

**Hochschule für Technik und Wirtschaft des Saarlandes**

**Betreuer:** Prof. Dr. Robert Lemor (HTW)

**Betreuer:** Dr. med. Rainer Hanselmann

**Semester:** Sommersemester 2023

**Veranstaltung:** Praxisphase

# Praxisbericht

**Verfasst von:**

Philipp Bellia

**Anschrift:** Eleonorenstraße 4, 66119, Saarbrücken

**Studiengang:** Biomedizinische Technik

**Matrikelnummer:** 3810321

## Inhaltsverzeichnis

Einführung .....	1
1. Ibeth Biermann Cancer Research und die Kibero GmbH .....	2
2. Praktische Aufgabenstellung .....	3
2.1 Einarbeitung in die Materie .....	3
2.1.1 Die Aufnahme von Daten.....	3
2.1.2 Der Datensatz.....	3
2.2 Eigenes Arbeiten mit den Daten.....	4
2.2.1 Modul zum Einladen der Daten .....	4
2.2.2 Daten weiterverarbeiten .....	5
2.2.3 Daten darstellen .....	7
2.2.4 Ansätze zum Verbessern der Auflösung .....	9
2.3 Erstellen eines Visualisierungswerkzeuges .....	12
2.3.1 Einarbeitung in Qt für Python.....	12
2.3.2 Erstellen einer eigenen Software zur Visualisierung der Daten .....	13
2.3.3 Visualisierungstool.....	14
3. Fazit.....	18
3.1 Probleme.....	18
3.2 Kritische Analyse.....	19
3.3 Lösungsansätze .....	19
3.4 Abschließend .....	19
Erklärung der Praktikumstelle .....	20
Abbildungsverzeichnis .....	21

## Einführung

Die praktische Studienphase erstreckt sich über einen Zeitraum von drei Monaten und wird anschließend durch die Bachelorthesis ergänzt. Sie dient als Vorbau für die Thesis und ermöglicht, ein Grundgerüst an Wissen zu schaffen, welches zum Absolvieren der Bachelorphase benötigt wird. Das Wissen, welches hierbei hilfreich sein wird, ist der Aufbau von Software und die Wartung und Erweiterung dieser.

Die praktische Phase wird in der IB Cancer Research Stiftung in Saarbrücken durchgeführt. Die Stiftung befindet sich auf dem Saarbrücker Campus im Science Park 2.

Die Praxisphase setzt das Ziel einen Überblick über die Gerätschaften und Daten zu schaffen und wie man mit diesen umzugehen hat. Zudem sollen nach der Praktischen Phase erste Strukturen vorhanden sein, die als Grundgerüst für die Bachelorthesis dienen.

## 1. Ibeth Biermann Cancer Research und die Kibero GmbH

Die IBCR wurde 2018 von der Stifterin Ibeth Biermann gegründet mit dem Hintergrund Ursachen und Therapien für etliche Krebserkrankungen zu untersuchen. Zudem ist die Stiftung eine freie und unabhängige Stiftung, die von Wissenschaftlern geleitet wird, um bestmögliche Forschungsbedingungen zu gewährleisten.

Wissenschaftliche Themengebiete werden mit modernen molekularbiologischen, zellbiologischen und mikroskopischen Techniken untersucht.

Die Räumlichkeiten umfassen mehrere biologische Labore, Büroräume und ein Labor für die Mikroskopie.

Aktuell sind neun wissenschaftliche Mitarbeiter in der Stiftung beschäftigt. Darunter auch Leiter der Stiftung Dr. Rainer Hanselmann.

Die Kibero GmbH kooperiert mit der Stiftung und stellt akustische Mikroskope zur Verfügung, um die Untersuchungen und die Forschung zu unterstützen. Leiter der Kibero GmbH ist Prof. Dr. Robert Lemor.

Die aktuelle Arbeit der Stiftung beschäftigt sich mit der Rolle der RNA in dynamischen Zellsystemen, die, so lautet die Vermutung, eine entscheidende Rolle spielt in Bezug auf dynamischen Umstrukturierungen der Zelle. In Tumorzellen sorgen nun solche Eigenschaften zu dynamischeren Tumorzellen.

Die Aufgaben, die während der Praxisphase anfallen werden, sind hauptsächlich im Mikroskopie-Labor und im Büro der Kibero GmbH zu bearbeiten. Im Mittelpunkt steht das neue Konzipieren einer Oberfläche für die Betrachtung von Zellen mit einem akustischen Mikroskop.

Unter anderem soll an Algorithmen gearbeitet werden, die synthetisch für eine Auflösungsverbesserung sorgen sollen.

## 2. Praktische Aufgabenstellung

Die praktische Aufgabestellung befasst sich mit dem Bereich der Softwareentwicklung und zielt darauf ab, Wissen zu erlangen wie man bei der Softwareentwicklung vorgeht und welche Schritte in diesem spezifischen Fall, vorgenommen werden müssen. Ziel ist es den Datensatz einzulesen und die Daten zu visualisieren.

### 2.1 Einarbeitung in die Materie

Zu Beginn muss das Problem angegangen werden, dass vorliegende Format zu decodieren und korrekt einzulesen. Hierzu muss bekannt sein, wie der Prozess einer Aufnahme mit einem akustischem Mikroskops abläuft und in welchem Format es abgespeichert wird.

#### 2.1.1 Die Aufnahme von Daten

Die Daten werden mithilfe eines akustischen Mikroskops aufgenommen. Gesteuert wird das System von Computer aus, auf dem das Programm für die Bedienung läuft. Bevor eine Aufnahme aufgenommen werden kann, muss das System fokussiert werden. Hierbei ist wichtig zu beachten, dass man die Grenzen um den Fokusausschlag richtig platziert.

#### 2.1.2 Der Datensatz

Der Datensatz, der von dem akustischen System mittels LabView erfasst wird, hat das sogenannte „easySAM“ – Format und hat die Dateiendung „\*.esm“. Der Datensatz wird binär auf folgende Weise abgespeichert:

1. Die ersten Byte der Datei sind die Headerinformationen, die Auskunft über bestimmte Systemrelevante Parameter liefern.
2. Folgend enthält der Datensatz einen Snapshot der Aufnahme, welcher aber in x- sowie y-Richtung gespiegelt ist.
3. Nach dem Snapshot folgen die Messdaten. Zu beachten ist aber, dass Snapshot und Messdaten von einem Trennzeichen getrennt werden und jedes Signal auch mit einem Trennzeichen von dem folgenden Signal getrennt wird. Dies muss bei dem Auslesen der Daten beachtet werden.

Anzumerken ist, dass die Binärdatei noch im Hexaformat codiert ist. Um nun die Datei auslesen zu können erfordert es noch ein Modul in Python, welches ermöglicht, Hex Code zu decodieren. Mit der Installation von Python installiert man automatisch die Bibliothek „struct“. Diese bietet eine Vielzahl von Funktionen, um das Problem zu lösen. Die Funktion, die in diesem Fall gebraucht wird, ist die Funktion „struct.unpack“.

#### *Der Header*

Der Header der Datei ist in mehrere Parameter unterteilt. Darunter physikalische aber auch Systemparameter. Das Schema des Headers sieht folgendermaßen aus:

### Schema

1. Samplerate [Hz] (als Double [64-bit real])
2. Delay [Samples] (als Long [32-bit Integer])
3. Stepsize X [mm] (als Double [64-bit real])
4. Stepsize Y [mm] (als Double [64-bit real])
5. Z Offset (als Double [64-bit real])
6. Number Pixel X (als Long [32-bit Integer])
7. Samples (als Long [32-bit Integer])
8. Number Pixel Y (als Long [32-bit Integer])
9. System ID (als String mit 4 Byte für die Längenangabe)
10. Lense ID (als String mit 4 Byte für die Längenangabe)
11. Datum (2 \* Double [64-bit real] für Uhrzeit und Datum)
12. X Ofset [mm] (als Double [64-bit real])
13. Y Ofset [mm] (als Double [64-bit real])
14. Scanmode (als String mit 4 Byte für die Längenangabe)
15. Excitation (als String mit 4 Byte für die Längenangabe)
16. Gain (als Double [64-bit real])
17. Input Range (als Double [64-bit real])
18. Averaging (als Long [32-bit Integer])
19. ADQ-Serial (als String mit 4 Byte für die Längenangabe)
20. Stepsize Z [mm] (als Double [64-bit real])
21. Number Pixel Z (als Long [32-bit Integer])

In dieser Reihenfolge werden die Informationen im Header zusammengetragen.

## 2.2 Eigenes Arbeiten mit den Daten

Da die Struktur der Daten nun bekannt ist, kann damit begonnen werden, die Daten einzulesen und so zu sortieren, dass man sie so einfach wie möglich darstellen und Elemente der Daten selektieren kann.

### 2.2.1 Modul zum Einladen der Daten

Der erste Schritt, mit den Daten zu arbeiten, ist es die Daten erst einmal einzulesen. Wie bereits erwähnt ist hier Vorsicht geboten, denn die Funktion ``esm-einlesen`` liest die Datei Schritt für Schritt ein.

Zuerst wird der Header ausgelesen. Hierbei werden immer genau so viele Bytes eingelesen, die benötigt werden, die entsprechende Variable zu speichern:

- **Double:** 8 Byte
- **Long:** 4 Bytes

- **String:** Für jedes Zeichen ein Byte und 4 Bytes, die die Länge des Strings speichern

Man liest den Header nach dem oben beschriebenen Schema ein. Der entsprechende Code ist im Anhang zu finden. Sobald eine Variable vollständig decodiert wurde, wird Sie in das, im Code zu findende, Dictionary gespeichert.

Die Größe des Snapshots ergibt sich aus Pixel in X-Richtung multipliziert mit Pixel in Y-Richtung. Jeder Pixel wird als Double gespeichert, braucht also 8 Byte Speicherplatz. Ist der Snapshot also 100 mal 100 Pixel groß, so hat das Bild insgesamt 10000 Pixel und somit braucht es einen Speicherplatz von 80000 Byte.

Wie bereits erwähnt, gibt es anscheinend ein großes oder mehrere Trennzeichen zwischen Snapshot und den Messdaten. Beachtet man dies nicht, so kommt es zu einer fehlerhaften Messreihe. Auch die Signale innerhalb der Messreihe sind von einem Trennzeichen getrennt. Auch dies muss beachtet werden.

Vorgehen hier ist nun das erste Trennzeichen mit der Größe von 16 Byte auszulesen und weg zu schmeißen. Danach wird mit einer While-Schleife der restliche Datensatz ausgelesen. Nach jedem Signal, welches die Länge „Samples“ besitzt, dessen Wert im Header zu finden ist, folgt ein Trennzeichen von vier Byte. Auch diese Trennzeichen wird immer weggeschmissen. Nachdem die Messdaten alle eingelesen wurden, liegen sie in einem falschen Format vor, welches noch angepasst werden muss. Mit zwei For-Schleifen wird durch die Liste der Messdaten durch iteriert um anschließend ein Format zu erhalten, welches die dreidimensionalen Daten im Format (z, y, x) enthält.

Hier werden auch Funktionen aus der „Numpy“ Bibliothek verwendet.

- `numpy.array`
- `numpy.reshape`

Ist dieser Schritt beendet, gibt die Funktion ein Tupel mit der Größe von drei Elementen aus, die man bei dem Aufrufen alle definieren muss.

### 2.2.2 Daten weiterverarbeiten

Damit man die Daten einfacher und nicht mehrfach im Programm auftreten, wird ein Visualisierungsdatensatz erstellt, der im gleichen Zug interpoliert wird und in der Z-Ebene reduziert, um Berechnungen zu beschleunigen.

Auch wird ein Datensatz erstellt, der komplett mit der Hilbert Transformation transformiert wird.

Der Visualisierungsdatensatz wird abhängig vom Originaldatensatz berechnet. Hat der Originalbereich, der betrachtet wird, eine kleinere Auflösung als 256 mal 256, so wird die

Auflösung des Originaldatensatz auf 256 mal 256 interpoliert. Sollte er eine höhere Auflösung besitzen, so wird er auf 512 mal 512 interpoliert.

Die Z-Ebene wird halbiert. Um das Downsampling durchzuführen, wird eine Funktion aus dem Python Package „Scikit Image“ importiert. Die Funktion heißt „blockreduce“ und kann einen dreidimensionalen Datensatz in allen drei Dimensionen reduzieren. Hier wird nur die Z-Ebene reduziert.



### 2.2.3 Daten darstellen

Nun sind die Daten eingelesen. Das Erste, was mit den Daten gemacht wurde, war Sie visuell darstellen zu lassen. Aus dem Datensatz können folgende Darstellungen errechnet werden:

- C-Scan: eine Draufsicht auf die Probe
- B-Scan: ein Schnittbild der Probe entlang beider Achsen
- A-Scan: ein Diagramm des Signals in der Z-Ebene, sprich das akustische Signal an der Stelle XY.

#### C-Scan

Der C-Scan ist eine Darstellung, die in etwa genauso aussieht, wie der Snapshot, nur, dass der Snapshot in beiden Achsen gespiegelt ist. Errechnet wird dieser, indem alle Werte an Punkt XY in der Z-Ebene addiert werden, so dass ein Wert für die Stelle überbleibt.

Zu der normalen Darstellung kommt noch die Gate Variante. Gate bedeutet, dass man einen Bereich in Z angibt, den man darstellen lassen möchte. Man grenzt die Daten ein. Dies macht man, um sich einzelne Schichten darstellen zu lassen. Die Berechnung für die Gate-Darstellung ist die gleiche, nur nicht mit allen Daten, sondern mit den Daten in dem gewollten Intervall.

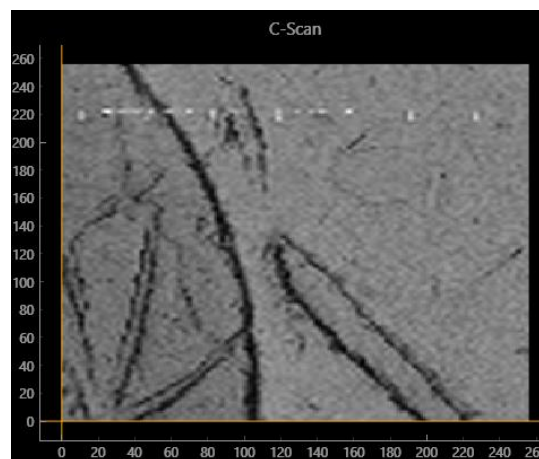


Abbildung 1: C-Scan einer 1 Cent Münze

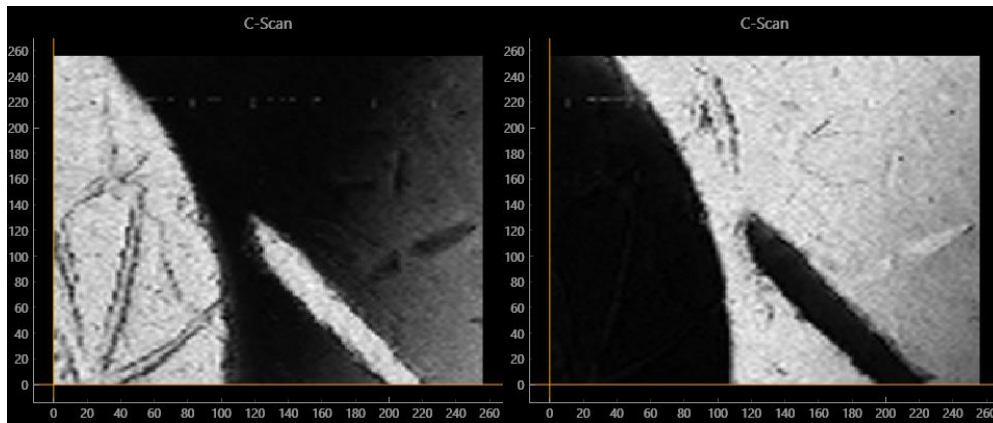


Abbildung 2: C-Scan der gleichen Münze mit links Gate 0-50 Ebenen und recht Gate 50-100 Ebenen

### B-Scan

Der B-Scan ist ein Schnittbild durch die Z-Ebene an einer bestimmten Stelle im Datensatz. Diese Stelle kann nur auf einer der Achsen liegen. Entweder der Wert liegt auf der X oder auf der Y-Achse.

Errechnet wird das resultierende Bild durch eine bestimmte Selektion von Werten aus dem Datensatz. Es muss nichts miteinander verrechnet werden, wenn man einen B-Scan darstellen will.

Der B-Scan ist auch im klinischen Ultraschall vorhanden. Hier nennt man ihn den B-Mode oder Brightness Mode.

Ein Beispiel sieht so aus:

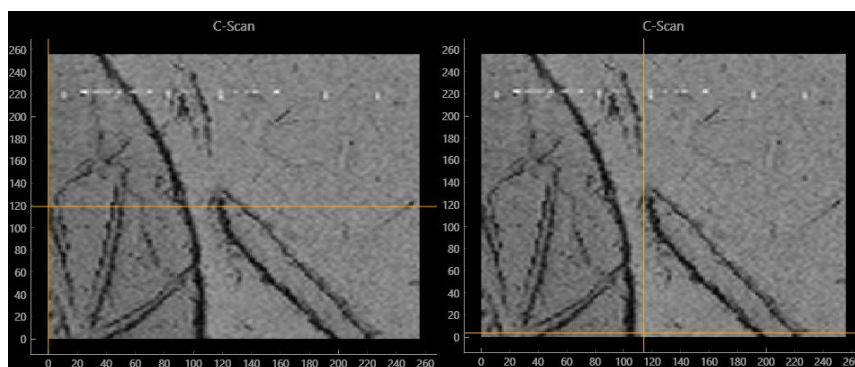


Abbildung 3: Darstellung an welcher Position der jeweilige B-Scan dargestellt wird. Links:  $y=120$  Rechts:  $x=120$

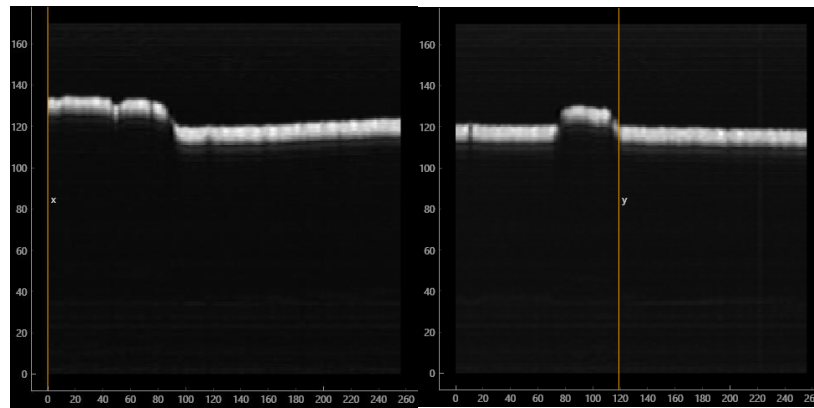


Abbildung 4: Die zu Abbildung 3 passenden B-Scan Bilder

### A-Scan

Der A-Scan ist eine einfache Darstellung des akustischen Signals an einer bestimmten Stelle des Datensatz. Auch hier benötigt man keine Berechnung, sondern nur eine Extraktion der benötigten Daten. Die Daten, die hierfür benötigt werden, sind an einer Stelle XY alle Z-Werte.

Die Darstellung erfolgt durch einen einfachen Graph:

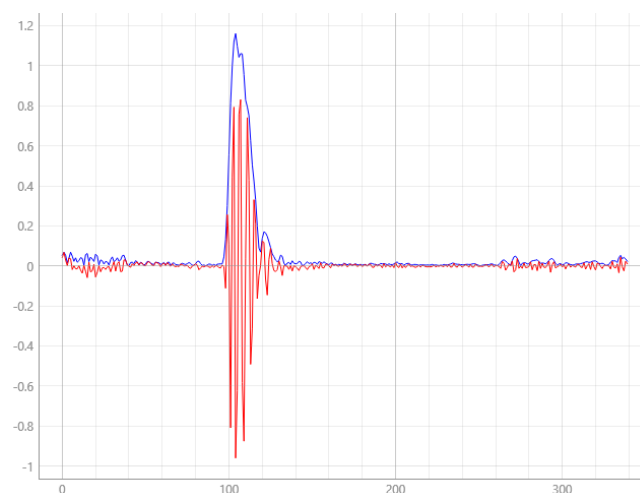


Abbildung 5: A-Scan an einer Stelle XY

Das rote Signal ist das Original aufgenommene Signal. Die blaue Kurve, die zu sehen ist, die Hilbert transformierte des Originalsignals. Die Hilbert Transformation liefert eine einhüllende des Signals, das transformiert wird.

### 2.2.4 Ansätze zum Verbessern der Auflösung

Um die Auflösung der Visualisierungen zu verbessern, gibt es einen Ansatz, der in der Geografie und auch schon im Ultraschall Anwendung findet. Dieser Ansatz nennt sich synthetische Apertur.

Das Verfahren der synthetischen Apertur ermöglicht eine präzisere Bestimmung dessen, was, nach dem Beschallen einer Oberfläche, zu sehen ist. Hierbei spielen Stellen, die nicht im Fokus liegen eine große Rolle. Im Fall der akustischen Mikroskopie, ist der Schall so gebündelt, dass es einen Fokusbereich gibt, der eine hohe Genauigkeit aufweist. Doch außerhalb dieses Fokusbereichs ist diese Genauigkeit nicht gegeben.

An diesem Punkt kann mit der synthetischen Apertur gearbeitet werden. Hierbei stehen vor allem die Punkte im Vordergrund, die nicht im Fokusbereich liegen.

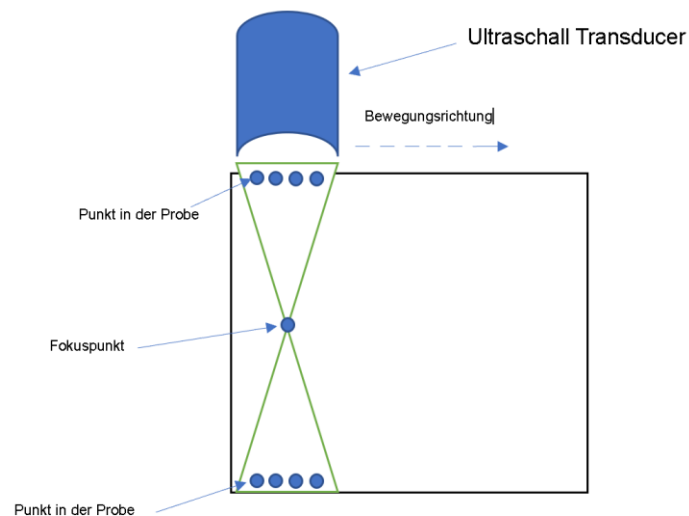


Abbildung 6: Skizze der synthetischen Apertur

In Abbildung 6 ist zu sehen, dass der Schall aus dem Ultraschall Transducer im Fokus gebündelt wird aber nach dem Fokus wieder auseinander geht. Die Punkte vor dem Fokus und nach dem Fokus können nun genauer bestimmt werden.

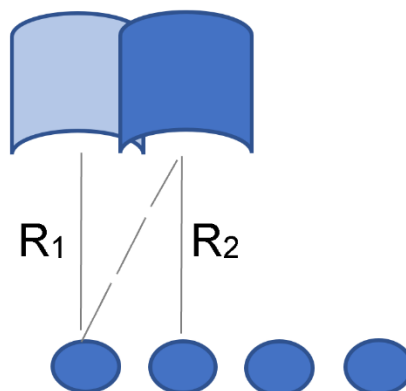


Abbildung 7: Skizze genauerer Betrachtung der Punkte außerhalb des Fokus

Die Abbildung 7 ist eine Skizze, die verdeutlichen soll, dass ab Beginn des Aussendens des Schalls mehrere Punkte vom Schall erfasst werden.  $R_1$  ist der Weg des Schalls, der ohne die

synthetische Apertur betrachtet wird. Geht man nun einen Schritt weiter mit dem Transducer, so wird bei dem nächsten Beschallen, der Punkt von  $R_1$  wieder getroffen nur zu einem späteren Zeitpunkt. Im Signal von Weg  $R_2$  ist nun zu diesem späteren Zeitpunkt ein Signalwert von dem Signal  $R_1$ . Dieses kann man nun gewichten und auf  $R_1$  hinzuaddieren.

Abhängig von der Entfernung der Punkte und der Linsenapertur, dies ist die Länge der Öffnung der Linse, kann man nun eine Anzahl an Punkten um den Punkt in Betracht mit einbeziehen.

Auch den Wert von  $R_1$  muss gewichtet werden, da er nicht zu 100 Prozent stimmen kann. Wie die Wichtungen ausfallen, hängt mit der Anzahl an betrachteten Punkten zusammen.

Nutzt man dieses Verfahren für jeden Punkt in dem untersuchten Medium, so verbessert man die Auflösung.

Geht man einen Schritt weiter, so kann man gleichzeitig eine Interpolation der Daten vornehmen, indem man, je nach dem wie viele Pixel man interpolieren möchte, Werte zwischen zwei Pixeln so annähert.

Die Z-Ebene wird bei diesem Verfahren nicht beachtet und man muss abwägen, ob man noch ein Downsampling durchführen möchte.

### 2.3 Erstellen eines Visualisierungswerkzeuges

Da ein grundlegendes Verständnis für die Daten vorhanden ist, kann nun eine graphische Oberfläche programmiert werden, die die Visualisierungsmöglichkeiten vereint. Um eine solche Aufgabe anzugehen, muss man ein geeignetes Framework für die GUI-Entwicklung nutzen. Im Laufe der Arbeit, wird mit dem Framework PyQt gearbeitet.

PyQt basiert auf sogenannten Widgets. Widgets sind alle Elemente, die man der Oberfläche hinzufügen kann. Diese Widgets funktionieren nach dem Signal-Slot System. Hierbei handelt es sich um die Kommunikationsform innerhalb einer GUI, die mit PyQt entworfen wird. Widgets kommunizieren mit Signalen und empfangen Signale mit sogenannten Slots.

#### 2.3.1 Einarbeitung in Qt für Python

Da nun die Daten eingelesen werden können und darauf aufbauend Bilder von den Daten erzeugt wurden, ist dies der Punkt, an dem die Planung eines ersten Werkzeuges zur Visualisierung startet.

Das Framework, in dem diese Software programmiert wird, nennt sich PyQt und ist die Qt – Version für Python.

Grundvoraussetzung, um eine Solche Software in Qt zu schreiben ist der vertraute Umgang mit Objektorientierter Programmierung. PyQt basiert auf Objekten, die man einfach übernehmen und nach Belieben ändern kann. Framework eigene Objekte tragen den Namen „Widgets“.

Die erste Aufgabe bestand darin, sich ausführlich mit der Materie auseinander zu setzen und grundlegende Strukturen des Frameworks zu verstehen. PyQt ist ein Event- Slot basiertes Framework, was bedeutet, dass Aktionen, die auf der Benutzeroberfläche stattfinden, ein Event auslösen, welches von einem passenden „Slot“ gefangen wird. Events sind auch Signale.

Ein Beispiel hierfür ist, das Drücken eines Buttons auf der UI löst ein „clicked“ – Signal aus, welches von einem Slot, der definiert werden muss, gefangen wird und daraufhin eine Funktion oder Methode ausführt. Dies kann alles sein.

Alle Bestandteile der Benutzeroberfläche nennen sich Widgets. Jeder Button, jedes Label ist ein Widget. Diese Widgets muss man auf richtige Art und Weise miteinander verknüpfen und auf dem Fenster platzieren. Dazu gibt es bestimmte Layouts, die von PyQt bereitgestellt werden.

Wie bereits erwähnt, kommunizieren Widgets untereinander mit Signalen und Slots. Es ist möglich, sich eigene Signale zu definieren, die Informationen über das gesamte Programm

aussenden. Jedes Signal kann Informationen tragen, die es dann weitergeben kann. Dies wird im weiteren Verlauf der Erstellung der Software von großer Bedeutung sein.

### 2.3.2 Erstellen einer eigenen Software zur Visualisierung der Daten

Eine Software zu schreiben ist das eine. Sie zu planen ist um einiges aufwendiger. Umso wichtiger ist es sich sorgfältig Gedanken zu machen, wie das Programm funktionieren soll und wie sich die Widgets untereinander verständigen.

Damit es hier nicht zu Problemen kommt, wird ein bekanntes Softwarepattern, das Model-View-Controller Pattern verwendet. Softwarepatterns sind im Grunde Baupläne für Software, um bekannten Problemen vorzubeugen.

Das Model View Controller Pattern ist vor allem dann in Gebrauch, wenn eine graphische Benutzeroberfläche das Ziel ist, denn die Kommunikation zwischen der Eingabe des Users und die Verarbeitung der Daten, um die Anfrage zu beantworten, ist nicht ganz einfach.

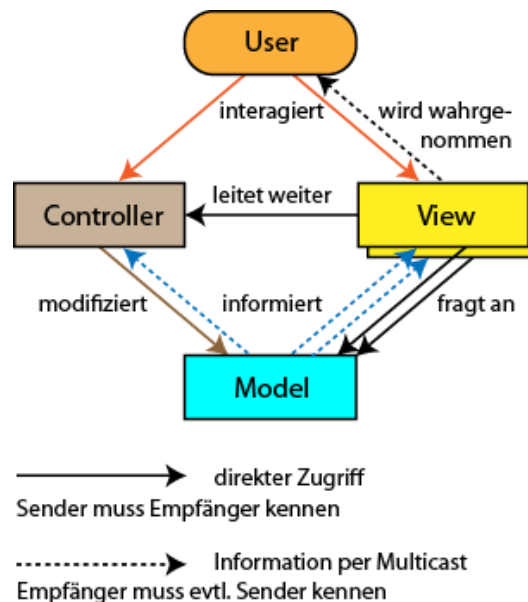


Abbildung 8: Modell-View-Controller Pattern Darstellung der Kommunikation. Quelle:  
<https://glossar.hs-augsburg.de/Model-View-Controller-Pattern>

Abbildung 8 zeigt das Verhältnis der Elemente des Modell View Controller Patterns. Die Funktionsweise musste sich komplett angeeignet werden.

Damit eine Software ohne Probleme gewartet und oder erweitert werden kann sollte man so modular arbeiten wie möglich. Dies bedeutet, dass man alles, was in einem Python Skript steht in eine eigene Datei schreibt. Diese Module werden dann mit dem Importieren der

erforderlichen Funktionen miteinander verknüpft. Das ist das gleiche Vorgehen, wenn man eine offizielle Python Bibliothek, wie Numpy, importiert.

Das Modell View Controller Pattern ist schon voll modular und deswegen gut geeignet.

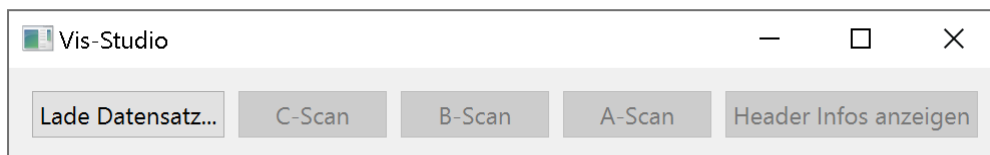
### 2.3.3 Visualisierungstool

Der Name der Software ist „Vis-Tool“, welches nun in seinem Aufbau erklärt wird.

#### Startfenster

Sobald man das Programm öffnet, sieht man ein kleines Fenster, mit den Buttons:

- Lade Datensatz...
- C-Scan
- B-Scan
- A-Scan
- Header-Infos anzeigen



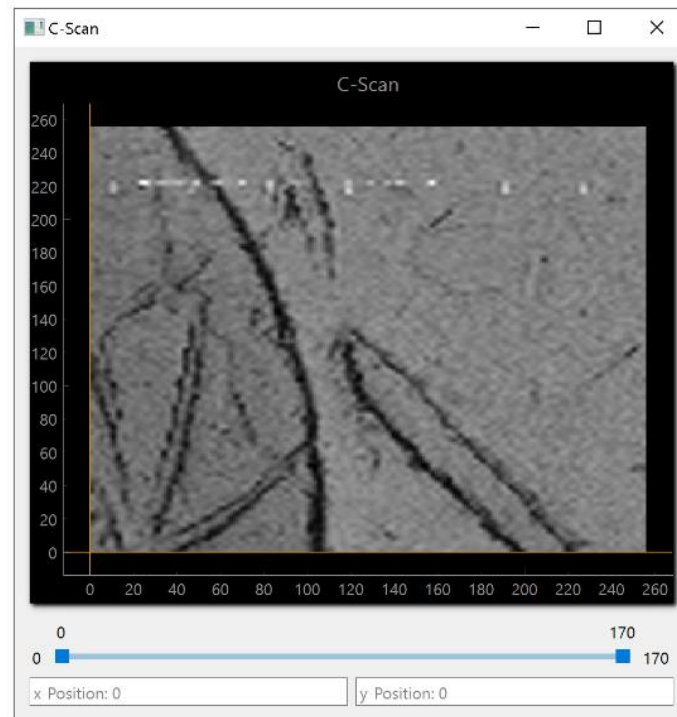
*Abbildung 9: Startfenster des Kibero Visualisierungstool*

Zu Beginn kann nur der Button „Lade Datensatz...“ betätigt werden. Dieser öffnet einen File-Dialoge, um einen Datensatz im esm-Format zu laden.

Ist dieser geladen, sind die anderen Buttons nicht mehr ausgegraut und können genutzt werden. Jeder Button, öffnet ein eigenes Fenster, welches seine eigene Visualisierung darstellt. Alle Fenster können parallel auf sein. Sobald man die gelben Linien auf dem C-Scan bewegt, werden alle anderen Widgets, also auch auf den anderen Fenstern aktualisiert.

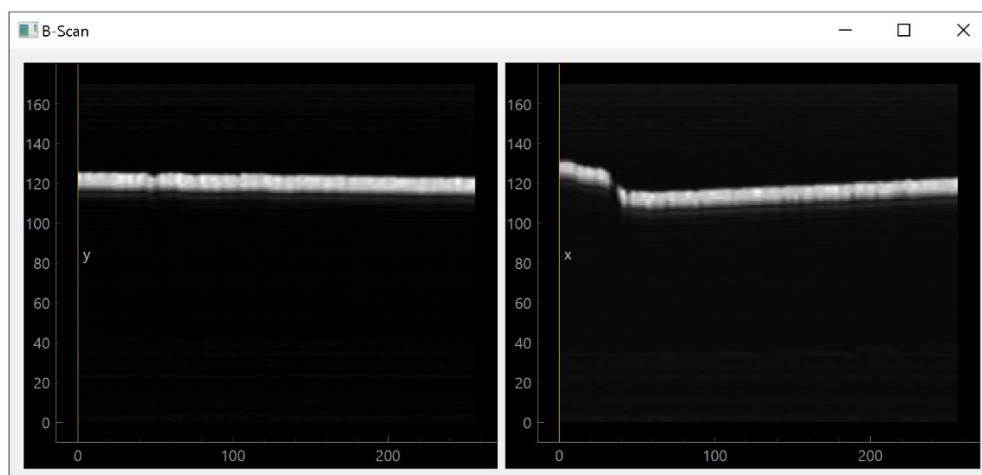
Hier spielen nun die eigenen Signale eine wichtige Rolle. Damit sich das ganze Programm aktualisiert, wenn auf einem Fenster etwas passiert, muss ein eigenes Signal definiert werden, welches die Cursorposition an alle weiteren Widgets weitergibt und diese aktualisiert.





*Abbildung 10: Das C-Scan Fenster geöffnet*

Abbildung 10 zeigt nun das C-Scan Fenster. Man sieht eine Darstellung des Bildes, welches aufgenommen wurde, die zwei verschiebbaren Linien, deren Schnittpunkt den Cursor darstellt, einen blauen Slider mit zwei verschiebbaren Enden um das zuvor erklärte Gate anzupassen und zwei Labels, die eine Position anzeigen und diese auch ändern können per Eingabe.



*Abbildung 11: Das B-Scan Fenster geöffnet*

Auf dem B-Scan Fenster, Abbildung 11, sieht man zwei Darstellungen. Auf der linken Seite sieht man einen B-Scan entlang einer Stelle auf der x-Achse und auf der rechten das gleiche nur über die y-Achse unseres C-Scans. Jedes Bild hat zudem die aktuelle Position des

anderen Cursors eingetragen, damit man sehen kann, wo man sich im Bezug zum C-Scan befindet.

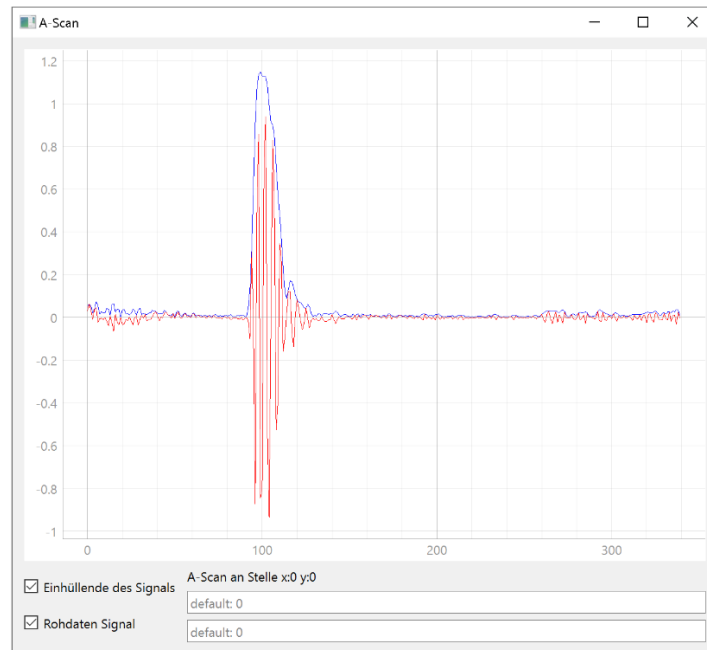
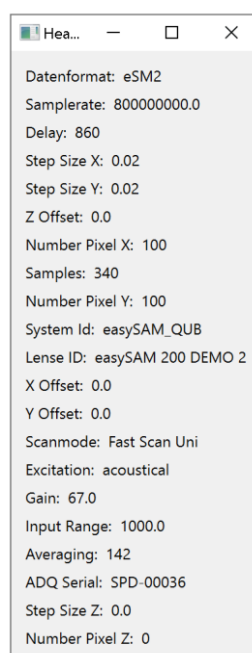


Abbildung 12: Das A-Scan Fenster geöffnet

Das A-Scan Fenster hat im Vordergrund den Plot des akustischen Signals an der Stelle des Schnittpunktes der zwei gelben Linien auf dem C-Scan. Das Fenster besitzt noch zwei Checkboxes, mit denen man entweder das originale Signal oder das „hilbert“ – Signal ausblenden kann. Auch hier gibt es zwei Eingabefelder, in die man auch Koordinaten auf dem Bild eingeben kann, wodurch der Cursor auf die Position springt.



### *Abbildung 13: Das Header Infos Fenster geöffnet*

Das Header-Info Fenster stellt untereinander alle Headerinformationen da, die im Datensatz enthalten sind.

### 3. Fazit

#### 3.1 Probleme

Probleme waren im Grunde immer vorhanden und immer musste eine Lösung gefunden werden. Die Probleme begründen dennoch überwiegend auf fehlendem Wissen, welches sich im Verlauf angeeignet wurde.

##### *Daten einlesen*

Begann tat dies schon mit dem Einlesen der Daten. Hier war mir zuerst nicht bewusst, in welcher Form die Daten codiert waren. Ich musste mich informieren, welche Art von Codierung verwendet wurde und wie ich sie in meiner Funktion decodieren kann, damit etwas Sinnvolles dabei herauskam. Wiederholen des Hexadezimalsystems war erforderlich und wie Zahlen im Computer verarbeitet und gespeichert werden.

##### *Bilder darstellen*

Auch bei der Erstellung der Bilder, kamen Probleme auf. Im ersten Moment musste ich mir klar machen, wie die Daten überhaupt aufgebaut sind und ob meine Anordnung überhaupt Sinn machte. Da dies nicht der Fall war, habe ich mich mit der Thematik auseinandergesetzt und ich konnte die Daten erfolgreich in das aktuelle Format (z, y, x) umwandeln. An diesem Punkt war es nicht mehr schwer, Daten zu extrahieren.

Ich musste die Daten einmal komplett umordnen, auch so, dass zukünftig keine Probleme auftreten.

##### *Erstellen einer GUI*

An dem Punkt, als es darum ging, eine eigene GUI zu entwickeln, musste ich erst viele online Kurse durcharbeiten, um die Grundlagen zu beherrschen. Dies nahm ein bis zwei Wochen in Anspruch, um sicher mit den Elementen des Frameworks zu arbeiten, obwohl mit der Zeit immer mehr neue Elemente dazu kamen, die wiederum eigene Signale haben. Es ist ein ständiges Lernen.

Als dann die Widgets fertig waren und ich das Programm zum ersten Mal startete, trat das Problem auf, dass die Widgets nicht synchron liefen. Also das ein Fenster die anderen aktualisiert, wenn eine Usereingabe auftrat. Zu diesem Zeitpunkt musste ich mich viel mit Softwarearchitektur auseinander setzen und eine Lösung finden, wie man alle Fenster synchron aktualisiert. Nach einiger Recherche begegneten mir zahlreiche Softwarepattern, die man für solche Probleme verwenden kann.

Mit dem Model View Controller Pattern musste ich mich auch einige Zeit auseinander setzen, da mir nicht direkt klar war, wie die Architektur funktioniert. Der Punkt Callbacks war mir auch

zu diesem Zeitpunkt nicht geläufig, obwohl sie für meinen Lösungsansatz unerlässlich sind. Ein Callback referenziert im Grunde auf eine Funktion, die in einem anderen Modul vorhanden ist. Callbacks sind wichtig für die Kommunikation zwischen View und Controller.

#### 3.2 Kritische Analyse

In vielen Punkten ist das Programm noch nicht optimal und effizient. Auch einige Algorithmen, die geplant waren, sind nicht im Rahmen der praktischen Phase fertig geworden.

Ein aktuelles Problem, welches noch nicht behoben wurde, ist das effiziente Einlesen der Daten. Während der praktischen Phase wurde das Programm mit einem Testdatensatz getestet, um Zeit zu sparen. Dies, weil die aktuell aufgenommenen Datensätze sehr groß sind und der Algorithmus zum Einlesen nicht effizient genug ist. An dieser Stelle stößt man auf eine sehr lange Laufzeit.

Es war geplant, dass man mehrere Fenster einer Art aufmachen kann. Dies klappt ebenfalls noch nicht.

Im Verlauf des Praktikums wurde zu spät erkannt, dass das Dokumentieren der Dateien, des Codes und der enthaltenen Klassen von großer Relevanz ist. Daher fehlt eine anständige Dokumentation des Codes. Das Dokumentieren ist unerlässlich, wenn das Programm etwas komplexer wird.

Auch im Blick auf den Code muss nochmal das Thema „Clean Code“ beachtet werden.

#### 3.3 Lösungsansätze

Um das Programm effizienter zu machen und dies nicht nur mit Blick auf das Einlesen der Daten, würde ich in Zukunft mehr auf die Verwendung von Threads zurückgreifen. Die Thematik von Multithreading ist aber noch nicht ganz klar.

Konzept von Multithreading ist, Programmabläufe parallel laufen zu lassen. Hier könnte man die Daten, beispielsweise, in mehrere Blöcke aufteilen und sie alle parallel einlesen lassen.

Multithreading ist auch für den Zweck gut, mehrere Fenster einer Art gleichzeitig zu öffnen.

Auch die Thematik von Multiprocessing ist in diesem Aspekt eine Lösung man muss sich nur im Klaren über Vor- und Nachteile von beiden Ansätzen auseinandersetzen.

#### 3.4 Abschließend

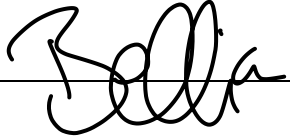
Abschließend ist zu sagen, dass ich in dieser praktischen Phase viel in Bezug auf Softwareentwicklung gelernt habe, was auch mein Anliegen war. Obwohl das Programm nicht fertig ist, ist das aktuelle Ergebnis zufriedenstellend und bietet eine gute Grundlage für die anknüpfende Bachelorarbeit.

## Erklärung der Praktikumsstelle

Die Praktikumsstelle hat den Bericht eingesehen und erklärt hiermit die ordnungsgemäße Durchführung der praktischen Phase:

**Unterschrift & Datum Praktikumsstelle:**

---

**Unterschrift & Datum Student:** 12.07.2023 

---

## Abbildungsverzeichnis

Abbildung 1: C-Scan einer 1 Cent Münze.....	7
Abbildung 2: C-Scan der gleichen Münze mit links Gate 0-50 Ebenen und recht Gate 50-100 Ebenen .....	8
Abbildung 3: Darstellung an welcher Position der jeweilige B-Scan dargestellt wird. Links: y=120 Rechts: x=120 .....	8
Abbildung 4: Die zu Abbildung 3 passenden B-Scan Bilder .....	9
Abbildung 5: A-Scan an einer Stelle XY .....	9
Abbildung 6: Skizze der synthetischen Apertur.....	10
Abbildung 7: Skizze genauerer Betrachtung der Punkte außerhalb des Fokus.....	10
Abbildung 8: Modell-View-Controller Pattern Darstellung der Kommunikation. Quelle: <a href="https://glossar.hs-augsburg.de/Model-View-Controller-Pattern">https://glossar.hs-augsburg.de/Model-View-Controller-Pattern</a> .....	13
Abbildung 9: Startfenster des Kibero Visualisierungstool .....	14
Abbildung 10: Das C-Scan Fenster geöffnet .....	15
Abbildung 11: Das B-Scan Fenster geöffnet.....	15
Abbildung 12: Das A-Scan Fenster geöffnet.....	16
Abbildung 13: Das Header Infos Fenster geöffnet.....	17