



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

22.10.2021 - 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. Zudem wird eine Auswertung realer Messdaten mit Hilfe des Tools durchgeführt und die Erkenntnisse zusammengetragen. 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
Formelverzeichnis	VII
Listings	VIII
1 Medizinische Grundlagen	1
1.1 Herzparameter	1
1.1.1 Herzfrequenz (Hf)	1
1.2 Herzfrequenzvariabilität (HRV)	3
1.2.1 Messung	4
1.2.2 Nutzen	5
1.2.3 Parameter	6
1.2.4 Frequenzabhängige Parameter	10
1.2.5 Nicht-lineare Parameter	13
1.2.6 Zusammenfassung	16
2 Aufgabenstellung	18
3 Rahmenbedingungen	19
3.1 Export-Datenstruktur	19
3.2 Programmiersprache	20
3.2.1 Python	21
3.2.2 MATLAB	21
3.2.3 Performance-Vergleich	22
4 Konzept	29
4.1 System-Grundlage und Struktur	29
4.2 Ablauf	30
4.3 Applikations-Design	31

5	Implementierung des Tools	33
5.1	Grundstruktur der Applikation	33
5.1.1	Layout und Komponenten	34
5.2	Setup der Applikation	36
5.3	Erstellung der Samples	37
5.4	Visualisierung der Daten	40
5.4.1	Datenstruktur und Datenverarbeitung	40
5.4.2	Plot-Funktion	44
5.4.3	Zusätzliche Funktionen	46
6	Auswertung von Messungen mit Hilfe des HRV Evaluation Tools	49
7	Kritische Würdigung und Ausblick	50
	Anhang	A
	Glossar	B

Abkürzungsverzeichnis

BSP Board Support Package

Abbildungsverzeichnis

1.1	R-R-Intervall	5
1.2	HRV - Zeitabhängige Parameter	7
1.3	Leistungsdichtespektrum	11
1.4	HRV - Frequenzabhängige Parameter	12
1.5	HRV - Nicht-lineare Parameter	14
1.6	Poincaré-Diagramm	15
1.7	SD1 und SD2	16
3.1	Visualisierung der Zeitmessung	25
4.1	Konzept Klassendiagramm	29
4.2	Ablaufdiagramm des Konzepts	30
4.3	Konzept Ablauf für das Erstellen der Samples	31
4.4	Konzept Ablauf für das Visualisieren von Daten	31
4.5	Konzept Ablauf für das Design	32
5.1	Klassendiagramm der fertigen Applikation	33
5.2	Fertige Applikation mit geladener Messung	36
5.3	Auszug aus der Struktur der Setup-Datei	37
5.4	Unterapplikation Sample Configurator	38
5.5	Angepasstes Ablaufdiagramm der Sample-Erstellung mit manuellen Schritten	40
5.6	Verschachtelungsprobleme der unverarbeiteten Export-Daten	41
5.7	Baumstruktur zur Veranschaulichung der Datenstruktur	42
5.8	Beispiel einer Visualisierung mit Belastungsintervall	47

Tabellenverzeichnis

3.1	Übersicht der technischen Daten des Testmediums	22
3.2	Zeitmessungen der Datensatzgrößen und Implementierungen	25
3.3	Ergebnisse der Geschwindigkeits-Messungen	27
5.1	Beispielhafter Aufbau einer Sample-Konfigurationsdatei	37

Formelverzeichnis

1.1 Herzminutenvolumen (HMV)	1
1.2 Maximale Herzfrequenz	1
1.3 SDNN	7
1.4 RMSSD	9
1.5 pNN50	9
3.3 Berechnung der abhängigen Variable - OLS	23

Listings

3.1	Methode der kleinsten Quadrate MATLAB Implementierung	24
3.2	Methode der kleinsten Quadrate Python Implementierung	24
3.3	Line-Plot MATLAB Implementierung	26
3.4	Line-Plot Python Implementierung	26
5.1	Auszug einiger verwendeter App-Komponenten	34
5.2	Erstellung der Sample-Konfigurations-Datei	39
5.3	Konvertieren einer MATLAB Tabelle in ein Struct	42
5.4	Auflösen verschachtelter MATLAB Tabellen in ein Struct	43
5.5	Auflösen eines Parameters in einer Liste anhand des Beispiel-Parameters MSE	43
5.6	Ermittlung von Sample-Länge und Erstellung der x-Werte	44
5.7	Beispiel der Link-Zuordnung für Parameter-Gruppe	45
5.8	Index des Parameters und dazugehörige Informationen filtern	45
5.9	MATLAB Scatter-Funktion	45
5.10	Funktion zum Togglen des Belastungsintervalls	46
5.11	Funktion zum Speichern der Visualisierung	47
5.12	Funktion zur Wiederherstellung der Visualisierung	48

1 Medizinische Grundlagen

1.1 Herzparameter

Das menschliche Herz kann mit vielen verschiedenen Werten parametrisiert werden. Der wohl bekannteste Parameter ist dabei die Herzfrequenz (Hf). 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.

1.1.1 Herzfrequenz (Hf)

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 Herzschlägen. Sie ist einer der wichtigsten Parameter des menschlichen Herzens. Ein gesunder Mensch hat keinen vollständig gleichmäßigen Herzschlag. Für Personen, die sich nicht mit der HRV auskennen, hört sich dies erst einmal verwunderlich an. Im Folgenden soll jedoch die HRV und deren Bedeutung erläutert werden.

Um die HRV besser erklären zu können, wird folgendes Beispielszenario verwendet: Das Herz eines Menschen schlägt 60 Mal pro Minute. Die Person ruht währenddessen vollständig, sodass von einem konstanten Puls von 60 ausgegangen wird. Nun stellt sich die Frage nach der Zeit zwischen den einzelnen Herzschlägen. Auf den ersten Block liegt es Nahe davon auszugehen, dass zwischen jedem Herzschlag die gleiche Zeit, also eine Sekunde vergeht. Dies ist jedoch nicht richtig. Die Zeit zwischen den einzelnen Schlägen des Herzens, die HRV, variiert deutlich. Bei einem gesunden Menschen beträgt die HRV einen Wert von ungefähr 100ms. Am gezeigten Beispiel ist dies eine Abweichung von bis zu 10 %.

Mithilfe dieser Abweichungen lassen sich Aussagen über den Gesundheitszustand des Menschen treffen. Dabei spielt das bereits erklärte vegetative Nervensystem eine wichtige Rolle. Ein gutes Zusammenspiel zwischen Sympathikus und Parasympathikus sind Teil eines gesunden Menschen. Mithilfe der HRV kann dieses Zusammenspiel gut messbar und

in Zahlen ausgedrückt werden". Grundsätzlich lässt sich sagen, dass eine hohe HRV ein Zeichen für einen gesunden Organismus ist. So können zum Beispiel eine ausgewogene Ernährung oder ausreichend Schlaf zu einer Erhöhung der Herzfrequenzvariabilität. Im Gegensatz dazu wirken sich Stress, gesundheitsschädliche Medikamente oder Schlafmangel negativ auf die HRV aus.

Auch im Bereich des Sports spielt die HRV eine wichtige Rolle, da sie eine große Aussagekraft über die Regeneration des Athleten hat. Auch gibt Trainingspläne, in welchen sich die Trainings nach dem Wert der HRV richten. So wurde bereits 2007 ein Artikel veröffentlicht, welcher das Training mit Beachtung der HRV beschreibt. In diesem Artikel wurde das Ausdauertraining von 26 Männern untersucht. Diese wurden dabei in drei Gruppen aufgeteilt. Eine Gruppe trainierte dabei einen festgelegten Trainingsplan von sechs Einheiten (4 Hochintensitätstraining, 2 Trainings mit niederer Intensität) .

Die zweite Gruppe trainierte individuell nach dem Wert ihrer HRV. War die Herzfrequenzvariabilität bei der Messung am Morgen hoch, so wurde mit hoher Intensität trainiert. Bei einer niederen HRV wurde nur mit niederer Intensität oder gar nicht trainiert. Die dritte Trainingsgruppe war eine Kontrollgruppe. Am Ende des vierwöchigen Trainingsprogramm konnte bei der individuell nach HRV Wert trainierenden Trainingsgruppe eine stärkere Verbesserung der Ausdauer (Untersuchung verschiedener Parameter, welche Aussagen über die Ausdauer zulassen (Beispiel: VO2max)) im Vergleich zur Gruppe mit festem Trainingsplan festgestellt. werden.<https://pubmed.ncbi.nlm.nih.gov/17849143/>

Die Wichtigkeit der HRV wurde nun bereits deutlich. Wie diese gemessen wird und welche Aussagekraft die verschiedenen Parameter der HRV haben soll nun genauer erläutert werden.

1.2.1 Messung

In der folgenden Grafik ist eine EKG Messung eines menschlichen Herzens zu sehen.

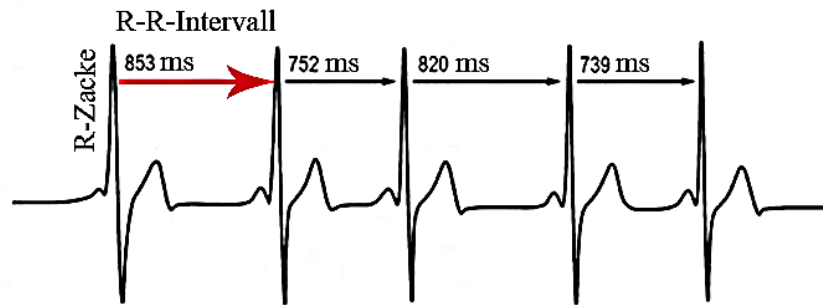


Abbildung 1.1: R-R-Intervall

Gut erkennbar sind dabei die hohen R Zacken. Diese werden benutzt, um die Herzfrequenzvariabilität zu berechnen. Für die Berechnung wird die Zeit zwischen zwei aufeinanderfolgenden R Zacken benötigt. Diese Zeitspanne wird als R(-)R-Intervall bezeichnet. Mit diesem Begriff muss jedoch vorsichtig umgegangen werden, da er leicht verwechselt wird. Beim Messen des Blutdrucks wird ebenfalls die Abkürzung RR verwendet. In diesem Zusammenhang ist jedoch von der Blutdruckmessmethode nach *Riva-Rocci* die Rede. Um eine Verwechslung zu vermeiden, wird das R(-)R-Intervall der HRV in der Literatur häufig als NN-Intervall bezeichnet. Dieser unterscheidet sich inhaltlich jedoch leicht vom Begriff des RR-Intervall. Das RR-Intervall ist die Zeit zwischen allen R-Zacken der Messung. Während einer Messungen können Störungen auftreten, welche zusätzliche SZacken"verursachen. Diese verfälschen die Messung und nennen sich *Artefakte*<https://flexikon.doccheck.com/de/Artefakt>. IM RR-Intervall werden diese trotzdem berücksichtigt.

Das NN-Intervall ist die Zeit zwischen den *normalen* R-Zacken. Hier werden die fälschlich gemessenen R-Zacken nicht berücksichtigt und herausgefiltert. Im Laufe dieser Ausarbeitung ist meist vom bereits gefilterten NN-Intervall die Rede.

Im Gegensatz zum Herzfrequenz gibt es bei der HRV nicht nur einen Wert. Die Herzfrequenzvariabilität besitzt viele Parameter, welche alle unterschiedlich berechnet werden. **Bevor deren Berechnung erklärt wird, muss kurz erläutert werden, weshalb es überhaupt mehrere Parameter gibt und warum diese eine Daseinsberechtigung haben. Dafür muss zuerst Grundsätzliche Wichtigkeit der HRV verstanden werden.**

1.2.2 Nutzen

Mit den verschiedenen Parametern lassen sich grundsätzliche Feststellungen über den Gesundheitszustand treffen. Besonders für Aussagen über das vegetativen Nervensystem

ist die HRV sehr geeignet. Da die Unterschiede der Parameter der HRV noch nicht genau erklärt wurde, ist im Folgenden mit dem Begriff HRV immer die Gesamtheit aller Parameter gemeint. Es stellt sich jedoch die Frage wieso nicht einfach die unter 1.1.1 erklärte Herzfrequenz genutzt wird um Aussagen über das vegetative Nervensystem treffen zu können. Um diese Frage zu beantworten, muss sich ein Unterschied zwischen HRV und Hf bewusst gemacht werden. Zwar sind sowohl HRV als auch Hf vom vegetativen Nervensystem abhängig, jedoch unterscheiden sie sich stark in ihrer Beeinflussbarkeit. Die Hf ist beispielsweise durch kurzfristige Faktoren wie Aufregung beeinflussbar, welche die Messung verfälschen können. Die HRV ist zwar auch Abhängig von Aufregung, jedoch in viel kleinerem Maße. Auch kann der Messproband seinen eigenen Herzschlag wahrnehmen, was ebenfalls einen Einfluss auf die Messung haben kann. Der Zeitabstand zwischen zwei Herzschlägen ist von außen nicht spürbar und kann so vom Probanden nicht bewusst beeinflusst werden.

Grundsätzlich lässt sich sagen, dass die HRV aufgrund ihrer sehr starken Abhängig vom vegetativen Nervensystem und relativ geringem Einfluss äußerer Bedingungen sehr gut für Messungen rund um das vegetative Nervensystem geeignet ist.

1.2.3 Parameter

Die HRV kann mit vielen verschiedenen Parametern dargestellt werden. Alle Parameter lassen sich dabei eine der folgenden Gruppen zuordnen:

1. Zeitabhängige Parameter
2. Frequenzabhängige Parameter
3. Nicht-lineare Parameter

Diese drei Gruppen sollen nun genauer betrachtet werden. Dabei soll vor allem die Abgrenzung zu den anderen Gruppen betont werden. Die einzelnen Gruppen sollen dabei mit Beispielen erklärt werden. Die Gruppen der verschiedenen Parameter unterscheiden sich grundlegend.

Zeitabhängige Parameter

Zeitabhängige Parameter sind mathematisch am einfachsten zu greifen, da keine mathematischen Operationen auf die Messwerte angewendet werden. Es werden einfache Rechnungen mit den Messungen durchgeführt. Wichtige Parameter für Zeitabhängige Parameter sind:

Parameter	Unit	Description
SDNN	ms	Standard deviation of NN intervals
SDRR	ms	Standard deviation of RR intervals
SDANN	ms	Standard deviation of the average NN intervals for each 5 min segment of a 24 h HRV recording
SDNN index (SDNNI)	ms	Mean of the standard deviations of all the NN intervals for each 5 min segment of a 24 h HRV recording
pNN50	%	Percentage of successive RR intervals that differ by more than 50 ms
HR Max – HR Min	bpm	Average difference between the highest and lowest heart rates during each respiratory cycle
RMSSD	ms	Root mean square of successive RR interval differences
HRV triangular index		Integral of the density of the RR interval histogram divided by its height
TINN	ms	Baseline width of the RR interval histogram

Abbildung 1.2: HRV - Zeitabhängige Parameter

Alle dargestellten Parameter zu erklären würde den Umfang der Arbeit übersteigen, sodass nur einige ausgewählte Parameter in ihrer Berechnung erläutert werden. Bei weiterem Interesse an den anderen Parametern können genauer Informationen unter <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5624990/> nachgelesen werden.

SDNN (Standard Deviation of NN-Intervals)

Der SDNN beschreibt die Standard-Abweichung der NN-Intervalle und wird in der Einheit ms angegeben. Er zeigt wie stark die einzelnen Werte vom Durchschnitt abweichen. Mit folgender Formel kann der SDNN berechnet werden:

$$SDNN = \sqrt{\frac{1}{N-1} * \sum_{i=1}^N (RR_i - \overline{RR})^2} \quad (1.3)$$

Wenn der SDNN einen beispielhaften Wert von 40 ms beträgt ist die Zeit zum nächsten Herzschlag 40 ms länger oder kürzer als die Zeit zum vorhergehenden Herzschlag. Grundsätzlich lässt sich sagen, dass je höher der SDNN, desto besser.

Der SDNN wird häufig genutzt um eine Aussage über die Gesamtheit des autonomen Nervensystems zu treffen, weswegen er weder Sympathikus noch Parasympathikus zugeordnet ist. PDF MARIO, SEITE 15 **Über die Dauer einer Messung nimmt der SDNN stetig zu.** Über die Dauer einer Messung nimmt der SDNN stetig zu. Daher ist es äußerst wichtig, dass wenn Werte des SDNN verglichen werden Messungen mit gleicher Länge als Berechnungsgrundlage vorliegen. Auch sollten die äußeren Umstände so ähnlich wie möglich sein. <https://knowledge.time2tri.me/de/articles/die-wichtigsten-hrv-parameter-teil-1-zeitbezogene-parameter>

SDNN Werte welcher einer Messung über 24 Stunden entspringen, werden häufig genutzt um Patienten mit Herzkrankheiten zu untersuchen und einzuteilen. So werden Patienten mit einem SDNN unter 50 ms als nicht gesund eingestuft. Bei einem Wert zwischen 50 und 100 ms wird die Gesundheit als beeinträchtigt angesehen und ab einem Wert von über 100 ms gilt der Patient als gesund. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5624990/pdf/fpubh-05-00258.pdf> SEITE 4

RMSSD (Root Mean Square of successive differences)

Wenn in Pulsuhren, Apps oder Artikeln nur von der HRV die Rede ist, jedoch kein genauer Parameter spezifiziert wird, ist mit einer hohen Wahrscheinlichkeit der RMSSD gemeint. Er ist nämlich einer der bekanntesten Parameter der HRV und wird dazu benutzt, um Aussagen über die (kurzzeitige) Erholungsfähigkeit des Körpers auszusagen. Erholung ist für einen Menschen essenziell und daher auch Indikator für einen gesunden Organismus. Deshalb können über den Wert der Erholungsfähigkeit, den RMSSD, Aussagen über grundsätzlichen Gesundheitszustand und Fitness abgeleitet werden. Ein hoher RMSSD Wert zeigt dabei, dass der Körper sich gut erholen kann. Auch lassen sich positive Schlüsse auf den Umgang mit (psychischem) Stress ziehen. Auf der anderen Seite deutet ein niedriger RMSSD Wert eine weniger gute Erholungsfähigkeit des Körpers an. Ursachen können dabei physische oder psychische Anstrengung sein. Auch kann ein niedriger RMSSD Wert ein Indiz für Krankheiten sein.

Die Studie <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2943000/> untersucht beispielsweise den Zusammenhang zwischen RMSSD und SUDEP(sudden unexpected death in epilepsy, deutsch:plötzlich und unerwarteter Todesfall bei Epilepsie). Dabei wird festgestellt, dass das SUDEP-Risiko bei Epilepsiepatienten mit niederen RMSSD Werten höher ausfällt. Der RMSSD und grundsätzlich die HRV stehen also in direktem Zusammenhang zum SUDEP-Risiko.

Aufgrund der großen Aussagekraft über die Erholungsfähigkeit im Körper, kann der RMSSD im autonomen Nervensystem klar dem Parasympathikus zugeordnet werden und unterscheidet sich hiermit klar vom SDNN.

Die Berechnung des RMSSD sieht wie folgt aus:

$$RMSSD = \sqrt{\frac{1}{N-1} * \sum_{i=1}^N (RR_{i+1} - RR_i)^2} \quad (1.4)$$

In der Berechnung werden die einzelnen aufeinanderfolgenden NN-Intervalle quadratisch addiert. **Danach wird dies durch die Anzahl der NN-Intervalle geteilt.** Abschließend wird die Wurzel gezogen, um die durchschnittliche Abweichung zwischen aufeinanderfolgende Intervalle zu erhalten. Angegeben wird der RMSSD in Millisekunden.

<https://hrv-herzratenvariabilität.de/2017/09/rmssd-der-hrv-wert-fuer-die-erholungsfahigkeit/>

pNN50

Ein weiterer wichtiger zeitabhängiger Parameter ist der pNN 50. Dieser kann ebenso wie der RMSSD dazu benutzt werden, um Aussagen über die Erholungsfähigkeit des Körpers zu treffen und ist daher ebenfalls dem Parasympathikus zugeordnet. Grundsätzlich sind sich die zwei Parameter sehr ähnlich. Während der RMSSD jedoch zur Messung der Kurzzeitvariabilität genutzt wird, werden mit dem pNN 50 eher Aussagen über die Spontanvariabilität getroffen. (MARIO PDF S15) Ein weiterer großer Unterschied, welchen den pNN50 von den bisher vorgestellten Parametern SDNN und RMSSD unterscheidet, ist die Einheit. SDNN und RMSSD werden in Millisekunden angegeben, wohingegen der pNN50 Prozent als Einheit besitzt.

Der pNN50 lässt sich folgendermaßen berechnen:

$$pNN50 = \frac{NN50}{N-1} * 100\% \quad (1.5)$$

Wie auf den ersten Blick erkannt werden kann, hängt der pNN50 von einem weiteren HRV Parameter namens NN50 ab. Dieser ist einfach eine Zahl, welche die Anzahl der aufeinanderfolgenden NN-Intervall mit einer Differenz von >50ms beschreibt.

Der NN50 wird durch die Anzahl aller Paare geteilt und in Prozent gerechnet. Somit erhält man den prozentualen Anteil aller **Herzschlagpaare, welche eine Differenz von mehr als 50 ms aufweisen. Ein Beispiel könnte folgendermaßen aussehen:**

Differenz zum vorgehenden NN Intervall	1	2	3	4	5
ms	75	32	53	60	32

Der NN50 wäre in diesem Fall drei, und der pNN würde einen Wert von 60 % betragen.

Nun wurden einige wichtige zeitabhängige Parameter der HRV erläutert. Diese wurden dabei nur oberflächlich beschrieben, um einen kurzen Überblick zu erhalten. Auch wurde nicht genauer auf den jeweiligen empfohlenen Messzeitraum der Parameter und andere Eigenschaften der Messungen eingegangen, da dies den Rahmen der Arbeit übersteigt.

1.2.4 Frequenzabhängige Parameter

Die bisher genannten Parameter der HRV lagen alle im Zeitbereich. Eine weitere Möglichkeit Parameter der HRV darzustellen liegt jedoch im Frequenzbereich, weshalb die daraus berechneten Werte *Frequenzabhängige Parameter* heißen. Im Folgenden sollen wichtige Parameter aus dem Frequenzbereich beschrieben werden. Dafür sind allerdings zuerst einige wichtige mathematische Umformungen nötig, welche bekannt sein müssen

Mathematische Grundlagen

Für eine Darstellung im Frequenzbereich müssen zuerst die Messwerte, welche im Zeitbereich sind in den Frequenzbereich transformiert werden. Dies kann mit einer *Fast-Fourier-Transformation (FFT)* erfolgen, welche jedoch nur auf ein zeitdiskretes Signal angewendet werden kann. Da die Messungen der NN-Intervalle kein zeitdiskretes Signal ist, muss zuerst noch eine Interpolation durchgeführt werden. Danach existiert ein durchgehendes zeitdiskretes Signal, welches Messwerte in äquidistanten Abständen besitzt. Mit diesem Signal ist eine Fast-Fourier-Transformation möglich und die Messwerte können in den Frequenzbereich übertragen werden. Veranschaulicht wird dies in in einem *Leistungsdichtespektrum*, womit die Verteilung der verschiedenen NN-Intervall Frequenzen dargestellt werden kann. Mithilfe dieses Spektrums kann veranschaulicht werden, welche Leistung unter welcher Frequenzen anfällt. Ein beispielhaftes Spektrum ist unter 1.3 dargestellt. Die Berechnung und Darstellung des Leistungsdichtespektrums ist das Kennzeichen der frequenzabhängigen Analyse der HRV.

Da sowohl Interpolation als auch FFT mathematisch komplex sind und dem Thema der HRV Analyse keine weitere Erkenntnis bringt, werden die mathematischen Hintergründe nicht weiter vertieft.

Darstellung

Das Leistungsdichtespektrum kann im Kontext der HRV laut der Task Force of the European Society of Cardiology and the North American Society of Pacing and Electrophysiology (QUELLE: Force T. Standards of Measurement, Physiological Interpretation and Clinical Use. Task Force of the European Society of Cardiology and the North American Society of Pacing and Electrophysiology. Circulation. 1996;93:1043–1065. [PubMed] [Google Scholar] <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8230044/HASHTAGB18-sensors-21-03998>) in vier verschiedene Frequenzbänder aufgeteilt werden.

1. ULF - Ultra Low Frequency mit Frequenzen $<0.003\text{Hz}$
2. VLF - Very Low Frequency mit Frequenzen zwischen 0.0033 und 0.04 Hz
3. LF - Low Frequency mit Frequenzen zwischen 0.04 und 0.15 Hz
4. HF - High-Frequency mit Frequenzen zwischen 0.15 und 0.4 Hz

Das ULF Band ist vor allem bei langen Messungen wichtig und betrachtungsrelevant. Da die Messungen, mit welchen sich diese Arbeit befasst meist relativ kurz (~ 45 Minuten) sind, wird das ULF nicht betrachtet. Alle Frequenzen die dem ULF zugeordnet werden würden, werden im Folgenden als Teil des VLF betrachtet. Ein Beispiel für ein solches Diagramm kann folgendermaßen aussehen:

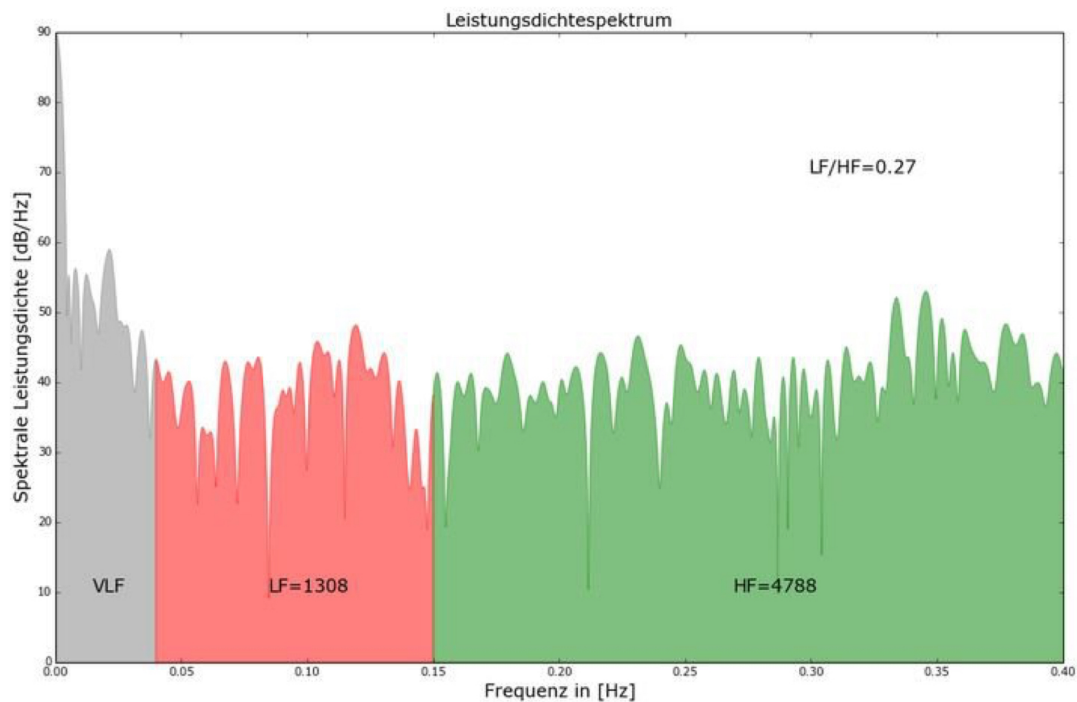


Abbildung 1.3: Leistungsdichtespektrum

QUELLE: <https://knowledge.time2tri.me/de/articles/die-wichtigsten-hrv-parameter-teil-2-frequenzbezogene-parameter>

In 1.3 kann die Einteilung der verschiedenen Frequenzbänder gut erkannt werden. Die Messung, welche dem Spektrum zu Grunde liegt, ist ein 10-minütige Messung in Ruhe (<https://knowledge.time2tri.me/de/articles/die-wichtigsten-hrv-parameter-teil-2-frequenzbezogene-parameter>). Aufgrund dieser kurzen Messdauer ist im Schaubild kein ULF eingetragen.

Mithilfe dieser Bereiche können nun folgende frequenzabhängigen Parameter der HRV berechnet werden.

Parameter	Unit	Description
ULF power	ms ²	Absolute power of the ultra-low-frequency band (≤ 0.003 Hz)
VLF power	ms ²	Absolute power of the very-low-frequency band (0.0033–0.04 Hz)
LF peak	Hz	Peak frequency of the low-frequency band (0.04–0.15 Hz)
LF power	ms ²	Absolute power of the low-frequency band (0.04–0.15 Hz)
LF power	nu	Relative power of the low-frequency band (0.04–0.15 Hz) in normal units
LF power	%	Relative power of the low-frequency band (0.04–0.15 Hz)
HF peak	Hz	Peak frequency of the high-frequency band (0.15–0.4 Hz)
HF power	ms ²	Absolute power of the high-frequency band (0.15–0.4 Hz)
HF power	nu	Relative power of the high-frequency band (0.15–0.4 Hz) in normal units
HF power	%	Relative power of the high-frequency band (0.15–0.4 Hz)
LF/HF	%	Ratio of LF-to-HF power

Abbildung 1.4: HRV - Frequenzabhängige Parameter

Nun sollen einige wichtige Parameter noch genauer untersucht werden.

HF power

Der Parameter HF power kann berechnet werden, indem die Fläche des HF Bands bestimmt wird. Anschaulich kann dies als grün gefärbte Fläche in Abbildung 1.3 betrachtet werden. Als Einheit trägt HF power meist ms², kann jedoch auch in Prozent angegeben werden. Mit dem Parameter HF power können parasympathische Aktivitäten gemessen werden, wodurch er im autonomen Nervensystem dem Parasympathikus zugeordnet ist. Aufgrund dieser Tatsache steht er in einem starken Zusammenhang mit RMMSD und pNN 50,

welche ebenfalls die Aktivität des Parasympathikus messen.

VLF- und LF power können analog dazu berechnet werden, indem die Fläche im jeweiligen Intervall bestimmt wird. Der Parameter VLF power kann beispielsweise genutzt werden um Aussagen über SADS (Sudden arrhythmic death syndrome, deutsch: Plötzliche Arrhythmiesyndrome) zu treffen. Auch ist ein niedriger VLF power Wert ein Anzeichen für PTSD (Post Traumatic Stress Order, deutsch: posttraumatische Belastungsstörung) . QUELLE:<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5624990/pdf/fpubh-05-00258.pdf>

LF/HF

Ein weiterer Parameter ist das Verhältnis zwischen den Werten von LF und HF. Da LF power eher vom Sympathikus getrieben wird und HF power stark vom Parasympathikus abhängt, kann das Verhältnis aus beiden Werten als Parameter für das Zusammenspiel und die Balance zwischen Sympathikus und Parasympathikus gesehen werden. Ein hoher LF/HF Wert spricht daher eher für eine Dominanz des Sympathikus, während ein geringer Wert des LF/HF Verhältnisses für eine höhere parasymphatische Aktivität spricht. <https://knowledge.time2tri.me/><https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5624990/pdf/fpubh-05-00258.pdf>

1.2.5 Nicht-lineare Parameter

Die letzte Gruppe der HRV nennt sich *nicht-lineare Parameter* und ist "moderner als zeitabhängig- oder frequenzabhängige- Parameter. Dies soll bedeutet, dass sie noch nicht so lange existieren als die bereits erläuterten Gruppen. Nichtsdestotrotz sind sie ein wichtiger Teil der Erhebung von HRV Daten und gewinnen immer mehr an Aufmerksamkeit und Bedeutung.

Die Frequenzabhängige Analyse kennzeichnet sich durch das Berechnen des Leistungsdichtespektrum. Aus diesem werden dann alle Parameter bestimmt. Auch bei der zeitabhängigen Analyse gibt es einen eindeutigen Weg wie Parameter bestimmt werden. Bei der nicht-linearen Analyse ist dies nicht so. Es gibt keinen eindeutigen "Rechenweg" für die Analyse und Berechnung der Parameter.

Eine weit verbreitete Möglichkeit ist es, die verschiedenen NN-Intervalle in einem Poincaré-Diagramm darzustellen. Aus diesem können anschließend einige Parameter berechnet werden. Eine ausführliche Liste der nicht-linearen Parameter der HRV ist in Abbildung 1.5 dargestellt.

Parameter	Unit	Description
S	ms	Area of the ellipse which represents total HRV
SD1	ms	Poincaré plot standard deviation perpendicular the line of identity
SD2	ms	Poincaré plot standard deviation along the line of identity
SD1/SD2	%	Ratio of SD1-to-SD2
ApEn		Approximate entropy, which measures the regularity and complexity of a time series
SampEn		Sample entropy, which measures the regularity and complexity of a time series
DFA $\alpha 1$		Detrended fluctuation analysis, which describes short-term fluctuations
DFA $\alpha 2$		Detrended fluctuation analysis, which describes long-term fluctuations
D ₂		Correlation dimension, which estimates the minimum number of variables required to construct a model of system dynamics

Abbildung 1.5: HRV - Nicht-lineare Parameter

Der Poincaré-Plot und die dazugehörige Berechnung der Parameter $SD1$ und $SD2$ soll im Folgenden genauer betrachtet werden.

Poincaré-Diagramm

In einem Poincaré-Diagramm werden aufeinanderfolgende NN-Intervalle gegeneinander eingetragen. Dies kann man sich wie folgt vorstellen. Der Wert des Intervalls NN_n wird als x-Achsen Wert eingezeichnet. Das darauffolgende Intervall NN_{n+1} ist der dazugehörige y-Achsen Wert. Dadurch entsteht ein Punkt im Diagramm mit den Koordinaten $P(NN_n | NN_{n+1})$. Das nächste Intervall NN_{n+2} ist wieder x-Achsen Wert mit dazugehörigem y-Achsen Abschnitt bei NN_{n+3} . Mithilfe dieser paarweisen Zuordnung werden alle NN-Intervalle der Messung in das Diagramm eingetragen, wodurch eine große Punktwolke entsteht. Ein Beispiel für solch ein Poincaré-Plot im Kontext der HRV kann in 1.6 gesehen werden. **Im Anschluss wird meist eine Ellipse um den Mittelwert gezeichnet. Diese hilft bei der späteren Berechnung der Parameter und kann in Abbildung 1.7 gesehen werden.**

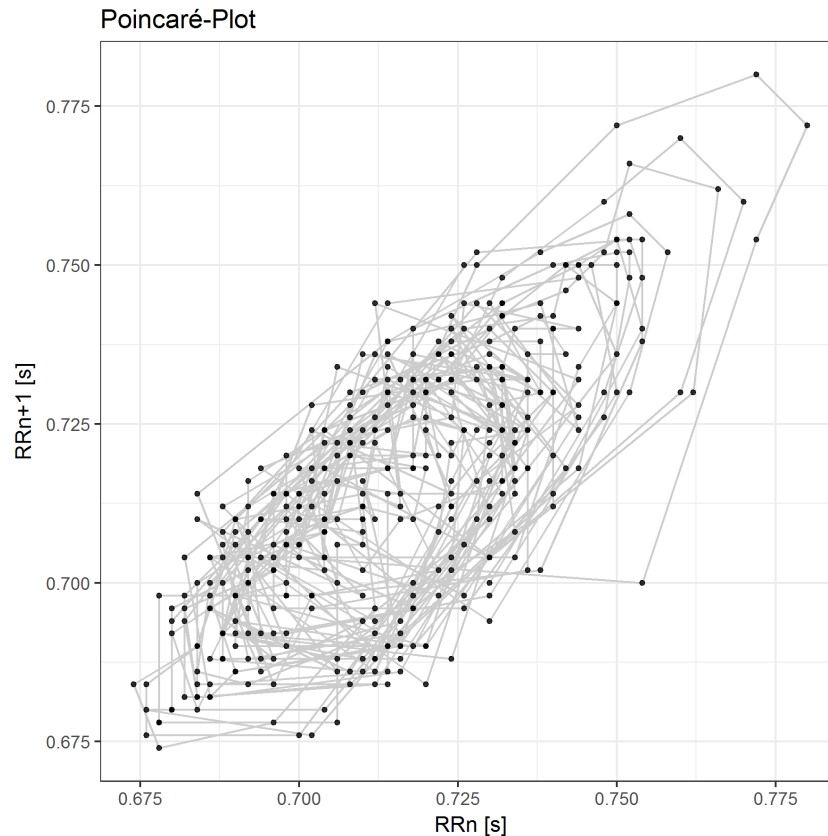


Abbildung 1.6: Poincaré-Diagramm

<https://hrv-herzratenvariabilität.de/2017/07/berechnung-des-poincare-diagramms-aus-rr-intervallen/> Um mithilfe des Poincaré-Diagramms Aussagen über die Herzratenvariabilität treffen zu können, kann die Form der entstandenen Punktwolke visuell analysiert werden. Bei einer geringen Variabilität des Herzens liegen die Punkte alle nah beisammen und die Punktwolke ist klein. Bei Menschen mit einer hohen HRV ist die ellipsenförmige Punktwolke größer. Auffällige Intervalle, welche auf Messfehler oder Herzfehler zurückzuführen sind, werden im Poincaré-Diagramm als Ausreißer dargestellt. Diese sind weit von der Punktwolke entfernt. In Abbildung 1.6 können einige dieser Ausreißer im oberen rechten Teil erkannt werden.

SD1 und SD2

Die bekanntesten Parameter der nicht-linearen Analyse sind SD1 und SD2. Beide dieser Parameter beziehen sich auf die Streuung der Punkte um den Mittelpunkt im Poincaré-Diagramm. Mathematisch betrachtet die Standardabweichung.

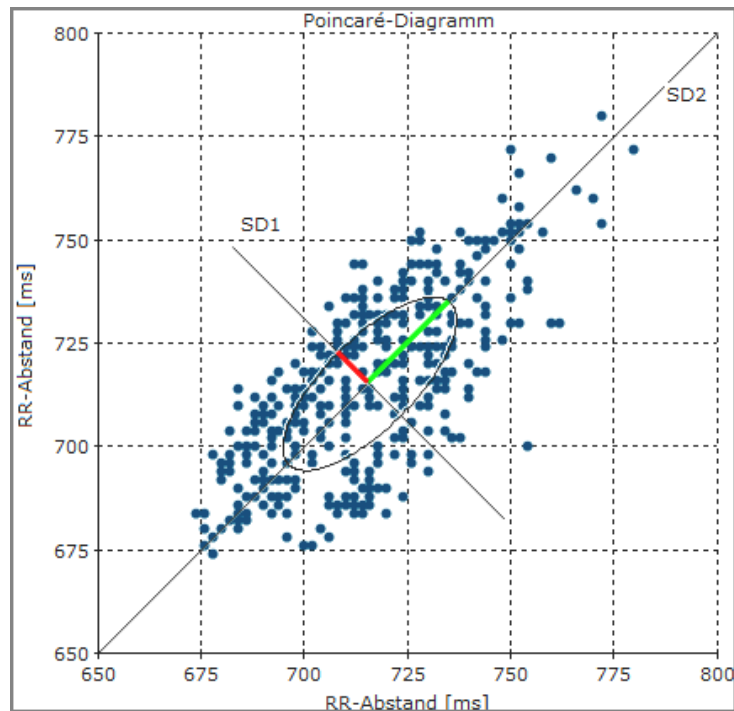


Abbildung 1.7: SD1 und SD2

<https://hrv-herzratenvariabilität.de/2017/07/berechnung-des-poincare-diagramms-aus-rr-intervallen/>

Zur Berechnung des SD1 wird die Standardabweichung in Richtung der linken Diagonalen gemessen. Der Parameter SD1 gibt Hinweise zur Kurzzeitvariabilität. Er ist identisch zum zeitabhängigen Parameter RMSSD. <https://onlinelibrary.wiley.com/doi/10.1002/mus.25573><https://www.105-00258.pdf>

Der Parameter SD2 wird durch die Bestimmung der Standardabweichung in der rechten Diagonalen bestimmt. Er drückt eher die Langzeitvariabilität des Herzens aus, sodass er in einem Zusammenhang mit dem frequenzabhängigen Parameter LF power steht. Das Verhältnis der beiden Parameter SD1/SD2 ist ähnlich wie das bereits erläuterte Verhältnis HF/LF. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5624990/pdf/fpubh-105-00258.pdf>

1.2.6 Zusammenfassung

In diesem Abschnitt wurden die verschiedenen Analysen und Parameter der HRV erläutert. Es werden immer neue Parameter entdeckt, welche berechnet werden können. Auffallend

ist dabei, dass viele Parameter zusammenhängen. Beispielsweise sind RMSSD und SD1 der gleiche Wert, obwohl diese auf völlig verschieden Arten berechnet werden. Dies zeigt, dass es nicht die eine richtige Analyse gibt, sondern alle ihre Berechtigung haben. Welche Analyse wann eingesetzt wird, hängt hauptsächlich mit der persönlichen Präferenz zusammen.

Zusätzlich lässt sich durch das Verwenden verschiedener Analysen die Aussagekraft des Ergebnisses verstärken. Wenn beispielsweise eine Aussage über die parasympathische Aktivität getroffen werden soll, ist das Ergebnis belastbarer, wenn RMSSD, HF power und SD1 berechnet wird, als wenn nur ein Parameter angeführt wird.

Die Berechnung aller Parameter aus einer Messung per Hand vorzunehmen, würde extrem viel Zeit in Anspruch nehmen. Daher gibt es bereits Programme und Anwendungen, die diese Berechnung der gewünschten Parameter übernehmen. Außerdem ist die Vergleichbarkeit und Verlässlichkeit der aus den Auswertungen resultierenden Ergebnissen größer, wenn diese mit einem öffentlichen Programm durchgeführt werden. Während dieser Arbeit wurde daher ebenfalls ein Programm zur Auswertung verschiedener HRV Messungen benutzt. Die dafür Anwendung heißt *Kubios HRV Premium* und soll im Nachfolgenden kurz erläutert werden.

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.[**csv**]

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.[**mat**]

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.[**dataSciLang**]

Python ist hierbei die beliebteste und meistverwendete Programmiersprache.[**dataSciLang**]
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.[[whatIsPython](#)][[pythonBeginners](#)]

3.2.2 MATLAB

MATLAB ist eine High-Level-Programmiersprache 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.[[whatIsMatlab](#)][[whatIsMatlab2](#)]

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 Berechnung einer Regressionsfunktion

Zuerst wird ein allgemeiner Geschwindigkeitstest durchgeführt. Hierzu soll sowohl in Python als auch in MATLAB die Methode der kleinsten Quadrate (engl. ordinary least squares - kurz OLS) zur Regression anhand eines zufällig generierten Datensatzes durchgeführt werden. Mit Hilfe von Ausgleichsrechnungen kann eine möglichst genau passende,

modellabhängige Modellkurve in eine Wolke von Datenpunkten gelegt werden. In der Methode der kleinsten Quadrate wird die Summe der quadratischen Abweichungen der Kurve von den beobachteten Punkten minimiert. Ausgleichsrechnungen sind ein wichtiger Bestandteil der Datenanalyse und sind größtenteils rechenintensiv wenn sie auf große Datenmengen angewendet werden, deshalb bietet diese Methode ein gutes Beispiel zum Vergleich der Programmiersprachen und ihrer Performance.[**kleinsteQuadrate**]

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

$$\beta = \begin{bmatrix} 10 \\ -0,5 \\ 0,5 \end{bmatrix} \quad (3.1)$$

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:

$$\mu_x = \begin{bmatrix} 10 \\ 10 \end{bmatrix}, \sigma_x = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \quad (3.2)$$

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:

$$Y = Xb + \epsilon \quad (3.3)$$

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 ihre 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 python_time = np.zeros((len(n_array),2))
17 count=0
18
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
30         python_boot", number=1)
    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

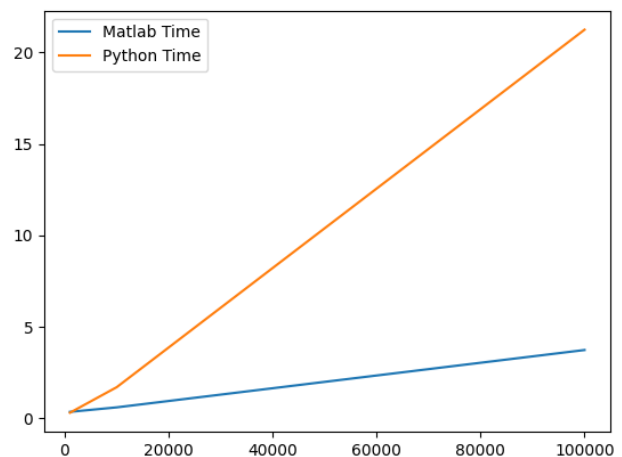


Abbildung 3.1: Visualisierung der Zeitmessung

An den Zeitmessung für verschieden große Datensätze lässt sich gut erkennen, dass MATLAB und Python für kleine Datenmengen eine ähnlich große Berechnungszeit benötigen. Mit zunehmender Datenmenge wächst auch die Berechnungszeit. MATLAB skaliert hier jedoch um ein Vielfaches besser und es lässt sich eine stetige steigende Lücke zwischen den Ausführungszeiten erkennen.[olsMatPy]

Analyse anhand der einfachen Datenvisualisierung

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.[**pandas**][**h5py**]

```
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. 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. Da die Geschwindigkeit bei kleinen simplen Skripten nicht wirklich aussagekräftig ist, soll hier vor allem das Augenmerk auf die Umsetzung gesetzt werden. Deshalb muss vor allem 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.

Fazit und Begründung der Entscheidung

MATLAB ist für die Umsetzung des Evaluierungs-Tools die schnellere und passendere Programmiersprache. Dies lässt sich vor allem auf die allgemein bessere zeitliche Performance innerhalb mathematischer Berechnungen aber auch beim Einlesen und Visualisieren von Datensätzen zurückführen. Zudem ist die MAT-Datei, welche im späteren Verlauf der Studienarbeit durch Kubios erstellt wird ein nativer Datentyp von MATLAB. Dadurch lässt sich allgemein ein einfacherer Umgang mit den Daten garantieren, sowie die Reduzierung von Code-Komplexität durch einfacher gehaltene Datenverarbeitung ermöglichen. So lässt sich die innere Struktur der MAT-Datei in MATLAB direkt einsehen und es sind keine zusätzlichen Bibliotheken und Funktionen notwendig.

Betrachtet man die Komplexität bezogen auf die Erstellung von Applikationen und das dazugehörigen Design, lässt sich keine objektiv bessere Programmiersprache für die

Aufgabenstellung erkennen. Hierbei handelt es sich rein um eine persönliche Präferenz, da die Möglichkeiten beider Sprachen vergleichbar bis gleich sind.

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.

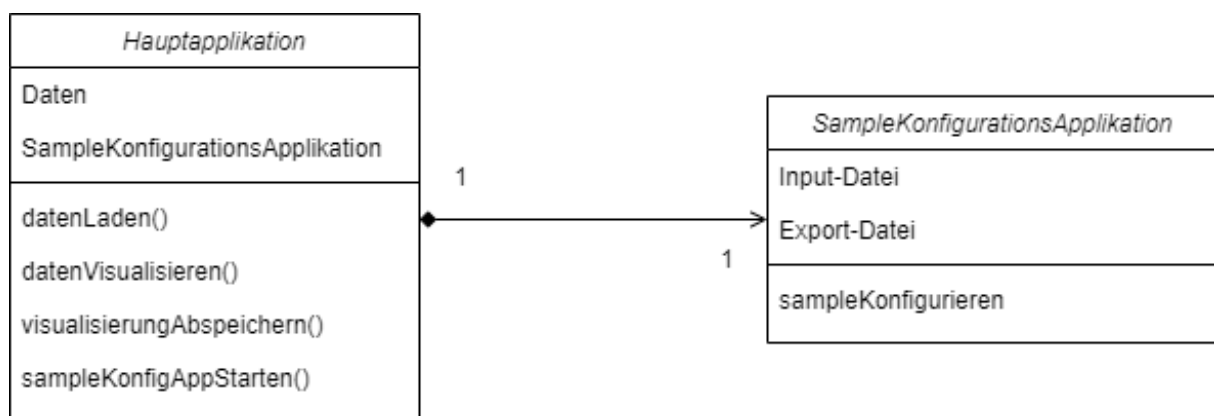


Abbildung 4.1: Konzept Klassendiagramm

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:

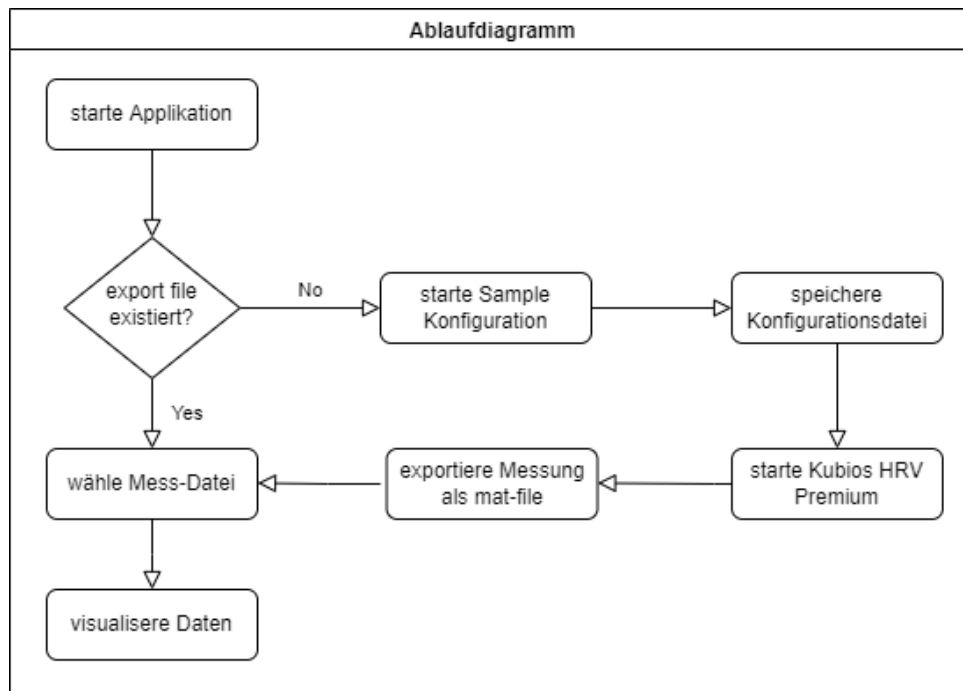


Abbildung 4.2: Ablaufdiagramm des Konzepts

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.

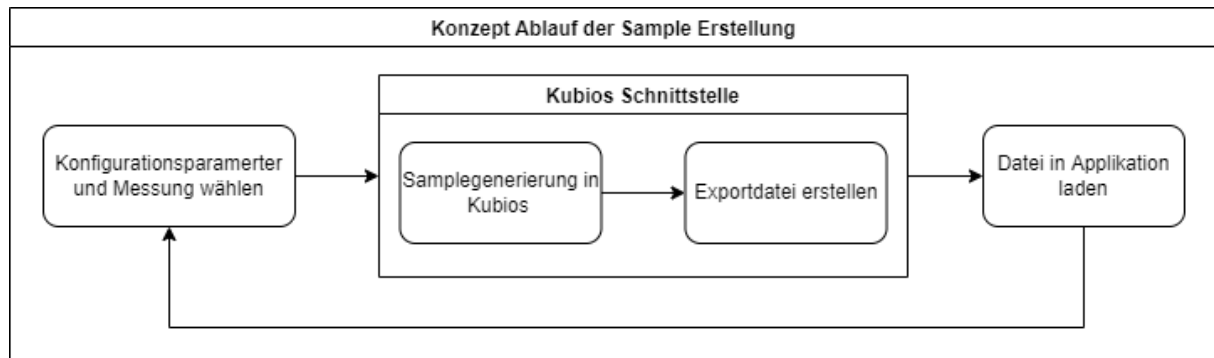


Abbildung 4.3: Konzept Ablauf für das Erstellen der Samples

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.

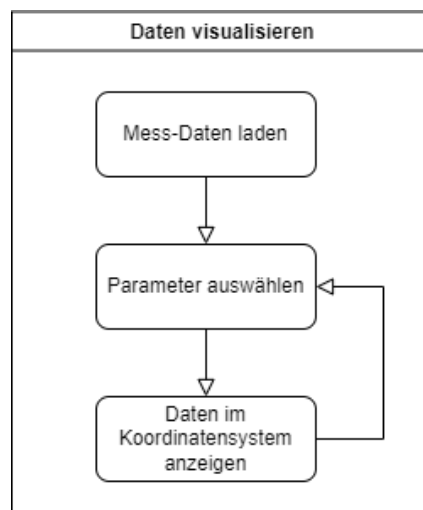


Abbildung 4.4: Konzept Ablauf für das Visualisieren von Daten

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.

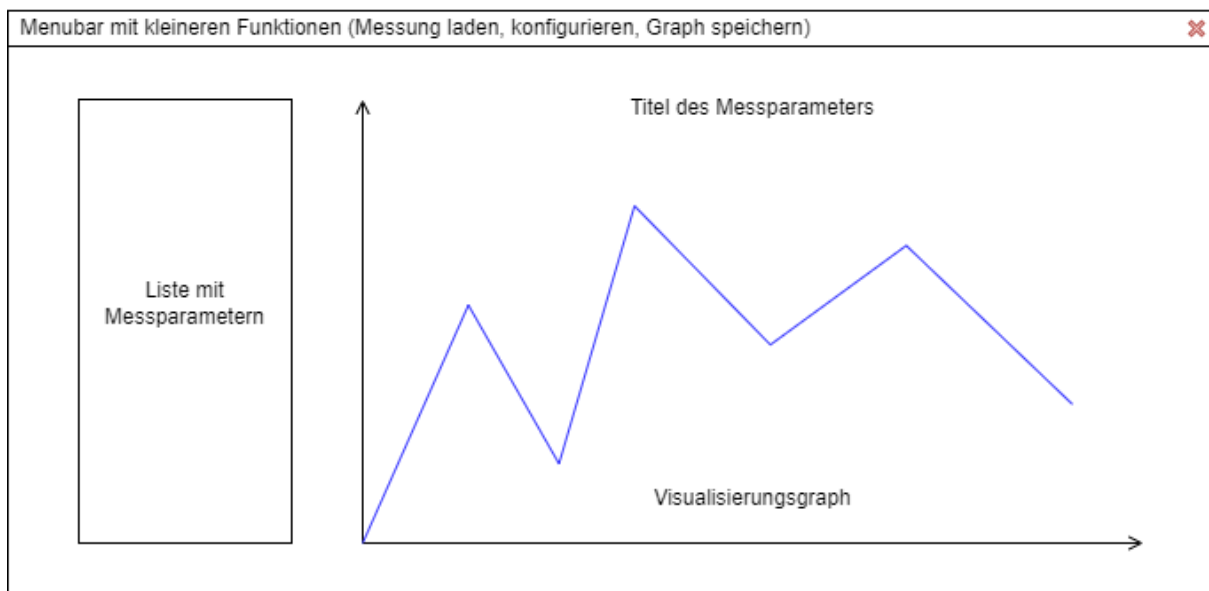


Abbildung 4.5: Konzept Ablauf für das Design

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.

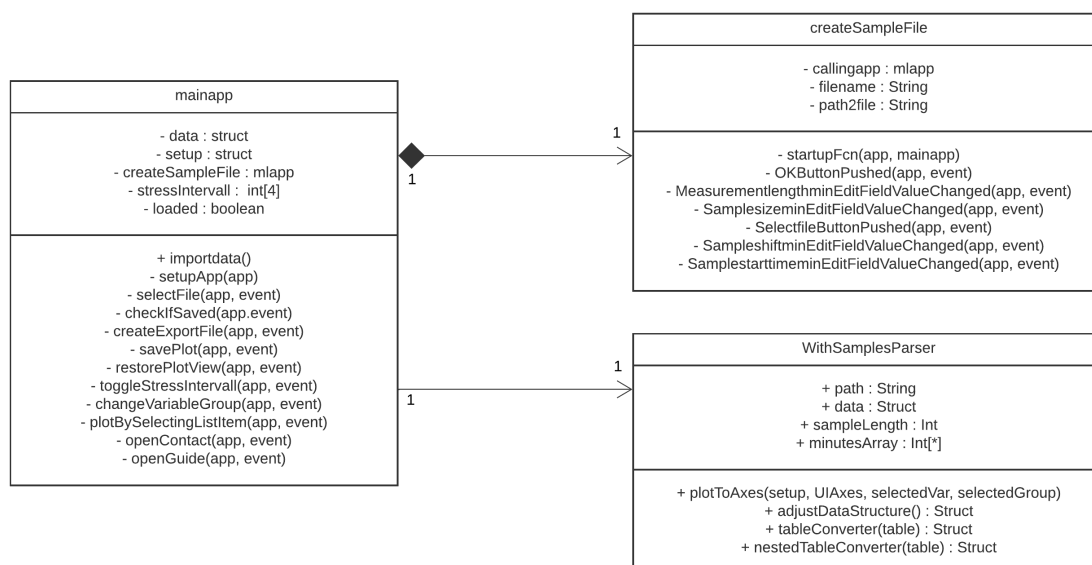


Abbildung 5.1: Klassendiagramm der fertigen Applikation

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.

Innerhalb des Grid-Layouts befinden sich alle zur Visualisierungen von Messdaten notwendigen Komponenten. In der kleineren linken Hälfte der Applikation sind alle Elemente zur Auswahl des jeweiligen Mess-Parameters vorhanden. Dazu gehört eine **DropDown-Komponente** in der alle Gruppierungen der Parameter, wie sie in Kubios vorgegeben sind, aufgeführt werden. Diese ermöglicht das Ausklappen einer Liste und die Wahl einer

der vorgegeben Gruppierungen. Für die Auswahl eines speziellen Parameters wird eine sogenannte **ListBox-Komponente**. Diese zeigt eine Liste von Werte an, mit dem Vorteil, dass innerhalb dieser auch ein Element angeklickt und eine Callback-Funktion ausgewählt werden kann. Außerdem wird hier automatisch ein Slider eingefügt, sollte die Anzahl der Elemente die Größe der Box überschreiten. Die Inhalte der Listbox werden zu den jeweiligen Parametern der ausgewählten Gruppierungen geändert, sollte diese sich innerhalb der DropDown-Komponente verändern.

Hier befindet sich ebenfalls eine **Panel-Komponente**, welche die Einstellung des Belastungsintervalls ermöglicht. Dazu gehören zwei Textfelder zur Eingabe der Intervalleigenschaften, sowie ein **Toggle-Button**, welcher das Aktivieren und Deaktivieren des konfigurierten Belastungsintervalls ermöglicht.

In der größeren rechten Hälfte befindet sich das Koordinatensystem, welches zum Visualisieren der Parameter verwendet wird. Innerhalb der MATLAB App Designer Umgebung wird dieses als **UIAxes-Komponente** bezeichnet. Zum Start der Applikation ist keine Visualisierungen, sowie Achsenbeschriftung zu sehen. Diese Elemente werden beim Laden von Messungen sowie Auswählen spezieller Parameter mit Hilfe der Plot-Funktion angepasst, welche in einem späteren Teil der Arbeit beschrieben wird.

Für das Ausführen aller Funktionen, welche über das Visualisieren der Messdaten hinaus geht, wurde eine **Menu-Komponente** am oberen Fensterrand eingerichtet. Diese beinhaltet drei verschiedene Menüs mit Unterpunkten. Unter dem Punkt „Open“ kann die Konfiguration von Samples für eine Messung gestartet werden. Hier lassen sich ebenfalls konfigurierte Messdaten in die Applikation laden. Unter „Plot“ befinden sich alle visualisierungsspezifischen Funktionen. Hierzu gehören das Speichern des erstellten Graphen, sowie das zurückkehren zum ursprünglichen Bildausschnitt der Visualisierung, sollte diese verschoben bzw. vergrößert worden sein. Der letzte Punkt „Help“ ermöglicht das Öffnen eines Guides sowie eines Pop-Up der Entwickler-Kontaktdaten.

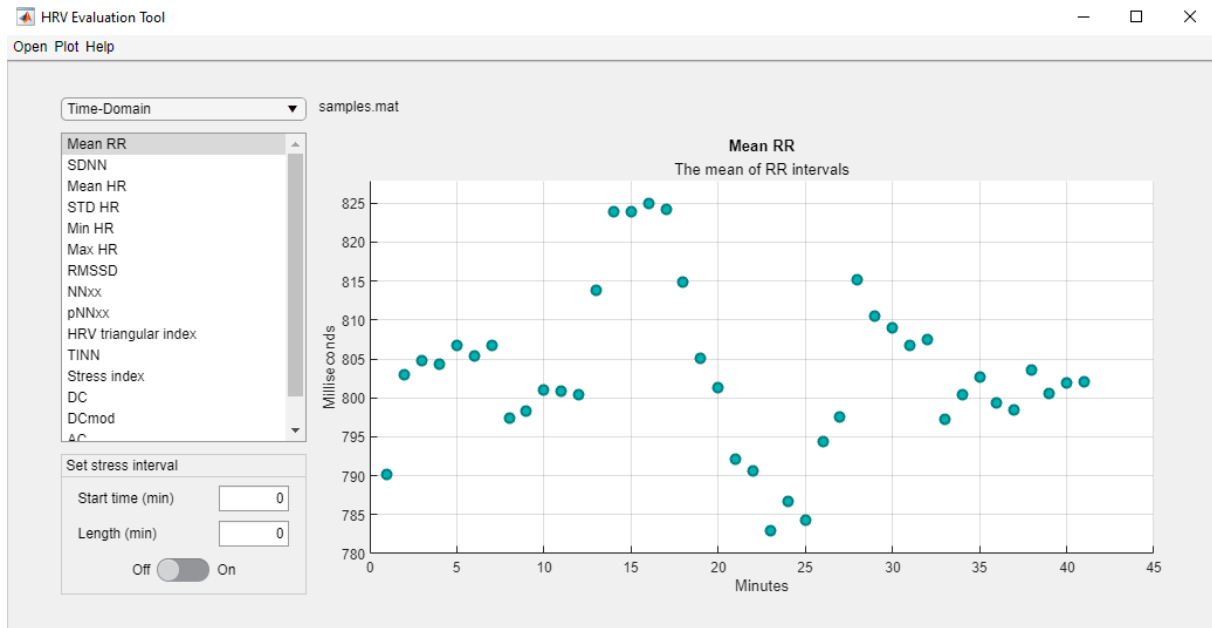


Abbildung 5.2: Fertige Applikation mit geladener Messung

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 ist eine Helper-Funktion implementiert, 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 5.3: Auszug aus der Struktur der Setup-Datei

5.3 Erstellung der Samples

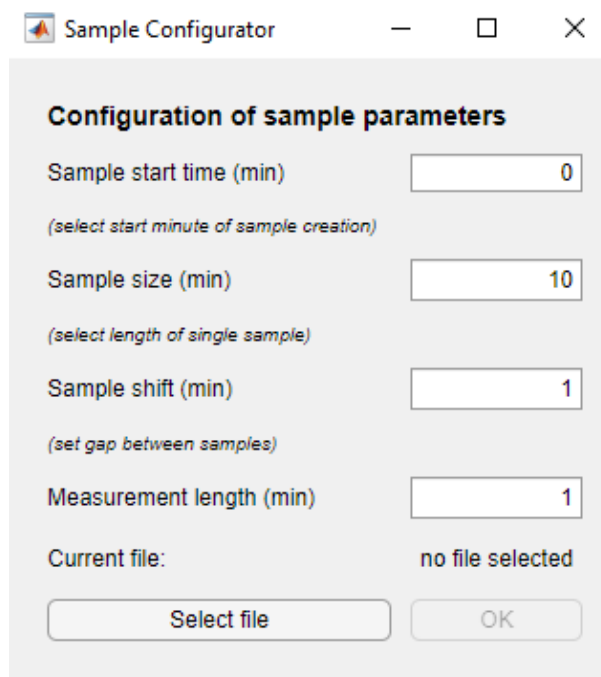
Die automatisierte Erstellung der Samples für eine Messung basiert auf einem Feature von Kubios HRV Premium. Dabei kann manuell eine strukturierte CSV-Datei erstellt werden, in welcher die genauen Informationen über die einzelnen Samples definiert sind. Befindet sich diese beim Laden der Messung in Kubios im gleichen Projekt-Verzeichnis, so wird diese erkannt, importiert und die Samples entsprechend angelegt. In der CSV-Datei können dabei Sample-Bezeichnungen, sowie Farbe, Start- und Endzeit der Samples festgelegt werden, wobei die Bezeichnung und Farbe keinen Einfluss auf die spätere Visualisierung hat, jedoch zwingend angelegt werden muss. Die Struktur der Datei ist, wie im folgenden Beispiel zu erkennen, eindimensional. In der ersten Spalte befindet sich immer der Name der Messdatei, sodass Kubios eine Zuordnung der CSV-Datei zur Messung ermöglicht wird. Im nächsten Schritt werden dann die einzelnen Samples konfiguriert. Hierzu wird zunächst in einer Spalte der Name und die Farbe des Samples gesetzt. Die Farbe wird dabei in RGB mit drei Werten von 0 bis 255 angegeben. Die folgenden beiden Spalten geben die Start- und Endzeit in Sekunden an. Das nächste Sample wird direkt an das vorherige angehängt. Alle Spalten werden, wie bei CSV-Dateien üblich, mit einem Komma getrennt. **[kubios]**

11-48-21.EDF	Sample1 # 0 255 0	0	600	Sample2 # 0 255 0	60	660	...
--------------	-------------------	---	-----	-------------------	----	-----	-----

Tabelle 5.1: Beispielhafter Aufbau einer Sample-Konfigurationsdatei

Da die manuelle Erstellung dieser Konfigurationsdateien sehr aufwendig ist und im Zuge der Aufgabenstellung ein hoher Automatisierungsgrad erreicht werden soll, wird ein Tool innerhalb der Applikation geschrieben, welche diese stark vereinfacht. Dazu kann über den Menüpunkt „Create parametersettings csv-file“ eine Unterapplikation, der **Sample Configurator**, gestartet werden. Diese führt den Nutzer Schritt für Schritt durch die Generierung der CSV-Datei.

Der Sample Configurator besteht hierbei aus vier Eingabefeld-Komponenten und zwei Buttons des App Designers. In den Eingabefeldern werden die wichtigen Parameter der Sample eingegeben. Dazu gehören die Länge der einzelnen Sample, die Verschiebung (d.h. wie viele Minuten nach dem Start des ersten Samples das zweite Sample angelegt werden soll), sowie die Startzeit des ersten Samples und die Länge der gesamten Messungen. Über die Buttons lässt sich dann die Messdatei laden, um Informationen über den Namen, sowie den Speicherort zu erhalten. Der zweite Button dient zur Bestätigung und Generierung der Konfigurationsdatei.



Sample Configurator

Configuration of sample parameters

Sample start time (min)
(select start minute of sample creation)

Sample size (min)
(select length of single sample)

Sample shift (min)
(set gap between samples)

Measurement length (min)

Current file: no file selected

Abbildung 5.4: Unterapplikation Sample Configurator

Die tatsächliche Generierung erfolgt außerhalb des Sample Configurator durch die **createCSV**-Funktion. Diese gibt die erhaltenen Daten lediglich an die Hauptapplikation, wo die entsprechende Funktion aufgerufen wird. Hierbei wird die Anzahl der Samples mittels der Messungslänge sowie der Verschiebung der einzelnen Samples berechnet und über die Anzahl dieser geloopt. Dabei werden in jeder Iteration die zuvor beschriebenen Daten

berechnet und mit Hilfe des String-Datentyps in eine Liste hinzugefügt. Es ergibt sich eine Liste im Stil der oben beschriebenen Tabelle. Zum Schluss wird diese als CSV-Datei mit dem passenden Namen in das Projekt-Verzeichnis der Messung exportiert.

```
1 function isCreated = createCSV(sampleStart, sampleLength, ...
2     fileLength, sampleShift, filename, path)
3
4     isCreated = true;
5
6     if (isempty(filename))
7         isCreated = false;
8         return;
9     end
10
11     export = [filename,"1"];
12     color = " # 0 255 0";
13     configFile = fullfile(path, 'Kubios_Samples.csv');
14     starttime = sampleStart * 60;
15     endtime = starttime + sampleLength * 60;
16
17     for i = 1:(floor(fileLength/sampleShift))
18         if (endtime > fileLength * 60)
19             break;
20         end
21         export = [ export "Sample" + i + color starttime endtime ];
22         endtime = endtime + (sampleShift * 60);
23         starttime = starttime + (sampleShift * 60);
24     end
25     try
26         writematrix(export, configFile, 'Delimiter', 'comma')
27     catch ME
28         isCreated = false;
29     end
30 end
```

Listing 5.2: Erstellung der Sample-Konfigurations-Datei

Um nun die Sample-Konfiguration zu importieren, muss zuerst die entsprechende Messdatei manuell in Kubios geladen werden. Die erstellten Samples lassen sich hier direkt auf der Visualisierung von Kubios erkennen. Im nächsten Schritt muss dann ein Export der Messung als MAT-Datei durchgeführt, da diese als Grundlage der Visualisierung der implementierten Applikation dient. Hier ist also ein direkter manueller Eingriff in die Generierung notwendig, da keine Schnittstelle von Kubios einen vollautomatisierte Analyse

und Export einer Messung ermöglicht. Das Ablaufdiagramm ändert sich dann wie in nachfolgender Abbildung zu erkennen, wobei die manuell auszuführenden Schritte farbig hinterlegt sind.

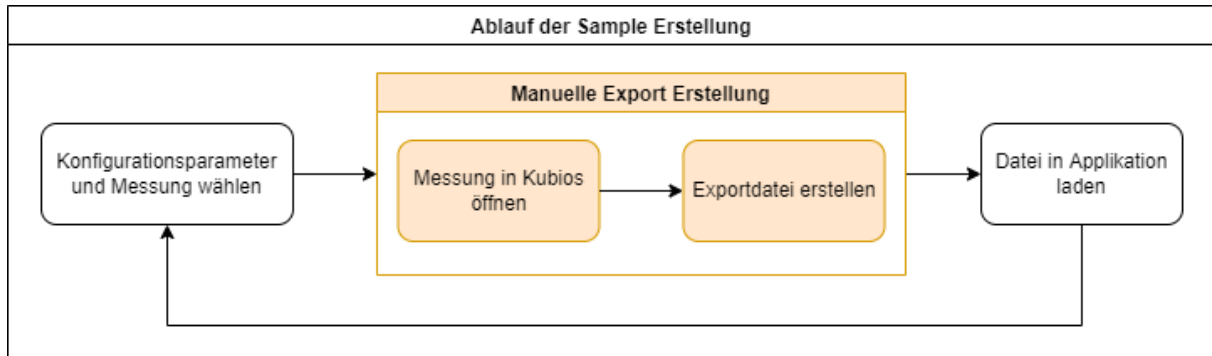


Abbildung 5.5: Angepasstes Ablaufdiagramm der Sample-Erstellung mit manuellen Schritten

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 eingelesenen 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 5.6: 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. Nach dem erfolgreichen Laden der ausgewählten Datei wird der Name dieser oberhalb des UIAxes-Komponente angezeigt.

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.[**pythonDict**]

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.

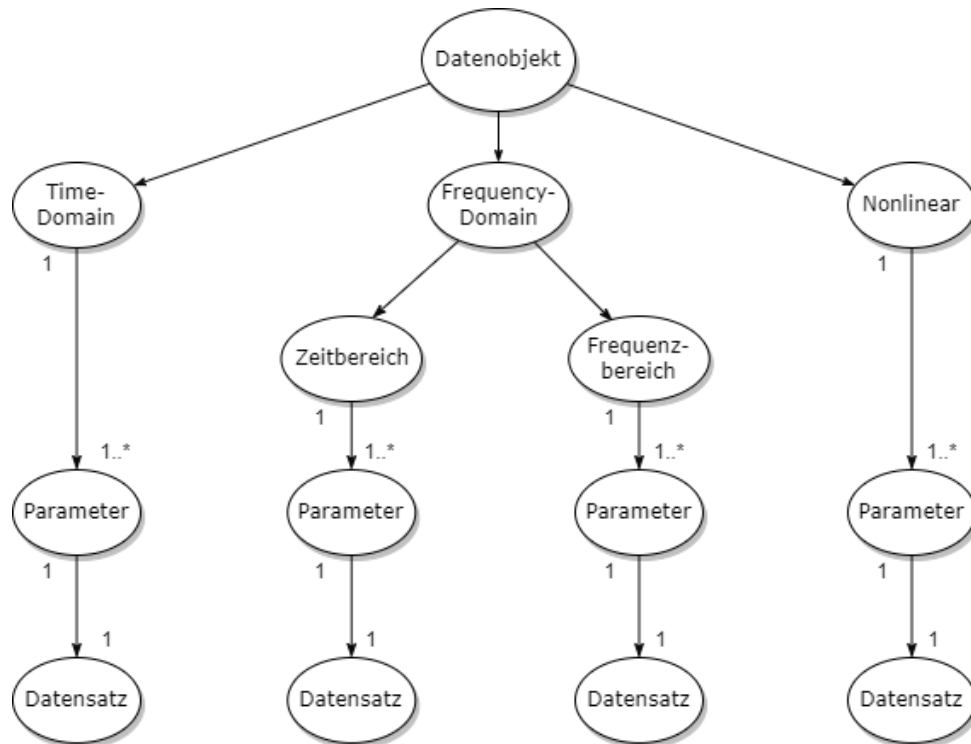


Abbildung 5.7: Baumstruktur zur Veranschaulichung der Datenstruktur

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 5.3: 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 Schleife 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 5.4: 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 5.5: 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 5.6: 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 5.7: 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 5.8: 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 5.9: 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

Im folgenden Kapitel sollen alle zusätzlichen Funktionen erläutert werden, welche nicht direkt im Kern der Aufgabenstellung beschrieben wurden, jedoch aber zur Grundfunktion oder Benutzerfreundlichkeit der Applikation beitragen. Dazu gehören das Hinzufügen eines Belastungsintervalls zur Visualisierung, das Speichern der Visualisierung als Bild-Datei und das Wiederherstellen der Visualisierungsansicht.

Einfügen eines Belastungsintervalls

Da in einem Großteil der Messungen der Proband einer Belastung zu einem zufälligen Zeitpunkt ausgesetzt ist und besonders in diesem Intervall die Veränderungen der Parameter betrachtet werden soll, muss dieses Intervall in der Visualisierung farblich hervorgehoben werden. Dazu werden die Eckpunkte eines Rechtecks mit Hilfe des Start- und Endpunktes des Intervalls, sowie der Höhe der momentanen Koordinatenachsen berechnet und ein Rechteck in die Visualisierung gezeichnet. Dieses Rechteck erhält dabei eine sehr niedrige Deckkraft, um eine gute Sichtbarkeit, aber keine Überdeckung der Messpunkte zu ermöglichen. Beim erneuten Betätigen des Toggle-Buttons wird diese Rechteck wieder aus der Visualisierung gelöscht.

```
1 function toggleStressIntervall(app, event)
2
3     value = app.Switch.Value;
4     if strcmp(value, 'On')
5         % calculate height and width of stress intervall
6         rectX = [app.StarttimeminEditField.Value, app.StarttimeminEditField.
7                 Value + app.LengthminEditField.Value];
8         rectY = ylim(app.UIAxes);
9         % add stress intervall to plot
10        app.stressIntervall = patch(app.UIAxes, rectX([1,2,2,1]), rectY([1 1
11        2 2]), 'red', 'EdgeColor', 'none', 'FaceAlpha', 0.2);
12        % reset limits of UIAxes to prevent layout issues
13        ylim(app.UIAxes, rectY);
14    else
15        % delete stress intervall if switch is set to off
```

```

14     delete(app.stressIntervall);
15 end
16 end

```

Listing 5.10: Funktion zum Togglen des Belastungsintervalls

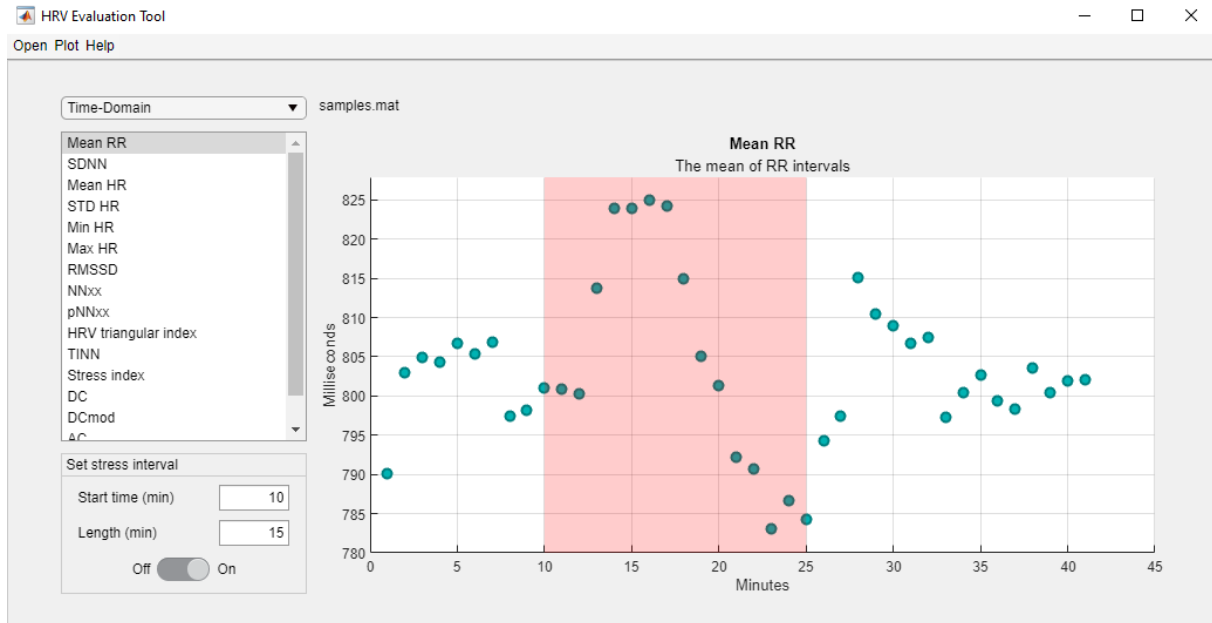


Abbildung 5.8: Beispiel einer Visualisierung mit Belastungsintervall

Abspeichern der Visualisierung als Bild-Datei

Um die erstellten Visualisierungen speichern zu können soll ein Export des Koordinatensystems als Bild-Datei möglich sein. Hierzu wird die MATLAB-native Funktionen **uiputfile** und **exportgraphics** verwendet. Diese erhalten die UIAxes-Komponente als Argument und öffnen ein Datei-Explorer- bzw. Finder-Fenster in welchem der Benutzer einen Speicherort auswählen kann.

```

1 function savePlot(app, event)
2     % this functions saves the plot to a given format
3     % it can be called in the menu bar
4
5     filter = {'*.jpg'; '*.png'; '*.tif'; '*.pdf'};
6     [filename, filepath] = uiputfile(filter);
7     if ischar(filename)
8         exportgraphics(app.UIAxes, [filepath filename]);
9     end
10 end

```

Listing 5.11: Funktion zum Speichern der Visualisierung

Wiederherstellung der Visualisierungsansicht

Die UIAxes-Komponente ermöglicht das Verschieben und Zoomen der Koordinatenachsen, um einen besseren Bildausschnitt zu erzeugen oder die Visualisierung genauer zu untersuchen. Da das Wiederherstellen der Visualisierung auf den Startzustands meist jedoch entsprechend schwierig ist, wird der Menüpunkt „Restore plot“ eingesetzt, um genau dieses Problem zu beheben. Hierzu wird auf die **zoom**-Funktion von MATLAB zurückgegriffen, welche das automatische Skalieren der Koordinatenachsen auf einen Gesamtüberblick ermöglicht.

```
1 function restorePlotView(app, event)
2     % restore the view of the plot
3     zoom(app.UIAxes, 'out');
4 end
```

Listing 5.12: Funktion zur Wiederherstellung der Visualisierung

6 Auswertung von Messungen mit Hilfe des HRV Evaluation Tools

7 Kritische Würdigung und Ausblick

Zusammenfassend kann man sagen, dass die Grundfunktionen des Tools nahezu problemlos umgesetzt werden konnten. Das Tool ermöglicht das Auswerten und Visualisieren von medizinischen Messdaten. Zudem können Samples konfiguriert und Messungen passend exportiert werden. Auch die fachlich korrekte Darstellung aller Messdaten mit einem zusätzlich hervorhebbaren Belastungsintervall sind im Rahmen der Arbeit vollständig implementiert worden.

Besonders hervorgehoben werden muss auch die Benutzerfreundlichkeit der Applikation, sowie die Möglichkeit diese mit Hilfe eines Installers sowohl auf Windows als auch auf MacOS zu verteilen und zu installieren.

Abweichungen bezüglich des Konzepts treten hauptsächlich bei der automatischen Generierung von Samples in Kubios. Aufgrund der eingeschränkten Schnittstellen der Software, konnte keine vollständige Automatisierung implementiert werden, weshalb das Exportieren der mit Samples versehenen Messung noch manuell ausgeführt werden muss. Dieses Problem ist jedoch vollständig Kubios verschuldet und kann möglicherweise durch zukünftige Versionen der Software durch eine Kommandozeilen-Schnittstelle behoben werden.

Auch das Design der Applikation wurde über den Zeitraum der Entwicklung immer wieder verändern und angepasst, um diese möglichst übersichtlich sowie intuitiv gestalten und zusätzliche Funktionen passend in die Benutzeroberfläche integrieren zu können.

Die Entwicklung innerhalb von MATLAB erwies sich als sehr praktisch und intuitiv, was sich vor allem auf den nativen MATLAB-Datentyp des Messungsexport, sowie das schnelle Erstellen einer Benutzeroberfläche durch den graphisch unterstützte App Designer zurückführen lässt. Außerdem konnte die große Zahl an hilfreicher und umfassender Dokumentation als wichtiger Teil der Problemlösung kennengelernt werden.

Für die Zukunft und der Weiterentwicklung sollte vor allem Fokus auf die Anpassbarkeit der Applikation gelegt werden. So sollte eine Funktion implementiert werden, die es dem Benutzer ermöglicht die Setup-Datei direkt innerhalb des Tools zu bearbeiten, um die Erweiterung durch weitere Parameter sowie die Anpassung auf eine mögliche andere Datenstruktur des Exports von Kubios ermöglichen zu können.

Ein weiterer Punkt ist die Speicherung einer Sitzung. Im Moment geht die erstellte Visualisierung und die geladene Messung, sowie jegliche Einstellungen beim Beenden der Applikation verloren. Ein übergreifender Speichermechanismus könnte dieses Problem lösen und möglicherweise auch eine passende Grundstruktur zur Speicherung und Anpassung des Setups ermöglichen.

Zuletzt kann davon ausgegangen werden, dass sich einige fehlende Funktionen oder benötigte Anpassung zukünftig vor allem bei der Anwendung durch die Beteiligten der Pilotstudie herausarbeiten lassen.

Anhang

Glossar

Glossareintrag

Ein Glossar beschreibt verschiedenste Dinge in kurzen Worten.