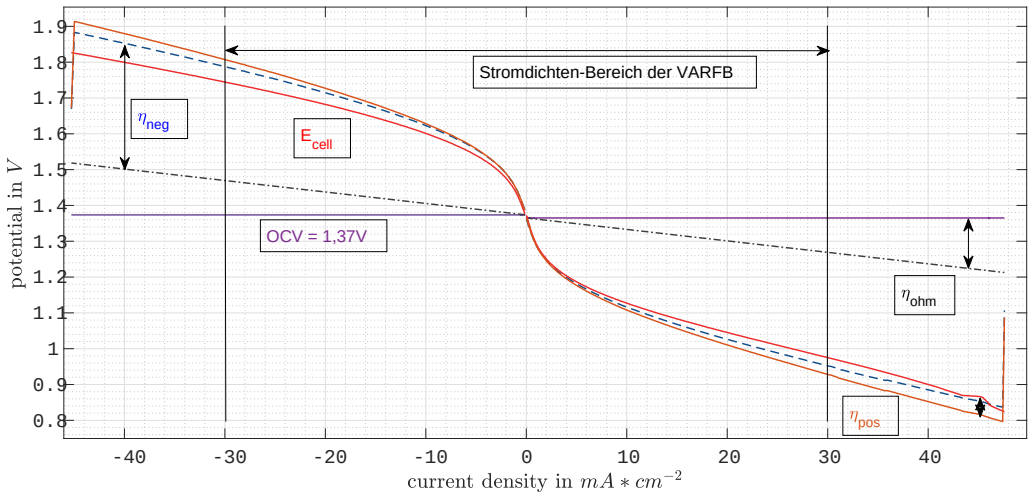


Abbildung 4.5: All-Vanadium RFB vom 23.11.2016 inklusive Zell- und Überspannungen

Tabelle 4.5: Vergleich mit All-Vanadium-RFB der HAW; VL = Vanadium-Luft, VV = All-Vanadium

| | Zellspannung [V] | |
|---|---|---|
| i [mA/cm$^2$] | VL | VV |
| -30,000 | 2,089 | 1,745 |
| -20,000 | 2,014 | 1,682 |
| -10,000 | 1,914 | 1,602 |
| 5,000 | 0,910 | 1,187 |
| 10,000 | 0,810 | 1,128 |
| 20,000 | 0,632 | 1,046 |
| 27,900 | 0,459 | 0,989 |

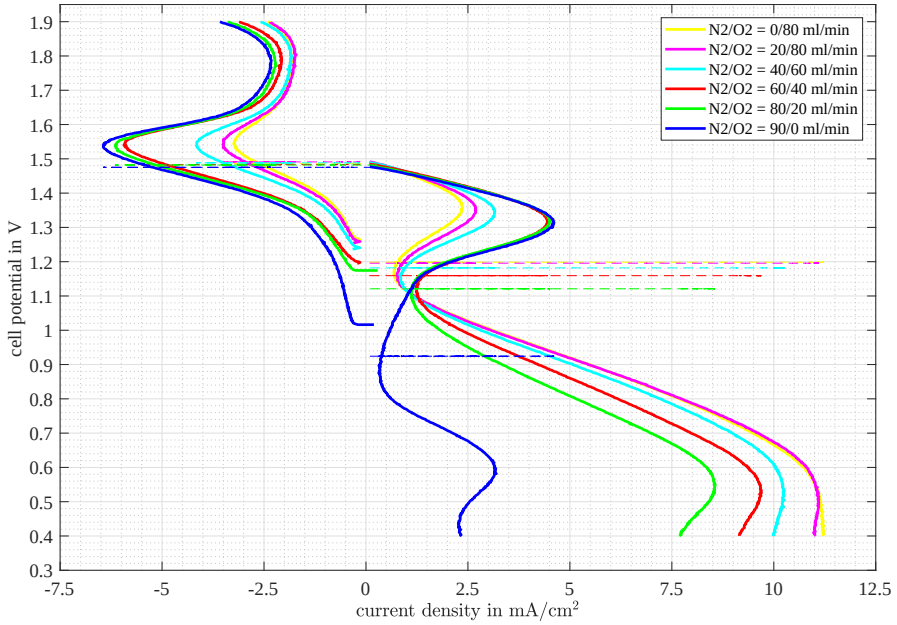


Abbildung 4.6: Polarisationskurven-Schar der Vanadium-Luft-Messung vom 08.06.2017

Überspannungen der negativen Halbzelle. Bei einer Entlade-Stromdichte von $i = 20mA/cm^2$ weist sie eine Überspannung von $\eta_{neg} = 0,274V$ auf. Sie ist damit $50mV$ höher als bei der Vanadium-Luft-Messung ($\eta_{neg} = 0,222V$).

## 4.3 Vergleich mehrerer Polarisationskurven der selben Messung: Einfluss von Stickstoff auf die Luft-Seite

Als nächstes sollen mehrere Polarisationskurven mit unterschiedlichen Parametern verglichen werden. Dafür wird eine andere Messung verwendet, nämlich die vom 08. Juni 2017. In dieser Messung wurden Polarisationskurven mit verschiedenen Verhältnissen von Stickstoff und Sauerstoff in der positiven Halbzelle aufgenommen. Ansonsten stimmen die Betriebsparameter mit der weiter oben beschriebenen Messung vom 01. Juni überein. Aus dieser Messung wurde in Abbildung 4.6 für jedes aufgenommene Verhältnis eine Polarisationskurve aufbereitet. Der jeweilige OCV-Wert ist als gestrichelte Linie ebenfalls abgebildet.

Das Verhalten beim Laden ist aus diesem Graphen leider kaum abzulesen, da keine ausreichend hohen Stromdichten erreicht wurden, um die Ablese-

methoden aus den Polarisationskurven anwenden zu können. Zu sehen ist außerdem, dass mit höherem Stickstoff-Anteil auch der für Vanadium-Luft-Paarungen charakteristische Knick in der Stromdichte wächst, in diesem Fall also der Bereich bis $1,8V$. Tritt die maximale Ausdehnung des Knicks bei hohen Sauerstoff-Anteilen noch bei Stromdichten kleiner $5mA/cm^2$ auf, so erhöht sich dieser Maximalausschlag bei höheren Stickstoffanteilen auf fast $7mA/cm^2$.

Für das Entladen hingegen lässt sich ein Zusammenhang ablesen, lediglich die Kurve mit purem Stickstoff fällt erwartungsgemäß aus der Reihe. Alle Kurven mit einem Anteil Sauerstoff weisen für Zellspannungen unterhalb ihrer OCV die charakteristische Polarisationskurven-Form auf: erst ein exponentieller Anteil, dann ein großer Bereich mit einem linearen Zusammenhang gefolgt von einem abschließenden exponentiellen Anteil. Je größer der Stickstoff-Anteil ist, desto geringer ist der sich einstellende Grenzstrom; damit schrumpft auch der lineare Anteil der Polarisationskurve. Genau dieser lineare Anteil ist es auch, der bei reinem Stickstoff fast komplett fehlt. Theoretisch dürfte es bei reinem Stickstoff zu gar keiner Reaktion kommen, da kein Sauerstoff für die ORR vorhanden ist. Durch den komplexen Aufbau der Batterie ist es allerdings denkbar, dass an einer unbekannten Stelle in Wasser gelöster Sauerstoff eine minimale Reaktion ermöglicht.

## 4.4 Schlüsse aus der Betrachtung

Aus dem Vergleich der Daten ist abzulesen, dass die untersuchte VARFB mit einer Fremd-VARFB, diversen Vanadium-Sauerstoff-Brennstoffzellen und der hauseigenen All-Vanadium-RFB durchaus vergleichbar ist. Erfreulich ist vor Allem, dass die zusätzliche Funktion des Beladens des Elektrolyts gegenüber den Brennstoffzellen für vergleichsweise wenig Leistungseinbußen im Entlade-Betrieb sorgt. Zudem konnte der Einfluss von Stickstoff auf die Zellspannung in der Luft-Halbzelle dargestellt werden. Der Einfluss entspricht im auswertbaren Entlade-Betrieb den Erwartungen.

# Kapitel 5

# Fazit und Ausblick

Die Aufgabenstellung konnte in allen Punkten erfolgreich bearbeitet werden.

**Implementierung der Software:** Es ist eine funktionierende Software für die Auswertung eines Redox-Flow-Batterie-Prüfstandes erstellt worden. Die Software erfüllt die an sie gestellten Anforderungen, die Daten aus den beiden verschiedenen Messgeräten Potentiostat und VI einlesen zu können. Über eine externe Steuerdatei können die Kanäle umbenannt werden. Außerdem erfolgt ein Matching der Dateien auf einen gemeinsamen Zeitvektor anhand eines Trigger-Signales und das Erstellen einer Matrix. Die Matrix umfasst den kompletten Zeitvektor des VIs und jeder Zeit werden die am nächsten gelegenen Werte aus den Potentiostat-Kanälen zugeordnet. Im Anschluss sind außerdem einfache Auswertungs-Operationen mit der Software möglich, die vor allem dazu dienen, erste Einblicke über die Daten zu erlangen und durch Zuschneiden eine Weiterverarbeitung möglich machen. Die Software wurde bereits im Realbetrieb getestet ohne Fehler festzustellen. Die Software ist außerdem so weit es geht modular aufgebaut um eine einfache Weiterentwicklung zu ermöglichen.

**Dokumentation der Software:** Die Dokumentation der Software ist über diese Bachelorarbeit erfolgt. Es wurde sichergestellt, dass am Anfang der Beschreibung von Unterfunktionen der Software die nötigen Eingaben, die der Anwender zu tätigen hat, kurz beschrieben wurden. Für eine mögliche Weiterentwicklung der Software wurden die Funktionsweisen der einzelnen Programmpunkte im Detail beschrieben und anhand von einfachen Beispielen verdeutlicht. Außerdem wurde darauf geachtet, den gesamten Quellcode mit hilfreichen Kommentaren zu versehen.

**Beispielhafte Auswertung anhand von Polarisationskurven:** Eine beispielhafte Weiterverarbeitung wurde anhand von Messdaten einer Vanadium-Luft-RFB gezeigt. Die ausgewerteten Daten entsprechen

den Erwartungen und sind mit der Literatur vergleichbar. Positiv aufgefallen ist, dass die VARFB mit der tubulären Zelle in ihrer Leistungsfähigkeit nur wenig hinter reinen Brennstoffzellen zurückbleibt, die ebenfalls eine Vanadium-Sauerstoff-Paarung benutzen. Außerdem konnte der Zusammenhang von Stickstoff auf die Zellspannung und die zu erreichenden Stromdichten gezeigt werden. Die Weiterverarbeitung zeigt, dass die neue Software eine gute Grundlage bietet, um auch eine detaillierte Auswertung der Messdaten zu ermöglichen.

Die Zeitersparnis, die sich über die Matrix-Erstellung für die Auswertung ergibt ist nicht zu verachten. Eine sinnvolle nächste Erweiterung der Software stellt das Einfügen von detaillierten Auswertemechanismen dar, die sich jeweils auf einen bestimmten Messzyklus beziehen. Bei der Implementierung solcher Auswertemechanismen ist jedoch darauf zu achten, das Verhältnis aus Aufwand für die Programmierung und Zeitersparnis und Nutzen im Auge zu behalten. Ebenfalls sinnvoll wäre das Einführen einer graphischen Benutzeroberfläche für das gesamte Programm.

# Literaturverzeichnis


[1] S. Ressel, A. Chica, T. Flower, and T. Struckmann, "A tubular cell design for redox flow batteries," in *GDCh Wifo Session: Elektrochemische Energiespeicher und Wandler*, Sept. Sept. 2017, Berlin.

[2] D. Aaron, Z. Tang, A. B. Papandrew, and T. A. Zawodzinski, "Polarization curve analysis of all-vanadium redox flow batteries," *Journal of Applied Electrochemistry*, vol. 41, pp. 1175–1182, aug 2011.

[3] S. Ressel, P. Kuhn, F. Bill, A. von Stryk, T. Flower, and T. Struckmann, "Status report haw hamburg," in *tubulAir± Projekttreffen Frankfurt*, HAW Hamburg, HAW Hamburg, 19.-20. July 2017.

[4] J. grosse Austing, C. N. Kirchner, E.-M. Hammer, L. Komsiyska, and G. Wittstock, "Study of an unitised bidirectional vanadium/air redox flow battery comprising a two-layered cathode," *Journal of Power Sources*, vol. 273, pp. 1163–1170, jan 2015.

[5] C. Menictas and M. Skyllas-Kazacos, "Performance of vanadium-oxygen redox fuel cell," *Journal of Applied Electrochemistry*, vol. 41, pp. 1223–1232, sep 2011.

[6] J. Noack, G. Cognard, M. Oral, M. Küttinger, N. Roznyatovskaya, K. Pinkwart, and J. Tübke, "Study of the long-term operation of a vanadium/oxygen fuel cell," *Journal of Power Sources*, vol. 326, pp. 137–145, sep 2016.

[7] P. Alotto, M. Guarnieri, and F. Moro, "Redox flow batteries for the storage of renewable energy: A review," *Renewable and Sustainable Energy Reviews*, vol. 29, pp. 325–335, jan 2014.

[8] C. Blanc and A. Rufer, "Understanding the vanadium redox flow batteries," in *Paths to Sustainable Energy*, InTech, nov 2010.

[9] "tubulair± projektziele." http://www.tubulair.de/verbundprojekt/projektziele/. Accessed: 2017-10-10.

[10] P. Kurzweil and P. Scheipers, *Chemie: Grundlagen, Aufbauwissen, Anwendungen und Experimente (German Edition)*. Vieweg+Teubner Verlag, 2012.

[11] V. M. Schmidt, *Elektrochemische Verfahrenstechnik*. Wiley-VCH Verlag GmbH & Co. KGaA, oct 2003.

[12] S. Hosseiny, M. Saakes, and M. Wessling, "A polyelectrolyte membrane-based vanadium/air redox flow battery," *Electrochemistry Communications*, vol. 13, pp. 751–754, aug 2011.

[13] W. Vielstich and W. Schmickler, *Elektrochemie II*. Steinkopff, 1976.

[14] D. Chen, S. Wang, M. Xiao, and Y. Meng, "Preparation and properties of sulfonated poly(fluorenyl ether ketone) membrane for vanadium redox flow battery application," *Journal of Power Sources*, vol. 195, pp. 2089–2095, apr 2010.

[15] L. Arenas, C. P. de León, and F. Walsh, "Engineering aspects of the design, construction and performance of modular redox flow batteries for energy storage," *Journal of Energy Storage*, vol. 11, pp. 119–153, jun 2017.

[16] S. Ressel, A. Laube, S. Fischer, A. Chica, T. Flower, and T. Struckmann, "Performance of a vanadium redox flow battery with tubular cell design," *Journal of Power Sources*, vol. 355, pp. 199–205, jul 2017.

[17] S. Ressel, F. Bill, L. Holtz, N. Janshen, A. Choca, T. Flower, C. Weidlich, and T. Struckmann, "State of charge monitoring of vanadium redox flow batteres using half cell potentials and electrolyte density." Preprint submitted to Journal of Power Sources, Oct. 2017.

[18] L. Holtz, "Konzeptionierung und erstellung eines programms zur auswertung von messdaten eines redox-flow-battery-prüfstandes," Master's thesis, HAW Hamburg, 2016. Studienarbeit Bachelor.

[19] "Matlab dokumentation." https://de.mathworks.com/help/matlab/.

[20] "roundtowardvec-funktion." https://de.mathworks.com/matlabcentral/fileexchange/37674-round-toward-vector-of-values. Accessed: 2017-10-24.

[21] "matlog-funktionssammlung." http://www4.ncsu.edu/ kay/matlog/. Accessed: 2017-10-22.

[22] P. Kuhn, "Validierung einer messmethodik zur potentialbasierten verlustanalyse von vrfb-zellen," Master's thesis, Hochschule für Angewandte Wissenschaften Hamburg, 2017. Masterprojekt-Arbeit.

Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

## Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

*Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI*

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

**Erklärung zur selbstständigen Bearbeitung der Arbeit**

Hiermit versichere ich,

Name: Brandes

Vorname: Fabian

dass ich die vorliegende Bachelorarbeit bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Implementierung einer Auswertungs-Software für einen Redox-Flow-Batterie-Prüfstand

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

*- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -*

Die Kennzeichnung der von mir erstellten und verantworteten Teile der -bitte auswählen- ist erfolgt durch:

| Hamburg | 09.11.2017 | |
|---|---|---|
| Ort | Datum | Unterschrift im Original |