



BOSCH
Technik fürs Leben



Duale Hochschule
Baden-Württemberg

Entwicklung eines Tools zur Auswertung medizinischer Messdaten für eine Pilotstudie im Bereich der Stress-Regulationsfähigkeit des Menschen

Studienarbeit

Bachelor of Science

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Jona Krumrein, Tim Weiss

10.06.2022

Bearbeitungszeitraum
Matrikelnummer, Kurs
Ausbildungsfirma
Betreuer

22.10.2022 - 10.06.2022
3857223, 8336074, TINF19ITA
Robert Bosch GmbH, Stuttgart
Mario Babilon

Abstract

Diese Arbeit beschreibt die Konzeptionierung und Implementierung einer Benutzeroberfläche zum Visualisieren und automatisierten Auswerten medizinischer Messdaten im Bereich der Stress-Regulationsfähigkeit des menschlichen Herzens. Dazu wird zunächst eine Einführung in den Bereich des menschlichen Herzens und der Messtechnik der Herzparameter gegeben. Hierbei wird ebenfalls das Kubios HRV Premium Tool vorgestellt und bewertet, welches die Grundlage der Auswertung bildet und auf dem aufgebaut wird. Im nächsten Schritt werden dann unterschiedliche Tools und Programmiersprachen zur Umsetzung der Aufgabenstellung evaluiert und aufbauend darauf ein Konzept für die Applikation vorgestellt. Danach wird die schrittweise Implementierung dargestellt, sowie ein Überblick über die Verwendung und Auswertung mit der Applikation gegeben. Der Bericht schließt mit einer Zusammenfassung der Entwicklung, sowie einem Ausblick auf zukünftige Erweiterungen und der damit verbundenen Nutzung.

Inhaltsverzeichnis

Abkürzungsverzeichnis	IV
Abbildungsverzeichnis	V
Tabellenverzeichnis	VI
Formelgrößenverzeichnis	VII
Formelverzeichnis	VIII
Listings	IX
1 Medizinische Grundlagen	1
1.1 Herzparameter	1
1.2 Herzfrequenzvariabilität (HRV)	3
2 Aufgabenstellung	4
3 Rahmenbedingungen	5
3.1 Export-Datenstruktur	5
3.2 Programmiersprache	6
3.2.1 Python	7
3.2.2 MATLAB	7
3.2.3 Performance-Vergleich	8
4 Konzept	14
4.1 System-Grundlage und Struktur	14
4.2 Ablauf	15
4.3 Applikations-Design	16
5 Implementierung des Tools	18
5.1 Grundstruktur der Applikation	18
5.1.1 Layout und Komponenten	19
5.2 Setup der Applikation	20
5.3 Erstellung der Samples	21

5.4	Visualisierung der Daten	21
5.4.1	Datenstruktur und Datenverarbeitung	21
5.4.2	Plot-Funktion	25
5.4.3	Zusätzliche Funktionen	27
6	Auswertung von Messungen mit Hilfe des HRV Evaluation Tools	28
7	Kritische Würdigung und Ausblick	29
	Anhang	A
	Glossar	B

Abkürzungsverzeichnis

BSP Board Support Package

Abbildungsverzeichnis

1.1	R-R-Intervall	3
3.1	Visualisierung der Zeitmessung	11
4.1	Konzept Klassendiagramm	14
4.2	Ablaufdiagramm des Konzepts	15
4.3	Konzept Ablauf für das Erstellen der Samples	16
4.4	Konzept Ablauf für das Visualisieren von Daten	16
4.5	Konzept Ablauf für das Design	17
5.1	Klassendiagramm der fertigen Applikation	18
5.2	Auszug aus der Struktur der Setup-Datei	20
5.3	Verschachtelungsprobleme der unverarbeiteten Export-Daten	22
5.4	Baumstruktur zur Veranschaulichung der Datenstruktur	23

Tabellenverzeichnis

3.1	Übersicht der technischen Daten des Testmediums	8
3.2	Zeitmessungen der Datensatzgrößen und Implementierungen	11
3.3	Ergebnisse der Geschwindigkeits-Messungen	13

Formelgrößenverzeichnis

a	rad	Bedeutung von a
b	rad	Bedeutung von b
λ	rad	Bedeutung von lambda
ϕ	rad	Bedeutung von phi

Formelverzeichnis

1.1 Herzminutenvolumen (HMV)	1
1.2 Maximale Herzfrequenz	1
3.1 Berechnung der abhängigen Variable - OLS	9

Listings

3.1	Methode der kleinsten Quadrate MATLAB Implementierung	10
3.2	Methode der kleinsten Quadrate Python Implementierung	10
3.3	Line-Plot MATLAB Implementierung	12
3.4	Line-Plot Python Implementierung	12
5.1	Auszug einiger verwendeter App-Komponenten	19
5.2	Konvertieren einer MATLAB Tabelle in ein Struct	23
5.3	Auflösen verschachtelter MATLAB Tabellen in ein Struct	24
5.4	Auflösen eines Parameters in einer Liste anhand des Beispiel-Parameters MSE	24
5.5	Ermittlung von Sample-Länge und Erstellung der x-Werte	25
5.6	Beispiel der Link-Zuordnung für Parameter-Gruppe	26
5.7	Index des Parameters und dazugehörige Informationen filtern	26
5.8	MATLAB Scatter-Funktion	26

1 Medizinische Grundlagen

1.1 Herzparameter

Das menschliche Herz kann mit vielen verschiedenen Werten parametrisiert werden. Der wohl bekannteste Parameter ist dabei die Herzfrequenz (H_f). Sie gibt an, wie oft das Herz in einer bestimmten Zeit schlägt. Meist wird dafür das Zeitintervall einer Minute gewählt und die in dieser Zeit gemessenen Herzschläge in der Einheit *bpm* (beats per minute) angegeben.

Die Herzfrequenz ist abhängig von verschiedenen Faktoren. So spielen beispielsweise Alter und körperliche Fitness eine große Rolle. Bei trainierten (Ausdauer) Sportlern schlägt das Herz in Ruhe bedeutend seltener als bei untrainierten Personen. Dies kann damit erklärt werden, dass durch das Ausdauertraining das **Schlagvolumen (SV)** erhöht werden kann. Das Schlagvolumen gibt an, wie viel Blut das Herz mit einem Schlag durch den Körper pumpen kann. Wenn sich dieses Volumen also erhöht, muss das Herz seltener schlagen, um das **Herzminutenvolumen (HMV)** zu halten. Dieses berechnet sich aus Herzfrequenz und Schlagvolumen:

$$H MV = H f * S V \quad (1.1)$$

Bei körperlicher Belastung, beispielsweise sportlicher Aktivität, erhöht sich das benötigte Herzminutenvolumen, da die Muskeln mehr Sauerstoff brauchen. Da sich das Schlagvolumen des Herzens nicht anpassen lässt, erhöht sich die Herzfrequenz um dem benötigten Herzminutenvolumen gerecht zu werden. Grundsätzlich, wie jeder bei sich selber feststellen kann, erhöht sich die Herzfrequenz bei körperlicher Aktivität. Durch sehr starker Aktivität kann die Herzfrequenz auf ein Maximum ansteigen. Die maximale Herzfrequenz kann mit folgender Faustformel abgeschätzt werden:

$$H f_{max} = 220 - A l t e r \quad (1.2)$$

gesprochen. Allerdings muss zwischen Puls und Herzfrequenz unterschieden werden, da diese nicht identisch sind. Die Herzfrequenz beschreibt, wie bereits in den vorhergehenden

Abschnitten beschrieben, die tatsächlichen Schläge des Herzens. Zur Messung der Herzfrequenz kann ein EKG genutzt werden, Der **Puls** hingegen wird peripher, beispielsweise am Hals oder Handgelenk gemessen. Dabei werden Pulswellen erfasst, welche sich durch den Körper bewegen. Eine Pulswelle entsteht wenn das Herz schlägt, und das Blut durch den Körper pumpt. Dabei wird das Blut gegen die Arterienwände gedrückt und eine Pulswelle kann detektiert werden. Bei gesunden Menschen entspricht der Puls der Herzfrequenz, da jeder Herzschlag Blut durch den Körper pumpt und somit auch eine messbare Pulswelle erzeugt. Bestimmte Herzrhythmuskrankheiten können dafür verantwortlich sein, dass Puls und Herzfrequenz voneinander abweichen. Dabei können Kontraktionen des Herzens entstehen, welche kein oder nicht genug Blut durch den Körper pumpen um eine messbare Pulswelle zu erzeugen. Die Differenz zwischen Herzfrequenz und Puls nennt sich **Pulsdefizit** (und ist bei gesunden Menschen 0).

Die Variation der Herzfrequenz wird über das **vegetative Nervensystem** geregelt. Dies besteht aus dem *Sympathikus* und seinem Gegenspieler, dem *Parasympathikus*. Außerdem ist das *enterische Nervensystem* (*Eingeweidenervensystem*) teil des vegetativen Nervensystems. Dies spielt für die Steuerung des Herzens allerdings keine Rolle und wird daher im Laufe dieser Arbeit nicht mehr erwähnt. Ebenso haben Sympathikus und Parasympathikus Funktionen in der Verdauung. Da dies ebenfalls keine Relevanz für das Herz hat, wird darauf nicht weiter eingegangen. Die Folgende Erklärung von Sympathikus und Parasympathikus beziehen sich daher nur auf die für das Herz relevanten Eigenschaften.

1. Sympathikus

Der Sympathikus kann den Organismus auf eine Aktivitätssteigerung ("fight or flight") einstellen. Er ist der dominante Teil des vegetativen Nervensystems, wenn es um Stresssituationen geht. Wenn diese eintritt, wird die Aktivität von situationsbedingt unwichtigen Organen, beispielsweise Darm oder andere Organe die zur Verdauung beitragen, gesenkt. Die Aktivität von Organen die in Stresssituationen wichtig sind wird verstärkt. So werden beispielsweise die Pupillen vergrößert. **Die Herzfrequenz wird stark erhöht.**

2. Parasympathikus

Der Parasympathikus stellt den Organismus auf Ruhesituationen ("rest and digest") ein. So werden in den dominanten Phasen des Parasympathikus Organe mit Verdauungsfunktionen in ihrer Aktivität gestärkt. Der Körper wird auf Entspannung und Regeneration heruntergefahren. **Die Herzfrequenz wird in diesen Phasen stark gesenkt.**

Das vegetative Nervensystem ist in seiner Funktion **unwillkürlich**. Das bedeutet, es kann nicht bewusst von der jeweiligen Person gesteuert werden. Dadurch sind Messungen, beispielsweise an der Herzfrequenz sehr gut geeignet, um bestimmte äußere Einflüsse zu untersuchen. Die Ergebnisse können nicht von der Testperson verfälscht werden, da sie das vegetative Nervensystem nicht beeinflussen kann. (Andere äußere Einflüsse wie Aufregung müssen berücksichtigt werden und können das Ergebnis logischerweise verfälschen.)

Um nun den konkreten Einfluss von elektromagnetischen Wellen zu untersuchen kann die Herzfrequenz als Messparameter benutzt werden. Ein weiterer Parameter welcher stark abhängig vom vegetativen Nervensystem ist, ist die **Herzfrequenzvariabilität (HRV)**. Im Verlauf dieser Arbeit, wird die Herzratenvariabilität als entscheidender Parameter genutzt. Alle durchgeführten Messungen und daraus abgeleiteten Interpretationen beziehen sich auf diese.

1.2 Herzfrequenzvariabilität (HRV)

Die Herzfrequenzvariabilität, auch Herzratenvariabilität ist die zeitliche Variation zwischen zwei aufeinanderfolgenden Herzschlägen [wiki?](#).

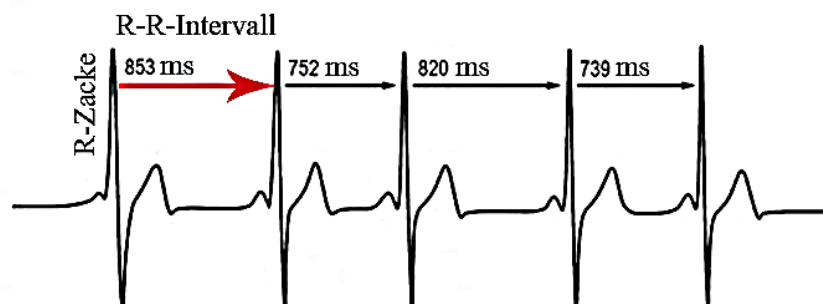


Abbildung 1.1: R-R-Intervall

In dieser Grafik ist eine EKG Messung zu sehen. Gut erkennbar sind die hohen R Zacken. Diese werden benutzt, um die Herzfrequenzvariabilität zu berechnen. Dabei wird die Zeit zwischen zwei Herzschlägen als die Zeit zwischen den aufeinanderfolgenden R-Zacken definiert. In diesem Schaubild liegt diese Zeit, also das R(-)R-Intervall zwischen 739 ms und 853 ms. Das RR-Intervall wird in der Literatur oft als NN-Intervall bezeichnet. Der Begriff RR wird auch bei Messen des Blutdrucks verwendet. Dort steht RR für *Riva-Rocci*. Um eine Verwechslung zu vermeiden, wird daher im Kontext der HRV vom NN-Intervall gesprochen.

2 Aufgabenstellung

Mit der fortschreitenden Entwicklung in der Digitalisierung unserer Gesellschaft erhöht sich ebenfalls die Anzahl und Verfügbarkeit der notwendigen Geräte und der dazugehörigen Sendeanlagen. Diese können mit Hilfe von elektromagnetischen Wellen kabellos Daten übertragen und erzeugen somit künstlich elektromagnetische Felder, welche sich mit den bereits natürlichen elektromagnetischen Feldern überlagern. Die Folgen des wachsenden Ausbaus digitaler kabelloser Technologien, sowie die Nutzung der passenden Endgeräte, wie Smartphones, Tablets und drahtlose Telefone und die damit verbundenen Auswirkung auf die Bevölkerung, rückte bisher nur in geringem Maße in den Fokus der wissenschaftlichen Forschung.

Innerhalb einer Pilotstudie sollen die Auswirkungen und Einflüsse elektromagnetischer Strahlung auf die Stress- und Reizverarbeitung des Menschen untersucht werden. Dazu werden verschiedene biologische Messwerte des Herzens ohne Strahlenbelastung bestimmt und danach den alltäglichen Belastungen durch WLAN-Router, DECT-Telefone, Bluetooth- und Mobilfunk-Endgeräte gegenübergestellt. Bei der Durchführung der verschiedenen Messungen und Untersuchungsarten der Studie wird eine große Datenmenge produziert, welche passend aufbereitet werden muss, um eine schnelle und einfache Auswertung durch die beteiligten Ärzte und Physiotherapeuten ermöglichen zu können.

Ziel der Studienarbeit ist die Konzeptionierung und Implementierung einer Schnittstelle, welche die Automatisierung der Aufbereitung und Auswertung der Messdaten ermöglicht. Dazu soll eine Applikation erstellt werden, welche an die bereits verwendete und etablierte Software Kubios HRV Premium anknüpft und deren Funktionen erweitert. So sollen die Rohdaten der Messungen in Kubios eingelesen und die Exportdateien und deren Herzparameter innerhalb einer Applikation graphisch dargestellt werden. Der Fokus liegt dabei auf der automatische Generierung der in Kubios erstellbaren Samples, welche die Einteilung der Messung in kleinere Zeitbereiche ermöglicht, sowie der fachlich korrekten Darstellung der Herzparameter im Diagramm zur Unterstützung der Studie. Zudem soll die Möglichkeit bestehen das Belastungsintervall hervorzuheben und die fertige Visualisierung zu speichern. Zur Umsetzung sollen dabei ebenfalls unterschiedliche Programmiersprachen sowie Dateiformate betrachtet und bewertet werden.

3 Rahmenbedingungen

3.1 Export-Datenstruktur

Wie bereits im Kapitel zuvor erwähnt bietet Kubios HRV Premium zwei unterschiedliche Exportmöglichkeiten. Diese werden durch den einheitlichen CSV-Standard und das MATLAB-eigenen MAT-File repräsentiert.

Bei einer CSV-Datei (engl. Comma-separated values) handelt es sich um einen Standard, welcher den Aufbau einer Textdatei beschreibt. Der Inhalt der Datei beschränkt sich meist auf Tabellen und Listen, welche nicht wie üblicherweise in Zeilen und Spalten aufgeteilt wird, sondern mit Hilfe von Kommas getrennt wird, wobei jede neue Zeile eine neue Datenbankzeile darstellt und jede Datenbankzeile aus einem oder mehreren Feldern besteht. CSV-Dateien finden vor allem Anwendung in Tabellenkalkulationsprogrammen und Datenbanksysteme, wie Microsoft Excel oder MySQL, welche das Format einlesen, sowie exportieren können. Aufgrund der weiten Verbreitung des Formats ist auch das Einlesen und Verarbeiten der Dateien in nahezu jeglicher Programmiersprache möglich. Vor allem Python bietet passende Bibliotheken und Funktionen.

Das MAT-File beschreibt ein binäres Datencontainerformat, welches von MATLAB verwendet wird. Es ermöglicht das Speichern von Arrays, Variablen und Funktionen. Je nach MATLAB Version ermöglicht das Format zusätzlich die Speicherung von mehrdimensionale numerische Arrays, Zeichenarrays, Zellenarrays, Felder mit geringer Dichte, Objekte und Strukturen. Die Lesbarkeit beschränkt sich hier jedoch auch auf MATLAB und einige wenige Bibliotheken für beispielsweise Python.

Da diese beiden Export-Formate und deren Inhalt von einander verschieden sind und die Grundlage der Datenverarbeitung sowie Datenvisualisierung bilden, sollen diese im Folgenden Hinblick auf Übersichtlichkeit, Formatierung, inhaltlicher Korrektheit und Zugänglichkeit verglichen werden.

Die Darstellung der CSV-Datei ist unverschachtelt und befindet sich auf einer Ebene. Alle Daten lassen sich auf einmal einsehen. Innerhalb der Datei befinden sich tabellarische Strukturen der einzelnen Messdaten, aber auch Einzelinformationen, Überschriften und

Bezeichner. Dies stellt für das Einlesen und Verarbeiten der Daten ein Problem dar, da zusätzlich nach relevanten gefiltert oder mehrere Formatierungsiterationen durchgeführt werden müssen. Das MAT-File hingegen ist in sich verschachtelt. Die Messdaten sind sortiert und unter ihrem jeweiligen Bereich geordnet. Der Zugriff auf die Daten erfolgt mittels Key-Value-Verfahren, das heißt unter einem Schlüsselwort befinden sich der jeweilige Datensatz. Ein Datensatz kann dabei ebenfalls von einem weiteren Key-Value-Paar dargestellt werden. Irrelevante Daten können hier gefiltert werden in dem man den jeweiligen Key einfach nicht aufruft. Das Format kann also ohne weitere Vorformatierung eingelesen werden. Betrachtet man die Konstanz und Korrektheit bezogen auf die Darstellung der Daten in Kubios, lassen sich im CSV-Format einige Unregelmäßigkeiten erkennen. So sind häufig Kommas verschoben und Felder mit einer stark variierenden Anzahl Leerzeichen aufgefüllt, um eine übersichtlichere Darstellung zu ermöglichen. Dies hat jedoch den Effekt, dass diese Leerzeichen beim Einlesen mit beispielsweise Python ebenfalls im Datensatz auftauchen und diesen verunreinigen bzw. im schlimmsten Fall unbrauchbar machen. Auch hier müsste deshalb im Vorhinein eine Formatierung der grundlegenden Datei durchgeführt werden. **Diese Unregelmäßigkeiten lassen sich auf die Exportfunktionen der Kubios-Software zurückführen.** Der Inhalt des MAT-Files ist hingegen akkurat und nicht von den Unregelmäßigkeiten betroffen.

Daraus ergibt sich, dass das MAT-File ein grundlegend konsistenterer Datenspeicher für die medizinischen Messdaten darstellt und einen großen Vorteil im Bereich der Zugänglichkeit sowie der Reduzierung zusätzlicher Hilfsfunktionen und Datenmanipulationen gegenüber dem CSV-Format bietet. Aus diesem Grund wird dieses auch als Grundlage zur Umsetzung des Tools zur Auswertung der medizinischen Messdaten verwendet.

3.2 Programmiersprache

Im Bereich der Data Science gibt es viele unterschiedliche Programmiersprachen, welche das Bearbeiten und Auswerten großer Datenmengen ermöglichen. Je nach Anwendungsgebiet und Lösungsansätzen, sowie der dazugehörigen Eigenschaften der zu analysierenden Daten muss eine passende Programmiersprache gewählt werden. Im Hinblick auf die Anforderungen der Studienarbeit stehen vor allem das Auslesen der Daten aus vorgegebenen Datei-Formaten, das graphische Darstellen der Messdaten, sowie die Erstellung einer passenden Benutzeroberfläche im Vordergrund.

Python ist hierbei die beliebteste und meistverwendete Programmiersprache. Sie ermöglicht das effiziente Auslesen von CSV-Datei sowie MAT-Dateien, verfügt über die Möglichkeit eine Benutzeroberfläche zu erstellen und bietet mit einer großen Anzahl zusätzlicher

Bibliothek eine hohe Flexibilität. Somit ist Python perfekt auf die Anforderungen der Aufgabenstellung zugeschnitten. Da die Messdaten jedoch in der zukünftigen Implementierung aus der MAT-Datei ausgelesen werden sollen, wird auch die Umsetzung in MATLAB betrachtet. MATLAB bietet dabei auch eine Reihe von Tools zum Auswerten verschiedener Daten und das erstellen einer graphische Benutzeroberfläche. Beide Programmiersprachen stellen alle benötigten Funktionen sowie Ansätze zur Lösung der Aufgabenstellung dar und werden häufig direkt miteinander verglichen, da die Überschneidung ihrer Anwendungsgebiete sehr groß ist. Um eine Entscheidung über die zu verwendende Programmiersprache treffen zu können werden diese im folgenden vorgestellt und anhand wichtiger Kriterien miteinander verglichen.

3.2.1 Python

Python ist eine interpretierte, objektorientierte High-Level-Programmiersprache mit einer dynamischen Semantik, welche in den Bereichen der Datenanalyse und Machine Learning, aber auch im Web Development oder für Automatisierungstasks eingesetzt wird. Sie glänzt vor allem durch ihre hochentwickelten, integrierten Datenstrukturen und dynamischer Typisierung, weshalb sie sehr attraktiv für Anwendungs- sowie Skriptentwicklung ist. Auch die einfache und leicht zu erlernende Syntax von Python, bietet einen guten Einstieg in die Programmierung, beschleunigt den Entwicklungsprozess und erleichtert die Programmpflege. Außerdem unterstützt Python eine Vielzahl von Modulen und Paketen, was die Modularität von Programmen und die Wiederverwendung von Code fördert. Diese werden vor allem auch durch die große Community und deren Open-Source-Projekte weiter gefördert, weshalb grundlegende Lösungen für nahezu jedes Problem bereits bestehen. Der Python-Interpreter und die umfangreiche Standardbibliothek sind für alle wichtigen Plattformen kostenlos verfügbar und können frei verteilt werden.

3.2.2 MATLAB

MATLAB ist eine High-Level-Prgrammiersprache für technische Berechnungen, welche von *The MathWorks Inc.* entwickelt sowie vertrieben wird und deshalb auch eine kostenpflichtige Lizenz benötigt. Sie integriert Berechnungen, Visualisierung und Programmierung in einer einfach zu bedienenden Umgebung, in der Probleme und Lösungen in vertrauter mathematischer Notation ausgedrückt werden. Ihre Anwendungsgebiete sind dabei mathematische Berechnungen und Algorithmen-Entwicklung, sowie die Simulation und Datenanalyse, aber auch die Entwicklung von Anwendungen, einschließlich der Erstellung

von graphischen Benutzeroberflächen sind möglich.

MATLAB basiert grundlegend auf einem Array-Datenelement, welches nicht dimensioniert werden muss. Dadurch lassen sich viele technische Berechnungsprobleme, insbesondere solche mit Matrix- und Vektorformulierungen, in einem Bruchteil der Zeit lösen, die für das Schreiben eines Programms in einer skalaren, nicht interaktiven Sprache wie C erforderlich wäre. MATLAB bietet eine Reihe anwendungsspezifischer Lösungen, die so genannten Toolboxes. Toolboxes sind umfassende Sammlungen von MATLAB-Funktionen, die die MATLAB-Umgebung erweitern, um bestimmte Problemklassen zu lösen. Zu den Bereichen, in denen Toolboxes verfügbar sind, gehören Signalverarbeitung, Steuerungssysteme, neuronale Netze, Simulation und viele andere.

3.2.3 Performance-Vergleich

Um einen besseren Überblick über die Performance der beiden Programmiersprachen zur Auswertung von Messdaten zu erhalten, werden mehrere Performance-Vergleiche durchgeführt. Getestet wird auf einem MacBook mit folgenden technischen Daten, welcher sich im Akku-betriebenen Modus befindet.

Technische Daten

Model	MacBook Pro (Retina, 13-inch, Early 2015)
Betriebssystem	macOS Monterey Version 12.1
Prozessor	2,7 GHz Dual-Core Intel Core i5
Arbeitsspeicher	8 GB 1867 MHz DD3
Grafikchip	Intel Iris Graphics 6100 1536 MB

Tabelle 3.1: Übersicht der technischen Daten des Testmediums

Analyse anhand der Methode der kleinsten Quadrate

Zuerst wird ein allgemeiner Geschwindigkeitstest durchgeführt. Hierzu soll sowohl in Python als auch in MATLAB die Methode der kleinsten Quadrate (OLS) zur Regression exemplarisch durchgeführt werden. Hierbei werden die quadrierten Abstände zwischen den Datenpunkten und der Regressionsfunktion/-geraden minimiert, um die Regressionsgerade

zu finden. Ausgleichsrechnungen sind ein wichtiger Bestandteil der Datenanalyse, deshalb bietet diese Methode ein gutes Beispiel zum Vergleich der Programmiersprachen.

Die Methode der kleinsten Quadrate wurde mit einer Replikation von 1000 durchgeführt. Für die Erstellung des Beispieldatensatzes werden die folgenden wahren Parameter verwendet.

$$\beta = \begin{bmatrix} 10 \\ -0,5 \\ 0,5 \end{bmatrix}$$

Die Regression wird für drei unterschiedliche Datensatzgrößen mit $n = 1000, 10000$ und 100000 umgesetzt. Für jede Beobachtung werden die unabhängigen Variablen aus folgenden Werten gezogen:

$$\mu_x = \begin{bmatrix} 10 \\ 10 \end{bmatrix}, \sigma_x = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$$

Die abhängige Variable wird konstruiert, indem ein Vektor von zufälligen Normalvariablen aus $\text{Normal}(0,1)$ gezogen wird. Dieser Vektor wird als ϵ bezeichnet, und die abhängige Variable wird wie folgt berechnet:

$$Y = Xb + \epsilon \quad (3.1)$$

Für die Berechnung der Parameter werden integrierte Funktionen der beiden Programmiersprachen verwendet. MATLAB basiert hierbei auf der **OLS-Funktion**, während bei Python die **statsmodels**-Bibliothek, sowie wie **numpy** und **pandas** zur Datenverarbeitung zum Einsatz kommen. Hierbei muss angemerkt werden, dass im folgenden davon ausgegangen wird, dass diese aufgrund ihrer Definition als State-of-the-Art bereits auf Geschwindigkeit optimiert sind und eine aussagekräftige Darstellung des Performance-Vergleich bieten können.

Für den Vergleich fällt die Beschränkung der Zeitmessung auf das Auswählen von Stichproben aus dem Datensatz, das Berechnen der Parameterschätzung durch OLS und das Abspeichern der Ergebnisse. Die Erzeugung der Datensätze wird hingegen nicht in die Zeitmessung aufgenommen. Beide Implementierungen wurden im Vorhinein bereits nach grundlegenden Code-Optimierungen angepasst.

```
1 reps = 1000;
2 beta = [10,-.5,.5];
3 n_array = [1000, 10000, 100000];
4
5 mat_time = zeros(3,2);
6
7 for i = 1:3
8     n = n_array(i);
9     row_id = 1:n;
10
11     X = [normrnd(10, 4, [n 2]) ones(n,1)];
12     Y = X * beta' + normrnd(0,1,[n 1]);
13
14     store_beta = zeros(reps, 3);
15     tic
16     for r = 1:reps
17         this_row = randsample(row_id, n, true);
18         store_beta(r,:) = (OLS(Y(this_row), X(this_row,:)))';
19     end
20     mat_time(i,:) = [n toc];
21 end
```

Listing 3.1: Methode der kleinsten Quadrate MATLAB Implementierung

```
1 import numpy as np
2 import pandas as pd
3 import statsmodels.api as sm
4 from timeit import timeit
5 import matplotlib.pyplot as plt
6
7 reps, beta, n_array = 1000, [10, -0.5, 0.5], [1000, 10000, 100000]
8
9 def python_boot():
10     for r in np.arange(reps):
11         this_sample = np.random.choice(row_id, size=n, replace=True)
12         X_r = X[this_sample,:]
13         Y_r = Y[this_sample]
14         store_beta[r,:] = sm.regression.linear_model.OLS(Y_r, X_r).fit(
15             disp=0).params
16
17 python_time = np.zeros((len(n_array),2))
18 count=0
19 for n in n_array:
```

```

20     row_id = range(0, n)
21     X1 = np.random.normal(10, 4, (n, 1))
22     X2 = np.random.normal(10, 4, (n, 1))
23     X = np.append(X1, X2, 1)
24     X = np.append(X, np.tile(1, (n, 1)), 1)
25     error = np.random.randn(n, 1)
26     Y = np.dot(X, beta).reshape((n, 1)) + error
27
28     store_beta = np.zeros((reps, X.shape[1]))
29     TimeIt = timeit("python_boot()", setup="from __main__ import
        python_boot", number=1)
30     python_time[count, :] = [n, TimeIt]

```

Listing 3.2: Methode der kleinsten Quadrate Python Implementierung

n	MATLAB	Python
1000	0.3571	0.3072
10000	0.6031	1.7127
100000	3.7375	21.5871

Tabelle 3.2: Zeitmessungen der Datensatzgrößen und Implementierungen

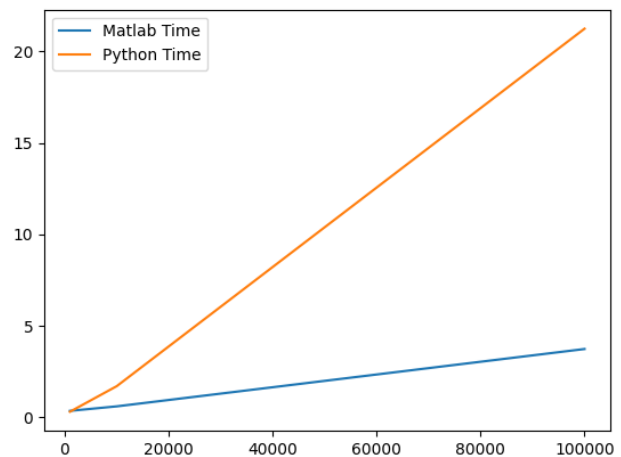


Abbildung 3.1: Visualisierung der Zeitmessung

TODO: Auswertung

Analyse anhand des Daten einlesen und visualisieren

Im nächsten Schritt soll ein Performance-Test durchgeführt werden, welcher stärker in Richtung der Aufgabenstellung ausgelegt ist. Hierzu wird ein kleiner Datensatz geöffnet und ein einfache Line-Plot erstellt. Geplottet wird jeweils die durchschnittliche Herzfrequenz aus dem normalen Datensatz der Auswertung ohne Samples. Hierbei handelt es sich um 41 Werte, welche als Line-Plot ohne weitere Konfiguration dargestellt werden. In der Python Umgebung werden die Bibliotheken **pandas** und **h5py** verwendet, da diese für ihre jeweiligen Aufgaben, Erstellen von Data Frames und Laden von mat-Files, als

State-Of-The-Art gelten. Zum Plotten wird auf die **matplotlib** zurückgegriffen, da diese der MATLAB Darstellung am nächsten kommt. MATLAB ermöglicht die Auswertung ohne zusätzliche Bibliotheken. Die Zeit wird in beiden Fällen mit nativen Funktionen ausgewertet. Die Implementierungen der einzelnen Programmiersprachen, sowie die berechneten Programmlaufzeiten sind im Folgenden dargestellt. Die beiden Skripte werden jeweils fünf mal ausgeführt und der Mittelwert der berechneten Zeiten bestimmt.

```
1 tic;  
2 load(' ../dat/11-48-21_hrv.mat ');  
3 plot(Res.HRV.TimeVar.mean_HR);  
4 tac;
```

Listing 3.3: Line-Plot MATLAB Implementierung

```
1 import time  
2 import pandas as pd  
3 import h5py  
4 import matplotlib.pyplot as plt  
5  
6 start = time.time()  
7 f = h5py.File(' ../dat/11-48-21_hrv.mat ')  
8 df = pd.DataFrame(f.get('Res/HRV/TimeVar/mean_HR')).T  
9  
10 df.plot(y=0, kind='line')  
11 end = time.time()  
12 print(end - start)  
13 plt.show()
```

Listing 3.4: Line-Plot Python Implementierung

Durchlauf	MATLAB	Python
1	0.303867 s	0.359526 s
2	0.307696 s	0.351593 s
3	0.295474 s	0.354158 s
4	0.301647 s	0.352104 s
5	0.299218 s	0.350182 s
Mittelwert	0.301580 s	0.353512 s

Tabelle 3.3: Ergebnisse der Geschwindigkeits-Messungen

MATLAB ist bei jeder Ausführung um ca. 17% schneller als das Python Skript. Zudem muss die Komplexität der beiden Skripte betrachtet werden. In MATLAB benötigt man lediglich zwei Zeilen Code und keine zusätzlichen Bibliotheken, während das Python Skript vier Code Zeilen und drei zusätzlichen Bibliotheken in Anspruch nimmt. Außerdem muss hier beachtet werden, dass das Anzeigen des Plots unter Python nicht mit in die Berechnung der Zeit aufgenommen werden kann, da alle Code-Zeilen nach „plt.show“ auch erst nach dem Schließen des Plot-Fensters angezeigt werden.

TODO: finale Entscheidung begründen

4 Konzept

4.1 System-Grundlage und Struktur

Als Grundlage der Schnittstelle soll eine MATLAB-Standalone-Applikation dienen, welche eine zentrale Benutzeroberfläche für die Funktionalitäten der Anforderungen stellt. Dies ermöglicht das spätere Kompilieren der Applikation und die Möglichkeit diese als einfaches Programm auf einem Rechner zu starten und von hier aus eine Auswahl über die einzelnen Funktionen der Applikation zu bekommen. Zudem kann so die Applikation einfach verteilt und auf mehreren Rechner installiert werden ohne dass zusätzliche Bibliotheken oder Programmierkenntnisse benötigt werden.

Innerhalb der Implementierung ergeben sich zwei große Bereiche in die die Applikation aufgeteilt wird und die beiden Hauptfunktionalitäten der Anforderung, das erstellen geeigneter Samples in Kubios HRV Premium und das grafische Darstellen der erzeugten medizinischen Messdaten, repräsentieren. Diese beiden Hauptfunktionalitäten sollen dazu voneinander abgekapselt implementiert werden, wobei das Visualisieren die Basis der Applikation darstellt und die Konfiguration der Samples auf dieser aufbaut bzw. von hier aus aufgerufen wird.

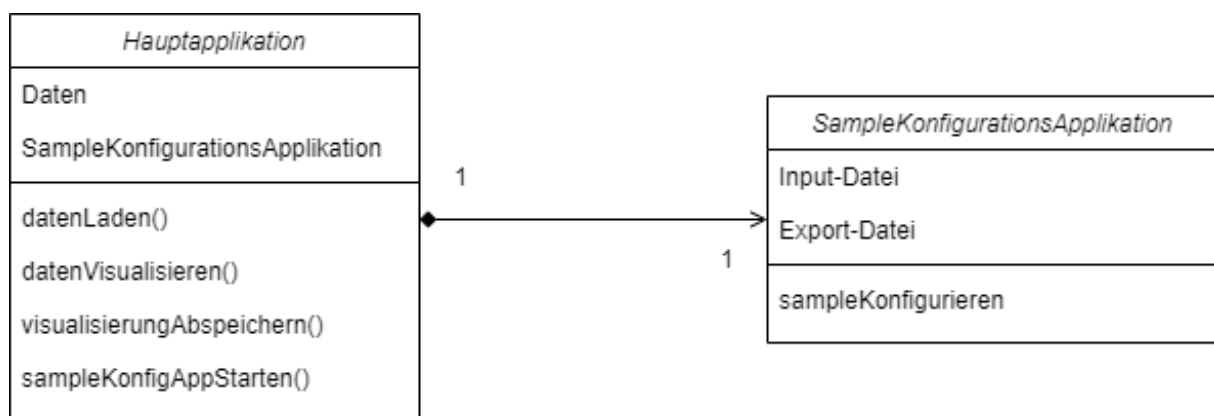


Abbildung 4.1: Konzept Klassendiagramm

4.2 Ablauf

Die grundlegenden Benutzung der Applikation soll wie folgt stattfinden. Der Benutzer kann entweder Samples für eine Messung konfigurieren und diese dann im nächsten Schritt innerhalb des Tools visualisieren oder, bei bereits angelegten Samples, diese direkt im Graphen anzeigen. So kann die Erstellung neuer Visualisierungsdaten, aber auch die Wiederverwendbarkeit bereits angelegter Daten gewährleistet werden. Innerhalb eines Ablaufdiagramms lässt sich dies folgendermaßen beschreiben:

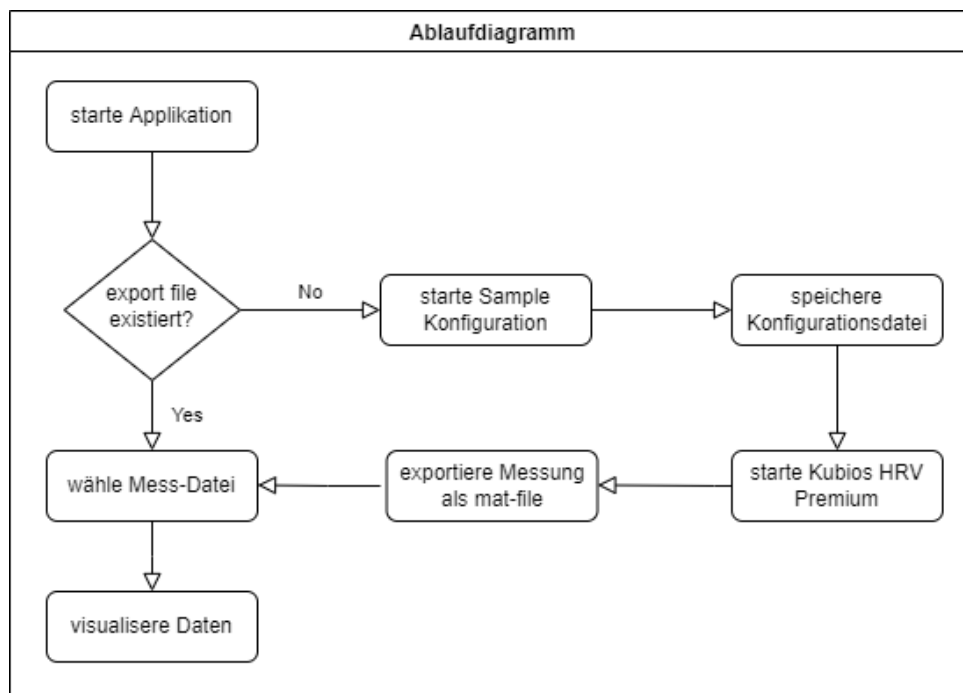


Abbildung 4.2: Ablaufdiagramm des Konzepts

Für das Erstellen der Samples wurde folgender Ablauf konzipiert. Der Nutzer ruft die Funktion in der Hauptapplikation auf und erhält ein PopUp, in welchem er die Messung und die Parameter zur Konfigurierung einträgt. Im nächsten Schritt wird dies an Kubios HRV Premium weitergeleitet, die Messung wird dann mittels einer API Schnittstelle oder Ähnlichem in die entsprechenden Samples aufgeteilt und eine Export-Datei im mat-Format erstellt. Danach erfolgt das automatische Einlesen der Datei in die Applikation, der Nutzer kann dann die entsprechend enthaltenen Messparameter visualisieren.

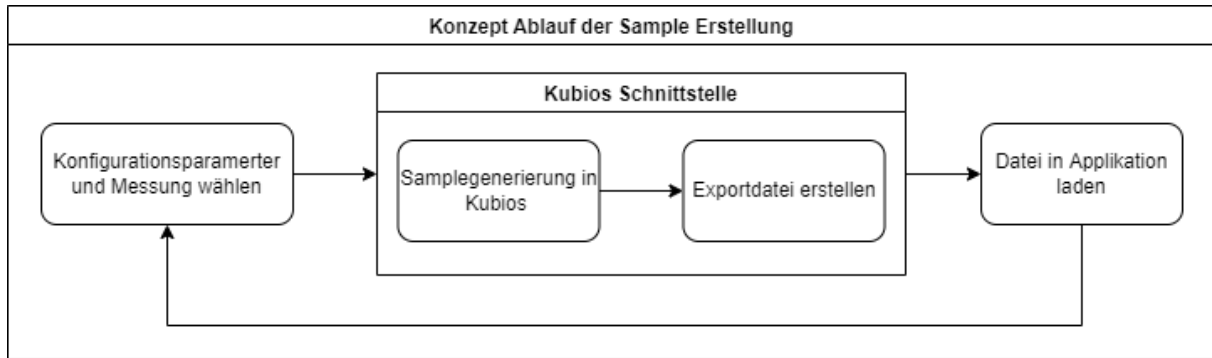


Abbildung 4.3: Konzept Ablauf für das Erstellen der Samples

Für das Visualisieren einzelner Messparametern innerhalb der Applikation wird ein Ablauf wie folgt modelliert. Nach dem das in Kubios erstellte MAT-File in die Applikation geladen wird, kann eine der enthaltenen Parameter in einer Liste ausgewählt werden. Die dazugehörigen Messdaten werden dann im bereits vorhandenen Koordinatensystem angezeigt. Das Visualisieren anderer Parameter erfolgt durch das Auswählen eines anderen Listen-Elements und kann beliebig oft wiederholt werden.

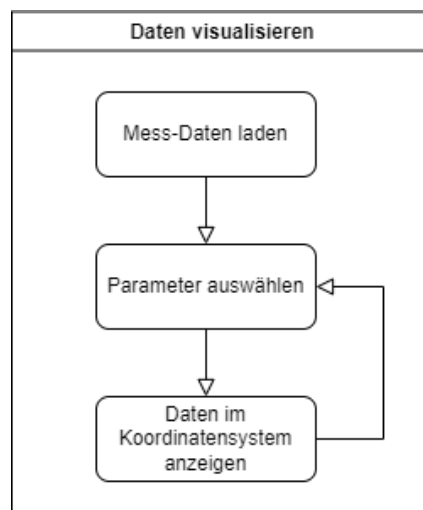


Abbildung 4.4: Konzept Ablauf für das Visualisieren von Daten

4.3 Applikations-Design

Das Applikations-Design soll anhand der grundlegenden Regeln des UI Designs implementiert. Dazu gehört in erster Linie der Fokus auf Übersichtlichkeit, Konsistenz der Design-Elemente und Positionierungen sowie einfache Bedienbarkeit, welche dem Nutzer im Gedächtnis bleibt und keine große Zeit zur Einarbeitung in Anspruch nimmt. Zudem

soll schnell zu erkennen sein, wo welche Funktion aufgerufen und verwendet werden kann. Zuletzt sollen Fehler passend verhindert werden, um das reibungslose Arbeiten mit der Applikation garantieren zu können.

Um diese Grundsätze und Regeln umsetzen zu können, soll die Oberfläche keine Verschachtlungen enthalten. Alle grundlegenden Funktionen liegen zum Start der Applikation offen und können von dort aus auch verwendet werden. So soll direkt auf der Startseite der Graph zur Visualisierung sowie die Auswahl der Messparameter zu sehen sein. Kleinere Hilfsfunktionen und Informationen, dazu gehören beispielsweise das Laden von Messdaten oder das Abspeichern der Visualisierung, sollen in eine Menubar am oberen Rand der Applikation integriert werden. Da diese typisch für einen Großteil der Programme ist und so einen Wiedererkennungswert für die Nutzer bietet.

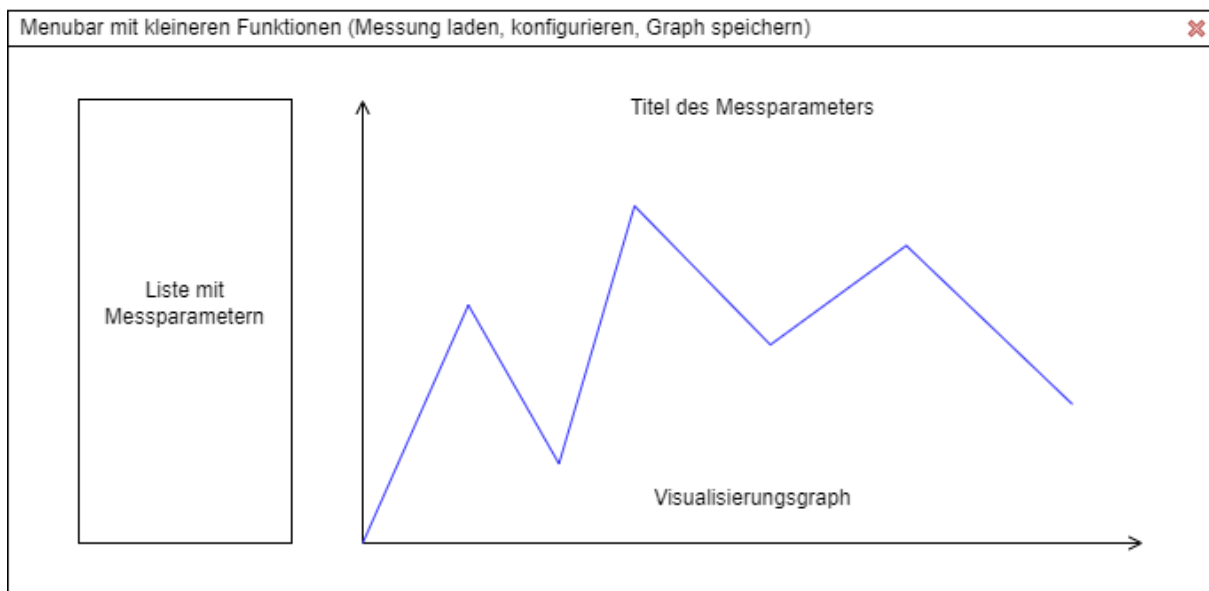


Abbildung 4.5: Konzept Ablauf für das Design

5 Implementierung des Tools

5.1 Grundstruktur der Applikation

Die Grundstruktur der Applikation basiert auf zwei im MATLAB App Designer erstellten Applikationen. Der Hauptapplikation, welche das Aufrufen einzelner Funktionalität ermöglicht, sowie den Graphen der erstellen Visualisierungen darstellt, und die Applikation zur Konfiguration der Samples, welche parallel zur Hauptapplikation als Pop-Up geöffnet werden kann, jedoch nicht ohne diese verwendbar ist. Als dritte Struktur wurde die sogenannte WithSamplesParser-Klasse erstellt, welche alle Funktionalitäten bezüglich der Messdatenverarbeitung und Darstellung enthält. Dazu gehören das Einlesen und Umstrukturieren der Messdaten auf ein passendes Format, sowie das Erstellen eines Graphen mit der passenden Konfiguration. Diese Funktionen werden dabei ebenfalls aus der Hauptapplikationen gestartet und benötigen diese deshalb auch als Grundlage.

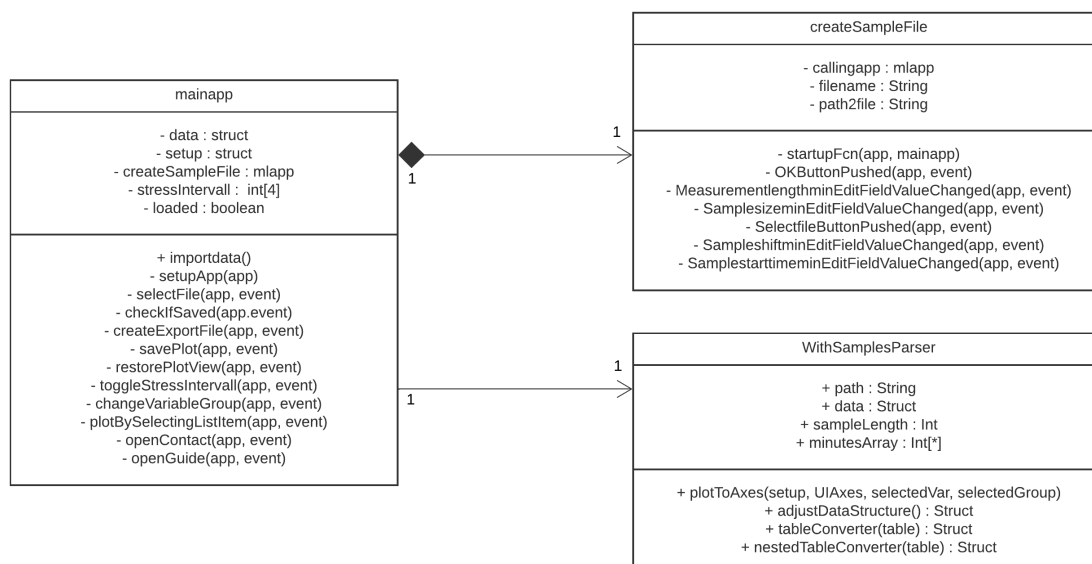


Abbildung 5.1: Klassendiagramm der fertigen Applikation

5.1.1 Layout und Komponenten

Der MATLAB App Designer setzt bei der Erstellung von Applikation auf Code-Generierung. Der App Designer ist eine eigenständige MATLAB Applikation, welcher über eine Auswahl bereits vorgefertigter Komponenten, wie verschiedene beispielsweise Buttons, Container oder Eingabefelder verfügt. Um eine Applikation zu erstellen, können diese einfach beliebig kombiniert und visuell angepasst werden. Aus den verwendeten Komponenten wird dann eine Klasse generiert, welche beim Ausführen instanziiert wird. Diese Klasse enthält einen genauen „Bauplan“ der späteren Applikation. Funktionen, welche beispielsweise durch das Klicken eines Buttons ausgelöst werden basieren auf sogenannten **Callbacks** (siehe Setup der Applikation). Diese werden direkt als Methoden der App-Klasse definiert und können nicht generiert werden. Jede Komponente kann mehrere dieser Callbacks besitzen.

```
1 % Properties that correspond to app components
2 properties (Access = public)
3     HREvaluationToolUIFigure    matlab.ui.Figure
4     GridLayout                  matlab.ui.container.GridLayout
5     ListBox                     matlab.ui.control.ListBox
6     DropDown                    matlab.ui.control.DropDown
7     UIAxes                      matlab.ui.control.UIAxes
8 end
```

Listing 5.1: Auszug einiger verwendeter App-Komponenten

Die Grundstruktur der implementierten Applikation bildet die **Figure-Komponente**. Diese legt die Fenstergröße fest und ist die Grundlage der Applikation in der die einzelnen Unterkomponenten eingefügt werden. Um eine gewisse Grundstruktur in das Aussehen zu integrieren wird ein **Grid-Layout** nächste Stufe der Komponenten-Hierarchie eingebaut. Dies ermöglicht das korrekte Positionieren der einzelnen Komponenten innerhalb der Applikation und damit auch die symmetrische Konsistenz von Rahmen- und Komponentenabständen, welche im grundlegenden Design-Konzept definiert wurden. Außerdem fügt es der Applikation eine umfassende Responsiveness hinzu, wodurch das Verwenden der Applikation in unterschiedlichen Monitorgrößen und Fenstergrößen ermöglicht werden kann ohne ungewollte Nebeneffekte herbeizuführen.

TODO: Die Applikation besteht aus den folgenden Elementen... (UIAxes, ListBox, etc.)

TODO: Bild der fertigen Applikation

5.2 Setup der Applikation

Die Export-Daten von Kubios haben ein Problem in der Konsistenz ihrer Datensätze. So sind innerhalb der MAT-Datei lediglich reine Zahlenwerte und die spezifischen Identifier der Parameter zu finden. Es fehlen wichtige Elemente wie beispielsweise ein genauer Name und die Einheit der Parameter. Um nun die Visualisierungen einzelner Parameter mit allen wichtigen Informationen versorgen zu können, muss eine entsprechende Mapping-Datenbank vorhanden sein, welche manuell konfiguriert werden muss.

Die Applikation lädt sich hierzu beim Starten eine entsprechende MAT-Datei als Variable in den Workspace. Diese Datei enthält die Mappings der einzelnen Parameter sortiert nach ihrer jeweiligen Gruppierung wie sie in Kubios HRV Premium angegeben sind. Diese Gruppierungen sind hier Time-Domain, Nonlinear sowie Frequency-Domain im Zeit- und Frequenzbereich, wobei die beiden Frequency-Domain-Gruppen auf ein Mapping reduziert werden kann, da diese die gleichen Parameter enthält. So ergeben sich drei Setupstrukturen im Tabellenformat mit den Spalten Index, Kurzbezeichner, Kurzbeschreibung, Einheit und Gruppierung. Mit Hilfe des von Kubios erzeugten Index lassen sich nun alle dazugehörigen Informationen aufrufen und verwerten. Zudem wird diese Setup-Datei verwendet um zu Beginn alle vorhandenen Parameter unter ihrer jeweiligen Gruppierung zur ListBox-Komponente hinzuzufügen, sodass der Nutzer später diese Auswählen kann.

Zusätzlich wurde eine Helper-Funktion geschrieben, welche es ermöglicht aus einer Excel-Datei das Setup zu generieren, um die Handhabung bzw. Veränderung des Mappings zu ermöglichen. Dazu können einfach neue Elemente in die Tabelle mit aufgenommen oder bereits bestehende Einträge verändert bzw. gelöscht. Diese Funktion ist jedoch der Entwicklung vorenthalten, da das Verändern bestehender Workspace-Variable nach der Installation oder während des Betriebs der Applikation nicht passend umsetzbar ist.






 index	 short	 description	 unit	 type
'mean_RR'	'Mean RR'	'The mean of RR intervals'	'Milliseconds'	'Statistics'
'std_RR'	'SDNN'	'Standard deviation of RR intervals'	'Milliseconds'	'Statistics'
'mean_HR'	'Mean HR'	'The mean heart rate'	'1/min'	'Statistics'
'std_HR'	'STD HR'	'Standard deviation of instantaneous heart rate values'	'1/min'	'Statistics'

Abbildung 5.2: Auszug aus der Struktur der Setup-Datei

5.3 Erstellung der Samples

TODO: Wim schreibt was er gemacht hat :) TODO: Die Applikation besteht aus den folgenden Elementen... (UIAxes, ListBox, etc.)

5.4 Visualisierung der Daten

Im folgenden Teil der Implementierung sollen die einzelnen Schritte und Funktionen dargestellt werden, welche zum schlussendlichen Visualisieren und Auswerten der Daten beitragen. Dazu gehören die Verarbeitung der Daten und die Datenstruktur im Backend der Applikation, sowie das Darstellen des Graphen mit Hilfe der Plot-Funktion und die dazugehörigen erweiternden Funktionen wie beispielsweise das Hervorheben des Belastungsintervalls.

5.4.1 Datenstruktur und Datenverarbeitung

Um das schnelle Visualisieren und Verarbeiten der Daten muss eine zentrale Datenstruktur entwickelt werden, die hohe Übersichtlichkeit, gute Erweiterbarkeit, sowie möglichst minimalistisch und kompakt aufgebaut ist. Zudem ist es wichtig ein passendes Format für die Visualisierungsfunktionen von MATLAB zu verwenden, da diese nur bestimmte Datenformate als Eingabe-Parameter akzeptiert. Die Rohdaten sind dabei sehr verschachtelt und es sind lange Zugriffsketten nötig, um an die korrekte Datenwerte zu gelangen. Außerdem fehlt es oft an Konsistenz. Während ein Großteil der Parameter als Tabelle vorliegt und mittels der passenden Reihe und Spalte ausgelesen werden können, sind einige in eine weitere Struktur verschachtelt oder es liegen Parameter in einer Liste unter einem allgemeinen Bezeichner vor und müssen mit dem passenden Index extrahiert werden. Deshalb benötigen die eingelesene Daten eine vorläufige Anpassung bevor sie visualisiert werden können. Aus diesem Grund muss das eingelesene MAT-File angepasst werden, um nicht für jeden Parameter eine separate Zugriffsfunktion implementieren zu müssen.





Fields	 ApEn	 SampEn	 MSE	 DFA
1	0.9958	1.2521	1x20 double	1x1 struct
2	1.0231	1.2626	1x20 double	1x1 struct
3	1.0030	1.1885	1x20 double	1x1 struct
4	0.9566	1.1586	1x20 double	1x1 struct
5	0.9420	1.1534	1x20 double	1x1 struct
6	1.0103	1.2465	1x20 double	1x1 struct

Abbildung 5.3: Verschachtelungsprobleme der unverarbeiteten Export-Daten

Das Einlesen des MAT-Files erfolgt über den Menüpunkt „Load measurement“. Dieser öffnet das Dateisystem des Betriebssystems und ermöglicht das Auswählen einer Datei. Gleichzeitig wird ein Objekt der Parser-Klasse **WithSampleParser** instanziiert, welche die Inhalte der Datei speichert und eine konsistente Datenstruktur anlegt.

Als Grundkonzept wird dazu ein Struct angelegt. Dieses lässt sich mit einem Python-Dictionary vergleichen, in dem die Elemente mittels Key-Value-Paaren abgespeichert sind. Ein Datensatz lässt sich dann einfach mit Hilfe des entsprechenden Keys aufrufen. Zusätzlich ist es möglich dieses zu verschachteln und unter einem Key ein weiteres Key-Value-Paar zu speichern. So lässt sich eine hierarchische Beziehung zwischen den unterschiedlichen Parametern und Datensätzen erzeugen.

Das erzeugte Struct besteht aus vier Ebenen. Auf obersten Ebene sind die Gruppierungen der Parametern zu finden. Für die Parameter der Herzmessdaten wurden hier die bereits von Kubios HRV Premium vordefinierten Gruppierungen Time-Domain, Nonlinear sowie Frequency-Domain im Zeit- und Frequenzbereich verwendet. Unterhalb dieser Bezeichner befinden sich dann alle zugehörigen Parameter. Im Falle der Frequency-Domain wird hier noch die Ebene der Darstellung, also Zeit- oder Frequenzbereich, eingeschoben, da hier die Parameter-Bezeichner identisch sind und so eine Unterscheidung unmöglich wäre. Auf tiefster Ebene ist dann jedem Parameter ein Datensatz als Liste mit genauen Werten zugeordnet. Ein Eintrag in der Liste entspricht dabei immer dem Mittelwert des Parameters für den Zeitraum der zuvor konfigurierten und exportierten Samples. Diese Liste bildet eine gute Grundlage für die Visualisierung der Daten mit Hilfe der vorhandenen MATLAB-Funktionen. Außerdem ist es möglich diese für gegebene Anforderungen zu manipulieren.

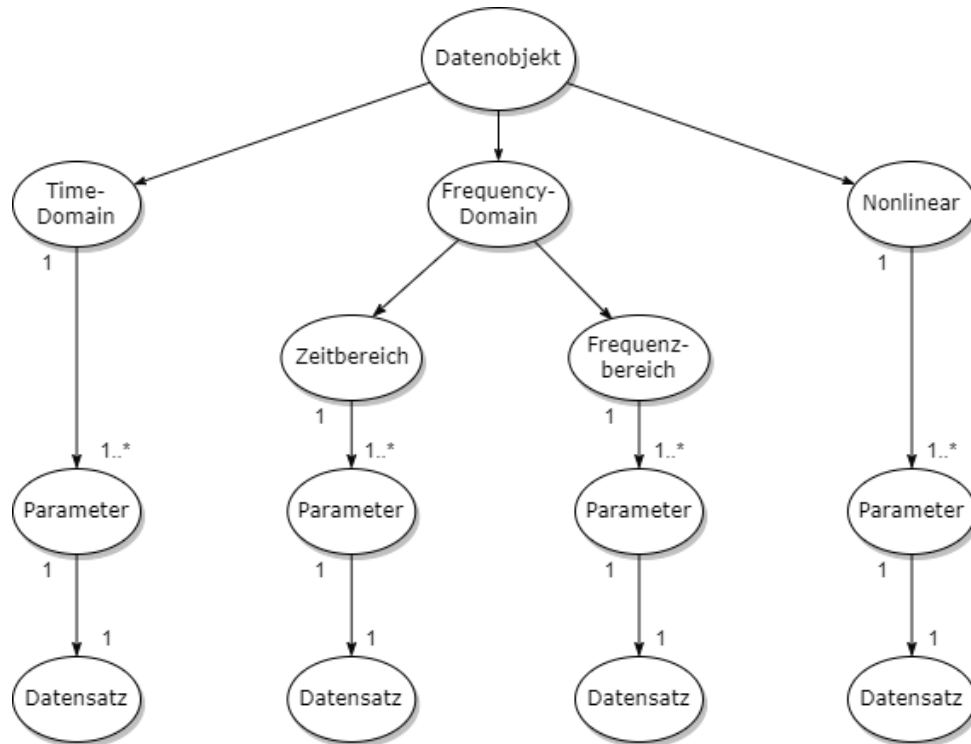


Abbildung 5.4: Baumstruktur zur Veranschaulichung der Datenstruktur

Um diese Struktur bilden zu können werden einige Funktionen zur Anpassungen durchlaufen. Zuerst wird die Tabelle invertiert. Das lässt sich darauf zurückführen, dass ein Großteil der Daten innerhalb einer Spalte gespeichert und der Zugriff beim Visualisieren der Daten mit Hilfe der MATLAB-Funktion auf als Reihe gespeicherte Daten präferiert wird. Hierzu wurde eine Methode implementiert, welche alle Spalten und die Parameter-Bezeichner einliest, invertiert und als Struct abspeichert.

```

1 function tempStruct = tableConverter(obj, table)
2     transposed = struct2cell(table');
3     names = fieldnames(table)';
4     tempStruct = struct();
5
6     for idx = 1:length(names)
7         theData = cell2mat(transposed(idx,:));
8         tempStruct.(char(names(idx))) = theData;
9     end
10 end
    
```

Listing 5.2: Konvertieren einer MATLAB Tabelle in ein Struct

Für das Extrahieren von verschachtelten Parametern innerhalb der Tabelle müssen zusätzliche Schritte ausgeführt werden. Hierbei muss die Struktur, welche anstatt eines

direkten Wertes vorliegt, aufgelöst werden. Im ersten Schritt werden dabei alle unwichtigen Parameter, die keinen Sample-Bezug besitzen gelöscht. In einer Schleifen werden dann alle restlichen Parameter kopiert und zu den restlichen Einträgen des zuvor erstellten Structs hinzugefügt. Somit ist die Verschachtelung von zwei Ebenen auf eine reduziert.

```
1 function cleanTable = nestedTableConverter(obj, table, column,
    structToRemove)
2     tempStruct = rmfield(table(1).(column), structToRemove);
3
4     for idx = 2:length(table)
5         tempStruct = [tempStruct; rmfield(table(idx).(column),
            structToRemove)];
6     end
7
8     cleanTable = obj.tableConverter(tempStruct);
9 end
```

Listing 5.3: Auflösen verschachtelter MATLAB Tabellen in ein Struct

Um eine Reduzierung mehrerer Parameter auf einen allgemeinen Bezeichner rückgängig zu machen wurde folgendes Vorgehen implementiert. Innerhalb des Datensatzes handelt es sich hier um den Parameter „Multiscale entropy“, welcher zwanzig unterschiedliche Skalierungsfaktoren besitzt. Diese sind innerhalb eines Bezeichners als Liste aus zwanzig Werten angelegt. Für die korrekte Visualisierung müssen diese jedoch als einzelne Parameter der Struktur angezielt werden, wobei der Datenwert an der ersten Stelle der Liste dem Wert für den Skalierungsfaktor eins entspricht. Die Auflösung erfolgt mittels zweier Iterationen. Zuerst werden alle Listen aneinander gehängt, sodass eine Matrix der einzelnen Werte entsteht. Mittels der zweiten Iteration wird diese Matrix in die einzelnen Structs aufgeteilt und der allgemeine Bezeichner mit dem Schleifen-Index als Key gesetzt.

```
1 tempMatrix = [];
2 cleanMSE = struct();
3 for idx = 1:length(obj.data.Res.HRV.NonLinear)
4     tempMatrix = cat(1, tempMatrix, obj.data.Res.HRV.NonLinear(idx).MSE);
5 end
6 tempMatrix = tempMatrix';
7 for id = 1:size(tempMatrix, 1)
8     [cleanMSE.(['MSE' num2str(id)])] = tempMatrix(id,:);
9 end
```

Listing 5.4: Auflösen eines Parameters in einer Liste anhand des Beispiel-Parameters MSE

Zum Schluss muss noch beachtet werden, dass die grundlegenden Daten keine direkten x-Werte, also exakte Zeitpunkte der Parameter-Messung, enthalten. So kann ein Datensatz lediglich mit seiner Stelle in der Liste, also für den ersten Wert $x=0$, zweiter Wert $x=1$, usw. verknüpft und visualisiert werden. Da die einzelnen Daten aber die Mittelwerte einzelner Samples sind und deren Länge im Normalfall größer einer Minute sind, muss hier ein passender Eintrag in der Datenstruktur hinzugefügt werden, um das Zuordnen der Messeinträge mit der relativen Messzeit zu ermöglichen. Hierzu wird beim Einlesen der Daten die Sample-Länge mit Hilfe der vorhandenen Informationen in der Messdatei berechnet. Dazu wird zuerst die Messzeit in Minuten umgerechnet und durch die Anzahl der Einträge, also Messwerte pro Parameter, geteilt. Da die Sample-Länge meist aber kein Teiler der Messzeit ist und es somit oft zu einem unvollständigen Sample am Ende der Messung kommt, wird die berechnete Sample-Länge noch abgerundet. Man erhält so eine genaue Minutenzahl. Hieraus lässt sich nun auch eine Liste beginnend mit der Sample-Länge erstellen, deren Schrittweite ebenfalls einer Sample-Länge entspricht. Dieser wird an das fertige Daten-Struct angehängt und als Werte für die x-Achse verwendet.

```
1 % calculate the length of a single sample
2 obj.sampleLength = floor((obj.data.Res.CNT.Length / 60) / obj.data.Res.
    HRV.Param.Nbr_Segment);
3 % convert into an array to use it in the plot later
4 obj.minutesArray = obj.sampleLength:obj.sampleLength:obj.data.Res.HRV.
    Param.Nbr_Segment * obj.sampleLength;
```

Listing 5.5: Ermittlung von Sample-Länge und Erstellung der x-Werte

5.4.2 Plot-Funktion

Die Hauptaufgaben der Plot-Funktion (dt. graphisch darstellen) sind das Auswählen des richtigen Datensatzes anhand einer gegebenen Gruppe und dem zugehörigen Messparameter, sowie das Verknüpfen der Daten mit dem Koordinatensystem und das Anpassen der Koordinatenachsen auf den passenden Stand.

Die Plot-Funktion wird dabei durch das Auswählen eines Messparameters in der Liste aller konfigurierten Parameter aufgerufen. Als Methode der Parser-Klasse **WithSampleParser** hat sie direkten Zugriff auf die eingelesenen Messdaten. Sie erhält dabei die setup-Konfiguration, das Koordinatensystem als UIAxes-Objekt des App Designers, den ausgewählten Parameter und dessen Gruppe. Zu Beginn wird anhand der übergebenen Gruppe, der Link auf die passende Setupstruktur und die Tabelle in der Datenstruktur ermittelt.

```
1 if selectedGroup == "Time-Domain"
2     setuplink = setup.variablesStruct.timeDomain;
3     datalink = obj.data.Res.HRV.Statistics;
4     ...
```

Listing 5.6: Beispiel der Link-Zuordnung für Parameter-Gruppe

Mit den beiden Links wird nun nach dem speziellen Parameter gefiltert und dessen Index innerhalb der Tabelle zurückgegeben. Hiermit lassen sich nun das Daten-Array, der vollständige Bezeichner, die Kurzbeschreibung und die Einheit des Parameters bestimmen.

```
1 % get id of the given variable in the data structure
2 idx = find(ismember({setuplink(:).index}, selectedVar));
3
4 % get all associated informations
5 short = setuplink(idx).short;
6 description = setuplink(idx).description;
7 unit = setuplink(idx).unit;
```

Listing 5.7: Index des Parameters und dazugehörige Informationen filtern

Das Daten-Array wird nun als Liste der y-Werte dem Koordinatensystem übergeben und dort als Datensatz hinterlegt. Die passenden Werte der x-Achse wurden bereits innerhalb des Einlesevorgangs der Messdaten ermittelt und repräsentieren die Länge der Messung als relative Minutenangabe. Diese werden ebenfalls mit an das Koordinatensystem angehängt. Mit Hilfe der MATLAB-eigenene Scatter-Funktion wird dem UIAxes-Objekt die Information übertragen, dass es sich um einen Scatter-Plot (dt. Streu-Graphen) handelt und die x- und y-Werte-Paare als Punkte ohne verbindende Linie dargestellt werden sollen. Hier können ebenfalls Konfigurationen für die Darstellung der Daten-Tupel im Graphen übergeben werden.

```
1 scatter(UIAxes, obj.minutesArray, datalink.(selectedVar), ...
2         'MarkerEdgeColor',[0 .5 .5],...
3         'MarkerFaceColor',[0 .7 .7],...
4         'LineWidth',1.5);
```

Listing 5.8: MATLAB Scatter-Funktion

Im letzten Schritt wird das Aussehen des Koordinatensystems angepasst, so dass es die wichtigen Informationen wie Achsenbeschriftungen aber auch Titel und Kurzbeschreibung enthält und der Bildausschnitt angenehm für den Ersteindruck ist.

5.4.3 Zusätzliche Funktionen

TODO: Belastungsintervall, Plot speichern, Plot Restoren

6 Auswertung von Messungen mit Hilfe des HRV Evaluation Tools

7 Kritische Würdigung und Ausblick

Anhang

Glossar

Glossareintrag

Ein Glossar beschreibt verschiedenste Dinge in kurzen Worten.