Technical Report

Ressourcenbeschränkte Analyse von Ionenmobilitätsspektren mit dem Raspberry Pi

Alexey Egorov, Alexander König,
Marcel Köppen, Henning Kühn,
Isabell Kullack, Elias Kuthe,
Suzana Mitkovska, Robert Niehage,
Andreas Pawelko, Manuel Sträßer,
Christian Striewe, Marianna D'Addario,
Dominik Kopczynski, Sven Rahmann

05/2014

technische universität
dortmund

SFB 876 Verfügbarkeit von
Information durch Analyse unter
Ressourcenbeschränkung

Part of the work on this technical report has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 Providing Information by Resource-Constrained Analysis, project TB1.

Speaker: Prof. Dr. Katharina Morik
Address: TU Dortmund University
Joseph-von-Fraunhofer-Str. 23
D-44227 Dortmund
Web: http://sfb876.tu-dortmund.de

INHALTSVERZEICHNIS

1 EINFÜHRUNG

Die Zusammensetzung der Umgebungs- oder Ausatemluft kann viele Informationen liefern, die z. B. helfen können, eine Erkrankung oder deren Ursache festzustellen. Die Moleküle der in der Luft enthaltenen Substanzen haben jeweils unterschiedliche Größen und Formen, so dass es möglich ist, sie voneinander zu trennen über Ausschläge in einer Luftmessung die Häufigkeit ihres Vorkommens zu bestimmen. Diese Ausschläge werden als Peaks bezeichnet. Ihre Erkennung ist Gegenstand aktueller Forschung. Das Einsatzgebiet solcher Messungen erstreckt sich von medizinischer Überwachung von Patienten im Krankenhaus bis zur Überprüfung der Umgebungsluft bestimmter Gegenden.

1.1 ZIELSETZUNG

Schwerpunkt dieser Arbeit soll die *Erkennung von Peaks* mit *Online-Algorithmen* sein. Das bedeutet, dass diese Verfahren schon zur Laufzeit Ergebnisse produzieren und dabei immer nur einen Ausschnitt einer Messung betrachten. Als Peaks werden hierbei visuell erkennbare Ausschläge in der Messung bezeichnet, die auf das Vorkommen eines bestimmten Stoffes hindeuten können. Im Gegensatz zu den mittlerweile gut erforschten und zahlreichen Offline-Verfahren ist dieses Gebiet der Peakerkennung noch nicht weit vorangeschritten. Wir wollen in dieser Arbeit Algorithmen entwerfen, die effizient und online auf einem Strom von Daten arbeiten. Die erzeugten Ergebnisse werden dann von uns klassifiziert, um verschiedene Typen von Messungen mit potenziell unterschiedlichen Stoffzusammensetzungen automatisch voneinander abgrenzen zu können. In der Realität spiegelt dies die Erkennung unterschiedlicher Krankheiten wieder. Zusätzlich ist die uns zur Verfügung stehende Rechenleistung mit dem Raspberry Pi sehr begrenzt. Bei heutzutage üblichen Verfahren werden meist leistungsstarke Laptops in Zusammenhang mit dem Analysegerät, einem Ionenmobilitätsspektrometer (IMS) mit Multikapillarsäule (MCC), verwendet. Während das MCC/IMS-Gerät an sich verkleinert werden könnte, so ist man bis jetzt auf die Rechenleistung eines Mehrkernrechners angewiesen. Falls es uns gelingen sollte, eine zuverlässige Peakerkennung auf begrenzter Hardware zu implementieren, so könnte man ein portables Gerät herstellen, welches nicht mehr nur stationär in Krankenhäusern oder Laboren verwendet werden kann. Solch ein Gerät könnten Ärzte zur Diagnose bei Hausbesuchen mit sich führen oder es in Ländern mit mangelhafter medizinischer Versorgung einsetzen. Über den Fokus dieser Arbeit hinaus geht eine Verifikation der Algorithmen mit großen, realen Datenmengen, auf welche wir zum Zeitpunkt dieses Berichts keinen Zugriff haben.

Bei unserer Arbeit begleiten und betreuen uns

- Professor Dr. Sven Rahmann, Lehrstuhlinhaber Genominformatik der Fakultät für Medizin, Universität Duisburg-Essen, Projektleiter des SFB 876 – Verfügbarkeit von Informationen durch Analyse unter Ressourcenbeschränkung der TU Dortmund.
- Marianna D'Addario und Dominik Kopczynski, Mitglieder des Sonderforschungsbereichs 876.

1.2 ZUSAMMENARBEIT MIT B & S ANALYTIK

Unterstützung bei unserem Vorhaben haben wir von der Firma B & S Analytik bekommen, die das Messgerät entwickelt und verkauft. Uns ist es ermöglicht worden, an drei MCC/IMS-Geräten zunächst in geringer und später auch in hoher Auflösung (Abschnitt 4.1) eigene Atemluft-Messungen von Bonbonsorten der Projektgruppenteilnehmer für Testdaten auszuführen. Diese Testdaten und weitere anonymisierte Messreihen werden im Abschnitt 13.2 vorgestellt.

1.3 STRUKTUR DES BERICHTS

Zum Einstieg in die Atemluftanalyse wird die MCC/IMS-Technologie im Kapitel 2 vorgestellt. Es werden neben den physikalischen Grundlagen auch unterschiedliche Anwendungsgebiete genannt. Zusätzlich beschreiben wir Möglichkeiten zur Anbindung des Messgerätes an den Raspberry Pi. In Kapitel 3 geben wir zunächst einen Überblick über diese Hardwareplattform und gehen auf das von uns gewählte Betriebssystem und die Möglichkeit zum Einsatz eines Crosscompilers zur Entwicklung ein.

In Kapitel 4 widmen wir uns den unterschiedlichen Dateiformaten, mit denen wir im Laufe des Projekts gearbeitet haben. Dazu beschreiben wir die Standardformate zum Austausch von MCC/IMS-Messungen und Peaklisten und evaluieren Formate zur kompakten Speicherung der Messungsdaten.

Da die Möglichkeit des direkten Anschlusses des Raspberry Pi an das MCC/IMS-Messgerät nicht gegeben ist, simulieren wir den Messungsverlauf durch ein Server-Client-Modell. Der Datenfluss und die Zuständigkeiten des Servers und Clients werden im Kapitel 5 behandelt.

In Kapitel 6 schauen wir uns die Messdaten genauer an und analysieren das vorhandene Rauschen und weitere Eigenschaften der Messdaten. Weiterhin wird dort ein Verfahren vorgestellt, um die Messdaten zu korrigieren. Dies ist unverzichtbar für die spätere Peakfindung.

Der weiteren Reduktion des Rauschens widmet sich dann Kapitel 7. Wir stellen bereits bestehende Filterverfahren und die generelle Idee dahinter vor und überprüfen einzelne Verfahren auf die Benutzbarkeit in unserem Szenario. Ein besonderes Augenmerk liegt hier auf dem Savitzky-Golay-Filter, den wir für eines unserer zwei Peakerkennungsverfahren benutzen.

Um deren Entwicklung geht es im darauffolgenden Kapitel 8. Zunächst befassen wir uns mit der Definition und verschiedenen Modellen von Peaks und gehen auf bestehende Offline-Ansätze zur Peaker-

kennung ein. Nun beschreiben wir unsere Peakerkennungsalgorithmen. Das „Savitzky Golay Laplace Threshold Regions“-Verfahren (SGLTR) setzt auf einen Filterkern, um die Annäherung des Laplace-Operators zu bestimmen und die Werte über einem Schwellwert zu Peaks zusammenzufassen. Das zweite Verfahren, „Peak Detection by Slope Analysis“ (PDSA), verwendet das charakteristische Muster aus steigenden und fallenden Peakflanken, um diese zu detektieren.

Nach der Vorstellung der Verfahren wenden wir uns in Kapitel 9 der Optimierung zu. Dieses beinhaltet eine Analyse der Übertaktungsmöglichkeiten sowieso einige Benchmarks zum Raspberry Pi. Es gilt diverse Caching-Effekte zu berücksichtigen und eine Entscheidung über die verwendeten Datentypen (Integer oder Float) zugunsten der Performanz zu treffen.

Anschließend werden im Kapitel 10 die Merkmale der Peaks anhand vorhandener Datensätze untersucht. Das Ziel ist dabei, eine Methode zu finden, die die gefundenen Peaks sinnvoll zu Merkmalen zusammenführt und somit die Grundlage zur Klassifikation darstellt, die im Kapitel 11 vorgestellt wird. Es werden verschiedene Methoden zur Klassifikation verglichen, die die aus einer Menge von Messungen gewonnenen Merkmale zur Bildung eines Klassifikationsmodells benutzen.

Im nachfolgenden Kapitel 12 werden die einzelnen vorgestellten Komponenten zu einer Pipeline zusammengesetzt. Wir erklären, wie die erstellte Server-Client-Architektur gebaut und die Algorithmen zur Peakerkennung angewandt werden können. Es wird auch kurz auf die dafür benötigte Software eingegangen.

Die Evaluation der Peakerkennung, die Klassifikationsraten der Algorithmen und die Laufzeiten auf dem Raspberry Pi sind im Kapitel 13 zu finden. Vor der Präsentation der Ergebnisse wird auf die zur Verfügung stehenden Datensätze eingegangen. Wir erklären auch, welches Gütekriterium zur Evaluation der Peakerkennung genutzt wird. Im Kapitel 14 fassen wir unsere Erkenntnisse und Ergebnisse zusammen und geben einen Ausblick auf offene Fragen für weiterführende Arbeiten.

Weitere Informationen, die man nicht notwendigerweise zum Verständnis der Hauptarbeit braucht, befinden sich im Anhang A.

1.4 VERSIONSHINWEIS

Diesem Bericht liegt die Version des Code-Repositorys mit dem Hashwert `33bb6bbf` bzw. dem Tag `eval-peakdetection` vom 29.03.2014, 18:45:52 zugrunde. Für die Merkmalsextraktion und Klassifikation sowie die Benchmarks der Peakerkennung auf dem Raspberry Pi wurde die Version des Code-Repositorys vom 10.02.2014, 15:23:07 mit dem Hashwert `75d9d9dc` bzw. dem Tag `eval-classification` verwendet.

Zu finden ist das Repository unter `git@projekte.itmc.tu-dortmund.de:in/git-pg572.git`. Sofern die Version der verwendeten Messungen für ein Kapitel relevant ist, wird sie dort ausgewiesen.

2

MCC/IMS-TECHNOLOGIE

In diesem Kapitel stellen wir zunächst die Ionenmobilitätsspektrometrie (IMS), die Vorseparation mittels Multikapillarsäule (MCC) und deren mögliche Einsatzgebiete vor. Nach einem Vergleich mit ähnlichen Messverfahren gehen wir auf Möglichkeiten ein, den Raspberry Pi mit einem MCC/IMS-Gerät zu verbinden.

2.1 MCC/IMS-VERFAHREN

Bei der Ionenmobilitätsspektrometrie (IMS) handelt es sich um ein Analyseverfahren zur Detektion chemischer Substanzen in Gasgemischen. Diese Substanzen werden im Folgenden auch Analyte genannt. Hierbei werden einzelne Moleküle der Probe ionisiert und durch eine ebenfalls mit Gas (z.B. Umgebungsluft oder Stickstoff N_2) gefüllte Driftkammer geleitet. Die Zeit, die die Ionen zur Durchquerung der Kammer benötigen, ist charakteristisch für bestimmte Analyte. Allerdings weisen verschiedene Analyte dieselben Driftzeiten auf. Um trotzdem aussagekräftige Messergebnisse zu erzielen, ist eine Vorseparation mittels anderer Verfahren wie Gaschromatografie oder Multikapillarsäulen erforderlich.

2.1.1 *Physikalische Grundlagen und Funktionsweise*

Dieser Abschnitt beschreibt das eingesetzte Messverfahren und die ihm zugrunde liegenden physikalischen Zusammenhänge. Als Quellen dienen die Arbeiten von Sabine Bader [4] und Jörg Ingo Baumbach [6].

Das von uns eingesetzte Messgerät ist ein sogenanntes Ionenmobilitätsspektrometer (IMS) mit vorgeschalteter Multikapillarsäule (Multi Capillary Column, MCC). Die Firma B & S Analytik, mit der wir im Rahmen des Projektes zusammenarbeiten, entwickelt und vertreibt diese Geräte unter der Bezeichnung „BreathDiscovery“ für stationäre medizinische Anwendungen. Abbildung 1 zeigt ein solches Messgerät von der Größe eines Laserdruckers.

Im Folgenden wird die Funktionsweise des Messverfahrens anhand des Weges beschrieben, den das Gas durch das Messgerät nimmt. Abbildung 2 illustriert schematisch den Aufbau des Geräts. Zunächst wird eine Probe des zu untersuchenden Gases genommen. Hierzu atmet der Patient oder Proband in ein spezielles Mundstück oder wie in unserem Fall einfach in ein offenes Schlauchende aus. Es sind je nach Anwendung verschiedene Mundstücke oder Adapter denkbar. Die Firma B & S Analytik bietet entsprechende Mundstücke unter der Produktbezeichnung *SpiroScout* an. Diese Mundstücke ermöglichen eine genauere Selektion und weitere Analysen der genommenen Proben. Beispielsweise könnte mittels Mundstück die Probenahme ausschließlich auf die letzten Milliliter eines Atemzuges begrenzt und die Temperatur und Feuchtigkeit

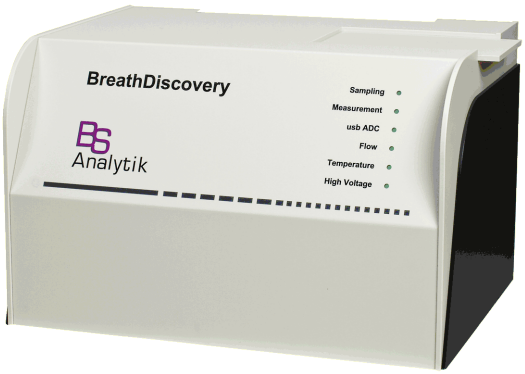


Abb. 1: BreathDiscovery, Produktabbildung entnommen von http://www.bs-analytik.de/produkte.html

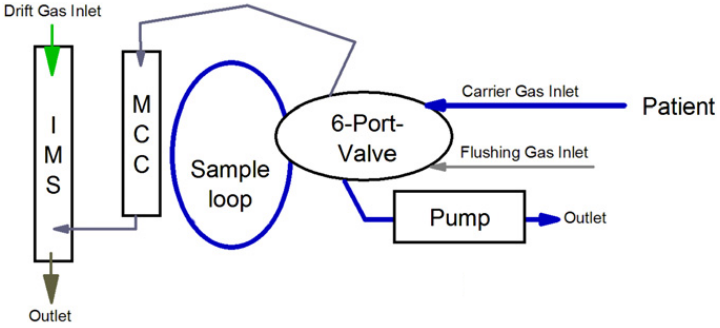


Abb. 2: Patient oder Proband atmet aus und eine Vakuumpumpe saugt die Atemluft in den Probenkreislauf. Die Messung läuft noch nicht. [6]

der Atemluft erfasst werden. Für unsere Messungen ist dies jedoch nicht relevant. Deshalb verzichten wir auf ein Mundstück.

Eine Vakuumpumpe saugt über ein passend geöffnetes 6-Wege-Ventil Atemluft in den Probenkreislauf (sample loop). Hierbei handelt es sich um ein dünnes Röhrchen, welches zwei Anschlüsse des 6-Wege-Ventils miteinander verbindet. Der Probenkreislauf weist ein Volumen von 10 ml auf und ist aus Edelstahl gefertigt, um eine Verfälschung der Proben durch Kunststoff-Ausdünstungen zu vermeiden.

Sobald der Kreislauf vollständig mit Atemluft gefüllt ist, startet die Messung.

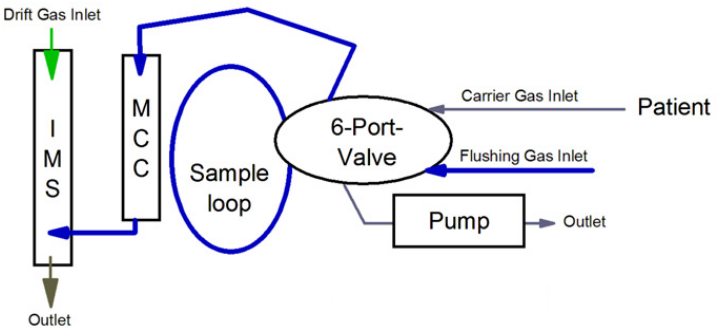


Abb. 3: Trägergas strömt durch den gefüllten Probenkreislauf und transportiert die Probe zum MCC/IMS. Die Messung läuft. [6]

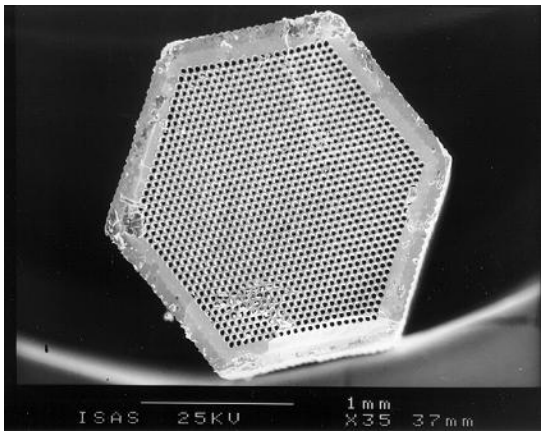


Abb. 4: Querschnitt durch eine Multikapillarsäule [6]

Wie in Abbildung 3 zu sehen ist, wird nach dem Umschalten des Ventils nun ein Trägergas – Stickstoff N_2 oder, wie in unserem Fall, Umgebungsluft – mit einer Durchflussrate von ca. $250\,\mathrm{ml/min}$ durch den Probenkreislauf geleitet. Das Gas transportiert die Moleküle zunächst durch eine Multikapillarsäule und dann ins eigentliche Ionenmobilitätsspektrometer. Die gesamte Messung dauert 10 Minuten. Abbildung 4 zeigt stark vergrößert den Querschnitt einer Multikapillarsäule. Die verwendeten Säulen sind $17\,\mathrm{cm}$ lang und umfassen ca. 1000 Kapillaren mit einem Durchmesser von jeweils $40\,\mu\mathrm{m}$. Der Gesamtdurchmesser der Multikapillarsäule beträgt lediglich $3\,\mathrm{mm}$. Die Säule ist von einem Thermoelement umgeben. Dies erlaubt den Betrieb zwischen $30\,°\mathrm{C}$ und $70\,°\mathrm{C}$. Während unserer Messungen wurden die Säulen stets mit einer Temperatur von $40\,°\mathrm{C}$ betrieben.

Die Multikapillarsäule dient der Vorseparation der zu analysierenden Substanzen. Je nach Molekülform sollen sich die Analyte hierzu unterschiedlich schnell durch die Säule bewegen. Dazu sind die Wände der einzelnen Kapillaren mit einem Gel des Typs „OV5“ beschichtet, welches die Moleküle unterschiedlich gut an sich bindet. Die Moleküle wechseln auf ihrem Weg durch die Kapillaren ständig zwischen der stationären (gebundenen) und mobilen Phase. Anschaulich betrachtet bleiben verwinkelte Moleküle dabei öfter und länger am Gel hängen als runde Moleküle. Die Zeit, die die Moleküle für die Durchquerung der Säule benötigen, wird *Retentionszeit* genannt.

Neben der Art des verwendeten Gels wirkt sich auch die Temperatur der Säule auf die Retentionszeit aus. Je höher die Temperatur gewählt wird, desto kürzer sind die resultierenden Retentionszeiten. Dieser Effekt kann je nach (medizinischer) Anwendung genutzt werden, um die Gesamtzeit pro Messung zu verkürzen oder um die Messgenauigkeit der Retentionszeiten zu erhöhen. Zur Erlangung vergleichbarer Ergebnisse ist stets eine konstante und möglichst über mehrere Messungen hinweg einheitliche Temperatur der Multikapillarsäulen erforderlich.

Nach dem Durchqueren der Multikapillarsäule gelangen die zu untersuchenden Analyte zusammen mit dem Trägergas durch den Analytgaseinlass in den Ionisationsraum des Ionenmobilitätsspektrometers. Dort wird ein Teil der Moleküle des Trägergases mittels β^--Strahlung

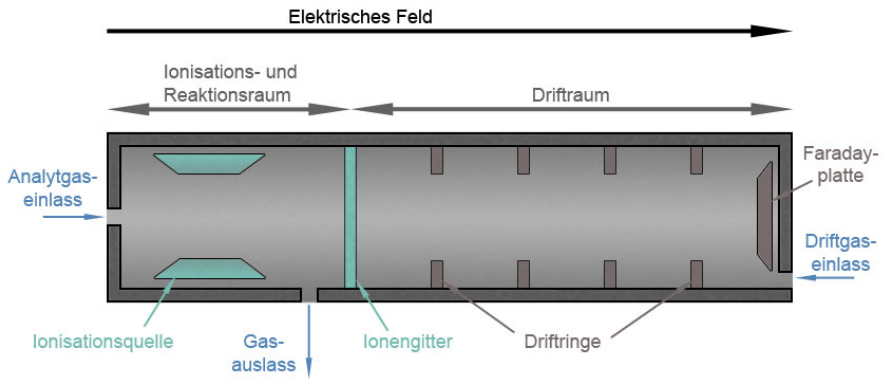


Abb. 5: Schematischer Aufbau eines Ionenmobilitätsspektrometers [28]

ionisiert (β^--Strahlung besteht aus energiereichen Elektronen). Als Ionisationsquelle dient hier das radioaktive Nickel-Isotop ^{63}Ni.

Beim Ionisationsvorgang entstehen sowohl positiv als auch negativ geladene Ionen: Positive Ionen entstehen, wenn Elektronen aus der Hülle der Moleküle herausgeschlagen werden; negative Ionen entstehen, wenn zusätzliche Elektronen eingefangen werden. Obwohl hier beide Ionisationsvorgänge ablaufen, sind für die Messung nur die positiv geladenen Ionen interessant. Die ionisierten Moleküle des Trägergases werden *Reaktionsionen* genannt. Ein Teil von ihnen reagiert mit umgebenen Molekülen des Analyts zu sog. *Produkt-Ionen*, die wir messen wollen. Der überwiegende Teil der Reaktionsionen reagiert nur mit einem Molekül des Analyts und bildet *Monomere*. Bei sehr hohen Stoffkonzentrationen reagieren die Ionen des Trägergases jedoch auch mit zwei Analyt-Molekülen und bilden *Dimere*, die sich, bedingt durch ihre größere Masse, in den Messungen als „Nebenpeak“ manifestieren. Die übrigen Trägergas-Moleküle, die keinen Reaktionspartner gefunden haben, erscheinen in den Messungen später als über den gesamten Retentionszeitraum ausgedehnter *Reaktionsionenpeak* (RIP). Nicht ionisierte Bestandteile des Trägergases und des Analyts verlassen die Ionisationskammer durch den Gasauslass.

Die Ionisationskammer ist durch ein sogenanntes Ionengitter vom Driftraum des IMS getrennt. Der Driftraum hat bei den hier verwendeten Geräten eine Länge von $120\,\mathrm{mm}$. Alle $100\,\mathrm{ms}$ öffnet sich das Ionengitter für einen Zeitraum t_{grid} von etwa $0{,}3\,\mathrm{ms}$ und lässt die zuvor ionisierten Moleküle in den Driftraum eintreten. Entlang des gesamten IMS verläuft ein elektrisches Feld mit einer Feldstärke von $365\,\mathrm{V/cm}$. Dieses wird durch das Anlegen einer Spannung von $4{,}38\,\mathrm{kV}$ aufgebaut. Die in Abbildung 5 abgebildeten „Driftringe“ dienen der Homogenisierung des Feldes.

Da wir uns bei unseren Messungen auf positive Ionen beschränken, befindet sich der Pluspol des elektrischen Feldes an der Ionisationskammer und der Minuspol am Ende des Driftraumes hinter der Faradayplatte (*Positivmodus*). Für den ebenfalls möglichen *Negativmodus* wird das Feld entsprechend umgepolt, jedoch wird dieser bei unseren Messungen nicht verwendet.

In der Driftkammer beschleunigen die Ionen sehr schnell auf eine konstante Geschwindigkeit von etwa $10\,m/s$. Da hier im Gegensatz zu einem Massenspektrometer kein Vakuum herrscht, sondern Luft oder Stickstoff N_2 als Driftgas verwendet werden, stoßen die Ionen auf dem Weg durch die Driftkammer immer wieder an Moleküle des Driftgases und werden abgebremst. Die so erreichte Durchschnittsgeschwindigkeit wird auch als Driftgeschwindigkeit v_d bezeichnet. Die Geschwindigkeit einzelner Moleküle kann nicht direkt gemessen werden. Stattdessen misst man am Ende der Driftkammer mittels einer Faradayplatte kleine Spannungsschwankungen, die durch auftreffende Ionen verursacht werden. So ergibt sich eine Funktion der Spannung über die Driftzeiten t_d, die die Ionen zum Durchqueren der Driftkammer benötigt haben.

Da die Drifzeiten der Ionen nicht konstant sind, sondern von Umgebungsbedingungen der Messung abhängen, wurde als aussägekräftigere Größe die sogenannte Ionenmobilität K eingeführt [4, S. 67ff]:

$$K = \frac{l_d}{x^* \cdot E} \cdot (1000)\ , \tag{1}$$

wobei l_d die Länge der Driftkammer, $x^* = x - t_{grid}/2$ die um die halbe Öffnungszeit des Ionengitters verschobene Driftzeit x und E die elektrische Feldstärke sind. Der Faktor 1000 dient zur Umrechnung der Driftzeit von Millisekunden in Sekunden.

Lässt man auch den Umgebungsdruck p (in hPa gemessen) und die Temperatur T (in Kelvin) in die Formel eingehen, so erhält man die reduzierte Mobilität K_0:

$$K_0 = \frac{(1000) \cdot l_d}{x^* \cdot E} \cdot \frac{p}{p_0} \cdot \frac{T_0}{T}\ . \tag{2}$$

Die Standardtemperatur $T_0 = 273{,}15\,K$ und der Normdruck $p_0 = 1013{,}25\,hPa$ dienen der Normalisierung. Zur besseren Anschauung und Interpretierbarkeit der Messwerte verwendet man statt K_0 die sogenannte *inverse reduzierte Mobilität* $1/K_0$:

$$\frac{1}{K_0} = \frac{x^* \cdot E}{(1000) \cdot l_d} \cdot \frac{p_0 T}{p T_0}\ . \tag{3}$$

Die *inverse reduzierte Mobilität* $1/K_0$ ist proportional zur Driftzeit t_d. Es gilt:

$$t_d = \frac{1}{K_0} \cdot F_{ims}\ . \tag{4}$$

F_{ims} ist eine geräteabhängige Konstante und wird in der Einheit $\frac{cm^2}{kV}$ angegeben.

Neben dem bereits erwähnten Problem der Dimer-Bildung bei hohen Stoffkonzentrationen gibt es bei IMS-Messungen noch andere zu berücksichtigende Effekte: Einige Stoffe weisen trotz unterschiedlicher Form oder Masse ihrer Moleküle dieselben Driftzeiten auf. In Kombination mit der Vorseparation durch eine Multikapillarsäule ist in vielen Fällen jedoch auch hier eine Unterscheidung möglich, wenn die Stoffe unterschiedliche Retentionszeiten haben. Durch den gekoppelten Einsatz zweier unterschiedlicher Messverfahren erhalten wir als Ergebnis einer

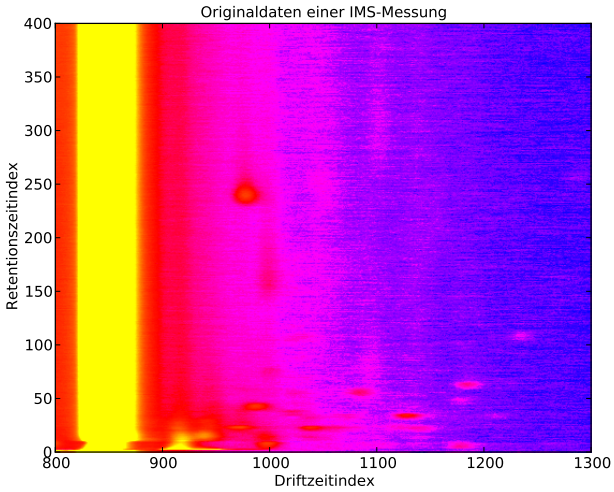


Abb. 6: Visualisierung einer MCC/IMS-Messung als Heatmap. Blaue Pixel stehen für niedrige, rote für hohe und gelbe für sehr hohe gemessene Spannungswerte an der Faradayplatte.

MCC/IMS-Messung eine zweidimensionale Matrix aus Datenpunkten, welche den an der Faradayplatte gemessenen Spannungen bzw. den daraus resultierenden Signalintensitäten entsprechen. Abbildung 6 zeigt das als Heatmap visualisierte Ergebnis einer MCC/IMS-Messung. Blaue Pixel stehen für niedrige, rote für hohe und gelbe für sehr hohe gemessene Spannungswerte an der Faradayplatte. Die Y-Achse entspricht in dieser Darstellung der Retentionszeit, die X-Achse der Driftzeit. Man kann deutlich einige rote Peaks auf blauem Grund erkennen.

Nach einer Arbeit von Spangler und Collins [35] kann die Halbwertsbreite $\omega_{1/2}$ eines solchen Peaks in der Driftzeit mit folgender Formel beschrieben werden:

$$\omega_{1/2} = \sqrt{\frac{11{,}09 \cdot D_L \cdot t_d}{{v_d}^2} + {t_{grid}}^2} \,. \tag{5}$$

D_L steht hier für den *Diffusionskoeffizienten*, welcher mittels der sogenannten Einstein-Relation berechnet werden kann, da außer Temperatur T und Ionenmobilität K auch Elementarladung q und Boltzmann-Konstante k_B bekannt sind:

$$D_L = \frac{K \cdot k_B \cdot T}{q} \,. \tag{6}$$

Im Unterschied zur Massenspektrometrie können aus einer Messung mit dem MCC/IMS-Verfahren keine eindeutigen Rückschlüsse auf Atom- oder Molekülmassen gezogen werden. Daher sind bei unbekannten Stoffen stets Vergleichsmessungen mit einem „sicheren“ Verfahren wie der Massenspektrometrie nötig, um Peaks einem bestimmten Stoff zuordnen zu können. Das MCC/IMS-Verfahren eignet sich also nur zur Identifikation von Stoffen, deren charakteristische Peaks bereits bekannt sind. Unabhängig davon können mit dem MCC/IMS-Verfahren natürlich vergleichende Messungen durchgeführt werden, ohne dass die „Verursacher“ der gemessenen Peaks bekannt sein müssen.

2.1.2 *Vergleich mit ähnlichen Verfahren*

Neben dem MCC/IMS-Verfahren existieren laut den Arbeiten von Sabine Bader [4] und Jörg Ingo Baumbach [6] noch einige andere Analyseverfahren, die prinzipiell zur Detektion derselben Stoffe eingesetzt werden können.

Als erstes wäre hier die *Gaschromatographie* zu nennen. Im Gegensatz zur MCC wird hier mit einer einzelnen bis zu 200 m langen (aufgewickelten) Kapillaren und wesentlich höheren Temperaturen gearbeitet. Die Gaschromatographie bietet sehr hohe Auflösungen und eignet sich auch als Vorseparationsverfahren (GC/IMS). Aufgrund des sehr kleinen Querschnitts muss hier jedoch mit wesentlich geringeren Durchflussmengen gearbeitet werden. Auch sind Gaschromatographen durch ihre Größe und ihren Energieverbrauch weniger gut für den mobilen Einsatz geeignet.

Ein anderes Verfahren, das Ähnlichkeiten zur Ionenmobilitätsspektrometrie aufweist, ist die „*differential mobility spectrometry*“ (DMS). Hierbei liegt das elektrische Feld anders als beim IMS orthogonal zum Ionenfluss. Dies ermöglicht es, gleichzeitig positiv und negativ geladene Ionen zu messen, während beim IMS vor jeder Messung die Polarität festgelegt werden muss. Das DMS-Verfahren erlaubt zwar die Konstuktion sehr kleiner energiesparender Messgeräte, bietet jedoch nur eine sehr begrenzte Auflösung.

Ein weiteres Alternativverfahren zur Ionenmobilitätsspektrometrie ist die *Massenspektrometrie*. Hierbei handelt es sich um ein extrem genaues, aber auch sehr aufwendiges Verfahren. Ein Massenspektrometer ist deutlich größer und schwerer als ein Ionenmobilitätsspektrometer. Zunächst wird hier in der gesamten Apparatur ein Hochvakuum erzeugt. Die ionisierten Moleküle des Analyts werden dann ebenfalls in einem elektrischen Feld beschleunigt und anschließend in einem oder mehreren Magnetfeldern abgelenkt. Je nach Ablenkwinkel kann nun das Ladungs-/Masse-Verhältnis der Moleküle bestimmt werden. Dieses Verfahren kann ebenfalls mit einer Vorseparation durch GC oder MCC kombiniert werden. Massenspektrometer sind sehr teure stationäre Laborgeräte. Der mobile Einsatz gestaltet sich aufgrund des hohen Energieverbrauchs und der vakuumbedingten schweren Konstruktion sehr schwierig.

2.2 ANWENDUNGSGEBIETE

Die Anwendungsgebiete der Ionenmobilitätsspektrometrie sind neben klassischen Drogentests[1] auch unter anderem der *Nachweis von Erkrankungen*, die *Überwachung von Therapieverläufen* und die *Anästhesiekontrolle*, wie im Folgenden gezeigt wird.[2]

Bei *Bakterien und Pilzen* sind z. B. ihre Stoffwechselprodukte in der Atemluft detektierbar. Dabei sind einzelne Zwischenprodukte in diesem biochemischen Stoffwechselvorgang, sogenannte Metaboliten, un-

1 z. B. http://www.gas-dortmund.de
2 http://www.bs-analytik.de/produkte.html

terscheidbar. Zum Teil weist das Messen dieser Metaboliten nur auf die Existenz einzelner Bakterien und Pilze hin. In einigen Fällen ist aber auch konkret pathogenes Auftreten detektierbar, wie es auf der Website von B & S Analytik erläutert wird.[2]

Messbare Unterschiede des Vorkommens bestimmter Metaboliten bei gesunden und befallenen Menschen liegen z. B. bei *Pseudomonas aeruginosa*, *Candida albicans* und *Aspergillus fumigatus* vor. Darüber hinaus gibt es Bakterien, denen bislang noch mehrere Peaks zugeordnet werden können, wie z. B. *Escherichia coli*, *Pseudomonas aeruginosa* oder *Aspergillus fumigatus*.

Dabei wird jeder auftretende Peak dieser Bakterien sowohl bei gesunden als auch befallenen Menschen gesammelt und es wird versucht Cluster in Form charakteristischer Mengen zu bilden. Mit diesen Clustern soll ein Rückschluss möglich sein, ob das gefundene Bakterium bei einem Peak im entsprechenden Cluster pathogen ist oder nicht. Dabei sind Fehler aufgrund von Menschen, die vorher fälschlicherweise für gesund gehalten worden sind, nicht auszuschließen.

Als weitere Motivation aus dem Bereich der Bakterien gilt die Erforschung der *Sepsis* bzw. ihrer Stoffwechselprodukte in der ausgeatmeten Luft. Aber auch die Untersuchung der schwefelhaltigen Produkte der *Halitose* gehört zu einem Schwerpunkt der Forschung. Bei der Messung der Atemluft auf Halitose wird explizit ein MCC/IMS-Gerät, der sogenannte BioScout von B & S Analytik, verwendet (Abbildung 1, Abschnitt 2.1) und eine hohe Konzentration der schwefelhaltigen Verbindungen in Form von Peaks lässt sich eindeutig der Halitose zuordnen.

Bereits gute Erfolge sind bei der Bestimmung von *Bronchial-Karzinomen* erzielt worden. Hier ist möglich, anhand einer eindeutigen Messung festzustellen, ob ein Patient *Lungenkrebs* hat, wie im Paper von Handa u. a. [22] dargestellt wird. Es sind bei Tumorerkrankungen bereits 61 Peaks als bedeutsam analysiert worden, wobei die Peaks teilweise geringe Signalintensitäten aufweisen können.

COPD (Chronisch obstruktive Atemwegserkrankungen) ist sogar durch einen einzelnen Analyten nachweisbar. Dabei sind die einzelnen Stufen von Verschmutzungen der Lunge durch Rauch und Staub (1. Stufe) bis zur stärksten Form stark eingeschränkter Belastbarkeit (4. Stufe), bestimmbar. Die Identität der Peaks muss allerdings weiter erforscht werden, wie eine Studie im Jahr 2013 zeigte. Dabei sind laut der Studie von Baumbach u. a. [5] 97 % der COPD-Patienten anhand der Messung korrekt mit COPD diagnostiziert worden. Allerdings sind nur 65 % der gesunden Menschen auch korrekt als gesund zugeordnet worden.

Ein besonderes Augenmerk sind Entwicklungen der Anwendung der IMS-Technologie im Bereich der *Dialyse*, bei *Abstoßungsreaktionen*, bei der *Medikamentenüberwachung*, bei der *Anästhesie* und allgemein bei der *Überwachung von Therapieverläufen*.

Bei der *Dialyse* sind unterschiedliche Peaks vor, während und nach der Dialyse messbar. Dabei sind einige Signale geringer als zuvor, andere werden verstärkt gemessen. Ob die Atemluft zur Dialyseüberwachung beitragen kann, wird noch erforscht.

Nach *Lungentransplantationen* wird ebenfalls nach auftretenden Stoffwechselprodukten gesucht. Es wird erforscht, ob durch gemessene Peaks

eine Abstoßungsreaktion der transplantierten Lunge prognostiziert werden kann.

Ebenfalls werden Messungen auf Peaks von Stoffwechselprodukten nach einer *Medikamenteneinnahme* überprüft, um die Einnahme und ggf. den positiven oder negativen Behandlungsverlauf nachzuweisen. Somit kann ein Nachweis für die Wirksamkeit bestimmter Medikamente erbracht werden. Auf die gleiche Weise kann auch überwacht werden, ob z. B. ein Demenzpatient seine Medikamente überhaupt eingenommen hat. Die Peaks und ihre Intensität können dafür dann über einen Zeitraum in Form von Zeitreihen abgebildet werden.

Hypnotika, wie z. B. *Propofol*, können ebenfalls in der Atemluft nachgewiesen werden. Dabei ist eine Überwachung der Anästhesie möglich, weil das Signal bei Hypnotika quantifizierbar ist, d. h. die Menge des Hypnotikums in der Ausatemluft proportional zu der Menge des Hypnotikums im Blut ist.

Darüber hinaus werden auch Therapieüberwachungen des *Hunter-Syndroms* mit der IMS-Technologie erforscht.

2.3 DATENÜBERTRAGUNG VOM MCC/IMS-MESSGERÄT AN DEN RASPBERRY PI

Bei einer Messung erfolgt die Übertragung der Messdaten derzeit von einem MCC/IMS-Messgerät an einen Laptop mit einer Steuerungssoftware, der die Messungen startet und beendet sowie die empfangenen Daten visuell darstellt und zur manuellen Weiterverarbeitung aufbereitet. Die MCC/IMS-Messgeräte der Firma B & S Analytik kodieren dabei, mittels eines Analog/Digital-Wandlers, das Signal mit 12 Bit (d. h. mit Werten zwischen 0 und 4095). Der Projektgruppe ist es nicht möglich gewesen, zum Testen einen digitalen Datenstrom aus dem für Signalverarbeitung und Steuerung zuständigen FPGA auszuleiten. Stattdessen sind noch zwei Möglichkeiten, Daten per Netzwerkanschluss oder durch Abgreifen des analogen Signals von dem MCC/IMS-Geräten zu erhalten, von der Projektgruppe in Betracht gezogen und in der Implementierung berücksichtigt worden. Aufgrund des Aufwands konnten allerdings kein realer Test am MCC/IMS-Gerät mit dem Raspberry Pi durchgeführt werden.

Der einfachere Ansatz, die Messdaten zu erhalten, besteht darin, sie vorgefertigt von einem Notebook übertragen zu lassen. Das Notebook selbst wäre dabei an das MCC/IMS-Messgerät angeschlossen. Das Notebook soll dann als Server jeweils ein UDP-Paket mit 25 000 Datenpunkten pro 100 ms Driftzeit (ein Spektrum) an den Raspberry Pi senden. Dabei wird ein Datenpunkt mit 16 Bits übertragen. Dieser Verlauf ist in der Abbildung 7 dargestellt.

Je nach Konfiguration liefert der bestehende Analog/Digital-Wandler entsprechend Abbildung 8 aber Werte zwischen ca. 0 und 2047 oder 2048 und 4095. Aufgrund dieser Vorgabe werden bereits in der Implementierung der Projektgruppe die ankommenden Werte normiert, so dass die Null-Linie (ca. 2048, aus der Messung ermittelt, Offset in der

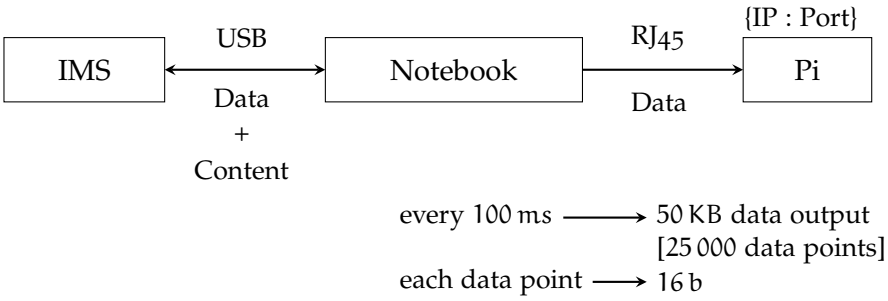


Abb. 7: Schematischer Messungsablauf mit dem Notebook, welches die Daten vom MCC/IMS-Gerät erhält und diese dann an den Raspberry Pi weiterleitet.

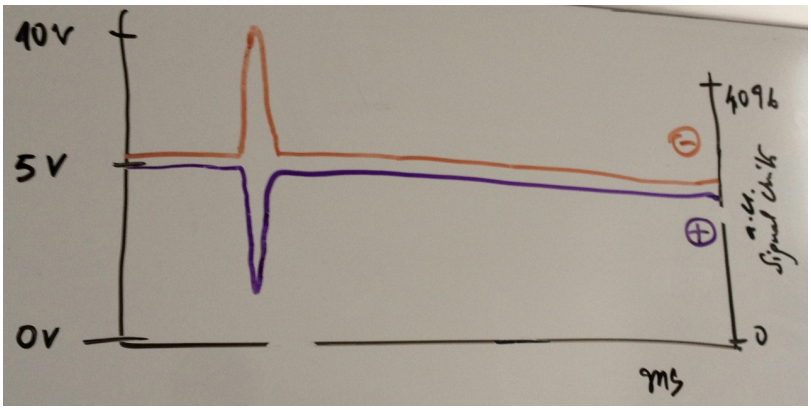


Abb. 8: Die verschiedenen Modi: Positiv (lila) bzw. negativ (orange). Illustration gezeichnet von Dr. Maddula

IMS-CSV-Datei) bei 0 liegt und der Wertebereich somit ca. 0 bis −2048 bzw. ca. 0 bis 2047 ist.

Eine weitere Möglichkeit war die Option, die Daten vom MCC/IMS-Messgerät an einem ausgegebenen, hochohmigen, analogen Signal direkt abzugreifen. Dies ist allerdings nur möglich, wenn dieses Signal verstärkt vorliegt bzw. auf den korrekten, auszugebenden Messbereich skaliert wird. D. h. die Signalintensitäten müssen mit einem Operationsverstärker oder einem Spannungsteiler auf den Eingangsbereich des verwendeten Analog/Digital-Wandler angepasst werden. Ein Problem dabei sind zeitliche „Verzerrungen“, die herausgerechnet werden müssen, d. h. die Peaks „wandern“. Aus diesen Gründen ist diese Möglichkeit für den Rahmen der Projektgruppe als zu fehleranfällig ausgeschlossen worden.

Von der Firma B & S Analytik ist geplant, zukünftig ein kleineres MCC/IMS-Gerät produzieren zu lassen, dessen Anschlussmöglichkeiten noch nicht restlos geklärt sind, so dass die Simulation der Netzwerklösung des bestehenden MCC/IMS-Geräts als beste Lösung für die Projektgruppe angesehen worden ist. Die Projektgruppe verwendet daher in ihrer Implementierung eine Simulation des Netzwerkanschlusses in Form einer Server- und Client-Implementierung, wie in Kapitel 5 dargestellt.

3 RASPBERRY PI

Als Hardwareplattform verwendet die Projektgruppe den Raspberry Pi, einen als günstigen Computer zum Einsatz in der Lehre konzipierten, kreditkartengroßen Ein-Platinen-Rechner auf Basis des Broadcom BCM2835 System on a Chip (siehe auch Abschnitt A.2). Für unter 40 € bietet das Modell B mit 512 MiB RAM eine Rechenleistung, die zum Ausführen einer grafischen Benutzeroberfläche und zur Wiedergabe von Videos in Full-HD-Auflösung ausreicht.

In diesem Kapitel geben wir im Abschnitt 3.1 zunächst einen Überblick über die dem Raspberry Pi zugrundeliegende ARM-Architektur und den zugehörigen Befehlssatz. Auf die Auswahl des Betriebssystems gehen wir in Abschnitt 3.2 kurz ein. Abschließend erläutern wir in Abschnitt 3.3 die Möglichkeit, einen Cross-Compiler zur Entwicklung für den Raspberry Pi einzusetzen.

3.1 ARM-ARCHITEKTUR/-ASSEMBLER

Im Raspberry Pi ist der SoC (System on a Chip) Broadcom BCM2835 verbaut. Dieser enthält den ARM-Prozessor ARM1176JZF-S aus der ARM11 Familie, die zur 32-Bit ARMv6 Architektur gehört. Die ARM-Architektur ist ein Reduced Instruction Set Computer (RISC) Befehlssatz. Das Ziel einer RISC-Architektur ist eine hohe Ausführungsgeschwindigkeit, welche durch die folgenden Maßnahmen erreicht wird: eine geringe Anzahl interner Zyklen pro Befehl, einfache Befehle, Fließbandverarbeitung und ein CPI-Wert der kleiner oder gleich eins ist. Unter dem CPI-Wert (cycles per instruction), einer Menge von Maschinenbefehlen, versteht man die mittlere Anzahl interner Bus-Zyklen pro Maschinenbefehl. Der ARM1176JZF-S Prozessor enthält drei verschiedene Befehlssätze, zu denen die Befehlssätze 32-Bit ARM, 32/16-Bit Thumb(2) und 8-Bit Jazelle zählen. Die ARM und Thumb(2) Befehlssätze enthalten die Single Instruction Multiple Data Befehle (SIMD), um beispielsweise Multimediadaten, die nur eine geringe Bitbreite benötigen, parallel bearbeiten zu können. Die VFPv2 (Vector Floating Point Version 2) Funktionsmerkmale des ARM1176JZF-S Prozessors ermöglichen es, dass Gleitkommazahloperationen direkt in Hardware ausgeführt werden und nicht in Software emuliert werden müssen. Weitere Informationen können im ARM Architecture Reference Manual [1], ARM1176JZF-S Technical Reference Manual [2] und im Buch „Computer architecture a quantitative approach“ [24] eingeholt werden.

Die Benutzung von Assembler ermöglicht die Optimierung von ressourcenintensiven Stellen im Quelltext und wurde für die Beschleunigung der Algorithmen angedacht. Da unser C/C++ Code performant genug ist, wurde dieser Ansatz nicht weiterverfolgt. Weitergehende Informationen befinden sich im Abschnitt A.3.

3.2 BETRIEBSSYSTEME

Für den Raspberry Pi stehen viele verschiedene Betriebssysteme zur Auswahl, welche je nach Verwendungszweck verschiedene Vor- und Nachteile haben. Für unsere Zwecke war ein leichtgewichtiges Betriebssystem von besonderer Bedeutung: Dieses soll das Empfangen und Verarbeiten einer Messung mit möglichst geringem Overhead ermöglichen. Außerdem sollen neben unserem Algorithmus keine weiteren Anwendungen auf dem Raspberry Pi laufen.

Unsere Wahl fiel deshalb auf Raspbian, eine speziell auf das Raspberry Pi angepasste Version von Debian. Gründe hierfür waren, neben der Leichtgewichtigkeit und Auslegung auf die Hardware, die große Unterstützung bei etwaigen Problemen, da Raspbian auch allgemein das am weitesten verbreitete Betriebssystem für das Raspberry Pi ist.

3.3 CROSS-COMPILING

Unter Cross-Compiling wird das Kompilieren auf einem System mit einer Architektur, die nicht der des Zielsystems entspricht, verstanden. Für unsere Projektgruppe ist es relevant, ausführbare Dateien für die vom Pi verwendete ARMv6-Architektur auf der weit verbreiteten und zur Entwicklung benutzten x86(_64)-Architektur zu erstellen.

Der Raspberry Pi ist wie ein vollwertiger Desktop-Rechner mit allen üblichen Programmen benutzbar, darunter auch eine Build-Umgebung mit allen notwendigen Tools und Bibliotheken. Tatsächlich ist es möglich, auf Cross-Compiling zu verzichten und z. B. zum Testen direkt auf dem Pi zu kompilieren. Jedoch macht sich dabei die schwache Rechenleistung des ARM-Prozessors stark bemerkbar. Während auf einem einigermaßen aktuellen x86-Desktop das gesamte Projekt in unter 10 s kompiliert werden kann, dauert dies auf dem Pi um die 5 min. Durch Makefiles mit automatischer Abhängigkeitserkennung müssen bei Änderungen an Quellcodedateien zwar nur die betroffenen Teile des Projekts neu gebaut werden, jedoch dauert auch das Kompilieren einer einzelnen C++-Datei mehr als 30 s. Somit ist ein effizientes Arbeiten mit häufigem Testen nicht mehr möglich.

Am einfachsten ist die Erstellung einer Cross-Toolchain mit vorgefertigten Scripts wie `crossdev`[1] oder `crosstool-ng`[2]. Ein Beispielaufruf zur Erstellung einer Toolchain für die Architektur des Pi mit GCC 4.7 inklusive binutils, glibc und passendem GDB ist:

```
crossdev -t armv6j-hardfloat-linux-gnueabi --g 4.7.3 --ex-gdb
```

Um den so erstellten Cross-Compiler mit CMake benutzen zu können, muss eine sogenannte Toolchain-Datei wie in Listing 1 angelegt werden. In dieser Datei werden die Namen der zu verwendenden C- und C++-Compiler sowie der Pfad zu den Bibliotheken und Header des Zielsystems angegeben. Diese Toolchain-Datei kann nun CMake beim Konfigurieren eines Build-Verzeichnisses mittels

1 Gentoo-spezifisch, http://packages.gentoo.org/package/sys-devel/crossdev
2 http://crosstool-ng.org/

```
set (CMAKE_SYSTEM_NAME Linux)
set (CMAKE_SYSTEM_VERSION 1)
set (CMAKE_C_COMPILER armv6j-hardfloat-linux-gnueabi-gcc)
set (CMAKE_CXX_COMPILER armv6j-hardfloat-linux-gnueabi-g++)
set (CMAKE_FIND_ROOT_PATH /mnt/raspbian/)
set (CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set (CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set (CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

Listing 1: Beispiel einer Toolchain-Datei

```
mount -o loop,offset=\
$(fdisk -l 2013-05-25-wheezy-raspbian.img \
| grep Linux \
| awk '{print $2 * 512}') \
2013-05-25-wheezy-raspbian.img /mnt/raspbian/
```

Listing 2: Mounten eines Raspbian-Images per loop-device

`-DCMAKE_TOOLCHAIN_FILE=rpi-toolchain.cmake`

übergeben werden.

Wenn man schon einen Pi mit dem Zielsystem zur Verfügung hat, kann man diesen z. B. per `sshfs` direkt mounten, um auf die benötigten Dateien zugreifen zu können. Von Vorteil ist, dass dies sehr flexibel und mit wenig Arbeit verbunden ist. Werden auf dem Pi Pakete aktualisiert, stehen diese sofort auf dem Host-System zur Verfügung. Allerdings ist der Zugriff über das Netzwerk mit deutlichen Geschwindigkeitseinbußen verbunden.

Eine andere Möglichkeit ist, das Raspbian-Image, das man auf eine SD-Karte schreiben könnte, wie in Listing 2 gezeigt als loop-device mit entsprechendem Offset für die ext4-Partition zu mounten.

Weiterhin ist es möglich, die `/usr/`- und `/lib/`-Verzeichnisse eines schon existierenden Zielsystems auf den Host zu kopieren. Dabei ist darauf zu achten, dass bei Aktualisierungen des Zielsystems die Kopie synchron gehalten wird.

Ein Problem, das sich durch die Verwendung von Raspbian ergeben hat, ist, dass viele Bibliotheken und Header bei diesem Betriebssystem nicht direkt in Standardverzeichnissen wie `/lib/` und `/usr/include/` liegen, sondern darin jeweils in einem architekturspezifischen Unterverzeichnis, `arm-linux-gnueabihf`, und somit nicht von CMake automatisch gefunden werden. Deshalb mussten in der CMake-Konfiguration einige Pfade fest angegeben werden. Dieses Problem besteht bei anderen Betriebssystemen wie Gentoo oder Arch Linux nicht.

4 DATEIFORMATE

In diesem Kapitel werden die drei wichtigsten Dateiformate vorgestellt, mit denen wir im Rahmen unseres Projektes gearbeitet haben. Im Abschnitt 4.1 wird zunächst das standardisierte IMS-CSV-Format beschrieben, welches das primäre Austauschformat unserer Messdaten ist. Da dieses Dateiformat aus Performanz- und Speicherplatzgründen nur schlecht zur Weiterverarbeitung geeignet ist, kam dafür mit `IMSH5` ein weiteres Format zur Anwendung, auf welches anschließend in Abschnitt 4.2 eingegangen wird. Darauf folgt in Abschnitt 4.3 eine Beschreibung des Ausgabeformats, welches wir zur Speicherung der Peakliste verwendet haben. Zum Abschluss dieses Kapitels gehen wir auf die Gründe für die Auswahl unseres binären Dateiformats `IMSH5` ein.

4.1 IMS-CSV

Beim IMS-CSV-Format handelt es sich um ein standardisiertes Dateiformat, welches von der Firma B & S Analytik zur Speicherung von Messdaten verwendet wird. Es ist menschenlesbar und richtet sich nach dem CSV-Standard[1]. Die Abkürzung CSV steht für *comma-separated values*. Neben der Erfassung von MCC/IMS-Messungen eignet es sich auch für andere Messverfahren wie GC/IMS (Ionenmobilitätsspektroskopie mit Vorseparation per Gaschromatographie).

Eines der Ziele bei der Entwicklung dieses Formats war die leichte Vergleichbarkeit unterschiedlicher artverwandter Messmethoden. So soll es z.B. möglich sein, Messergebnisse von Massenspektroskop-Messungen in dieses Dateiformat zu konvertieren und diese anschließend mit MCC/IMS-Messungen zu vergleichen. Um diese und auch zukünftige Anwendungen zu ermöglichen, enthält das Format im Dateiheader einige Datenfelder, die entweder komplett ungenutzt sind oder für MCC/IMS-Messungen keine Rolle spielen.

Im Folgenden werden einige der für unsere Berechnungen relevanten Datenfelder erläutert. Sämtliche von uns verwendete Messungen liegen in Version 0.3 des Dateiformats vor. Eine vollständige tabellarische Dokumentation des Dateiheaders mit Beispielwerten findet sich im Anhang A.1. Ein Großteil unserer Informationen über dieses Format beruht auf den Arbeiten von Vautz et al. [37] und Baumbach [6]. Einige Details wurden uns darüber hinaus von Mitarbeitern der Firma B & S Analytik erklärt.

Wichtig ist, dass es sich hier um ein Rohdatenformat handelt und per Definition nach Abschluss der Messung keine weiteren Änderungen an der Datei vorgenommen werden dürfen. Umrechnungen der Messwerte sowie Korrekturen oder Ergänzungen der Metadaten müssen in separa-

1 http://tools.ietf.org/html/rfc4180

ten Dateien gespeichert werden. In Abschnitt 4.2 wird näher auf diesen Umstand eingegangen.

Die Dateinamen des IMS-CSV-Formats folgen grundsätzlich dem Schema `NNNN_YYMMDDhhmm_ims.csv`, wobei `NNNN` die Seriennummer des verwendeten Gerätes ist und `_ims` die verwendete Messmethode beschreibt. In unserem Projekt werden wir ausschließlich mit IMS-Messungen arbeiten.

Eine IMS-CSV-Datei besteht aus 130 Zeilen Dateiheader gefolgt von zwei Index-Zeilen und einer Datenmatrix, die sich je nach Auflösung der Driftzeiten über 2500 oder 12500 Zeilen erstreckt. Die Anzahl der Matrix-Spalten ist sowohl von der Gesamtdauer der Messung als auch von der Auflösung der Retentionszeiten abhängig.

Der Dateiheader gliedert sich in folgende Abschnitte:

GENERAL INFORMATION (ZEILE 1 - 11) Hier finden sich sehr allgemeine Informationen wie der Zeitpunkt der Messung, die eingesetzte Software(-Version) und die Seriennummer des Messgeräts.

SAMPLE INFORMATION (ZEILE 12 - 22) Dieser Abschnitt beinhaltet mehrere Kommentarfelder sowie Informationen zu Ort und Dauer der Messung.

IMS - INFORMATION (ZEILE 23 - 78) In diesem Abschnitt befinden sich einige sehr wichtige Informationen wie Start und Ende des Aufzeichnungintervalls, die Anzahl der Datenpunkte pro Spektrum, die Anzahl der Spektren und die Baseline (Zeile 42). Bei der Baseline handelt es sich um einen von der Messsoftware berechneten Offset, der vor der Speicherung von jedem Datenpunkt abgezogen wird. Weiterhin finden sich hier alle automatisch erfassten Messparameter wie Temperaturen, Drücke und Durchflussraten und Informationen zum Typ des Messgeräts.

EXTERNAL SAMPLING CONTROL (ZEILE 79 - 97) Dieser Abschnitt ist nur bei Verwendung eines Mundstücks wie dem SpiroScout der Firma B & S Analytik von Bedeutung.

STATISTICS (ZEILE 98 - 130) Hier speichert die Messsoftware einige statistische Werte der Messung sowie die Position und Höhe des RIPs und dessen Signal-Rausch-Verhältnis. Weiterhin befindet sich in Zeile 120 noch die für die Umrechnung zwischen inverser reduzierter Mobilität $1/K_0$ und Driftzeit t_d benötigte Gerätekonstante F_{ims} (siehe Gleichung (4)).

4.2 IMSH5

Während das IMS-CSV-Format sich durch seine Standardisierung als Austauschformat eignet, erweist es sich für die Anwendungsfälle in unserem Projekt als ungünstig:

Zur Simulation einer IMS-Messung mittels einer Messdatei ist es nötig, die Messdaten spektrenweise zu verarbeiten. Im IMS-CSV-Format sind die Daten allerdings chromatogrammweise organisiert, so dass zum Auslesen eines Spektrums die komplette Datei im Speicher gehalten werden muss. Besonders für Messdaten in hoher Auflösung ist so der Speicherbedarf für Simulationen, die direkt auf dem Raspberry Pi durchgeführt werden, unverhältnismäßig hoch.

Diese Messdateien erreichen im IMS-CSV-Format einen Umfang von mehr als 250 MiB und erfordern selbst auf Desktop-Rechnern eine erhebliche Parsing-Zeit, bevor mit der Verarbeitung der Daten begonnen werden kann. Besonders zum Training des Klassifikators und zur Evaluation der Algorithmen wird eine große Anzahl von Messungen benötigt, so dass eine kompaktere Darstellung der Messungen wünschenswert ist.

Innerhalb der IMS-CSV-Dateien sind die Spektren mit den Drift- und Retentionszeiten sowie den inversen reduzierten Mobilitäten vermengt und können so nicht unabhängig voneinander ausgelesen werden. Metadaten werden als Kommentare dargestellt und müssen aufwendig geparst werden. Erweiterungen der Metadaten durch eigene Angaben zur Messung oder Analyseergebnisse sind im IMS-CSV-Format nicht vorgesehen und müssen über gesonderte Dateien realisiert werden.

Aufgrund dieser Nachteile entstand der Wunsch, die Messdaten in einem erweiterbaren, kompakten Dateiformat abzulegen. Die Wahl fiel dabei auf das *Hierarchical Data Format* `HDF5`, das sowohl kompakt als auch performant zu lesen und schreiben ist und besonders in Python und C/C++ gut unterstützt wird (vgl. Abschnitt 4.4).

Eine solche `IMSH5`-Datei enthält alle Daten, die eine entsprechende `IMS-CSV`-Datei enthalten würde. Allerdings ist der Daten-Bereich, der in `IMS-CSV` zusammengefasst ist, hier auf vier Datensätze aufgeteilt:

`/IMS/R` ist ein Vektor mit Retentionszeiten,

`/IMS/D` ein Vektor mit Driftzeiten,

`/IMS/T` ist ein Vektor mit inversen reduzierten Mobilitäten und

`/IMS/S` ist eine Matrix mit den Sampledaten der Spektren.

In `/IMS/S` liegen die Daten im Vergleich zum `IMS-CSV`-Format transponiert vor, so dass die Zeilen der Matrix die Spektren enthalten und die Spalten die Chromatogramme. Durch ein zeilenweises Abspeichern der Matrix sind die Spektren also zusammenhängende Blöcke in der Datei. Während Retentions- und Driftzeiten sowie inverse reduzierte Mobilitäten als Double-Werte gespeichert werden, lässt sich für die Spektren-Matrix der Wertebereich der Sampledaten ausnutzen, so dass diese im Datentyp Short abgelegt werden kann. Die in Abschnitt A.1 beschriebenen Metadaten sind einzeln als Attribute der Gruppe `/IMS` abgelegt. Dabei wird wenn möglich der entsprechende Integer- oder Fließkomma-Datentyp verwendet, ansonsten UTF8-String.

4.3 PEAKLISTE

Peaks, die ein Peakerkennungsverfahren findet, sollen als eine `Peakliste` ausgegeben werden. Dazu werden sie in einer CSV-Datei gespeichert, die in der Form aufgebaut ist, wie in Tabelle 1 dargestellt (vgl. Bunkowski [9], S. 24). Bei der Tabelle handelt es sich um einen Ausschnitt einer Peakliste, bei der die Zeilen durch Umbrüche und die einzelnen Zeileneinträge durch Tabulatoren getrennt werden. Der Header nimmt die erste Zeile der Datei ein und enthält Beschriftungen zu den gespeicherten Peakinformationen. Unter dem Header enthalten die Zeilen nur noch Peakinformationen. In jeder Zeile der Datei werden folgende Informationen zu einem einzelnen Peak gespeichert:

- *measurement_name (String)*: Dateiname der IMS-Messung, in welcher der Peak gefunden worden ist.
- *peak_name (String)*: Eine frei wählbare Bezeichnung für den Peak (in der Abbildung sukzessiv nummeriert mit `P0` bis `P4`).
- *t (Double)*: Inverse reduzierte Mobilität des Mittelpunkts des Peaks in s.
- *r (Double)*: Retentionszeit des Mittelpunkts des Peaks in ms.
- *signal (Double)*: Signalintensität des Peaks.
- *index_t (Integer)*: Position des Peak-Mittelpunkts als Index entlang der Driftzeit.
- *index_r (Integer)*: Position des Peak-Mittelpunkts als Index entlang der Retentionszeit.

Aus Kompatibilitätsgründen dieses Formats der Peaklisten zu dem vorgegebenen Format der Firma B & S Analytik ist zusätzlich ein Programm entwickelt worden, das eine Konvertierung von CSV-Peaklisten zu dem Peaklisten-Format von B & S Analytik durchführt. Einen Ausschnitt einer Peakliste, die im Format von B & S Analytik gespeichert ist, stellt Tabelle 2 dar. In den ersten beiden Zeilen befinden sich allgemeine Dateiinformationen:

- *ID (String)*: Eindeutige Bezeichnung der Messung.
- *Type*: Annotierungsweise von Peaks (*ovaldot/middle/rect*).

measurement_name	peak_name	t	r	signal	index_t	index_r
BA01_1101110902_ims	P0	0.88213	109	28.9817	1508	215
BA01_1101110902_ims	P1	0.57063	121	38.3533	978	239
BA01_1101110902_ims	P2	0.58297	79	10.1807	999	156
BA01_1101110902_ims	P3	0.53596	6	78.1608	919	13
BA01_1101110902_ims	P4	0.48306	0	73.1626	829	1

Tabelle 1: Format zur Speicherung von Peaklisten als CSV-Dateien.

ID	Type	default Color	Properties
<text>	middle	x	aced000570

Name	Comment	$1/K_0$	RT	$1/K_0$ radius	RT radius	Color
P0	Peak 0	0,54229	49,732	0,023	6,33260	-16777216

Tabelle 2: Format zur Speicherung von Peaklisten als XLS-Dateien.

- *default Color (Integer)*: Farbwert des Formats.
- *Properties*: Immer *aced*000570.

Anschließend folgt eine leere dritte Zeile und eine vierte Zeile, die Bezeichner für die in den Folgezeilen gespeicherten Peakinformationen auflistet. Diese sind:

- *Name (String)*: Bezeichnung des Peaks.
- *Comment (String)*: Kommentar (z.B. Angabe der gemessenen Substanz, welcher der Peak entsprechen soll).
- $1/K_0$ *(Float)*: Inverse reduzierte Mobilität des Mittelpunkts des Peaks in s.
- RT *(Float)*: Retentionszeit des Mittelpunkts des Peaks in ms.
- $1/K_0$ *radius (Float)*: Radius des Peaks für inverse reduzierte Mobilität.
- RT *radius (Float)*: Radius des Peaks für Retentionszeit.
- *Color (Integer)*: Farbwert.

Die Genauigkeiten der einzelnen Spalteneinträge wurden in der Tabelle aus Platzgründen abgekürzt, sind jedoch wie folgt festgelegt: Für 1/K0 sind 15 Nachkommastellen, für RT und 1/K0 *radius* jeweils 3 Nachkommastellen sowie für RT *radius* 14 Nachkommastellen vorgesehen. Ein Eintrag für einen Peak in dem Format ist in Zeile 5 (Peak P0) beispielhaft angegeben.

4.4 AUSWAHL DES BINÄREN DATEIFORMATS

In diesem Abschnitt beschreiben wir die Gründe für die Auswahl unseres binären Dateiformats auf Basis von HDF5 als Alternative zu IMS-CSV-Dateien, die sich, wie schon in Abschnitt 4.2 dargelegt, für unsere Zwecke als wenig geeignet erwiesen haben.

4.4.1 *NPZ*

Die naheliegende Lösung bei unseren Experimenten in Python war das NumPy-eigenen Dateiformat NPY[2] bzw. sein Archivformat NPZ, welches

2 https://raw.github.com/numpy/numpy/master/doc/neps/npy-format.txt

das getrennte Ablegen mehrerer NumPy-Arrays in einem wahlweise komprimierten ZIP-Container erlaubt. Dabei kann für jedes Array ein jeweils auf den Wertebereich der Daten abgestimmter Datentyp verwendet werden, was eine kompakte Darstellung ermöglicht. Eine besondere Unterstützung für Metadaten bietet auch das NumPy-Format nicht, so dass Metadaten behelfsmäßig als Array dargestellt werden müssen. Für C/C++ steht ein `NPZ`-Parser lediglich als privates Projekt zur Verfügung[3].

4.4.2 *HDF*

Bei der Suche nach einem Dateiformat mit einer guten C/C++-Unterstützung stießen wir auf das *Hierarchical Data Format* `HDF`. Dieses ursprünglich am National Center for Supercomputing Applications an der University of Illinois entwickelte und nun von der HDF Group[4] betreute und weiterentwickelte offene Dateiformat ist auf die effiziente Speicherung großer wissenschaftlicher Daten ausgelegt. Dabei lassen sich in einer Datei beliebig viele Datensätze, etwa Vektoren oder Matrizen, in verschachtelten Gruppen in einem virtuellen Dateisystem ablegen. Innerhalb einer `HDF`-Datei sind Aliase möglich und Referenzen auf externe `HDF`-Dateien werden transparent aufgelöst. Auf jeden Datensatz können einzeln verschiedene Filter und Kompressionsalgorithmen angewendet werden. Zu Gruppen und Datensätzen lassen sich beliebig Attribute verschiedener Datentypen definieren. Die Datensätze bieten einen wahlfreien Zugriff auf Teilmengen von Daten, ohne den kompletten Datensatz in den Speicher laden zu müssen. Für die aktuelle Version des Formats, `HDF5`, gibt es neben Bibliotheken für C und C++ auch eine gute Unterstützung für Python.

4.4.3 *Benchmarks*

Um die erreichbaren Lese- und Schreibgeschwindigkeiten sowie die Dateigrößen der Formate und den Einfluss verschiedener Kompressions- und Filterkonfigurationen bewerten zu können, haben wir verschiedene Benchmark-Reihen durchgeführt[5].

Während das `NPZ`-Format lediglich in einer komprimierten und einer unkomprimierten Variante zur Verfügung steht, bietet `HDF5` verschiedene Kompressions- und Filteralgorithmen an. Wir haben uns im Rahmen der Benchmarks auf die mitgelieferten Kompressionsverfahren `gzip`, `Szip` und das von der Python-Bibliothek *h5py*[6] bereitgestellte `LZF` beschränkt. Außerdem haben wir den Einfluss des *Shuffle*-Filters[7] untersucht, der die Datenbytes vor der Kompression umsortiert, um bei Daten, die nicht die volle Breite ihres Datentyps nutzen, einen höheren

3 https://github.com/rogersce/cnpy
4 http://www.hdfgroup.org/
5 Benchmark-System: MacBook Pro 3,1, Core2-Duo 2,4 GHz, 6 GiB RAM, OS X 10.8.4 im Single-User-Modus, 1 TB Western Digital WDC WD10JPVT, Journaled HFS+ mit AES-XTS. Python 3.3.2, NumPy 1.7.1, h5py 2.1.2-89-gd699049, HDF5 1.8.11
6 http://www.h5py.org/
7 http://www.hdfgroup.org/HDF5/doc_resource/H5Shuffle_Perf.pdf

Abkürzung	Format	Kompression	Stufe	Filter
csvu	IMS-CSV			
npzu	NPZ			
npzc	NPZ	gzip		
hdfu	HDF5			
hdfs	HDF5	Szip		
hdfss	HDF5	Szip		Shuffle
hdfl	HDF5	LZF		
hdfls	HDF5	LZF		Shuffle
hdfg	HDF5	gzip	4	
hdfgs	HDF5	gzip	4	Shuffle
hdfg5	HDF5	gzip	5	
hdfg5s	HDF5	gzip	5	Shuffle
hdfg7	HDF5	gzip	7	
hdfg9	HDF5	gzip	9	

Tabelle 3: Untersuchte Dateiformate

Kompressionsgrad zu erreichen. Tabelle 3 gibt eine Übersicht über die untersuchten Formate.

Für den Schreib-Benchmark wurden sechs IMS-Messungen in reduzierter Auflösung (2500×400) und sechs Messungen in hoher Auflösung (12500×6000) mittels eines Python-Skripts in den verschiedenen Formaten jeweils fünfmal gespeichert und die dazu benötigte Zeit gemessen. Im Lese-Benchmark wurden diese Dateien jeweils fünfmal komplett eingelesen. Von den so ermittelten Zeiten verwenden wir für jede Messdatei das Minimum. Außerdem wurde zu jedem Dateiformat die Dateigröße erfasst.

4.4.4 *Ergebnisse*

Bei den Dateigrößen, dargestellt in den Abbildungen 9 und 10, ergibt sich für reduzierte und volle Auflösung ein ähnliches Bild. Am größten ist jeweils das IMS-CVS-Format. Die unkomprimierten Varianten von HDF5 und NPZ kommen für die reduzierten Daten mit etwas mehr und für die voll aufgelösten Daten mit weniger als der Hälfte des Speicherplatzes aus, den das IMS-CSV-Format beansprucht. Die Szip- und LZF-komprimierten HDF5-Varianten ohne Filter schneiden nochmals deutlich besser ab. Am kleinsten sind die gzip-komprimierten HDF5-Varianten und das komprimierte NPZ. Sie liegen in etwa gleichauf und benötigen für die reduzierten Daten ca. $1/7$ und für die voll aufgelösten Daten ca. $1/8$ des Platzes von IMS-CSV, wobei NPZ für voll aufgelösten Daten etwas schlechter abschneidet. Ein höherer Kompressionsgrad wirkt sich kaum auf die erzielten Dateigrößen aus.

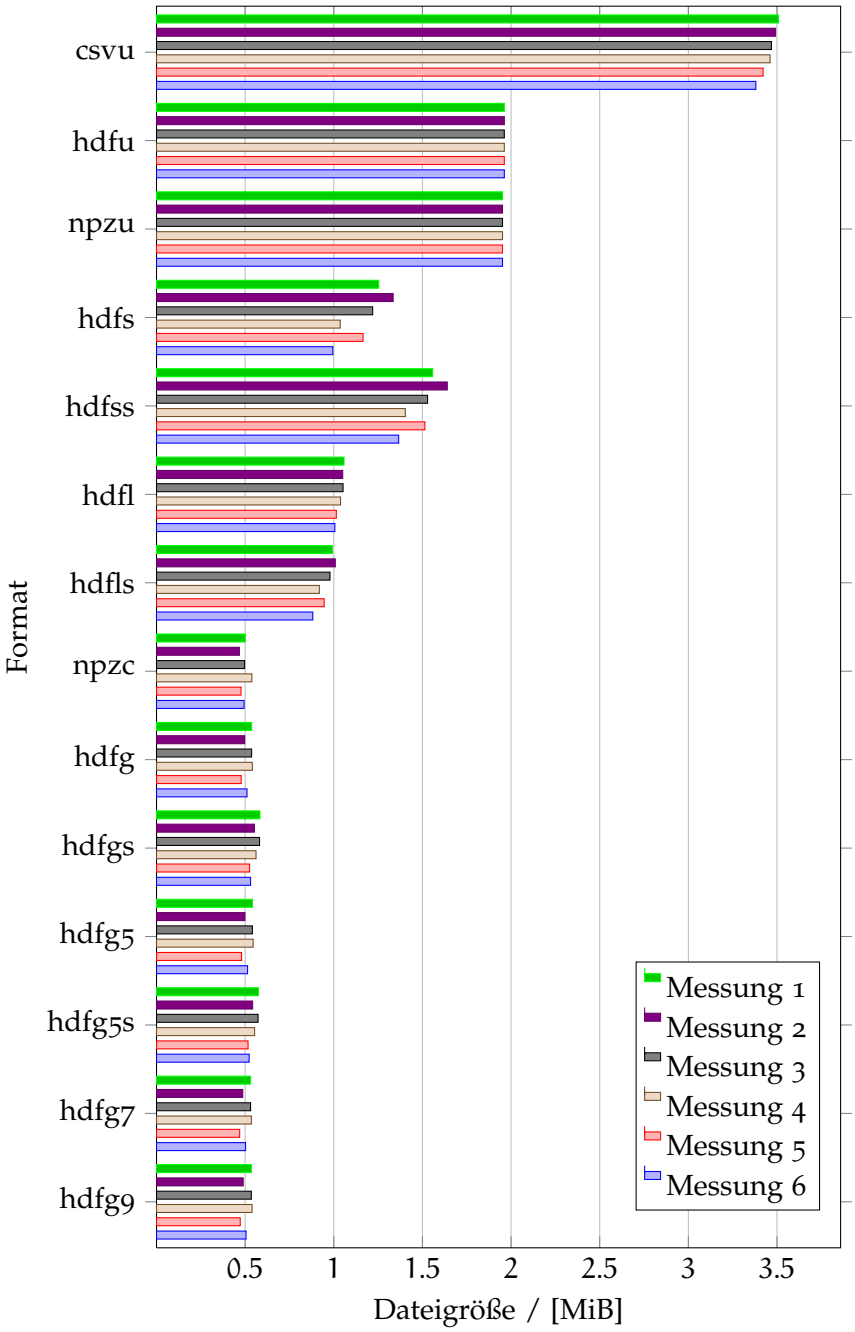


Abb. 9: Dateigrößen in verschiedenen Formaten für Messungen in reduzierter Auflösung

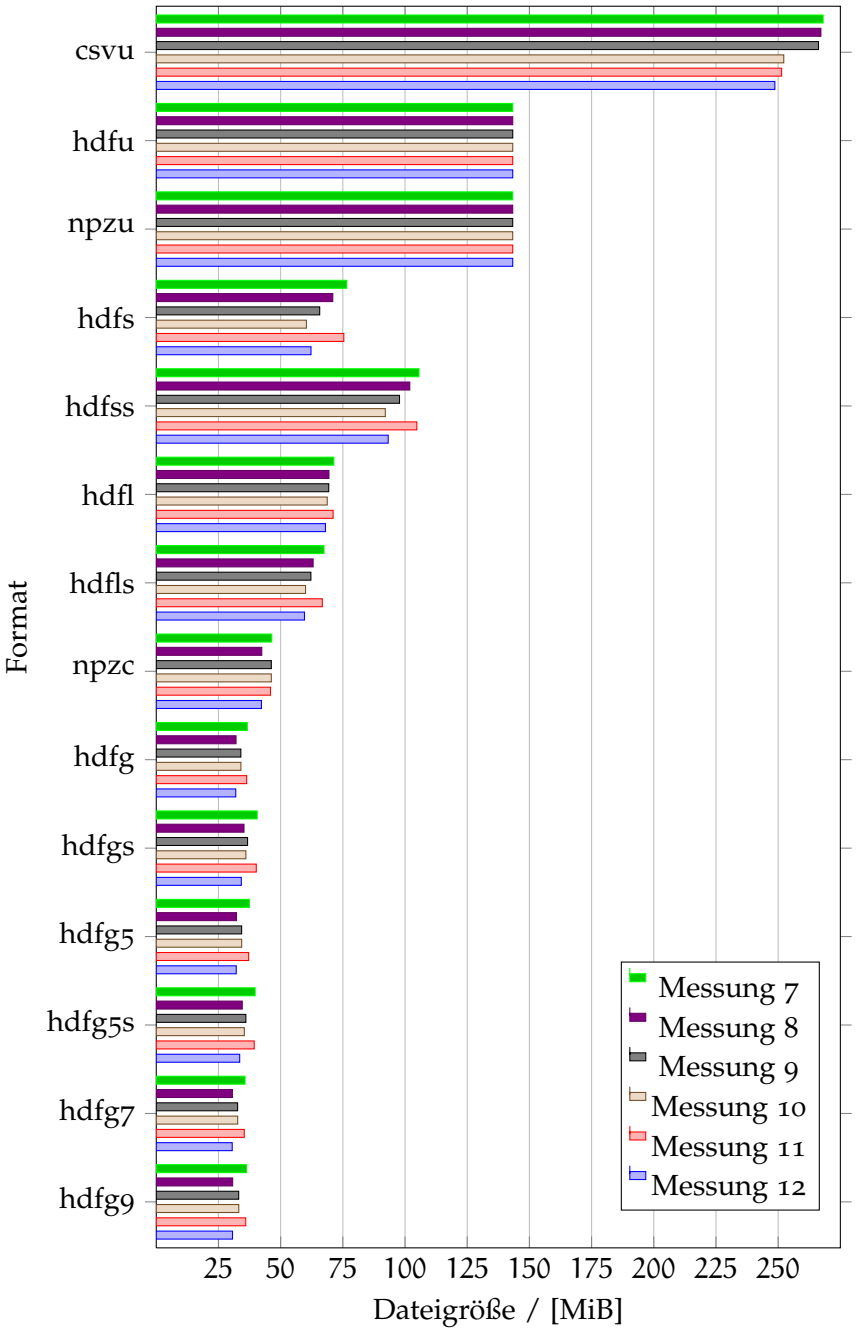


Abb. 10: Dateigrößen in verschiedenen Formaten für Messungen in voller Auflösung

Der Shuffle-Filter bringt nur für das mit LZF komprimierte Format leichte Verbesserungen. Bei der mit Szip komprimierten HDF5-Variante wird das Kompressionsverhalten sogar deutlich schlechter.

Die über die Messungen gemittelten Lesegeschwindigkeiten sind in den Abbildungen 11a und 11b dargestellt. Dabei wurde das IMS-CSV-Format ausgelassen, da es um den Faktor 27 langsamer ist als das jeweils schlechteste alternative Format.

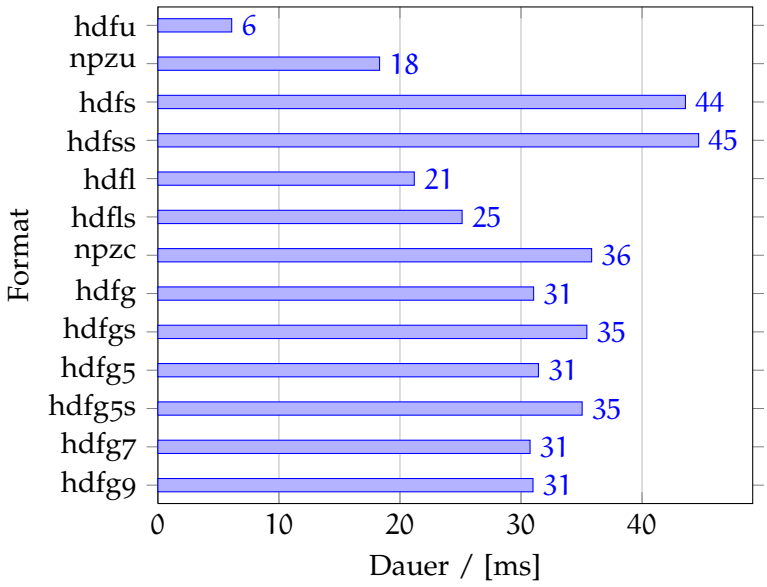
Bei den reduzierten Daten liegen die Ladezeiten bei den übrigen Formaten jeweils unter 50 ms. Mit Abstand am schnellsten ist HDF5, gefolgt von unkomprimiertem NPZ und dem LZF-komprimierten Format. Am schlechtesten schneiden Szip ab, während die mit gzip komprimierten Formate zwischen Szip und LZF anzusiedeln sind. Das komprimierte NPZ ist etwas schlechter als die gzip-HDF5-Formate. Die Anwendung des Shuffle-Filters sorgt, mit Ausnahme des mit Szip komprimierten Formates, für etwas längere Ladezeiten.

Für die Daten in voller Auflösung sind die Ergebnisse ähnlich. Auffällig ist hier, dass das komprimierte NPZ, gleich auf mit den Szip-Formaten, die längste Ladezeit unter den binären Formaten aufweist. Diese liegt ein Drittel über der schlechtesten der gzip-Zeiten.

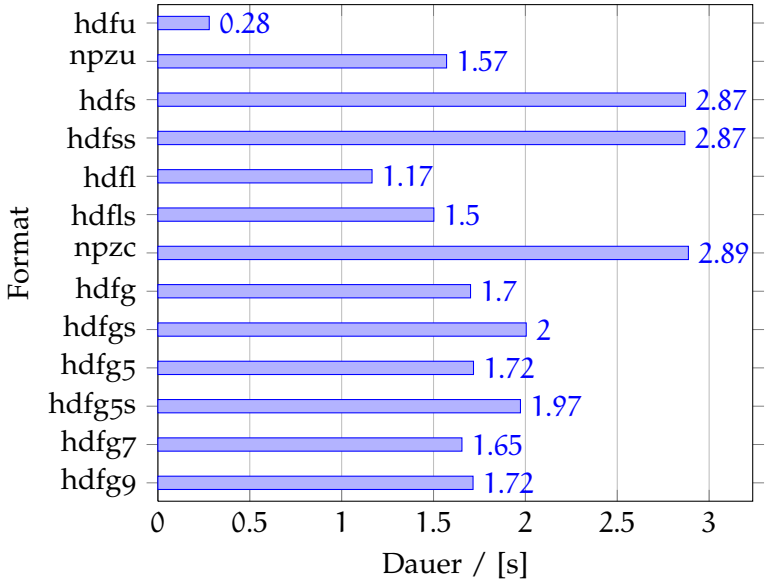
Bei den Schreib-Benchmarks haben wir das IMS-CSV-Format als reines Quellformat nicht betrachtet. Für reduzierte und voll aufgelöste Daten ergeben sich vergleichbare Resultate, dargestellt in den Abbildungen 12a und 12b. Am schnellsten lassen sich die unkomprimierten und die mit LZF komprimierten Formate schreiben. Dabei ist HDF5 wieder am schnellsten, bei den reduzierten Daten gefolgt von NPZ, bei den voll aufgelösten Daten von den LZF-Formaten. Etwas schlechter schneiden die Szip-Formate ab. Bei den voll aufgelösten Daten sind sie vergleichbar mit unkomprimiertem NPZ. Mit deutlichem Abstand folgen die gzip-HDF5-Formate, deren Schreib-Zeiten mit wachsendem Kompressionsfaktor exponentiell ansteigen. Das komprimierte NPZ-Format liegt für die reduzierten Daten gleichauf mit gzip-HDF5 Stufe 7, bei den voll aufgelösten Daten ist es deutlich schlechter. Die Formate, die den Shuffle-Filter nutzen, sind jeweils unwesentlich langsamer als die Varianten ohne Filter.

4.4.5 *Fazit*

Durch die Verwendung eines binären Dateiformates lassen sich Speicherplatzbedarf sowie Lese- und Schreib-Zeiten für Messdateien deutlich reduzieren. Die beste Performance bieten die unkomprimierten und mit LZF komprimierten Formate, führen aber zu deutlich größeren Dateien als die gzip-Formate. Größenmäßig vergleichbar mit den LZF-Formaten ist das Szip-Format, bietet allerdings eine deutlich schlechtere Performance. Mit nur leichten Nachteilen bei der Dateigröße gegenüber den gzip-HDF5-Formaten bietet das komprimierte NPZ-Format eine mittelmäßige bis schlechte Lese-Geschwindigkeit und eine enttäuschende Schreib-Performance. Die kleinsten Dateien liefern die gzip-HDF5-Formate. Dabei hat ein steigender Kompressionsfaktor keinen nennenswerten Einfluss auf die erzielten Dateigrößen, führt aber zu deutlich län-

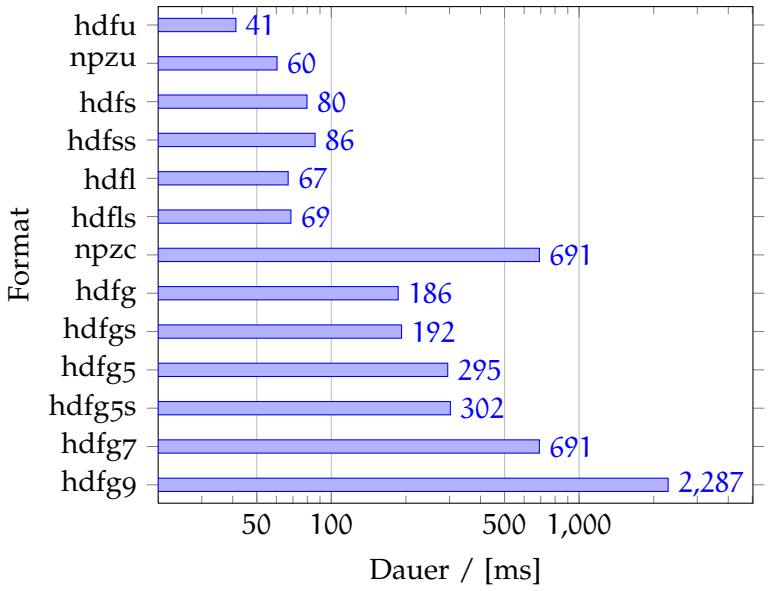


(a) Reduzierte Auflösung. *Nicht dargestellt ist csvu mit* 1,23 s.

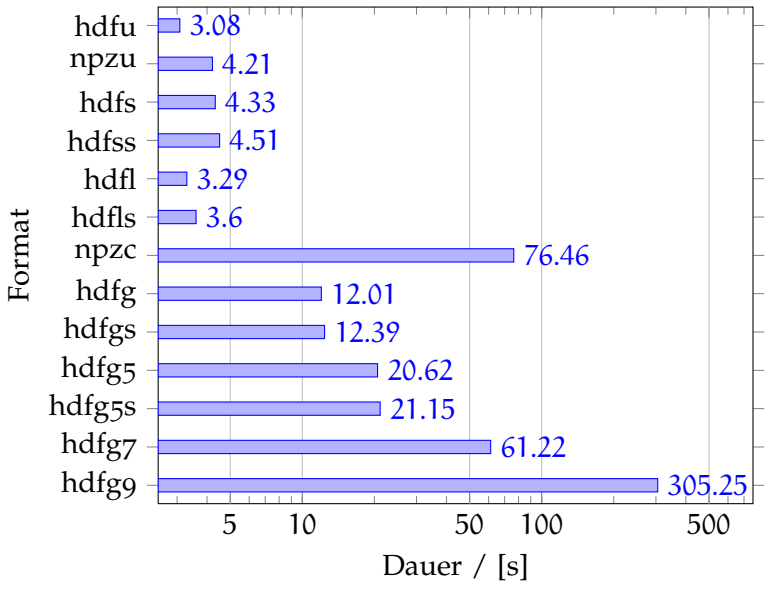


(b) Volle Auflösung. *Nicht dargestellt ist csvu mit 79,23* s.

Abb. 11: Dauer für das Lesen verschiedener Datenformate



(a) Reduzierte Auflösung



(b) Volle Auflösung

Abb. 12: Dauer für das Schreiben verschiedener Datenformate

geren Schreibzeiten. Die Lesezeiten liegen bei diesen Formaten durchweg im Mittelfeld. Der Shuffle-Filter führt nur beim mit LZF komprimierten Format zu einer leichten Verbesserung der Dateigröße.

Insgesamt bietet das mit gzip Stufe 4 komprimierte HDF5-Format den besten Kompromiss aus Dateigröße und Performance und unterstützt Metadaten nativ. Wir nutzen dieses Format daher als binäre Alternative zu IMS-CSV.

5 SERVER-CLIENT-ARCHITEKTUR

Vor dem Hintergrund beschränkter Ressourcen wurde im Rahmen des Projekts eine Architektur umgesetzt, die einen Kompromiss zwischen Effizienz und Modularität bietet. Eine gewisse Modularität war erforderlich, um die Komplexität des Programms möglichst gering zu halten und einen Austausch von Programmteilen für unterschiedliche Verfahren zur Peakerkennung zu ermöglichen. Mit Bezug zur Aufgabenstellung der PG (siehe Abschnitt 1.1) wurden zwei Verfahren zur Peakerkennung (siehe Kapitel 8) entwickelt. Beiden Verfahren liegt eine gemeinsame Server-Client-Architektur zugrunde. Die Darstellung der Architektur leiten wir mit einem Überblick über den Datenfluss in Abschnitt 5.1. Darauffolgend stellen wir die Aufgaben und Funktionsweisen des Servers und des Clients entsprechend in den Abschnitten 5.2 und 5.3 vor. Auf die Struktur der Umsetzung des Klassifikationsprozesses gefundener Peaklisten wird am Ende von Abschnitt 5.3 eingegangen.

5.1 DATENFLUSS

Nach Absprache mit der Firma B & S Analytik bietet sich die Möglichkeit an, die Messwerte eines IMS-Geräts über eine Netzwerkverbindung eines an das IMS-Gerät angeschlossenen Notebooks am Raspberry Pi zu empfangen (vgl. Abschnitt 2.3). Dabei stellt das Notebook den Server dar, der alle 100 ms ein Spektrum der IMS-Messung sendet. Der Raspberry Pi empfängt dann als Client die Spektren entsprechend und führt eine Peakerkennung durch. Um diesen Vorgang zu simulieren hat die PG ein eigenes Server-System entworfen, das die Messungen an den Raspberry Pi auf die gleiche Weise sendet. Darüberhinaus wurde für interne Testzwecke die Möglichkeit implementiert, die Spektren ohne Verzögerung direkt in den Client zu laden (Quickrun).

5.2 SERVER

In Abbildung 13 (links) ist die grundlegende Funktionsweise des Servers dargestellt. Bei der Abbildung handelt es sich um ein vereinfachtes UML-Aktivitätsdiagramm, das die Kommunikationsweise des Servers mit dem Raspberry Pi, dem Client, veranschaulichen soll.

Nach dem Start liest der Server die Konfigurationseinstellungen (Verbindungsinformationen, Pfadangaben zu Messungen usw.), um dann eine Messung im CSV- oder HDF5-Format zu laden und diese auf ihre grundsätzlichen Bestandteile zu reduzieren. Die Reduzierung umfasst sowohl die Entfernung aller Headerinformationen als auch das Rückgängigmachen der Vorverarbeitungsschritte (wie Basislinienkorrektur), die den CSV-Dateien zugrunde liegen. Nach der Reduzierung besteht die Messung im Server nur noch aus ihren einzelnen Roh-Spektren, wie

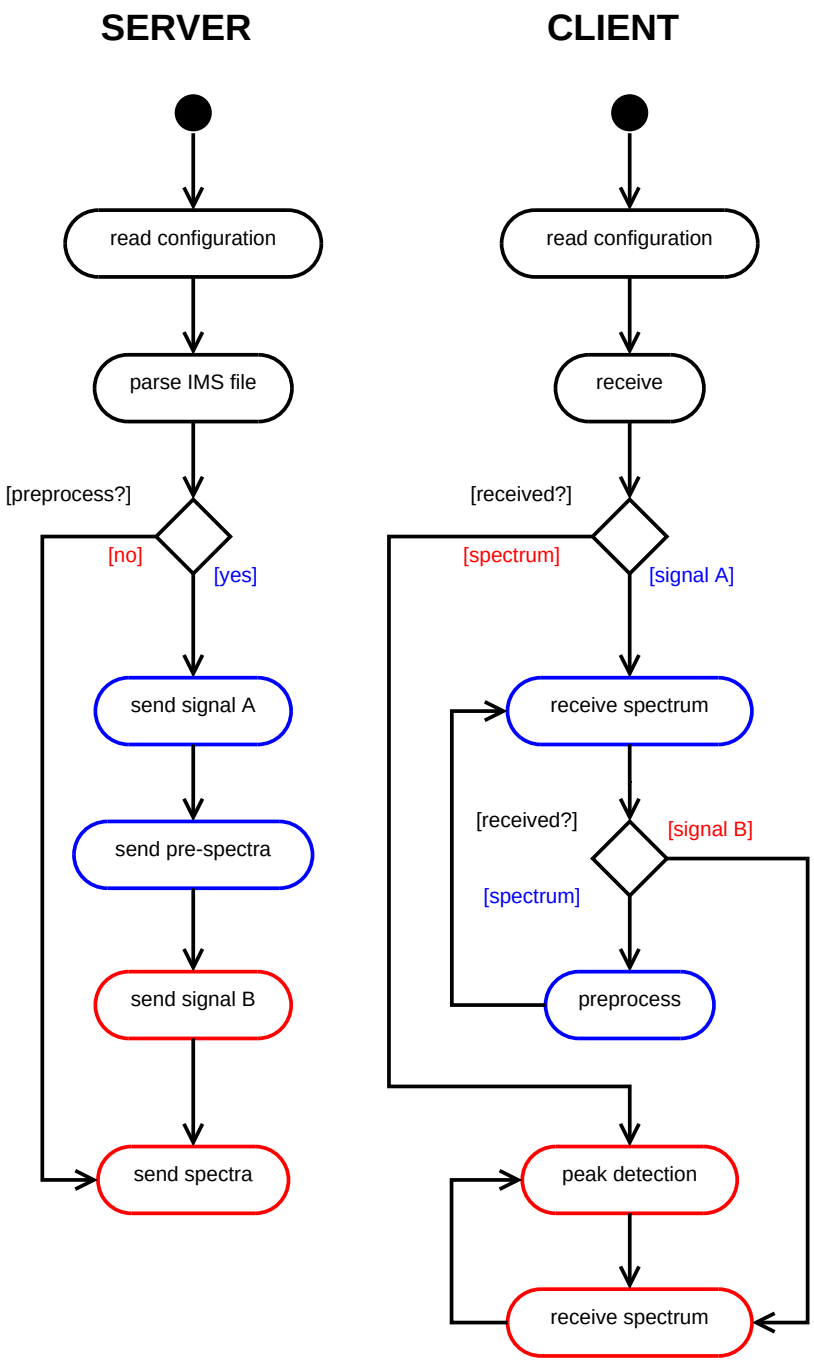


Abb. 13: Veranschaulichung des Servers (links) und des Clients (rechts) der Server-Client-Architektur als UML-Aktivitätsdiagramme.

sie das IMS-Gerät in praktischer Anwendung liefert. Da Spektren in voller Auflösung versendet werden müssen, simuliert der Server in diesem Schritt bei Verwendung gering aufgelöster Daten ein nicht gemitteltes Spektrum durch Vervielfältigung der gegebenen Signalwerte.

Nach einigen experimentellen Vergleichen wurde entschieden, dass eine Luftmessung - das sind Spektren, die keine Peaks enthalten (in Abbildung 13 (links) mit *Preprocessing* bezeichnet) - zur Schätzung von Parametern zur Vorverarbeitung der Spektren im Client durchgeführt werden sollte, bevor mit der Erkennung der Peaks fortgefahren wird. Deshalb war es erforderlich, in den Server eine entsprechende Funktio-

nalität einzubauen. Allerdings musste dabei auch die Kompatibilität zum Server-System von B & S Analytik sichergestellt werden, bei dem kein Senden einer Luftmessung vorgesehen ist.

Ein Server, der beide Aspekte erfüllt, wurde als endlicher Automat (durch äquivalentes UML-Diagramm in Abbildung 13 (links) dargestellt) modelliert, der im Grunde zwei Zustände besitzt. Ist die Konfiguration auf das Serverprogramm von B & S Analytik eingestellt, dann schickt der Server keine Luftmessung an den Client. Das bedeutet, dass unmittelbar nach Programmstart Spektren einer IMS-Messung gesendet werden, die potenziell Peaks enthalten. Ist die Konfiguration jedoch auf Luftmessung eingestellt, dann wird ein anderer Pfad in dem Aktivitätsdiagramm ausgeführt. Der Server sendet in diesem Fall ein Signal an den Client, das ihn darüber informiert, dass die nächsten Spektren zu einer Luftmessung gehören. Sobald alle Spektren der Luftmessung abgeschickt wurden, sendet der Server dem Client ein Signal als Indikator für das Ende der Luftmessung. Nach dem Signal werden nur noch Spektren gesendet, die Peaks enthalten können. Eine Terminierung des Servers ist nicht vorgesehen, da ein IMS-Gerät durchgehend Spektren misst.

5.3 CLIENT

Da die unabhängig voneinander implementierten Algorithmen jeweils die Client-Funktionalität benötigen, wurde diese in eine gemeinsam genutzte Bibliothek ausgelagert. In Abbildung 13 (rechts) ist die Funktionsweise des Clients vereinfacht als UML-Aktivitätsdiagramm angegeben. Zuerst werden die Konfigurationsparameter eingelesen, anschließend wartet das Clientprogramm auf den Erhalt eines Spektrums oder eines Signals.

Erhält der Client zu Beginn ein Spektrum, dann erfolgt eine Transition in die Aktivität *peak detection*. In dieser Aktivität wird eine vorher registrierte Callback-Funktion zur Peakerkennung mit dem empfangenen Spektrum aufgerufen. Ab diesem Zeitpunkt werden keine Luftmessungen mehr erwartet.

Handelt es sich initial um ein Signal zur Luftmessung, dann betrachtet der Client alle daraufhin empfangenen Spektren als Teil einer Luftmessung und verwendet diese Spektren dazu, Parameter für die Vorverarbeitung bei der späteren Peakerkennung zu schätzen. Erst der Empfang eines erneuten Signals signalisiert das Ende der Luftmessung, woraufhin der Client mit der Verwendung aller anschließend erhaltener Spektren zur Erkennung von Peaks beginnt.

Um potenzielle Wartezeiten bei Ein- und Ausgabeoperationen zu vermeiden, wurden mehrere parallel arbeitende Threads realisiert. Ein Thread übernimmt den Empfang der Spektren und wartet somit größtenteils auf Netzwerkaktivitäten, währenddessen die CPU für andere Aufgaben zur Verfügung steht. Weiterhin wird jede registrierte Callback-Funktion in einem eigenen Thread ausgeführt. So können z. B. die empfangenen Daten zu Debug-Zwecken in eine Datei geschrieben werden,

während gleichzeitig CPU-intensive Berechnungen durchgeführt werden.

Auch sind im Client optionale Funktionalitäten vorhanden, wie die Mittelung empfangener Daten zur Auflösungsreduktion oder ein Puffer, in dem eine begrenzte Anzahl Spektren zwischengespeichert werden kann, falls eine Berechnung einmal mehr als die ansonsten zur Verfügung stehende Zeit beansprucht.

KLASSIFIKATION Die Klassifikation von Peaklisten findet aufdem Raspberry Pi, also dem Client der Server-Client-Architektur, statt. Dazu wird zunächst offline, d.h. während einer Ruhezeit des Clients (oder auch auf einer anderen Maschine als dem Client), ein Modell eines Klassifikators mit Hilfe eines Peaklisten-Datensatzes trainiert und auf dem Client für einen späteren Einsatz gespeichert. Das gespeicherte Modell wird für die Online-Klassifizierung von Peaklisten benötigt. Nachdem ein Peakerkennungsverfahren zu einer Messung eine Liste von Peaks gefunden hat, wird die Peakliste im CSV-Format (siehe Abschnitt 4.3) dem Klassifikationsmodell übergeben. Die Peakliste wird dann mit Hilfe einer Merkmalsextraktion in einen für die Peakliste spezifischen Merkmalsvektor konvertiert. Anschließend wird der Merkmalsvektor einem Klassifikator übergeben, der den Vektor - dieser repräsentiert die Messung - einer Klasse zuordnet.

6 VORVERARBEITUNG DER MESSDATEN

In diesem Kapitel werden die zur Verfügung stehenden Messdaten genauer angeschaut und eine Möglichkeit der Vorverarbeitung dargelegt. Zuerst gibt es im Abschnitt 6.1 eine Analyse des Rauschens. Darauf folgt eine Veranschaulichung der Daten und deren Analyse im Abschnitt 6.2. Im Abschnitt 6.3 eine Methode zur Messdatenkorrektur und deren *online* Ansatz dargestellt.

6.1 RAUSCHANALYSE

Die eigentlichen Messwerte, wie man sie in exakten Messungen erwarten würde, werden von kleinen, anscheinend zufälligen Störungen überlagert (siehe Abb. 14). Dies ist bekannt als *Hintergrundrauschen* oder einfach nur *Rauschen*. Für jegliche Verarbeitung der Messdaten ist es wichtig, das Rauschen möglichst gut zu entfernen (beispielsweise kann ein kurzzeitiges Aufsteigen und Abfallen auch rein zufällig sein, was die Peakfindung erschwert). Es stellt sich daher die Frage, wie man das „wirkliche Signal“ vom Rauschen trennen kann.

Um ein möglichst spezifisches Verfahren zur Rekonstruktion des Signals zu erhalten, haben wir die Eigenschaften des Rauschens untersucht. In der Literatur finden sich zahlreiche Modelle, um Rauschen zu beschreiben (vgl. die Arbeit von Wang und Binford [38, Abschnitt 3.2], die Peaks in Synthetic Aperture Radar untersucht oder allgemein in der Bildverarbeitung das Buch von Gonzalez und Woods [19, S. 312 ff.]). Das eigentliche Signal $S_{r,t}$ wird zu jedem Messzeitpunkt (r, t) von einem *additiven Rauschterm* $N_{r,t}$ verfälscht. Gemessen wird

$$\tilde{S}_{r,t} = S_{r,t} + N_{r,t} \ , \tag{7}$$

wobei $N_{r,t}$ eine vom Zeitpunkt (r, t) unabhängige Zufallsvariable[1] ist. Im Weiteren begründen wir, dass das Rauschen *normalverteilt* ist.

1 In einigen Messungen war ein gewisses *Muster* im Rauschen erkennbar (Abb. 15). Insofern gibt es schon eine gewisse zeitliche Abhängigkeit. Als mögliche Erklärung haben uns Mitarbeiter von B & S Analytik Einflüsse der Wechselspannung von 50 Hz

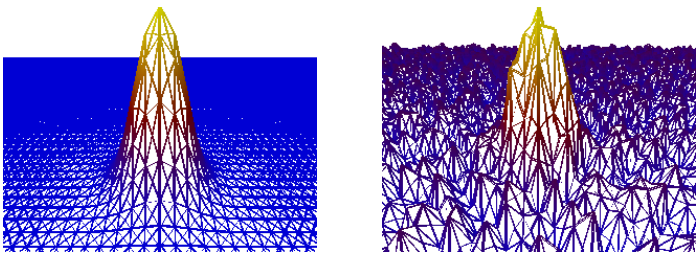


Abb. 14: Unverfälschtes Signal (links) und gemessenes Signal mit Rauschen (rechts).

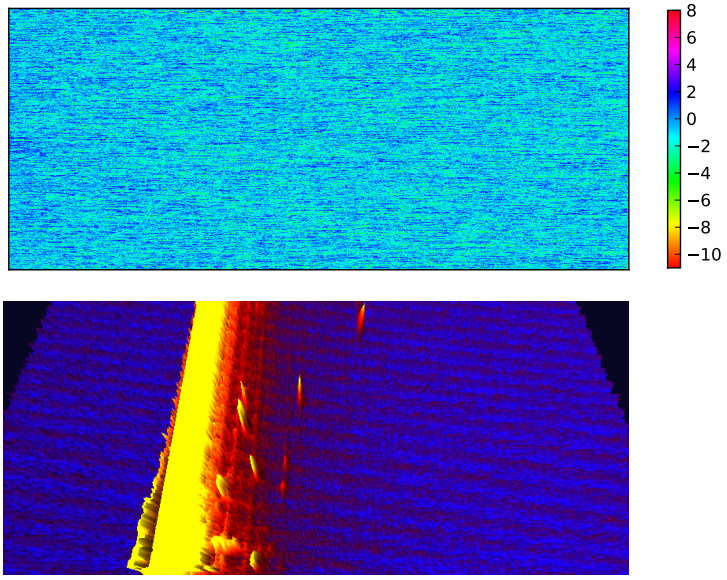


Abb. 15: Oben: Ausschnitt aus einer Messung, der nur Messrauschen enthält. Bei genauerer Betrachtung ist ein sich zeitlich wiederholendes Muster erkennbar. Unten: In einer dreidimensionalen Ansicht wird das Muster deutlicher sichtbar (andere Messung als oben).

Zur Analyse des Rauschens haben wir Bereiche in der Messung betrachtet, in denen das unverfälschte Messsignal vermutlich Null ist. Die Messung enthält also nur das reine Messrauschen (siehe Abb. 15). Laut Angaben von B & S Analytik ist dies bei Driftzeiten der Fall, die kleiner als die Driftzeiten vom RIP sind. Die genaue Position vom RIP kann von Messung zu Messung leicht variieren und muss daher jedes mal neu errechnet werden. Bei den Driftzeiten höher als ca. 35 ms ist in den meisten Messungen mit dem Auge kein Signal erkennbar.

In diesen Bereichen haben wir ein Histogramm erstellt, das zeigt, welche Signalintensitäten mit welcher (relativen) Häufigkeit auftreten (siehe Abb. 16). Mit dem bloßen Auge wurde bei allen betrachteten Messungen deutlich: Es handelt sich vermutlich um ein *normalverteiltes Rauschen*. Die zugehörigen Parameter der Normalverteilung (Standardabweichung σ und Erwartungswert μ) können per Maximum-Likelihood-Methode aus den Daten geschätzt werden und variieren je nach Messgerät nur minimal.

Ob eine Messreihe von einer Normalverteilung erzeugt wurde, lässt sich statistisch testen: Die Hypothese „die Rauschwerte sind normalverteilt“ wurde sowohl mit dem *Shapiro-Wilk-Test* als auch mit dem *Anderson-Darling-Test* verworfen[2]. Der Grund ist wahrscheinlich, dass wir nicht die analogen Werte, sondern nur die von einem Analog/Digital-Wandler gerundeten zur Verfügung haben – insofern kann es sich auch

der Stromversorgung genannt. Nach unseren Beobachtungen sind solche Muster außerdem stark vom verwendeten Messgerät abhängig und könnten teilweise mit Verunreinigungen im Messgerät zu erklären sein. Eine nähere algorithmische Betrachtung dieser Muster haben wir nicht vorgenommen.

2 Die Berechnung erfolgte über die Methoden `anderson` und `shapiro` aus dem Python-Paket `scipy.stats`. Die kritischen Werte für den Shapiro-Wilk-Test wurden berechnet wie in [31, S. 293] angegeben.

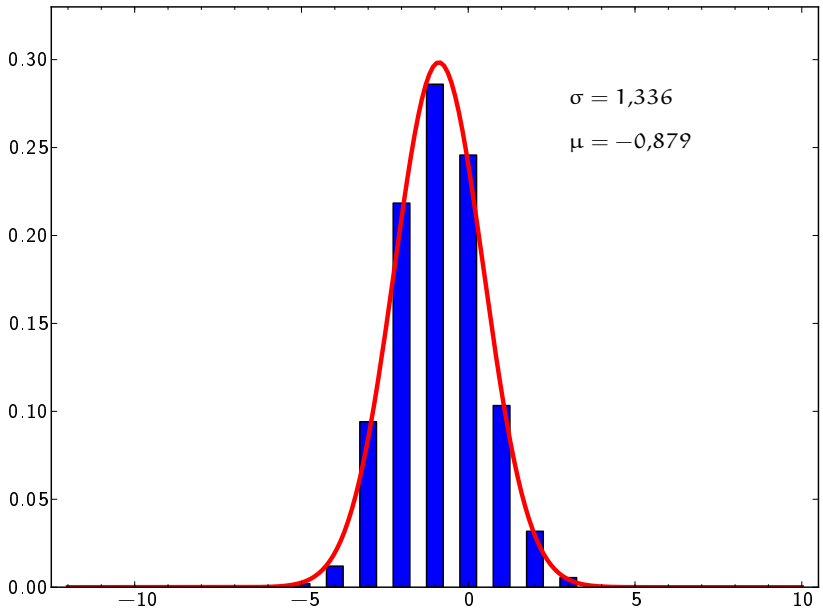


Abb. 16: Histogramm über das Rauschen

nicht um eine Normalverteilung im eigentlichen Sinne handeln. Getreu dem Motto von John W. Tukey

> *„There is no excuse for failing to plot and look“* (zitiert nach [21, S. 93])

konnten wir jedoch zumindest sinnvoll begründen, dass sich das Hintergrundrauschen nahezu wie gerundete Werte einer Normalverteilung verhält.

Die Parameter des Rauschens (Standardabweichung σ, Erwartungswert μ) können in einer Luftmessung näherungsweise berechnet werden. Ein Algorithmus zur Peakerkennung sollte diese Informationen verwenden. Die hier gewonnene statistische Beschreibung des Rauschens kann man auch dazu nutzen, um synthetische Messungen zu erzeugen (zur Peakmodellierung siehe Abschnitt 8.1), diese künstlich zu „verrauschen“ und damit die Güte eines Verfahrens zur Peakerkennung zu testen.

6.2 EIGENSCHAFTEN DER MESSDATEN

Auf einer unbearbeiteten MCC/IMS Messung (siehe Abb. 17) sind mehrere Auffälligkeiten erkennbar. Zuerst fällt der RIP auf, der bei allen Messungen auftritt und meist durchgehend von unten (niedrige Retentionszeiten) nach oben (hohe Retentionszeiten) zu einem bestimmten Driftzeitinterval auftritt. Die Position vom RIP kann von Messung zu Messung abweichen. Die Peaks zur Analyse des Atems findet man rechts vom RIP. An manchen Stellen wird der RIP durch starke Peaks eingeschnürt. Weiterhin sind ca. die ersten 10 - 20 Spektren verrauscht bzw. verschoben, so dass sie nicht als nützlich erscheinen. Diese Effekte können mit einigen Methoden korrigiert werden. In Abschnitt 6.3 betrachten wir eine dieser Methoden, die Basislinienkorrektur.

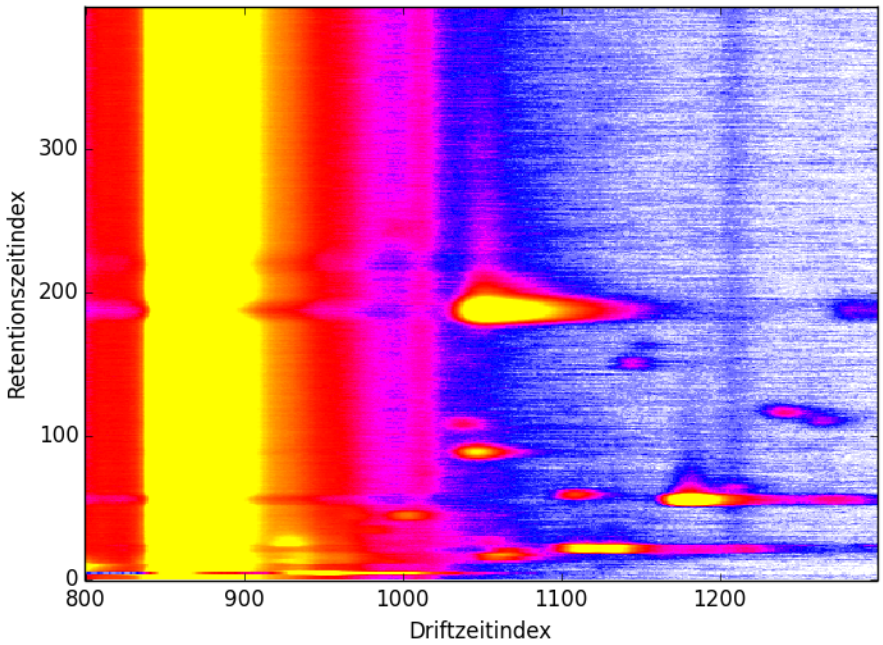


Abb. 17: Ein Ausschnitt einer unverfälschten MCC/IMS Messung. Zu sehen ist der RIP (vertikaler Streifen links) und dessen Einschnürrungen durch stark ausgeprägte Peaks (z. B. bei ca. Retentionszeitindex 190).

Wie schon erwähnt kommt der RIP in jeder MCC/IMS Messung vor. Einige der Verfahren zur Bildkorrektur, die in Abschnitt 6.3 vorgestellt werden, benötigen die Driftzeit, an der die höchste Summe der Signalstärken nachgewiesen werden kann. Dieser Driftzeitindex wird als t^* bezeichnet:

$$t^* = \arg\max_t \sum_r S_{r,t}$$

Die Analyse der von den PG-Teilnehmern selbst erzeugten Messungen, die im Abschnitt 13.2.1 genannt werden, weist dazu interessante Erkenntnisse auf. Für diese Messungen berechneten wir jeweils die Mitte des RIPs t^* und ermittelten den höchsten Wert zu diesem Driftzeitindex über alle Spektren. Abb. 18 präsentiert die Ergebnisse dieser Analyse. Die x-Achse indiziert dabei die einzelnen Messungen. Die y-Achse stellt in der oberen Abbildung den Driftzeitindex t^* und in der unteren Abbildung die höchste Signalstärke zur Driftzeit t^* in der Messung dar. Die Darstellung verdeutlicht, dass die Messungen vom Gerät und ggf. der Zeit der Durchführung abhängig sind. Man kann also nicht einen Wert für t^* festlegen, sondern dieser muss jedes Mal neu bestimmt werden.

Da die Position vom RIP in einer Messung selber nicht schwankt, reichen bereits wenige Spektren einer vorangehenden Raumluftmessung, um t^* zu ermitteln. Um die Laufzeiten der Algorithmen verbessern zu können, wird dieser Wert errechnet und die Peakerkennung erst ab dieser Driftzeit gestartet.

Um die Daten zu analysieren, haben wir jeweils pro Spektrum einer Messung eine Summe der Signalintensitäten verglichen. Die Werte einer einzelnen Messung schwanken meist um einen Wert herum. In Abb. 19 sind die Messungen und dazu jeweils ein entsprechender Überblick der

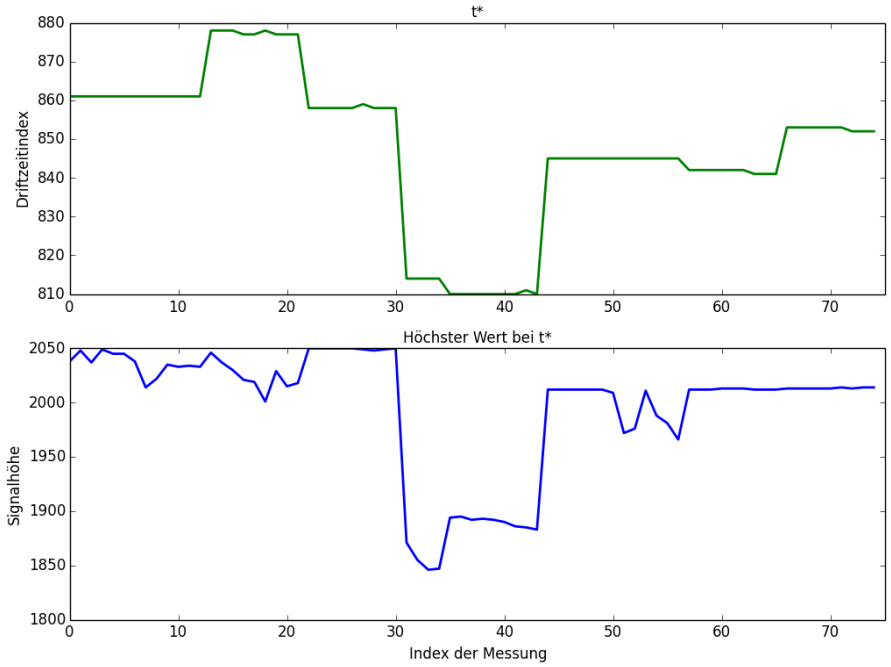


Abb. 18: Schwankungen der Position der Mitte des RIPs t* (oben) und die höchste Signalstärke zur Driftzeit t* (unten).

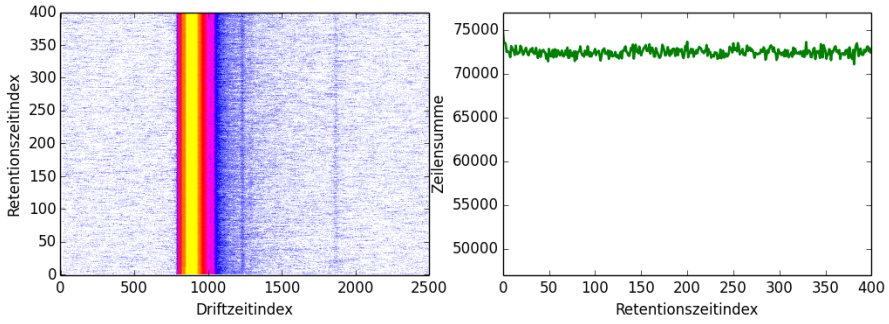
Zeilensummen zu sehen. Im Vergleich zu der Raumluftmessung fällt bei der normalen Messung auf, dass es in den ersten 10 - 20 Zeilen starke Schwankungen der Zeilensummen gibt und der gesamte Wert sinkt. Bei der übersteuerten Messung kann man zusätzlich an jeder Position, wo es zu einer Einschnürung von RIP kommt, ein Absinken der Zeilensummen identifizieren. Was auch bei dem Vergleich auffällt, sind die unterschiedlichen Niveaus, bei denen sich die Zeilensummen einpendeln.

6.3 MESSKORREKTUREN

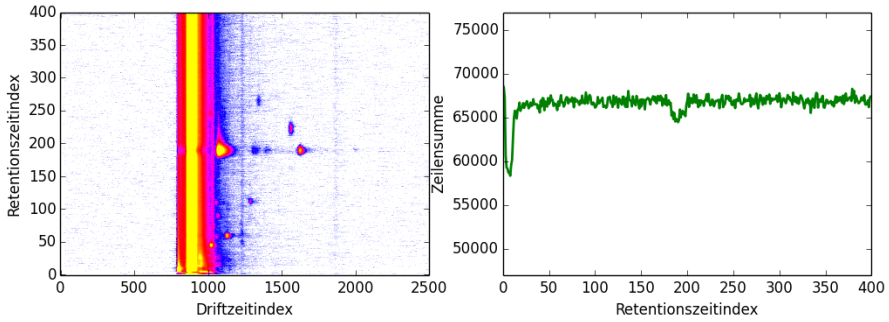
Bevor die Messdaten mit den Filterverfahren geglättet werden, soll ein grundlegendes Entrauschen der Messdaten durchgeführt werden. Im Folgenden wird die Basislinienkorrektur dargestellt, die später auch von der PG eingesetzt wird. Drei weitere Korrekturen sind im Anhang im Abschnitt A.4 zu finden. Alle diese Korrekturen sind detaillierter in der Arbeit von Kopczynski [28] vorgestellt. Neben der Arbeitsweise schauen wir uns auch an, inwiefern diese Methoden in einem Online-Algorithmus einsetzbar sind. Die Verfahren wurden auf der Messung aus Abb. 17 durchgeführt, welche im Rahmen der Messungen mit Bonbons entstand, die im Abschnitt 13.2.1 erläutert werden.

Basislinienkorrektur

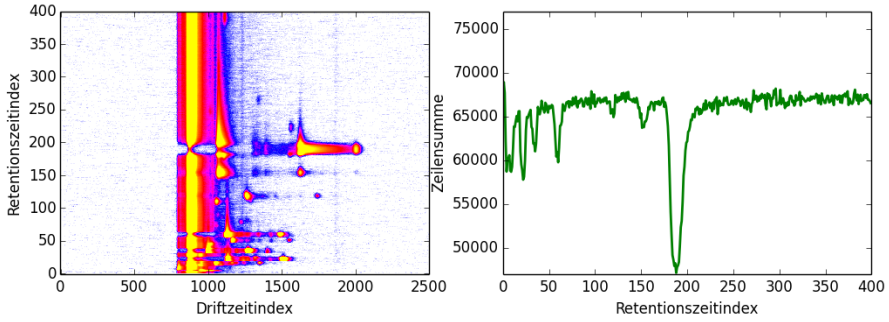
In den meisten Fällen versucht man vor der tatsächlichen Peakerkennung, die Daten mit Hilfe der Basislinienkorrektur zu entrauschen. Die Vorgehensweise ist wie folgt:



(a) Raumluftmessung



(b) Durchschnittliche Messung



(c) Übersteuerte Messung

Abb. 19: Vergleich zwischen den Zeilensummen dreier Messungen: Raumluftmessung in (a), eine durschnittliche Messung in (b) und eine übersteuerte Messung in (c), die fast unbrauchbar ist.

1. Berechne die Basislinie d_t als den Median über alle Signalwerte pro Chromatogramm

2. Ziehe von jedem Datenpunkt $S_{r,t}$ seinen entsprechenden Wert aus der Basislinie ab:

$$S_{r,t} := \max(S_{r,t} - d_t, 0).$$

Als Resultat dieses Verfahrens erhält man eine Messung, die im besten Fall keinen RIP und das meiste vom Rauschen nicht mehr enthält. Das wird dadurch gesichert, dass in einem Chromatogramm (eine Spalte der Datenmatrix) der Median durch die Peaks nicht stark verfälscht wird. Dementsprechend ist der Median auf dem RIP gleich der Signalstärke des RIPs und an den anderen Driftzeitpunkten auf dem Niveau des Rauschens.

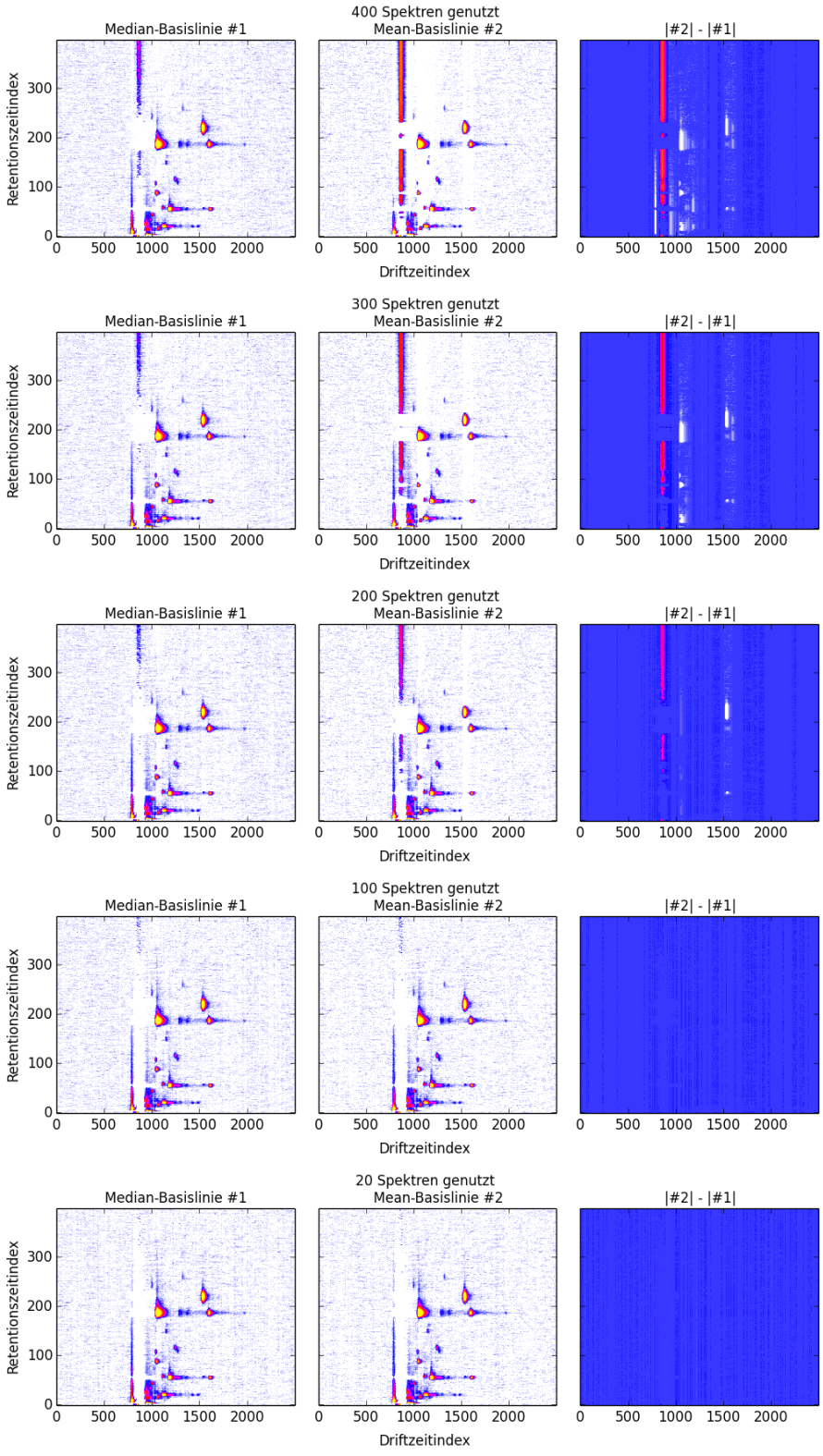


Abb. 20: Gegenüberstellung der Basislinienkorrekturen, die jeweils mit dem Median und dem Mittelwert auf der unterschiedlichen Anzahl an Spektren berechnet wurden. Die Bilder rechts sind jeweils Unterschiede zwischen den korrigierten Daten, wobei die Farben rot als zu viel verblieben und weiß als zu viel abgenommen zu verstehen sind. Blau entspricht ungefähr der Null, so dass an den Stellen nur sehr geringfügige Unterschiede betrachtet werden können.

VERSCHIEDENE BERECHNUNGEN Es sollen nun mehrere Möglichkeiten untersucht werden, auf den vorhandenen Messdaten die Basislinie zu berechnen. Während wir in einem *Offline*-Fall die gesamte Messung für die Berechnung der Basislinie benutzen können, soll das in einem *Online*-Szenario vermieden und schon beim Ankommen des einzelnen Spektrums die Basislinienkorrektur durchgeführt werden. Das Ziel ist somit, die ganzen Daten nicht im Hauptspeicher zu lagern, damit die Anwendung nicht sehr speicherhungrig wird und auch auf solchen Systemen wie dem Raspberry Pi (siehe Kapitel 3) laufen kann.

Eine Möglichkeit dies zu simulieren, ist es, eine bestimmte Anzahl an Spektren einer vorherigen Raumluftmessung oder auch einer normalen Messung zu nehmen. Wir haben diese Situation simuliert, indem wir die Spektren mit den höchsten Retenstionszeiten dieser Messung benutzt haben, um die Basislinie zu approximieren. Zusätzlich haben wir uns auch die Berechnung der Basislinie mit Hilfe des Mittelwerts angeschaut und mit der Basislinie aus dem Median verglichen, da der Mittelwert einfacher und speicherschonender zu berechnen ist. In Abb. 20 sieht man 5 Bilderreihen, wobei jede Reihe für die Anzahl an genutzten Spektren steht. Es werden von oben nach unten entsprechend 400, 300, 200, 100 und 20 Spektren genutzt. Die Bilder entsprechen pro Reihe folgendem Schema:

Korrektur mit der *Median*-Basislinie	Korrektur mit der *Mittelwert*-Basislinie	Differenz von korrigierten Daten
S^{median}	S^{mean}	$\lvert S^{mean}\rvert - \lvert S^{median}\rvert$

An der obersten Bilderreihe erkennt man klare Unterschiede zwischen S^{median} und S^{mean}. Dafür wurde die ganze Messung (400 Spektren) genutzt, wobei der Mittelwert stark auf die Einschnürungen des RIPs im unteren Bereich der Daten reagiert. Auch beim Benutzen von 300 und 200 Spektren vom Ende der Messung steht die *Mittelwert*-Basislinie eher im Nachteil. Als Differenz erkennt man klare rote Bereiche, an denen der Wert zu weit nach unten absinkt und daher zu wenig entfernt wird. Die weißen Bereiche sind die Stellen, an denen der Mittelwert bei den einzelnen Peaks auch schon teilweise die Peaks verkleinert und gar manche Peaks verschwinden lässt. Jedoch sobald in den benutzen Spektren keine Peaks mehr auftauchen, sehen die Bilder der Basislinienkorrekturen fast identisch aus.

Diese Untersuchung zeigt, dass es nicht immer sinnvoll ist, eine Basislinienkorrektur über die ganze Messung durchzuführen, auch wenn die Messung bereits vorliegt. Weiterhin kann man auch auf die einfachere Berechnung mittels des Mittelwerts zurückgreifen, solange die Daten dafür aus einer Raumluftmessung oder im Falle unserer Simulation aus den letzten Spektren der Messung stammen. Wir werden darum die *Mittelwert*-Basislinienkorrektur basierend auf den letzten 20 Spektren der Messung als Vorverarbeitung der Messdaten benutzen. Diese schafft nämlich mit nur 0.5% der ursprünglich genutzten Spektren eine Entfernung des RIP und des restlichen Rauschens in manchen Fällen sogar besser als die offline durchgeführte Basislinienkorrektur mit dem Median.

7

FILTERVERFAHREN

In der Rauschanalyse (Abschnitt 6.1) hat man gesehen, dass ein (vergleichsweise) hoher Messwert nicht zwangsläufig auf Peaks hindeutet. Der eigentliche Wert kann auch zufällig zu diesem Zeitpunkt durch ein starkes Rauschen verfälscht werden. Die hier vorgestellten Filter haben daher das Ziel, diese Unsicherheit des Rauschens möglichst zu eliminieren, sodass die Peaks klar erkennbar werden.

Das Rauschen ist durch „schnelle Zacken“ (hohe Frequenzen) charakterisiert. Um dieses zu entfernen, benötigt man einen Filter, der tiefe Frequenzen durchlässt und die hohen Frequenzen ab einem Grenzwert abschneidet (*idealer Tiefpassfilter*), d. h. im Frequenzraum soll der Übergang zwischen Durchlass- und Sperrbereich möglichst steil sein. Der ideale Tiefpassfilter entspricht also einer Rechteckfunktion im Frequenzbereich (siehe Abb. 21). Bekanntermaßen ist der ideale Tiefpassfilter nur in der Theorie gut beschreibbar, aber praktisch nicht berechenbar [30, S. 246 f.]. In der Praxis nähert man ihn daher möglichst gut an.

Da für uns schnelle, einfach zu berechnende Filter (die Ressourcen sind begrenzt) wichtig sind, konzentrieren wir uns im nächsten Abschnitt 7.1 auf lineare Filtermethoden. Im Abschnitt 7.2 wird der Savitzky-Golay Filter vorgestellt, der eine sehr wichtige Rolle für die PG spielt. Nichtlineare Filter werden nicht untersucht.

Die verwendeten Filter stammen größtenteils aus der Bildverarbeitung. Deshalb sprechen wir manchmal auch von *Pixeln* und meinen dabei Signalwerte $S_{r,t}$ zu einer Retentionszeit r und Driftzeit t. Im weiteren stützen wir uns auf die Literatur von Burger und James [10], Gonzalez und Woods [19], sowie Oppenheim und Schafer [30, Abschnitt 7.4].

Wir werden die Filter jeweils an dem Ausschnitt einer Messung demonstrieren, der in Abb. 22 zu sehen ist. Da es mehr um ein qualitatives Gefühl als um präzise Werte geht, wurde auf eine Beschriftung der Achsen sowie eine absolute Farbskala[1] verzichtet.

1 Die Filter sind zum Teil unterschiedlich „normiert“, weshalb eine Angabe von absoluten Skalen nicht sinnvoll ist. Wir sind uns bewusst, dass die Wahl der Farbskala die visuelle Wahrnehmung stark beeinflusst (siehe Probleme bei der manuellen An-

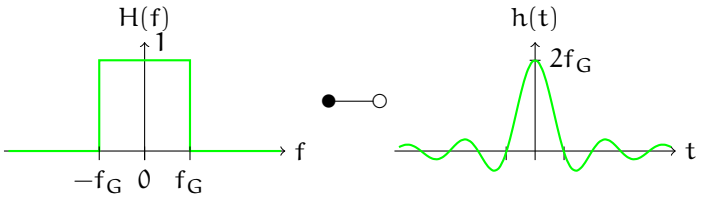


Abb. 21: Idealer Tiefpassfilter: Signalanteile mit einer tiefen Frequenz f (kleiner als die Grenzfrequenz f_G) sollen den Filter passieren können, alle anderen (hohe Frequenzen) heraus gefiltert werden. Zu sehen ist der Filter H im Frequenzraum (links) und dessen Fouriertransformierte (rechts).

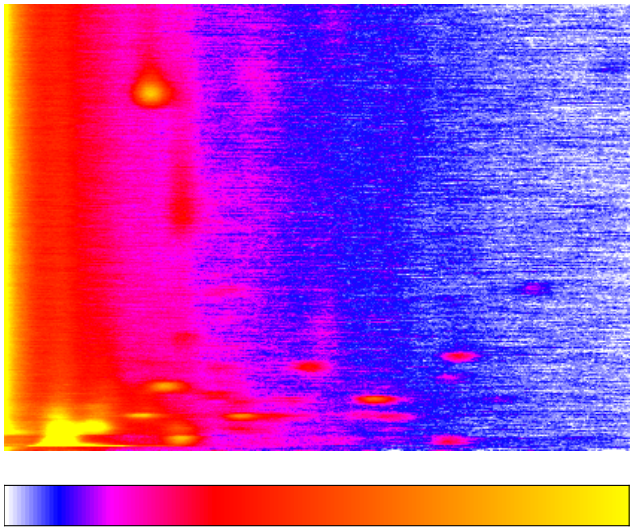


Abb. 22: Ausschnitt aus der Beispielmessung, an der die Filter ausprobiert werden. Darunter die verwendete Farbskala: von weiß (niedrig) über blau, lila, rot, orange zu gelb (hoch).

7.1 LINEARE FILTER

Das überlagerte Rauschen ist das Ergebnis eines stochastischen, zeitlich unabhängigen Prozesses (siehe Abschnitt 6.1). Eine wirkungsvolle Strategie, Zufall zu eliminieren, ist, das Experiment mehrfach zu wiederholen und den Mittelwert der Ergebnisse zu berechnen, denn nach dem Gesetz der großen Zahlen konvergiert dies gegen das tatsächliche Signal. Für uns würde das bedeuten, man müsste die Messung unter exakt denselben Bedingungen mehrfach reproduzieren, damit sich die Messwerte nur durch das Rauschen unterscheiden. Dies ist technisch nicht möglich. Allerdings sind die *„Nachbarmesswerte“* unter sehr ähnlichen Bedingungen entstanden und die Messwerte unterscheiden sich nur gering.

Ein möglicher Ansatz ist es, den Mittelwert über „Nachbarpixel“ zu bilden und den alten Wert durch das Ergebnis zu ersetzen; der Wert wäre ein wenig vom Rauschen befreit. In der Form spricht man vom Mittelwert- oder *Box-Filter*. Das Resultat (siehe Abb. 23) ist nicht besonders zufrieden stellend: Kleine Peaks werden „verschluckt“ und große Ausreißer des Rauschens heben die ganze Region an. Ein Grund dafür ist, dass alle Nachbarn gleich gewichtet werden, d. h. ein weit entfernter Messwert hat den gleichen Einfluss wie ein sehr naher.

Ein weiterer Ansatz ist darum, die Nachbarpixel unterschiedlich stark zu gewichten. In der allgemeinsten Form ist jeder Pixel dann eine *Linearkombination* seiner Nachbarn:

$$(S * K)_{r,t} = \sum_{i=-n_1}^{n_1} \sum_{j=-n_2}^{n_2} S_{r-i,t-j} K_{i,j} \ . \tag{8}$$

Dabei nennt man $K \in \mathbb{R}^{(2n_1+1)\times(2n_2+1)}$ den *Filterkern* (kurz Kern oder *Kernel*). Man beachte dabei, dass die Indizes der Matrix K nicht

notation von Peaks in Abschnitt 13.1). Der sichtbare Wertebereich wurde so gewählt, dass unserer Einschätzung nach der Effekt des Filters möglichst gut erkennbar ist.

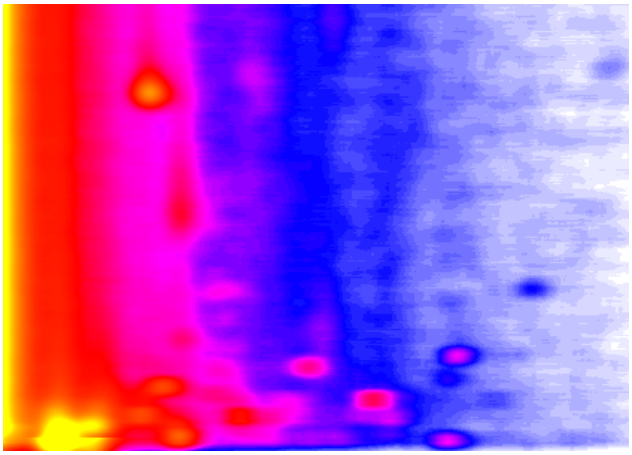


Abb. 23: Box-Filter mit Fenstergröße 11×11.

wie üblich bei 1 beginnen, sondern bei $-n_1$ bzw. $-n_2$, um die Notation zu vereinfachen. Mathematisch berechnet man die sogenannte diskrete Faltung $*$ des Signals S mit dem Kern K. Filter von diesem Typ heißen *lineare Filter*.

Die gefilterte Messung $(S * K)_{r,t}$ kann nach obiger Definition nicht für die *„Randbereiche“* der Größe n_1 in Retentionszeitindizes bzw. n_2 in Driftzeitindizes berechnet werden. In der Bildverarbeitung gibt es hier verschiedene Techniken, den Rand aufzufüllen (z. B. kopieren, spiegeln oder mit Nullen auffüllen). Da die Messungen in den ersten Spektren jedoch oft Störungen aufwiesen und in den anderen Randbereichen keine Peaks vorhanden waren, wurde der Rand nicht weiter verarbeitet.

Im Weiteren werden einige geläufige Filterkerne und ihre Eigenschaften vorgestellt. Dabei steht im Mittelpunkt die Frage, ob sie sich für unser Projektvorhaben eignen.

7.1.1 *Kaiser-Window*

Beim Filterentwurf sind aus Perspektive der Signalverarbeitung (siehe Oppenheim und Schafer [30, Abschnitt 7.4], [30, Abschnitt 7.4] und [20]) folgende zwei Eigenschaften der Filterfunktion zu beachten: Betrachtet man in Abb. 24 die Frequenzspektren (links) der Filterfunktionen (rechts), sollen sowohl die Amplituden der Nebenmaxima (die Peaks neben dem höchsten Peak) als auch die Breite des Hauptmaximums (der höchste Peak) möglichst gering sein. Denn durch ausgeprägte Amplituden der Nebenmaxima im Spektrum verbleiben unerwünschte Frequenzanteile nach dem Filtern im Signal und werden nicht wie ursprünglich gewünscht unterdrückt. Ist hingegen die Breite des Hauptmaximums zu groß gewählt, so selektiert der Filter die Frequenzen nicht genug, da die Übergänge zwischen Durchlass- und Sperrbereich des Filters zu weich sind.

In Abb. 24 wird deutlich, dass die Flankensteilheit der Filterfunktion sowohl Amplitude der Nebenmaxima im Frequenzbereich als auch die Breite des Hauptmaximums beeinflusst. Wählt man einen Filter mit einer flachen Flanke, sind die Amplituden der Nebenmaxima bedeutend

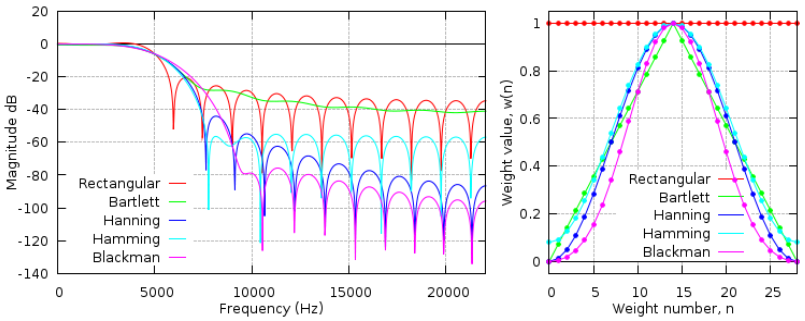


Abb. 24: Verschiedene Kaiserfilter in Frequenz- und Zeitbereich (mit freundlicher Genehmigung von Andrew Greensted [20]).

geringer. Andererseits kann man mit der gewählten Größe des Fensters n im Zeitbereich die Breite des Hauptmaximums konfigurieren: Je breiter das Fernster, desto schmaler das Hauptmaximum des Spektrums. Allerdings bringt eine zunehmende Verschmälerung des Hauptmaximums unerwünschte Nebeneffekte mit sich: Die Kompaktifizierung des Hauptmaximums führt zu einer deutlich längeren Filterfunktion, da sie dann langsamer ausklingt und somit zu einer vergrößerten Latenz des Filters führt.

Aus den erwähnten Argumenten geht hervor, dass der optimale Filter ein Kompromiss aus Amplitude der Nebenmaxima und Breite des Hauptmaximums ist.

Der Kaiserfilter ermöglicht den Kompromiss, in dem sich die Filterform über zwei Parameter konfigurieren lässt. Der (eindimensionale) Filterkern, das sogenannte *Kaiser-Fenster*, ist definiert als:

$$\left(w^{n,\beta}\right)_j = \begin{cases} \frac{I_0\left[\beta\left(1-[(j-\alpha)/\alpha]^2\right)^{1/2}\right]}{I_0(\beta)}, & 0 \leqslant j \leqslant n \\ 0, & \text{sonst} \end{cases} \tag{9}$$

wobei $\alpha = n/2$ und $I_0(\cdot)$ die modifizierte Bessel-Funktion[2] erster Art nullter Ordnung sind. Über den Formparameter β lässt sich die Flankensteilheit der Filterfunktion einstellen (siehe Abb. 25) und unterschiedliche Filterkerne darstellen wie Box, Bartlett, Hanning, Hamming und Blackman. Einen zweidimensionalen Filterkern berechnen wir auf die übliche Art als dyadisches Produkt eines horizontalen (w^{n_1,β_1}) und eines vertikalen (w^{n_2,β_2}) Filterkerns (bei uns $\beta_1 = \beta_2$)

$$W^{n_1\times n_2,\beta_1,\beta_2} := w^{n_1,\beta_1}\left(w^{n_2,\beta_2}\right)^T .$$

7.1.2 *Gaußfilter*

In der Bildverarbeitung wird oft der Gaußfilter als Weichzeichner (Tiefpassfilter) verwendet. Die Koeffizienten des Filterkerns G^{σ_1,σ_2} berech-

2 Bessel Funktionen sind Lösungen der Besselschen Differentialgleichung, die in vielen numerischen Bibliotheken verfügbar sind. Wir haben die Filterkerne mit der Funktion `kaiser` aus dem `numpy` Paket berechnet.

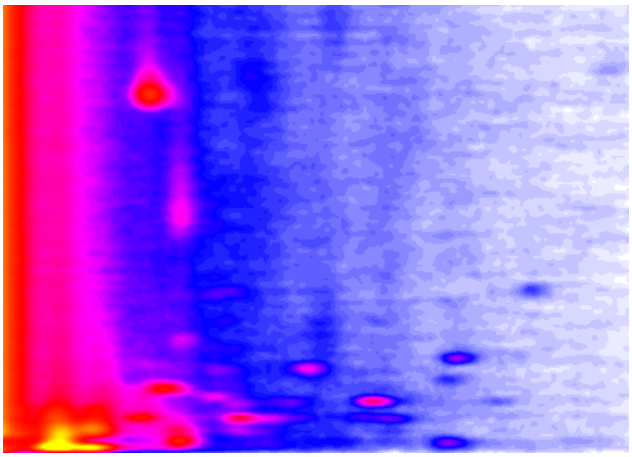


Abb. 25: Kaiser Fenster der Größe 11×11 mit $\beta = 8.6$.

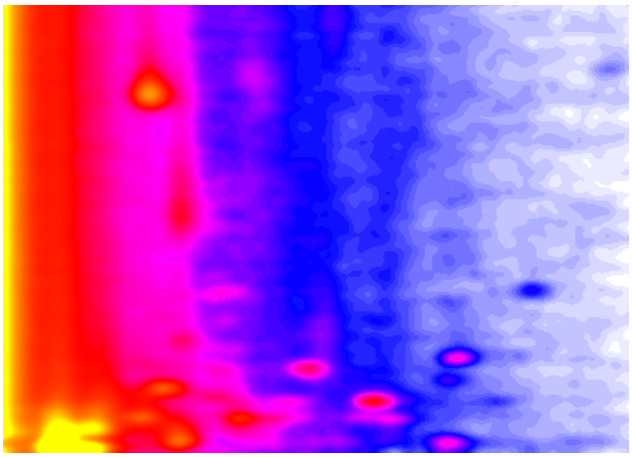


Abb. 26: Gaußfilter der Größe 11×11 mit Standardabweichung $\sigma_1 = \sigma_2 = 3$.

nen sich als die Dichte der Normalverteilung (daher der Name) mit Standardabweichungen σ_1 und σ_2 und Erwartungswert 0:

$$(G^{\sigma_1,\sigma_2})_{i,j} = \frac{1}{2\pi\sigma_1\sigma_2} \exp\left(-\frac{i^2}{2\sigma_1^2} - \frac{j^2}{2\sigma_2^2}\right) \tag{10}$$

Aus theoretischer Sicht ist er interessant, weil er im Frequenzbereich (bis auf Skalierung) auch einer Gaußglocke entspricht und deshalb zwar niemals die Null erreicht, dafür aber keine Nebenmaxima aufweist. Abb. 26 zeigt die Anwendung.

Im Vergleich zum Kaiser-Window (ab einem bestimmten β) ist die Selektion der Frequenzen schlechter, was für uns bedeutet, dass die Peaks stärker abgeflacht werden.

Sowohl der Gauß-Filter als auch verschiedene Varianten des Kaiser-Window haben sich als brauchbare Vorverarbeitungen zum Glätten des Rauschens erwiesen. Letztendlich kam jedoch der in Abschnitt 7.2 vorgestellte Savitzy-Golay-Filter zum Einsatz, da er neben einer Glättung noch zusätzliche Informationen über Steigung liefern kann.

7.1.3 *Effiziente Berechnung und Separierbarkeit*

Die Verarbeitung des Datenstroms der Messwerte sollte möglichst in Echtzeit und ressourcenschonend erfolgen. Damit scheidet die schnelle Berechnung über die Fouriertransformation (wie sonst üblich) aus, da dies nur auf den gesamten Daten möglich ist.

Die einfachste Möglichkeit, einen Filter zu berechnen, ist über die Definition der Faltung (siehe Gleichung (8)). Für jeden Messpunkt (bis auf den Rand) sind dazu $(2n_1+1)\cdot(2n_2+1)$ Multiplikationen notwendig (für jeden Koeffizient des Filterkerns eine). Dazu benötigen wir statt der ganzen Messung nur die letzten $(2n_1+1)$ Spektren, die zur Verarbeitung in einem Ringpuffer gespeichert werden.

In bestimmten Fällen lässt sich die Berechnung einer zweidimensionalen Faltung zu zwei hintereinander ausgeführten eindimensionalen Faltungen vereinfachen. Dies wird im Folgenden beschrieben.

Man nennt einen Kern $K \in \mathbb{R}^{(2n_1+1)\times(2n_2+1)}$ *separierbar*, wenn es zwei Vektoren $u \in \mathbb{R}^{2n_1+1}$ und $v \in \mathbb{R}^{2n_2+1}$ gibt, für die gilt

$$K = uv^T \quad \text{bzw.} \quad K_{i,j} = u_i v_j \text{ für alle Indizes } i \text{ und } j. \tag{11}$$

Damit lässt sich die Faltung (siehe Gleichung (8)) schreiben als

$$(S * K)_{r,t} = \sum_{i=-n_1}^{n_1} \sum_{j=-n_2}^{n_2} S_{r-i,t-j} u_i v_j = \sum_{i=-n_1}^{n_1} u_i \underbrace{\left(\sum_{j=-n_2}^{n_2} S_{r-i,t-j} v_j \right)}_{b_{r-i}}. \tag{12}$$

Um die zweidimensionale Faltung an der Stelle r,t zu berechnen, kann man zunächst für alle $i = -n_1 \ldots n_1$ die $(r-i)$-te Zeile von S eindimensional falten mit v (in obiger Gleichung b_{r-i}) und dann den daraus entstehenden Vektor b eindimensional falten mit u. Durch diese Zerlegung des Filters benötigt die Auswertung der Faltung für einen Messpunkt $(2n_1+1)+(2n_2+1)$ Multiplikationen im Gegensatz zu $(2n_1+1)\cdot(2n_2+1)$.

Algorithmisch entspricht dies zwei hintereinander geschalteten eindimensionalen Faltungen: Zunächst vertikal (mit v) und dann horizontal (mit u) wie in Abb. 27 dargestellt. Ebenso könnte man auch erst horizontal und dann vertikal falten (siehe Abb. 28).

In vielen Fällen (beispielsweise beim Gaußfilter) lässt sich analytisch direkt feststellen, ob ein Filter separierbar ist. Andernfalls kann man dies überprüfen, indem man die *Singulärwertzerlegung* des Kerns berechnet

$$K = U\Sigma V^T = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_n u_n v_n^T \tag{13}$$

mit orthogonalen Matrizen U und V und einer Diagonalmatrix $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$. Der Filterkern K ist genau dann separierbar, wenn alle bis auf einen Singulärwert σ_i (nahezu) Null sind. Auf diese Weise können im Übrigen auch nicht-separable Filter durch eine Summe von separablen Filtern approximiert werden.

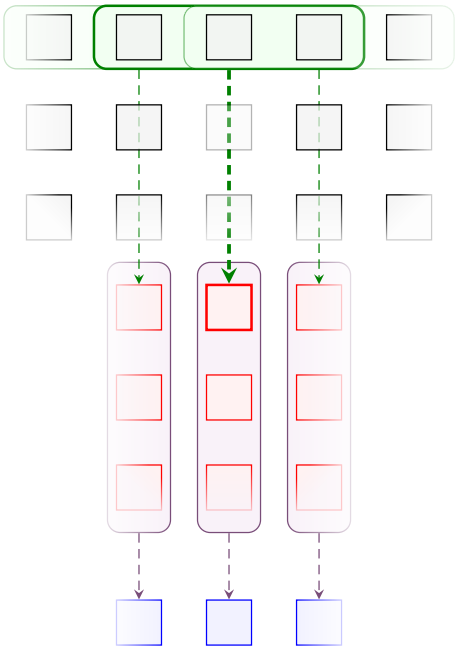


Abb. 27: Berechnung der Faltung über separierbaren Filter: Zunächst wird das neue Spektrum *horizontal* gefaltet (grün) und die Werte in einer neuen Zeile eines Puffers gespeichert (rot). Anschließend wird jede Spalte des Puffers *vertikal* gefaltet und der Filterwert (blau) errechnet.

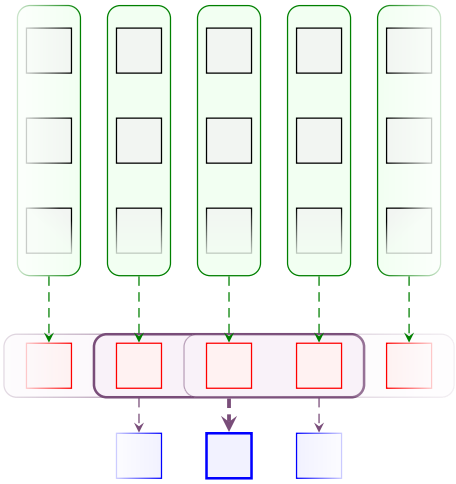


Abb. 28: Andere Variante der Faltungsberechnung: Anders als in Abb. 27 wird erst *vertikal* gefaltet und dann die Zwischenergebnisse (rot) *horizontal* gefaltet.

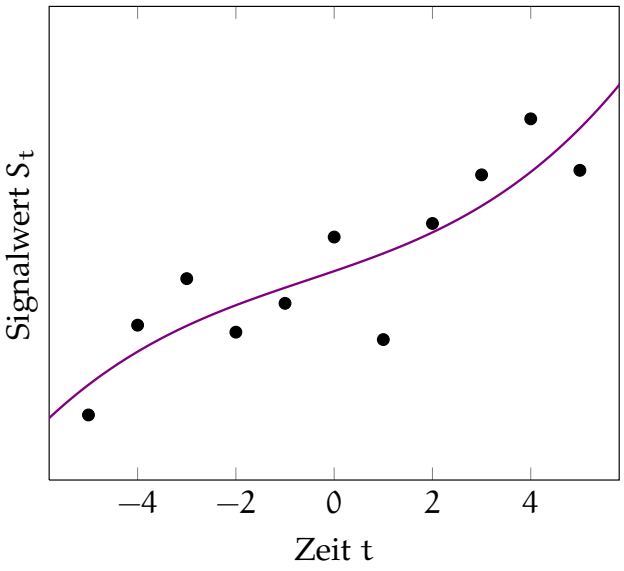


Abb. 29: Polynomielle Regression: Bestimme ein Polynom (hier vom Grad 3), sodass der Abstand zwischen vorgegebenen Punkten minimal wird (kleinste Quadrate).

7.2 SAVITZKY-GOLAY-FILTER

Schon 1964 haben sich A. Savitzky und M. E. Golay [33] damit beschäftigt, wie man mit einem Computer (anstatt mit anderen elektronischen Geräten) verrauschte (eindimensionale) Messdaten effizient rekonstruieren kann. Die Idee des Verfahrens ist es, die Messdaten lokal durch ein Polynom zu approximieren (siehe Abb. 29) und aus diesem korrigierte Werte zu berechnen (*polynomielle Regression*). Erstaunlicherweise führt dieser Ansatz auf einen linearen Filter, wie wir zeigen werden. Wir beschäftigen uns mit diesem Filter, da er verspricht, die Daten zu glätten *ohne* hohe Messwerte zu sehr „abzuschneiden".

> *„The specific nature of the assumed redundancy is that the underlying function should be locally well-fitted by a polynomial. When this is true, as it is for smooth line profiles not too much narrower than the filter width, the performance of Savitzky-Golay filters can be spectacular." [32, S. 770 f.]*

Dies ist für uns besonders wichtig, da dadurch Peaks besser erhalten bleiben und zuverlässiger erkannt werden können.

Zunächst wird das Verfahren für eindimensionale Signale beschrieben, insbesondere die Berechnung des Filterkerns. Danach wird gezeigt, wie man mit Hilfe des Savitzky-Golay-Filters Ableitungen annähern kann. Zuletzt werden Erweiterungen auf zweidimensionale Signale vorgestellt.

7.2.1 *Eindimensionale Variante*

In diesem Teil betrachten wir ein (eindimensionales) digitales Signal (in unserem Fall z. B. ein Spektrum) $S_0, S_1, \ldots$. Wie schon bei anderen Filtern wird das Signal zu einem Zeitpunkt t aus den k benachbarten Signalwerten $S_{t-k}, S_{t-k+1}, \ldots, S_{t+k}$ in beiden Richtungen berechnet.

Es sind auch Varianten denkbar, wobei nach links und rechts unterschiedlich viele Nachbarn berücksichtigt werden. Um die Formeln zu vereinfachen, indizieren wir so, dass der aktuell zu korrigierende Punkt bei $t = 0$ liegt. Wir nehmen an, das ursprüngliche, unverrauschte Signal verhält sich lokal, innerhalb des Fensters, relativ „gleichmäßig“ wie ein Polynom niedrigen Grads. Das Ziel ist es, ein möglichst „gutes“ Polynom p aus den Signalwerten $S_{-k}, \ldots, S_k$ zu approximieren. Dann kann man $p(t = 0)$ als korrigierten Messwert berechnen und hierdurch den Einfluss des Rauschens verringern.

Zur Bestimmung eines Polynoms p vom Grad d müssen Koeffizienten $a = (a_0, \ldots, a_d)$ gewählt werden, sodass gilt:

$$p(x) = a_0 + a_1 x + \cdots + a_d x^d \text{ mit } p(i) = S_i \text{ für } i = -k, \ldots, +k\,. \tag{14}$$

Hierzu werden mindestens $d + 1$ Signalwerte in der Nachbarschaft benötigt. Wird ein Fenster mit mehr als $d + 1$ Signalwerten verwendet, ist Gleichung (14) im Allgemeinen nicht lösbar. Aus diesem Grund sollten die Koeffizienten a zumindest so gewählt werden, dass der Abstand $|p(i) - S_i|$ für alle i möglichst gering wird. Da es sehr aufwendig ist, diesen Abstand exakt zu minimieren, minimieren wir stattdessen die Summe der quadrierten Abstände (*Methode der kleinsten Quadrate*):

$$\min_a \sum_{i=-k}^{k} (p(i) - S_i)^2 \ . \tag{15}$$

Wir führen eine vereinfachte Notation über Matrizen ein. Dazu setzen wir $f_i := (i^0, i^1, \ldots, i^d)$, dann ist

$$p(i) = a_0 + a_1 i + \cdots + a_d i^d = f_i a \ .$$

Die Zeilen f_i für $i = -k, \ldots, +k$ fassen wir zu einer Matrix F zusammen und setzen $s := (S_{-k}, \ldots, S_{+k})^T$. Damit lässt sich die Gleichung (14) kurz schreiben als

$$\min_a \|Fa - s\|_2^2 \tag{16}$$

Zur Lösung dieses Minimierungsproblems verwendet man die sogenannte *Normalengleichung* (siehe z. B. [13, S. 835 ff.]). Dadurch errechnen sich die Koeffizienten des angenäherten Polynoms als

$$a = \left[(F^T F)^{-1} F^T\right] s = F^+ s \ , \tag{17}$$

wobei F^+ die sogenannte *Pseudoinverse*[3] von F ist. Der korrigierte Signalwert ist demnach

$$p(0) = a_0 = e_1^T a = \underbrace{e_1^T F^+}_{c^T} s = \sum_{i=-k}^{k} c_i S_i \ , \tag{18}$$

3 Die Moore-Penrose Pseudoinverse F^+ lässt sich numerisch stabiler berechnen über die QR-Zerlegung: Faktorisiert man $F = QR$ in eine orthogonale Matrix Q und eine obere Dreiecksmatrix R, dann gilt

$$F^+ = \left(F^T F\right)^{-1} F^T = \left(R^T Q^T Q R\right)^{-1} R^T Q^T = R^{-1} \left(R^T\right)^{-1} R^T Q^T = R^{-1} Q^T \ .$$

Eine weitere mögliche Berechnung benutzt die Singulärwertzerlegung. Laut Dokumentation erfolgt die Berechnung der Funktion `pinv` aus dem Paket `numpy.linalg`, die wir verwenden, auf diese Weise.

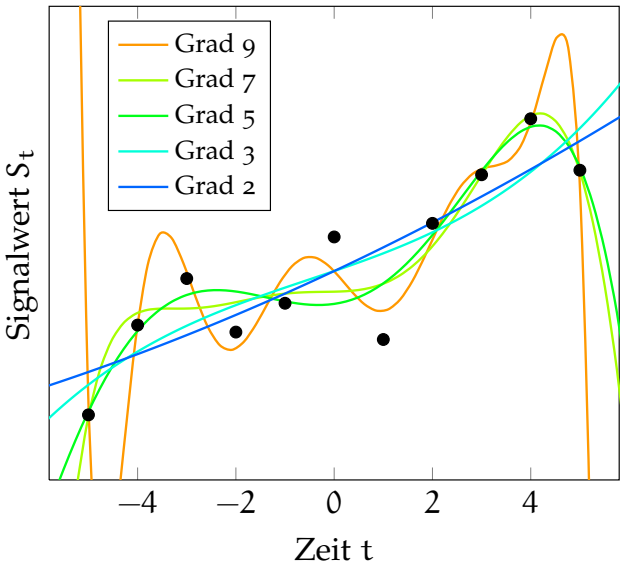


Abb. 30: Polynomielle Regression mit verschiedenen Polynomgraden.

wobei $e_1 = (1, 0, \ldots, 0)^T$ der erste Einheitsvektor und $c^T := e_1^T F^+$ der Vektor der Koeffizienten des Savitzky-Golay-Filters ist.

Wie man sieht, führt die polynomielle Regression zu einem linearen Filter, der einfach zu berechnen ist (wenn man die Koeffizienten c und somit den Filterkern vorberechnet).

EINFLUSS DES POLYNOMGRADS. Der Grad d des approximierenden Polynoms beeinflusst das Verhalten des Filters: Für ein lineares Polynom ($d = 1$) entspricht der Savitzky-Golay Filter einem Mittelwert-Filter (siehe Einleitung in Abschnitt 7.1). Bei zu kleinem Grad wird das Rauschen zwar sehr gut entfernt; allerdings gehen auch wichtige Informationen insbesondere aus den höheren Frequenzen (und damit auch Peaks) des eigentlichen, unverrauschten Signals verloren. In Abb. 30 sieht man, dass die blau-türkis farbigen Polynome den Verlauf der Messwerte relativ ungenau in Bezug auf Extrema wiedergeben, jedoch gut glätten.

Andererseits führt ein zu hoher Polynomgrad zu einer guten Interpolation der Signalwerte. Dafür approximiert er die Werte zwischen den vorgegebenen Zeitpunkten schlecht. Wie in Abb. 30 gut erkennbar ist, hat eine kleine Änderung der Signalwerte (wie z. B. Hintergrundrauschen) große Auswirkungen auf das Polynom und hierdurch auf das gefilterte Signal. Im schlechtesten Fall könnte das Rauschen bei zu hohen Graden verstärkt werden.

D. Thornley fasst diese Problematik so zusammen:

> *„The order of the polynomials used in these filters is a compromise between retention of detail favouring a higher order, and rejection of noise favouring a lower order.“* [36, Introduction]

Die konkrete Wahl des besten Polynomgrades hängt also stark von den Signalwerten und der Intention des Filterns ab (Glätten gegenüber Erhaltung von Informationen – insbesondere Peaks).

7.2.2 *Gewichtung der Abweichung*

Zusammenfassend läuft das Verfahren bisher wie folgt ab: Um den zu korrigierenden Signalwert wird *lokal* ein Polynom gelegt. Zur Korrektur wird das Polynom an der entsprechenden Stelle ausgewertet.

Dabei ist es gut möglich, dass sich die Regressionspolynome von nebeneinander liegenden Signalwerten stark unterscheiden und infolgedessen die gefilterten Werte stark abweichen können. Ein glatterer Übergang könnte erreicht werden, indem der Einfluss von weiter entfernten Punkten reduziert wird. Auch aus Sicht der Signaltechnik sollten die Werte eines Filterkerns zum Rand hin gegen Null streben, um ungewollte Filtereffekte (Überlagerung von hohen Frequenzanteilen) zu vermeiden. Beides ist realisierbar über den *gewichteten Savitzky-Golay-Filter*, wie er von Thornley [36, Abschnitt 3] oder Chinrungrueng Toonkum [12, 3. Abschnitt] beschrieben wird.

Dazu wird das Polynom p so gewählt, dass es wie im Problem (15) die Summe der quadratischen Abweichungen minimiert. Zusätzlich wird die quadrierte Abweichung des Polynoms an der Stelle k noch mit einem Faktor w_k gewichtet:

$$\sum_{i=-k}^{k} w_i \, (p(i) - s_i)^2 = \sum_{i=-k}^{k} w_i \, (Fa - s)_i^2 \ . \tag{19}$$

Bei einem hohen Gewicht w_k wird so der Wert des k-ten Punkts stärker berücksichtigt, d. h. das Polynom p hat einen geringeren Abstand, als bei vergleichsweise niedrigen Gewichten.

Wir zeigen nun die Berechnung des Filterkerns bei gegebener Gewichtung. Um die Gleichung (19) in eine vereinfachte Darstellung als Matrizen zu überführen, setzen wir

$$W := \operatorname{diag}(w_{-k}, \ldots, w_k) \quad \text{und} \quad D := \operatorname{diag}(\sqrt{w_{-k}}, \ldots, \sqrt{w_k}) \ ,$$

d. h. $W = D^T D$. Damit wird die zu minimierende Zielfunktion zu

$$(Fa - s)^T W (Fa - s) = (Fa - s)^T D^T D (Fa - s) = \|D(Fa - s)\|_2^2 \ .$$

Setzen wir $G := DF$ und $b := Ds$ haben wir jetzt dieselbe Form wie in Gleichung (16)

$$\min_a \|Ga - b\|_2^2 \quad \text{mit Lösung} \quad a = (G^T G)^{-1} G^T b = G^+ b \ .$$

Rücksubstitution von G und b führt auf die Koeffizienten a des gewichteten Regressionspolynoms p:

$$a = \left[(DF)^T DF\right]^{-1} (DF)^T Ds = \underbrace{(F^T W F)^{-1} F^T W}_{Q} s \ .$$

Die Filterkoeffizienten c_w des gewichteten Savitzky-Golay-Filters sind dann entsprechend $c_w = e_1^T Q$.

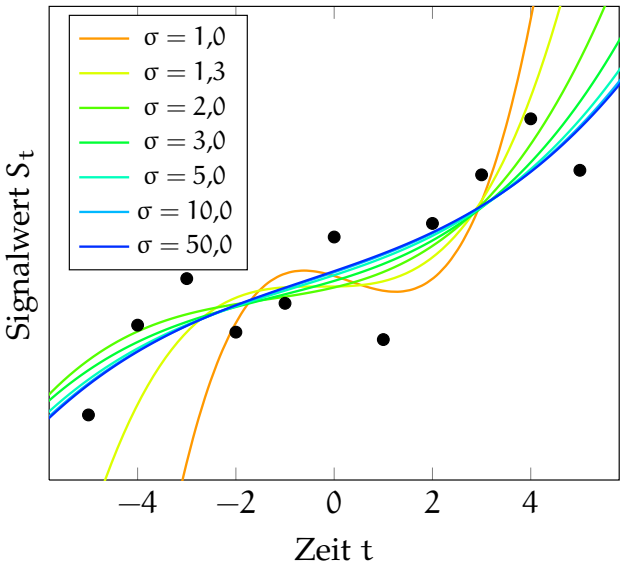


Abb. 31: Gewichtete Polynomielle Regression (Grad 3): Bei Gewichtung mittels Gaußglocke mit großer Standardabweichung σ unterscheidet sich das Polynom kaum vom ungewichteten Fall (vgl. Abb. 29). Je kleiner σ, desto geringer der Einfluss der äußeren Punkte auf das Polynom.

WAHL DER GEWICHTUNGSFAKTOREN. Tendenziell sollten die Gewichtung $w_{-k}, \cdots, w_k$ in der Mitte um w_0 am größten und zu den Rändern hin monoton abfallend gewählt werden. Hier sind viele Varianten denkbar (für eine Wahl bei Ultraschallbildern siehe [12, Gleichung (13)]). Wir haben nur ausprobiert, wie sich der Filter verändert, wenn man Gewichte wie bei einer Gaußglocke (siehe Gleichung (10)) mit verschiedenen Standardabweichungen σ wählt; die Auswirkungen auf das Regressionspolynom sind in Abb. 31 zu sehen.

7.2.3 *Approximation der Ableitungen*

Wir haben in den letzten beiden Abschnitten gezeigt, wie man (gewichtete) Koeffizienten $a = (a_0, \ldots, a_d)$ eines approximierenden Polynoms p aus den Signalwerten $S_{-k}, \ldots, S_k$ berechnen kann. Dabei wurde nur der erste Koeffizient $a_0 = p(0)$ verwendet. Gehen wir davon aus, dass das Polynom die Signalwerte gut approximiert, können wir weitere Informationen aus p gewinnen. Mit der Bezeichnung aus Gleichung (18) ist beispielsweise die erste Ableitung näherungsweise

$$\begin{aligned} p'(0) &= a_1 + 2a_2x + 3a_3x^2 + \cdots + da_dx^{d-1}\big|_{x=0} \\ &= a_1 = \underbrace{e_2^T(F^TF)^{-1}F^T}_{c^{(1)}} s = \sum_{i=-k}^{k} c_i^{(1)} S_i \end{aligned}$$

Die j-te Ableitung ist entsprechend

$$p^{(j)}(0) = j!a_j = \underbrace{j!e_{j+1}^T(F^TF)^{-1}F^T}_{c^{(j)}} s = \sum_{i=-k}^{k} c_i^{(j)} S_i \,,$$

also auch eine Faltung des Signals mit einem passenden Filterkern $c^{(j)}$.

7.2.4 *Erweiterung auf zwei Dimensionen*

Es reicht nicht aus, die Messdaten nur in einer Dimension (innerhalb eines Spektrums $S_{r,\bullet}$) zu glätten und zu analysieren: Peaks können nur sinnvoll erkannt werden, wenn auch die Veränderungen der Messwerte über die Retentionszeit r betrachtet werden. Wir schildern nun, wie man einen zweidimensionalen Savitzky-Golay Filterkern bestimmt. Im Grunde genommen reduziert man auf den eindimensionalen Fall.

Wie im eindimensionalen Fall approximieren wir die Messwerte um den aktuell zu korrigierenden Messwert lokal durch ein Polynom p, das aus den Nachbarpixeln $S_{i,j}$ mit $i \in \{-k_1, \ldots, k_1\}$ und $j \in \{-k_2, \ldots, k_2\}$ berechnet wird. Dies entspricht der rechteckigen Umgebung um den betrachteten Messwert der Größe $(2k_1 + 1) \times (2k_2 + 1)$ (auch andere Formen sind denkbar, wurden von uns jedoch nicht betrachtet).

Zur Approximation wählen wir ein Polynom p der Form

$$p(x, y) = \sum_{i=0}^{d_1} \sum_{j=0}^{d_2} a_{ij} x^i y^j \ .$$

Hierbei ist d_1 der Grad des Polynoms in x-Richtung und d_2 in y-Richtung. Durch unterschiedliche Grade kann man entlang der Spektren anders filtern als entlang der Chromatogramme. Bei der polynomiellen Regression wird die Matrix A der Koeffizienten a_{ij} mit $i \in \{0, \ldots, d_1\}$ und $j \in \{0, \ldots, d_2\}$ bestimmt. Man findet auch Filtervarianten, wo nur Polynome ohne *Mischterme*[4] verwendet werden, d. h. alle Koeffizienten außer a_{i0} und a_{0j} sind Null (diese Varianten werden nicht weiter betrachtet).

Um die Form wie im eindimensionalen Fall (siehe Gleichung (15)) zu erreichen, muss die Matrix der Koeffizienten a_{ij} als Vektor notiert werden, indem eine Reihenfolge festgelegt wird (Spaltenreihenfolge der Matrix F), beispielsweise

$$\begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0d_2} \\ a_{10} & a_{11} & \cdots & a_{1d_2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d_1 0} & a_{d_1 1} & \cdots & a_{d_1 d_2} \end{pmatrix} \leftrightsquigarrow \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0d_2} & a_{10} & \cdots & a_{d_1 d_2} \end{pmatrix}.$$

Auf ähnliche Art wird die Matrix S_{ij} der Nachbarpixel „linearisiert“ (Zeilenreihenfolge von Matrix F). Anschließend wird dieselbe Berechnung wie im eindimensionalen Fall durchgeführt. Die berechneten Filterkoeffizienten werden unter Verwendung der vorher festgelegten Reihenfolge zu einer Matrix rücktransformiert.

Auf diese Weise erhält man einen zweidimensionalen Filterkern (siehe Abb. 32), der zur Glättung der Messdaten eingesetzt werden kann (siehe Abb. 33). Ferner kann, wie im eindimensionalen Fall, die Ableitung angenähert werden. Dies verwenden wir zur Peakerkennung.

4 Am Anfang der Implementierung haben wir unsere Ergebnisse mit dem Filterkern verglichen, den John Krumm auf einer Website angegeben hat (siehe http://research.microsoft.com/en-us/um/people/jckrumm/SavGol/SavGol.htm, Stand Juli 2013). Später stellte sich heraus, dass er in seiner Berechnung keine Mischterme verwendet.

$$0{,}001 \cdot \begin{bmatrix} 7 & -1 & -8 & -13 & -16 & -17 & -16 & -13 & -8 & -1 & 7 \\ -1 & 0 & 2 & 3 & 4 & 4 & 4 & 3 & 2 & 0 & -1 \\ -8 & 2 & 10 & 16 & 20 & 21 & 20 & 16 & 10 & 2 & -8 \\ -13 & 3 & 16 & 25 & 31 & 33 & 31 & 25 & 16 & 3 & -13 \\ -16 & 4 & 20 & 31 & 38 & 40 & 38 & 31 & 20 & 4 & -16 \\ -17 & 4 & 21 & 33 & 40 & 43 & 40 & 33 & 21 & 4 & -17 \\ -16 & 4 & 20 & 31 & 38 & 40 & 38 & 31 & 20 & 4 & -16 \\ -13 & 3 & 16 & 25 & 31 & 33 & 31 & 25 & 16 & 3 & -13 \\ -8 & 2 & 10 & 16 & 20 & 21 & 20 & 16 & 10 & 2 & -8 \\ -1 & 0 & 2 & 3 & 4 & 4 & 4 & 3 & 2 & 0 & -1 \\ 7 & -1 & -8 & -13 & -16 & -17 & -16 & -13 & -8 & -1 & 7 \end{bmatrix}$$

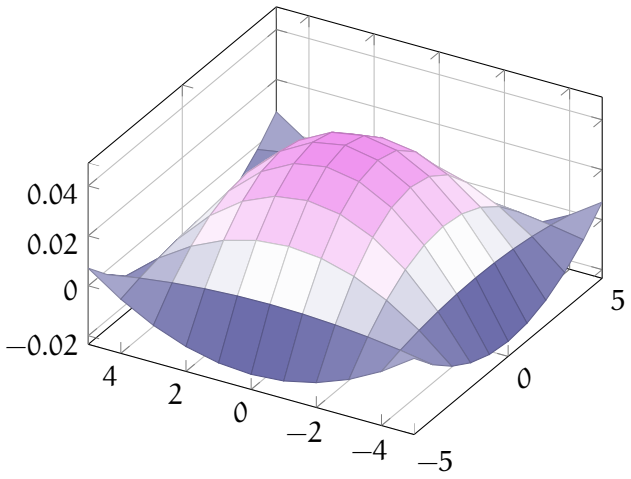


Abb. 32: Savitzky-Golay Filterkern der Größe 11x11 vom Grad 3×2. Zur Veranschaulichung sind die gerundeten Koeffizienten (oben) graphisch dargestellt (unten).

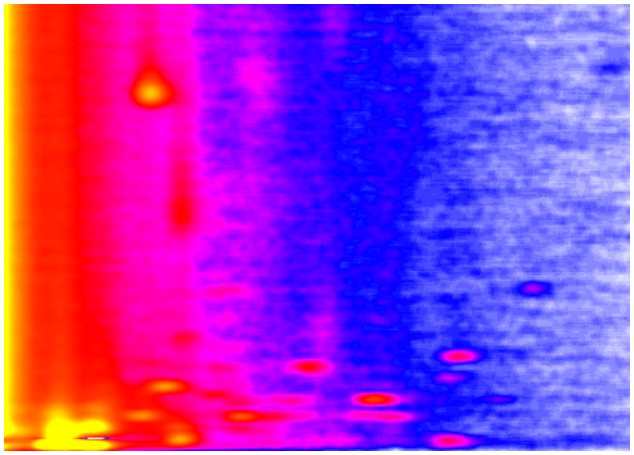


Abb. 33: Die Beispielmessung gefiltert mit Savitzky-Golay der Größe 11×11 vom Grad 3×2.

8 PEAKERKENNUNG

In Abschnitt 2.1 wurden die MCC/IMS-Messungen dargestellt. Auf den Bildern der Messungen sind schon mit bloßem Auge Peaks zu erkennen. Zu den Zielen der PG [aus Kapitel 1.1] gehört primär die Entwicklung eines Online-Algorithmus, der eine Beschreibung der wesentlichen Merkmale von Peaks berechnen kann. In diesem Kapitel widmen wir uns der automatisierten Erkennung von Peaks.

Zuerst wird in diesem Kapitel auf die Definition eines Peaks und seine Charakterisierung im Abschnitt 8.1 eingegangen. Die PG hat während der ersten Wochen verschiedene Ansätze erarbeitet, um die Peaks zu entdecken. Auf einige dieser bestehenden Verfahren wird in Abschnitt 8.2 kurz eingegangen. Danach werden zwei unserer selbst entworfenen Ansätze vorgestellt. In Abschnitt 8.3 wird ein Verfahren erläutert, das über einen speziellen Filter die Krümmung der Messkurve annähert und damit gute Kandidaten für Peaks bestimmt. In Abschnitt 8.4 wird eine Weiterentwicklung dieses Verfahrens vorgestellt, welches die Schwachstellen der ersten Version verbessert und die Parameter noch weiter optimiert. Darauf folgend präsentieren wir im Abschnitt 8.5 einen selbst entwickelten Ansatz, der mit weniger rechenintensiven Operationen die Struktur der Peaks erkennt.

8.1 PEAKDEFINITION

Um einen Peak zu erkennen, muss dieser erst geeignet definiert werden. In der Arbeit von Hauschild u. a. [23, Abschnitt 2.2] wird bei der Beschreibung der gemessenen Daten auf die Peaks eingegangen. Dabei werden Regionen mit höheren Signalwerten als Peaks detektiert. Sie werden über die Position und die Intensität des lokalen Maximum P identifiziert. Somit ist ein Peak ein Tripel $\mathsf{P} = (\mathsf{r}, \mathsf{t}, \mathsf{s})$ mit der Retentionszeit r, der inversen reduzierten Mobilität t und der Signalhöhe s. Im Weiteren wird unter anderem beispielsweise die Schreibweise $\mathsf{r}(\mathsf{P})$ benutzt, um die Retentionszeit eines Peaks abzufragen. Entsprechendes gilt auch für $\mathsf{t}(\mathsf{P})$ und $\mathsf{s}(\mathsf{P})$.

Eine Beschreibung der erkannten Peaks soll ausgegeben werden. Die Ausgabe soll wie in Abschnitt 4.3 spezifiziert aussehen. Zuerst schauen wir uns an, wie die grobe Struktur eines Peaks ist. Daraufhin gehen wir auf zwei Ansätze zur Modellierung von Peaks ein, die jeweils auf den Arbeiten von Kopczynski u. a. [27] und Bödeker und Baumbach [7] basieren.

EINFACHE PEAKSTRUKTUR Eine einfache Ansicht im zweidimensionalen Raum beinhaltet grob drei Strukturen:

- Steigende Flanke
- Fallende Flanke

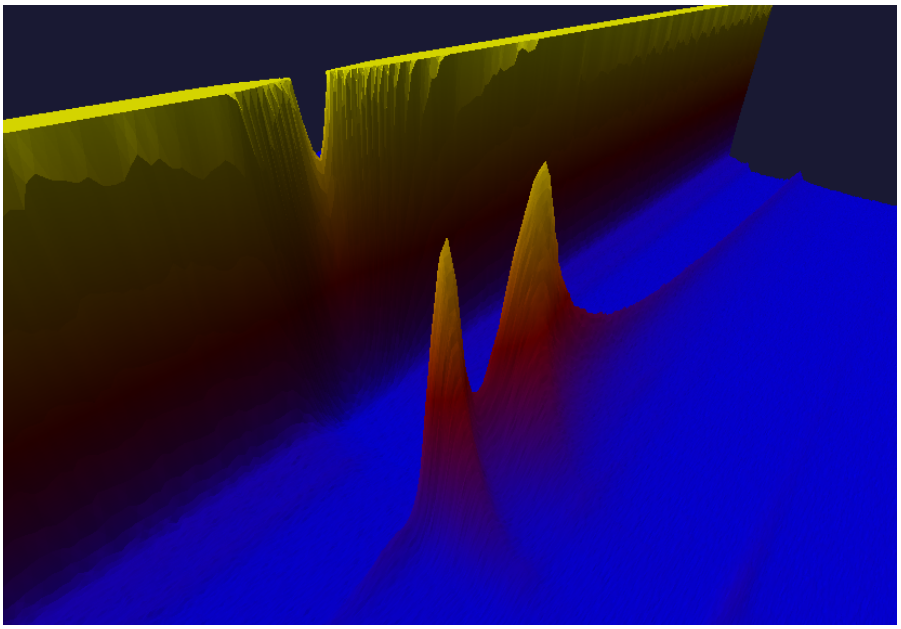


Abb. 34: Zur Retentionszeit in die Länge gezogener Peak.

- höchster Punkt (oder auch ein kurzzeitiges Plateau)

Diese Struktur wird im Abschnitt 8.5 genauer unter die Lupe genommen, analysiert und bei einem Verfahren zu Grunde gelegt.

PEAKMODELLIERUNG MIT DER VERSCHOBENEN INVERSEN NORMALVERTEILUNG Bei der näheren Betrachtung der gemessenen Peaks stellt man fest, dass sie in Richtung der inversen reduzierten Mobilität meist einen fast symmetrischen Verlauf aufweisen. Im Bezug auf die Retentionszeit fallen die Signalwerte nur langsam ab und hinterlassen wie in Abb. 34 zu erkennen einen Schweif.

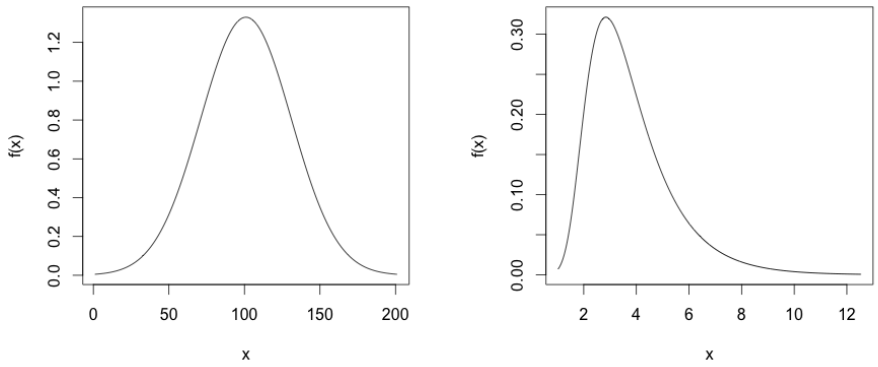
Eine nur grobe Annäherung an einen solchen Peak wäre die Gaußverteilung (auch Normalverteilung genannt):

$$g_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{20}$$

wobei μ der Erwartungswert und $\sigma > 0$ die Standardabweichung der Verteilung sind. Die Funktion kann den Verlauf in den Retentionszeiten mit $g_{\mu_r,\sigma_r}(r)$ für $r := r(P)$ und auch entsprechend den Verlauf in den inversen reduzierten Mobilitäten mit $g_{\mu_t,\sigma_t}(t)$ für $t := t(P)$ abbilden. Die Werte $\mu_r, \sigma_r, \mu_t, \sigma_t$ modellieren somit die unterschiedlichen Verläufe in den Achsen. Der Verlauf der Funktion sieht dann wie Abb. 35a aus. Kopczynski u. a. [27] schlägt jedoch eine genauere Möglichkeit vor, wie die Peaks modelliert werden können. Dieses Verhalten kann nämlich mit der *shifted inverse Gaussian distribution* (deutsch: *verschobene inverse Gaußverteilung*) dargestellt werden:

$$h_{\mu,\lambda,o}(x) = [x > o]\sqrt{\frac{\lambda}{2\pi(x-o)^3}} \exp\left(-\frac{\lambda((x-o)-\mu)^2}{2\mu^2(x-o)}\right), \tag{21}$$

wobei $\mu > 0$ dem Erwartungswert der Verteilung entspricht, der aber um den *Offset* $o > 0$ verschoben ist. Der Parameter λ bestimmt die



(a) Normalverteilung (b) Inverse Normalverteilung

Abb. 35: Normalverteilung und inverse Normalverteilung, deren rechte, absteigende Flanke langsamer fällt.

Form der Verteilung. Bei $\lambda \to \infty$ ähnelt die inverse Normalverteilung der normalen Gaußverteilung. In der Abb. 35b ist ein beispielhafter Verlauf der inversen Normalverteilung dargestellt. Durch diverse Anpassungen kann sowohl der Verlauf entlang den inversen reduzierten Mobilitäten ähnlich der Gaußverteilung als auch der längere Schweif entlang den Retentionszeiten modelliert werden. Dann sieht die Dichteverteilung für einen Peak P in etwa wie folgt aus:

$$p_{\mu_R,\sigma_R,o_R,\mu_T,\sigma_T,o_T,}(r,t) := h_{\mu_R,\sigma_R,o_R}(r)\, h_{\mu_T,\sigma_T,o_T}(t) \tag{22}$$

für $r := r(P)$ und $t := t(P)$. Diese Verteilung stellt grob dar, wie die Peaks modelliert werden können. Weitere Einzelheiten, wie die Normalisierung und Gewichtung der Verteilungen, findet man in der diesem Abschnitt zugrunde liegender Arbeit von Kopczynski u. a. [27]. Einige davon werden zur Erzeugung künstlicher (synthetischer) Messungen mit den vorgegebenen Peaklisten verwendet. Mehr Einzeilheiten dazu findet man im Abschnitt 13.2.2.

ANALYTISCHE BESCHREIBUNG DER PEAKS Das Modellieren der Peaks kann durch Mischverteilungen approximiert werden. Eine solche Möglichkeit wird in der Veröffentlichung von Bödeker und Baumbach [7] beschrieben. Die Peaks werden mit einer Funktion beschrieben, die auf der logarithmischen Normalverteilung basiert:

$$f_{\mu,\sigma}(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln(x)-\mu)^2}{2\sigma^2}\right), \quad x > 0\,. \tag{23}$$

Der Wert der logarithmischen Normalverteilung wird durch Übergabe des Mittelwerts der Verteilung μ und der Standardabweichung σ von der zugrunde liegenden Gaußverteilung bestimmt. Damit soll die Eigenschaft der Peaks beschrieben werden, dass die steigende Flanke steiler als die fallende ist. Es werden jeweils zwei solche Funktionen eingesetzt: eine beschreibt den Verlauf des Peaks spektrumweise und die andere

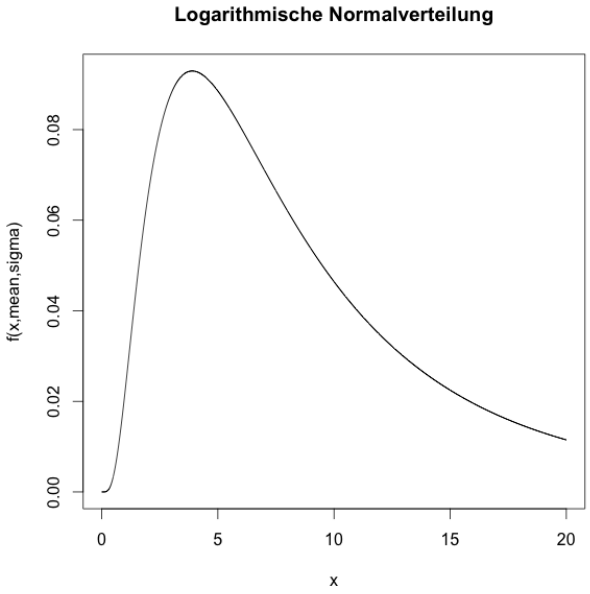


Abb. 36: Zur analytischen Beschreibung eines Peaks wird die logarithmische Normalverteilung zu Grunde gelegt. Diese Abbildung wurde mit Parametern $\mu = 2$ und $\sigma = 0.8$ mit Rstudio erstellt.

chromatogrammweise. In der Abb. 36 kann man den Verlauf der logarithmischen Normalverteilung sehen.

Für einen Datenpunkt mit Koordinaten (r, t) gilt dann:

$$\mathrm{peak}(r,t) = \frac{1}{s(P)} a_t \, f_{\mu_t,\sigma_t}(z_t) \, a_r \, f_{\mu_r,\sigma_r}(z_r) \,, \tag{24}$$

wobei t die inverse reduzierte Mobilität und r die Retentionszeit sind. Parameter a_t und a_r stauchen die logarithmische Normalverteilung. Die Parameter z, μ und σ hängen entsprechend von den Koordinaten t und r ab. Durch eine gute Abschätzung dieser Parameter kann das Residuum von echten Daten und des Modells minimiert werden. Abhängig vom höchsten Punkt P des jeweiligen Peaks, der mit der inversen reduzierten Mobilität $t(P)$, Retentionszeit $r(P)$ und der entsprechenden Signalhöhe $s(P)$ bekannt ist, werden die restlichen Punkte mit der Funktion modelliert. Weitere Details können der Veröffentlichung entnommen werden.

Die Idee hinter dem Peak-Modell, welches in der Arbeit von Bödeker und Baumbach [7] dargestellt wird, ist, solche Modelle in einer Datenbank zu sammeln. Bei neueren Messungen greift man auf die Datenbank zu und versucht die bekannten modellierten Peaks passend zu finden. Sowohl bei einzelnen Peaks als auch bei gemischten Verteilungen, wobei mehrere Peaks sich überlagern, ist es möglich, einen neuen Peak zu einem Analyten zu erkennen.

8.2 BESTEHENDE OFFLINE-VERFAHREN

Bevor wir eigene Ansätze entwickeln, schauen wir uns bestehende Verfahren und deren zugrundeliegenden Ideen an. Hierbei überprüfen wir, ob diese für unsere Zwecke als Online-Algorithmen einsetzbar sind.

8.2.1 *Watershed-Verfahren*

Das Watershed-Verfahren basiert auf der Idee, die Daten einer IMS-Messung als geografische Fläche zu betrachten und diese langsam mit Wasser zu fluten. Durch die schrittweise Füllung einer jeden Senke mit Wasser zeichnen sich Grenzen zwischen den einzelnen Peaks ab, die aus der gefluteten Messung heraus ragen. In unserem Anwendungsfall sorgt das Rauschen für eine Übersegmentierung der Daten, da auch kleinere Unebenheiten als Peaks gewertet werden und deren Grenzen den Wasserfluss behindern. Zwar existieren Verfahren um diesen Effekt zu kompensieren, diese basieren jedoch entweder auf händischer Annotation oder empirischen Werten, die vor Durchführung einer Berechnung festgelegt werden müssen. Zur Durchführung eines klassischen Watersheds muss das Gesamtbild vollständig vorliegen. Es wird sowohl in der Dimension der Retentionszeit als auch in der Driftzeit überprüft, ob sich eine gefüllte Region mit einer anderen überschneidet. Eine Implementierung oder einen Ansatz zur Berechnung eines Watersheds mit einzelnen Spektren konnten wir in der Literatur nicht finden. Der Vergleich einzelner Spektren miteinander als auch die schrittweise Erhöhung des Wasserstandes erscheinen uns in der Rechenzeit sehr kostspielig und in einem Online-Szenario schwierig zu realisieren. Aufgrund dieser Schwierigkeiten konzentrieren wir uns auf einfacher zu implementierende Verfahren und betrachten das Watershed-Verfahren nicht weiter.

8.2.2 *Gradientenbasierte Maximumsuche*

Die Messwerte können als ein Graph einer zweidimensionalen Funktion aufgefasst werden. Im Folgenden nehmen wir an, dass diese Funktion hinreichend „glatt“ (differenzierbar) ist; insbesondere nehmen wir an, die Messung wäre vollständig vom Rauschen und RIP befreit. Damit lassen sich die lokalen Maxima mit Hilfe des Gradienten bestimmen:

Wenn innerhalb eines Spektrums die Steigung entlang der Driftzeit erst positiv und anschließend negativ ist, befindet sich dazwischen ein Peak. Auf gleiche Weise findet man Peaks innerhalb der Chromatogramme. Um die lokalen Maxima in einer zweidimensionale Messung zu bestimmen, kombinieren wir die zwei partiellen Ableitungen, indem wir die (quadrierte euklidische[1]) Norm des Gradienten berechnen.

Zur Umsetzung dieses Gedankens müssen wir die partiellen Ableitungen bestimmen. Ein einfacher Differenzenquotient zwischen zwei Pixeln ist zu störungsanfällig gegenüber dem Messrauschen, funktioniert also nur bei guter Vorverarbeitung. Genauer wird eine Schätzung der Steigung, wenn man mehrere Pixel in die Berechnung einbezieht. Hierfür eignet sich ein Savitzky-Golay Filter (siehe Abschnitt 7.2.3) – als günstiger Nebeneffekt glättet er die Messung gleichzeitig. Eine so verarbeitete Beispielmessung (Abb. 22) findet sich in Abb. 37. Auf diese Weise berechnete Werte sind in Bereichen um die Peaks wie erwartet hoch

1 Wir haben auch die $\|\cdot\|_1$-Norm (Summe der Absolutbeträge) getestet, jedoch keinen wirklichen qualitativen Unterschied feststellen können (was auch nicht weiter verwundert, da im $\mathbb{R}^2$ alle Normen äquivalent sind).

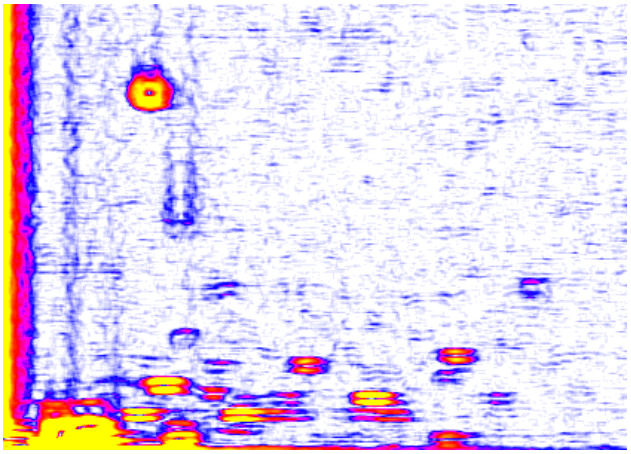


Abb. 37: Durch einen Savitzky-Golay Filter (Größe 11×11, Grad 3×3) approximierte erste Ableitung (Norm des Gradienten). Die Messwerte wurden skaliert und ab einem bestimmten Wert abgeschnitten.

(gelb). Allerdings eignen sich die Werte nicht zur Peakbestimmung, da die Regionen von einem Peak unterbrochen sind und Löcher oder Aufspaltungen aufweisen. Diese entstehen, da die erste Ableitung in Extrempunkten verschwindet. Wir stellen also fest, dass die Information der ersten Ableitung alleine nicht zur Peakerkennung genügt.

8.3 DAS SGLTR-VERFAHREN

Wir beschreiben einen selbst entwickelten Algorithmus, der über einen speziellen **S**avitzky-**G**olay Filter den **L**aplace-Operator annähert und Werte über einem Schwellwert (engl. **T**hreshold) zu **R**egionen zusammenfasst, die vermutlich die Peaks enthalten (SGLTR).

8.3.1 *Dahinterliegende Idee*

Das Verfahren entstand aus Überlegungen heraus, die Nachteile des gradientenbasierten Ansatzes (Abschnitt 8.2.2) auszugleichen: Aus dem Gradient kann man nur erkennen, ob die Messkurve steigt oder fällt, d. h. ob man sich einem Peak annähert oder sich davon entfernt. Dadurch einen Peak zu finden, ist jedoch nicht so einfach, da in den höchsten Teilen des Peaks die Ableitung wieder abnimmt (vgl. „Löcher“ vorherigen Abschnitt). Ein weiterer Nachteil der ersten Ableitung ist, dass man noch eine Norm des Gradienten berechnen muss, was eine nicht-lineare Berechnung erfordert.

Es liegt daher nahe, auch die zweite Ableitung (in zwei Dimensionen *Hessematrix*) mit einzubeziehen. Wie in Abschnitt 7.2.3 und 7.2.4 gesehen, lässt sich dies ebenfalls über einen Savitzky-Golay-Filter annähern. Anders als bei der ersten Ableitung sind die zweiten partiellen Ableitungen in der Nähe eines Peaks immer negativ (der Graph beschreibt eine Rechtskurve entlang der Koordinatenachsen). Daher reicht es, die Summe der zweiten partiellen Ableitungen statt einer Norm zu berechnen. Erste Versuche haben gezeigt, dass die gemischte Ableitung $\frac{\partial^2}{\partial x \partial y}$ zu Verzerrungen in den berechneten Werten um die Peaks führt. Ein

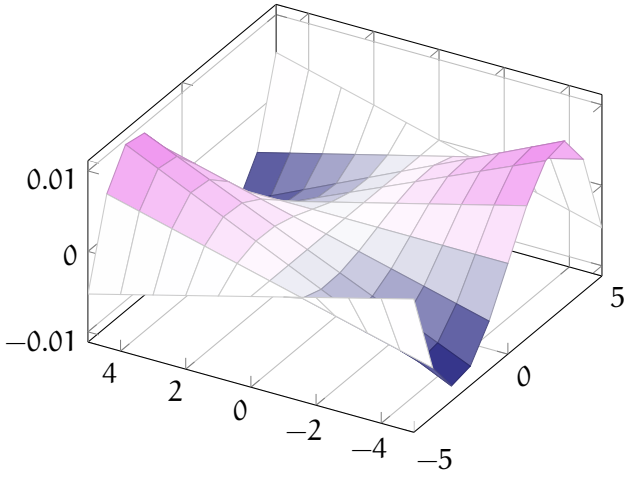


Abb. 38: Savitzky-Golay Kernel zur Berechnung der gemischten Ableitung $\frac{\partial^2}{\partial x \partial y}$ (Größe 11×11, Grad 3×2).

Blick auf den zugehörigen Filterkern (Abb. 38) macht dies verständlich, da nur die Diagonalen stark gewichtet werden. Aus diesem Grund haben wir die gemischte Ableitung aus der Berechnung entfernt. Damit bleibt die Summe der zweiten partiellen Ableitungen nach x und y – dies ist der *Laplace-Operator*. Ein so designter Filterkern ist in Abb. 39 zu sehen; Abb. 40 zeigt die mit diesem Kern gefilterte Beispielmessung

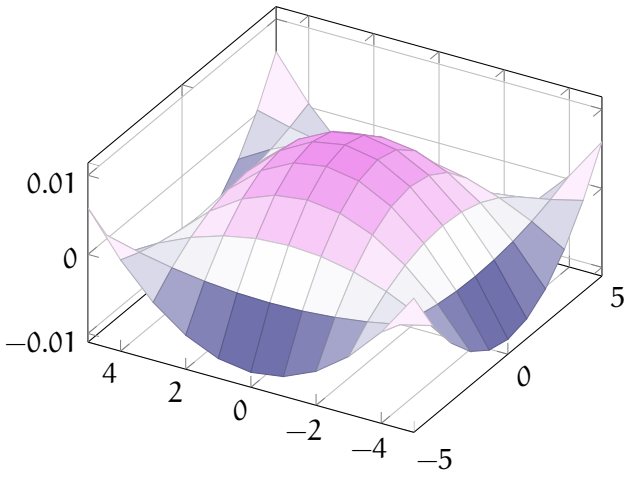


Abb. 39: Filterkern zur Annäherung des Laplace-Operators.

(vgl. Abb. 22). Im Vergleich zu der Annäherung der ersten Ableitung (siehe Abb. 37) sind die Peaks nun deutlich erkennbar. Hintergrundrauschen ist zwar noch immer vorhanden, aber mit geringerer Intensität im Verhältnis zu den Peakwerten, da der Savitzky-Golay Filter gleichzeitig glättet.

Intuitiv kann man den Laplace-Operator als ein Maß für die *Krümmung* im Graph der Messwerte auffassen. Da in einem Peak die Krümmung verhältnismäßig groß ist, hielten wir dies für einen viel versprechenden Ansatz zur Peakerkennung. Man beachte, dass nur eine Faltung mit einem vorberechneten Filterkern nötig ist. Daraus ist folgender Algorithmus entstanden, den wir auf dem Raspberry Pi implementiert haben.

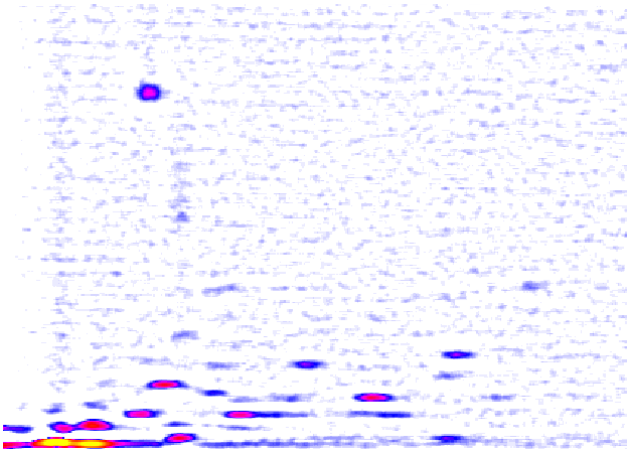


Abb. 40: Beispielmessung gefiltert mit einem Laplace-Operator (Filterkern aus Abb. 39).

8.3.2 *Signifikante Messwerte*

Ein Messwert $S_{r,t}$ ist *signifikant*, wenn die Krümmung an dieser Stelle hoch ist. Konkret führen wir die Faltung mit dem wie oben entworfenen Filterkern zur Approximation des Laplace-Operators aus und prüfen, ob der Wert über einem Schwellwert θ liegt. Nur die signifikanten Messpunkte werden in einer Hashmap gespeichert. Auf diese Weise ist der Algorithmus *effizient* was Laufzeit (siehe Abschnitt 7.1.3) und Speicherplatz angeht.

Den Schwellwert θ haben wir experimentell bestimmt. Inwiefern der optimale Parameter θ vom Rauschniveau, dem Messgerät oder anderen Einflüssen abhängt, konnten wir noch nicht untersuchen. Bei reduzierter Auflösung sind in unseren Experimenten nur rund 3000 bis 5000 (von rund 10^6) signifikant.

8.3.3 *Regionen bestimmen*

Nah beieinander liegende signifikante Messwerte gehören vermutlich zu demselben Peak. Diese fassen wir daher zu Regionen (Cluster) zusammen. Zuerst wurde dazu ein Offline-Verfahren entwickelt. Wir konstruieren nach Abschluss der Messung einen Graph mit den signifikanten Messwerten als Knoten, wobei ein Knoten mit den vier nächsten Nachbarpositionen im Gitter der Messpunkte verbunden ist. Auf diesem Graph bestimmen wir wie in Abb. 41 dargestellt über eine Breitensuche die Zusammenhangskomponenten.

Regionen mit zu wenig Pixeln können auch durch zufällige Störungen (Rauschen) entstehen. Wir filtern daher zu kleine Zusammenhangskomponenten aus und betrachten diese nicht weiter (die Mindestanzahl N_{min} von Pixeln für eine Region wurde nach Begutachtung einiger Messungen auf 10 festgelegt).

ONLINE-ALGORITHMUS. Ein im weiteren Verlauf realisiertes Online-Verfahren speichert nur Informationen über die Stellen, an denen sich im letzten Spektrum Regionen befanden. Diese Regionen werden

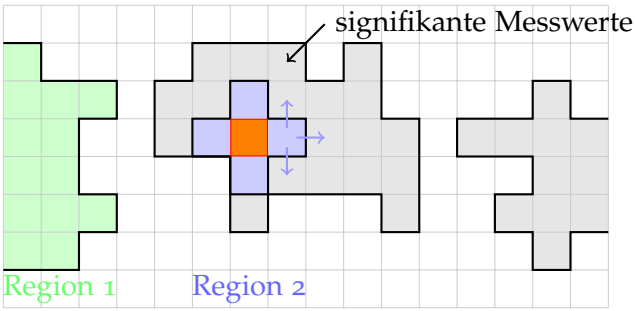


Abb. 41: Benachbarte signifikanten Pixel werden zu Regionen zusammengefasst, wobei ein Pixel (orange) jeweils mit den vier nächsten (blau) benachbart ist.

wie in Abb. 42 gezeigt mit neu gefundenen signifikanten Messwerten aus dem momentanen Spektrum gegebenenfalls fortgesetzt, sich dadurch nun berührende Regionen zusammengefügt oder nicht aktualisierte Regionen, die nun nicht mehr erreichbar sind, beendet. Die tatsächliche Form der Regionen ist hierbei irrelevant, es werden nur die minimalen und maximalen Drift- und Retentionszeiten gespeichert. Es kann in einem Spektrum maximal halb so viele Regionen geben wie Datenpunkte, da zwischen zwei verschiedenen Regionen mindestens ein Punkt Abstand bestehen muss. Somit ergibt sich für den benötigten Speicherplatz eine obere Schranke unabhängig von der Anzahl der verarbeiteten Spektren.

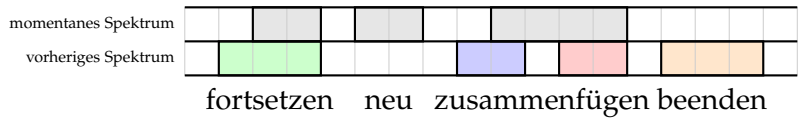


Abb. 42: Online-Algorithmus zum Bestimmen von Regionen.

Beim in Abschnitt 8.4.3 vorgestellten Zwei-Level SGLTR ist dies jedoch nicht mehr ausreichend.

8.3.4 *Peakzentrum ermitteln*

Wir beschreiben einen Peak über einen einzelnen Messpunkt (Details in Abschnitt 8.1). Als letzten Schritt bestimmen wir daher für jede Region einen Repräsentanten. Dies kann man realisieren, indem man den Mittelwert, den Schwerpunkt oder ein lokales Maximum der signifikanten Messwerte der Region berechnet.

In kleinen Experimenten lieferte die lokale Maximumsuche stets die besten Resultate, weshalb wir nur diese Methode weiter verwendet haben.

8.3.5 *Implementierung*

Diesen SGLTR-Algorithmus haben wir erst in Python evaluiert, um uns von der Qualität des Ansatzes zu überzeugen und die besten Parameter (Schwellwert, Grad und Größe des Filterkerns) zu ermitteln. Anschlie-

ßend wurde der Filterkern vorberechnet und zusammen mit den anderen Parametern als Konstanten in einen C-Header exportiert. Damit das Verfahren auch praktisch anwendbar ist, wurde vorher der RIP entfernt, indem eine Basislinie (Mittelwert von, bei uns, 120 Spektren, der aus einer Luftmessung berechnet wird) abgezogen wurde.

8.4 WEITERENTWICKLUNG DES SGLTR-ANSATZES

Wir haben den oben beschriebenen Algorithmus vollständig implementiert. Da er vielversprechende Ergebnisse produziert, beschäftigen wir uns im Weiteren mit Schwachstellen und entwickeln eine Technik um diese zu beheben.

8.4.1 *Debugging-Tools und fehlerhafte Versionen*

Das Experimentieren mit verschiedenen Parametern war anfangs sehr zeitaufwendig: Bei jeder Änderung musste die Datei mit den Konstanten `sgolay.hpp` neu erstellt und anschließend das Programm neu gebaut werden. Deshalb wurde das Python-Skript `sgparam.py` so erweitert, dass die Parameter aus einer für Menschen gut lesbaren JSON-Datei (genauer beschrieben in Abschnitt 8.4.4) in eine Binärdatei übersetzt werden können, die sich dann über den Parameter `-b` zur Laufzeit nachladen lässt. Das Programm mit fest einkompilierten Konstanten[2] ist erstaunlicherweise nicht immer schneller als diese dynamische Variante.

Das Laden der Binärdatei enthielt bis zur Version[3] 4.1 einen *Fehler*[4]: Die Koeffizienten des Filterkerns wurden in einer anderen Reihenfolge gelesen als sie geschrieben wurden. Erstaunlicherweise lieferte der Algorithmus mit vertauschten Koeffizienten hervorragende Peakerkennungsraten auf einem Datensatz – auf den anderen Datensätzen waren die Ergebnisse jedoch schlecht.

Die Weiterentwicklung machte es erforderlich, die Auswirkungen von den verschiedenen Parametern auf die gefaltete Messungen genauer zu untersuchen. Die Berechnung der Faltung in Python[5] ist viel langsamer als die selbst geschriebene C++-Implementierung, die bisher jedoch nur die Peaks ausgab. Um bei der Entwicklung schneller voran zu kommen, wurde daher der folgende Debugging-Mechanismus implementiert: Alle Zwischenergebnisse und Parameter der Berechnung werden in eine

2 Die fest in das Programm einkodierten Konstanten funktionieren nur bis zur Version 4.2, wenn ein entsprechender Header über `sgparam.py` *`param`*`.json` ohne weitere Parameter generiert wird; in diesem Fall wird das Präprozessormakro `CONSTANT_INLINE` auf `1` gesetzt.

3 Die letzte noch funktionierende SGLTR-Version mit diesem Fehler ist mit dem Tag `sgltr-3.90` versehen.

4 Der Fehler ist anfangs nicht entdeckt worden, weil ausgerechnet die stichprobenartig überprüften Koeffizienten nicht vertauscht wurden.

5 Die Faltung wurde über `convolve` bzw. `fftconvolve` aus dem Paket `scipy.signal` berechnet.

HDF5-Datei geschrieben, die nach einem Durchlauf[6] mit einem selbst geschriebenen Anzeigeprogramm[7] begutachtet werden können.

Die Visualisierung der tatsächlich berechneten Werte ließ erkennen: Die Ergebnisse der Faltungsberechnung unterschieden sich bis zur Version 4 von denen des Python-Skripts. Zunächst wurde vermutet, dass dies auf numerische Ungenauigkeiten[8] oder unterschiedliche Normierungen zurückzuführen ist. Nach einer intensiven Untersuchung steht fest, dass zum Teil stark abweichende Werte heraus kamen – was dort genau berechnet wurde, entzieht sich unserer Kenntnis. Der Fehler wurde durch eine komplette Neuimplementierung der Faltungsberechnung ab Version 5 behoben; die berechneten Werte unterscheiden sich nun um weniger als 10^{-8}.

Diese beiden Fehler wurden erst so spät entdeckt und behoben, dass sie bei der Evaluation der Klassifikation nicht mehr berücksichtigt werden konnten.

8.4.2 *Schwachstellen des SGLTR*

Als Startpunkt für eine Weiterentwicklung des SGLTR-Verfahrens haben wir die ausgegebenen Peaks und die Messungen angeschaut, bei denen die Erkennungsrate besonders schlecht war. Diese Untersuchungen haben folgende kritische Fälle ergeben: Flache Peaks (Plateaus), sich überlappende Peaks und ein extremer Unterschied in den Signalwerten der Peaks. Letzteres Problem ist gut in Abb. 43 erkennbar: Um die Peaks mit niedrigen Signalwerten (z. B. Peak 1) zu finden, müsste der Schwellwert θ entsprechend klein gewählt werden. Aus dicht beieinander liegenden Peaks wird infolgedessen eine große Region bestimmt (z. B. Region 2), die es für die nachfolgenden Schritte des Algorithmus unmöglich macht, diese zu differenzieren.

Andererseits setzt man den Schwellwert θ höher, um diese Peaks mit hohen Intensitäten zu trennen, dann werden aber Peaks mit geringen Intensitäten nicht mehr gefunden.

8.4.3 *Zwei-Level SGLTR*

Eine Peakerkennung muss sowohl die Peaks mit niedriger Intensität als auch die mit hoher erkennen. Wie gerade beschrieben, ist dies mit *einem* Schwellwert nicht machbar. Da „niedrige“ Peaks in jedem Fall erkannt werden müssen, bleibt zu überlegen wie „hohe“ Peaks nicht als eine Region, sondern einzeln erkannt werden. Dazu müssen die zu „großen“ Regionen zunächst als solche identifiziert werden, was wir über

6 Dazu muss `sgolay` mit Debug-Support kompiliert werden (setze in der `build_config.cmake` die Variable `SGLTR_DEBUG` auf `1`). Die Ausgabedatei mit den Debuginformationen wird über den Parameter `-d` *file* angelegt.

7 Benutzung: Siehe `python/common/sgview.py -h`.

8 Mit dem Datentyp `float` (32-Bit) lässt sich auf dem Raspberry-Pi schneller rechnen, insbesondere weil Cache-Effekte geringer sind (siehe Kapitel 9). In Python wird der genauere Datentyp `double` (64-Bit) verwendet. Außerdem könnte es sein, dass die Berechnung über separierte Filterkerne auch leicht andere Werte liefert als die Berechnung über die Fouriertransformation.

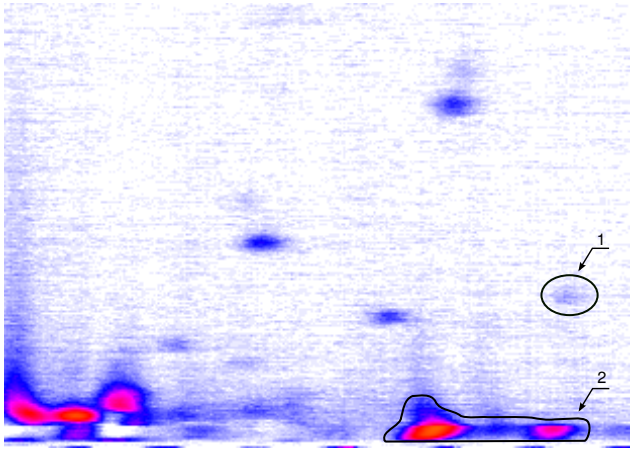


Abb. 43: Um in der Messung (nach Basislinienkorrektur) den Peak 1 zu finden, muss der Schwellwert θ niedrig gewählt werden. Dadurch können die Peaks in Region 2 nicht getrennt erkannt werden.

die Anzahl der Pixel prüfen: Wenn eine Region mehr als N_{max} viele Pixel erreicht, wird dies als eine Ansammlung von Peaks eingeordnet.

Zum Erkennen der „hohen“ Peaks könnte man Verfahren entwickeln, die „große“ Regionen weiter auftrennen. Der Schwellwert θ_{low} zur Erkennung der „niedrigen“ Peaks darf weiterhin nicht zu klein gewählt werden, da sonst ein Großteil der Messung in einer Region landen würde – dies würde das Peakerkennungsproblem also nicht vereinfachen.

Wir wollten für das Auftrennen der großen Regionen kein neues, komplexes Verfahren entwickeln. Daher haben wir das bestehende SGLTR-Verfahren nochmals auf die großen Regionen mit einem höheren Schwellwert θ_{high} angewendet. Praktischerweise ist die Idee effizient umsetzbar, da der Laplace-Operator bereits berechnet ist.

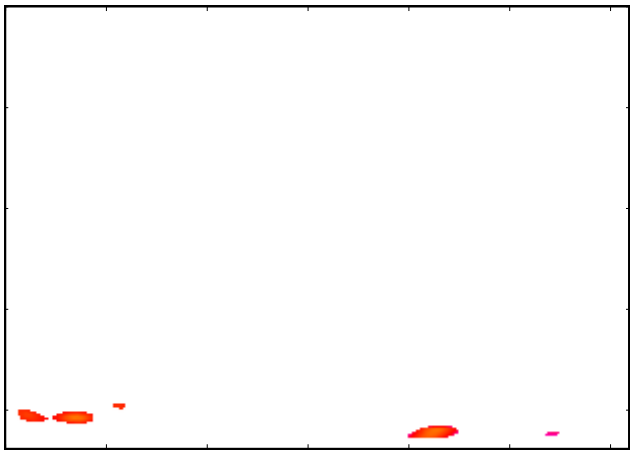
Der so entstandene **Zwei-Level SGLTR** arbeitet wie folgt (siehe Abb. 44). Für jeden Messwert $S_{r,t}$ (bis auf einen Randbereich):

> Berechne Laplace-Operator $\tilde{S}_{r,t}$ über Savitzky-Golay-Filter.
> Wenn $\tilde{S}_{r,t} > \theta_{high}$: Füge (r,t) zu „hohen“ Regionen hinzu.
> Wenn $\tilde{S}_{r,t} > \theta_{low}$: Füge (r,t) zu „niedrigen“ Regionen hinzu.

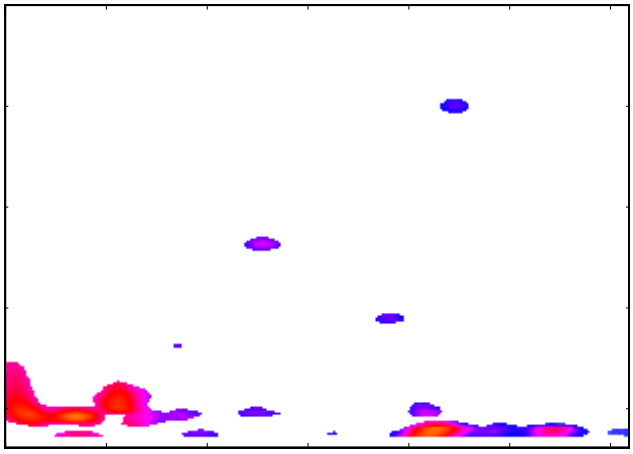
Die Peaks innerhalb einer Region zu bestimmen erfolgt wie bisher; beide Varianten (offline oder online) wurden implementiert. Eine ermittelte „niedrige“ Region wird nur dann als Peak ausgegeben, wenn die Anzahl der Pixel in $[N_{min}, N_{max}]$ liegt. Eine „hohe“ Region wird immer ausgegeben.

Dies führt dazu, dass manche Peaks doppelt ausgeben werden (einmal durch eine „hohe“ Region und einmal durch eine „niedrige“). Der Fehler wurde behoben, indem zuerst die „hohen“ Regionen verarbeitet wurden; bei einer Ausgabe eines Peaks wurde die zugehörige „niedrige“ Region als *„tot“* markiert, d. h. es erfolgt keine Ausgabe dieser Region.

Um sich überlappende Regionen in beiden Leveln erkennen zu können, muss im Online-Regionfinder nun doch die Form der Regionen gespeichert werden. Dazu werden wieder die gefundenen Punkte in eine Hashmap eingefügt. Somit ist der Algorithmus streng formal nicht mehr online, da potenziell beliebig viele signifikante Punkte in einer Region



(a) Regionen des oberen Schwellwerts θ_{high}: Innerhalb der großen Region zu θ_{low} in 44b werden verschiedene Peaks getrennt.



(b) Regionen des unteren Schwellwerts θ_{low}: Die große Region wird nicht als Peak ausgegeben, da sie zu viele Pixel enthält.

Abb. 44: Regionen zum oberen θ_{high} und unteren θ_{low} Schwellwert. Zu sehen ist derselbe Ausschnitt wie in Abb. 43.

gefunden werden können. Da Peaks nur eine begrenzte Größe haben, wird dies praktisch jedoch nicht auftreten.

8.4.4 *Parameteroptimierung*

Das SGLTR-Verfahren hat viele Parameter. Im Folgenden berichten wir, wie wir diese eingestellt haben.

PARAMETER. Sämtliche Parameter des SGLTR-Verfahrens können über eine JSON-Datei konfiguriert[9] werden.

Wie man in der Beispiel-Datei in Abb. 45 sehen kann, gibt es viele freie Parameter: Grundsätzlich sind zwei Kernel `K1` und `K2` vorgesehen (hier der Laplace-Operator). Die Größe des Filterkerns ist jeweils `nrows` × `ncols` (auch unterschiedlich große Kerne sind möglich). Über `type`

9 Zur Verwendung wird das Skript `sgparam.py -b` aufgerufen und daraus eine Binärdatei erzeugt, die dann dynamisch geladen werden kann.

```
{ "comment": "Default Parameters for sgolay",
  "full_resolution": false,
  "K1": {
      "type": "sgolay",
      "factor" : 1,
      "coef": [0, 2],
      "nrows": 13, "ncols": 13,
      "rdeg":   3, "cdeg":   3,
      "couple":    false,
      "normalize": false
  },
  "K2": {
      "type": "sgolay",
      "factor" : 1,
      "coef": [2, 0],
      "nrows": 13, "ncols": 13,
      "rdeg":   3, "cdeg":   3,
      "couple":    false,
      "normalize": false
  },
  "rip": { "npost_rip": 30, "nlast": 200 },
  "threshold_low":  0.2,
  "threshold_high": 0.8,
  "min_region_size": 10,
  "max_region_size": 600
}
```

Abb. 45: JSON-Parameterdatei für SGLTR.

sind auch andere Filterkerne einstellbar, die mit `factor` multipliziert aufsummiert werden. Beim Typ `sgolay` bzw. `wsgolay` (gewichteter Savitzky-Golay, siehe Abschnitt 7.2.2) geben `rdeg` bzw. `cdeg` den Grad des zweidimensionalen Polynoms entlang der Zeilen bzw. Spalten an; `coef` wählt den Koeffizient des Polynoms, also welche Ableitung gewählt wird.

Die Berechnung startet bei `npost_rip` nach dem RIP (höchster Punkt in der Luftmessung, die mindestens `nlast` Spektren enthält).

Als besonders wichtig für die Güte des Verfahrens haben sich die Schwellwerte `threshold_low` und `threshold_high` (im Text oben als θ_{low} und θ_{high} bezeichnet) erwiesen, ebenso wie die geforderte Größe einer Region, die über `min_region_size` und `max_region_size` eingestellt wird (im Text N_{min} und N_{max}).

OPTIMIERUNGSTECHNIK. Zur Optimierung wurden ausgehend von einem Standardparametersatz einige Parameter leicht variiert[10]. Da die Parameter wechselseitig voneinander abhängig sind, wurden alle Kombinationsmöglichkeiten durchprobiert (*brute force*).

Das Ziel der Optimierung war es, möglichst genau die manuell annotierten Peaks zu erkennen (Gütekriterium war das F1-Maß wie in Abschnitt 13.3.1 beschreiben) oder eine möglichst hohe Klassifikationsraten zu erreichen. Unterschiedliche Geräte- und Umgebungseinflüsse

10 Die Optimierung wurde mittels des `sgolay-optim.py` Skripts realisiert, das über eine JSON-Datei konfiguriert wird, wie z. B. `tea-coffee.optim.json`.

machten es notwendig, für jeden Datensatz eigene optimierte Parameter zu ermitteln.

Insgesamt ist die Optimierung damit ziemlich rechenaufwendig, sodass nur folgende Parameter optimiert werden konnten:

- Fenstergröße der Filterkerne `nrows`×`ncols`.
- Oberer und unterer Schwellwert θ_{high} und θ_{low}.

RESULTATE. Die meisten Verbesserungen entstanden durch händische Begutachtung der gefundenen Peaks in Messungen, bei denen die Peakerkennung besonders schlecht ausfiel. Dabei mussten wir feststellen, dass manuell bestimmte Peaks teilweise an ganz anderen Stellen gesetzt waren als wir es erwarten würden. Die Optimierung gegen diese Annotationen führt deshalb vermutlich nicht zu besseren Parametern zur Peakerkennung auf neuen Messungen.

Diese schlechten Ergebnisse haben uns motiviert, nicht nur auf die Peakerkennung hin, sondern auch auf die Klassifikationsergebnisse hin zu optimieren. Die Dauer für eine Auswertung von nur einem Parametersatz lag hier jedoch im Stundenbereich, sodass nur rund fünf Parametersätze ausprobiert werden konnten.

Die beiden Fehler in Abschnitt 8.4.1 beschriebenen Fehler machten es notwendig, eine komplette neue Optimierung durchzuführen. Für einen Datensatz, der später nicht zur Evaluation verwendet wurde, führte die fehlerhafte Version zu hervorragenden Ergebnissen der Peakerkennung. Dies ließ uns glauben, die schlechten Ergebnisse auf anderen Datensätzen lägen allein daran, dass wir die „richtigen“ Parameter noch nicht gefunden hätten.

Aus Zeitgründen gelang eine neue Optimierung nur für den Tee-Kaffee-Datensatz. Die Evaluation der Klassifikation auf den gefundenen Peaks war zu diesem Zeitpunkt schon durchgeführt worden, sodass die Korrekturen dort nicht in die Evaluation einfließen konnten.

8.4.5 *Ausblick*

Während der Entwicklung sind uns einige Ideen gekommen, wie sich der SGLTR-Algorithmus verbessern lässt. Wir schildern die aus unserer Sicht interessantesten.

Eine wissenschaftlich fundierte Methode der Parameteroptimierung ist sicherlich sinnvoll: Beispielsweise könnte man hierzu *evolutionäre Algorithmen* oder andere allgemeine, ableitungsfreie Optimierverfahren einsetzen. Dann ließen sich vermutlich auch mehr Parameter variieren.

Insbesondere *andere Filterkerne* würden vermutlich neue Ergebnisse liefern. Eine andere Art der Approximation des Laplace-Operators[11] ist denkbar. Wir haben begründet, dass der Laplace-Operator theoretisch

11 In der Bildverarbeitung wird der Laplace-Operator zur Kantendetektion verwendet. Dort wird der Laplace-Filter üblicherweise über seine Transferfunktion approximiert, wie von Jähne [25, Abschnitt 12.5] beschrieben. Diese Berechnungsart ist über `transfer` auch als Filter-type einstellbar in der JSON-Parameterdatei. Nach kurzen Tests braucht man hier jedoch eine vorherige Glättung, da das Rauschen sonst zu stark ist.

günstig ist zur Peakerkennung. Letzten Endes könnte man jedoch auch versuchen, unseren Filterkern nur als Ausgangspunkt zu nehmen und die einzelnen Filterkoeffizienten leicht zu „mutieren“, um noch bessere Ergebnisse zu erhalten.

Insgesamt bieten die vielen Parameter noch ein großes Entwicklungspotential. Bei jeder Form der Optimierung besteht jedoch die Gefahr der *Überanpassung* an den gegebenen Trainingsdatensatz. Um dies zu vermeiden, müsste man genauer untersuchen, wovon die Parameter *abhängig* sind und weiteres *Vorwissen* ausnutzen. Insbesondere ist es erstrebenswert, anhand der Luftmessung die Parameter automatisch jeweils auf die Messung anzupassen.

Die Schwellwerte ließen sich beispielsweise in Abhängigkeit des Mittelwerts μ und der Abweichung σ der gefilterten Werte der Luftmessung bestimmen (z. B. $\theta = \mu + c\sigma$ für eine geeignete Konstante c). Da die Ergebnisse bei konstantem Rauschniveau auf den synthetischen Daten deutlich besser sind, ist dies sicherlich einer der größten Einflussfaktor; insofern finden wir diesen Ansatz vielversprechend.

Ein anderer Ausgangspunkt sind *physikalischen Eigenschaften* des Messgeräts: So ist etwa die maximale Größe eines Peaks apriori begrenzt[12]. Ferner sind die Signalintensitäten nahe des RIPs generell höher – die Schwellwerte könnten daher abhängig von der Driftzeit fallend gewählt werden. Außerdem sind die Peaks mit niedrigen Retentionszeiten viel dichter gedrängt und schmaler, was durch unterschiedlich geformte Filterkerne für bestimmte Abschnitte oder noch besser durch eine andere Zeitskala ausgenutzt werden sollte.

Bisher bestimmen wir innerhalb einer Region nur ein Maximum. Zur besseren *Trennung von Peaks* in großen Regionen ist es vorstellbar, nach mehreren lokalen Maxima zu suchen und diese bei genügend großem Abstand als separate Peaks auszugeben. Alternativ könnten hierzu auch mehr als zwei Schwellwerte verwendet werden. Generell könnte die Form von Peaks (z. B. sollte eine Peakregion konvex sein) stärker in die Berechnung einfließen.

Zu guter Letzt könnte man während der Berechnung „Anomalien der Messungen“ erkennen und Fehler ignorieren, was die Zahl negativer Ausreißer verringern würde.

8.5 PEAK DETECTION BY SLOPE ANALYSIS

Bei dem folgenden Verfahren werden zur Peakdetektion für alle Peaks gemeinsame Merkmale genutzt. Die benötigten Merkmale werden zuerst im Abschnitt 8.5.1 erläutert. Danach erklären wir das Problem der Plateaus an Peakhöhepunkten im Abschnitt 8.5.2 und beschreiben in Abschnitt 8.5.3 eine Glättung der Basislinie, um die Einflüsse des Trägergases auf unsere Peakerkennung zu minimieren. Als letzte Vorüberlegung stellen wir dann eine Rauschabschätzung im Abschnitt 8.5.4 vor.

Mit diesen Informationen entwickeln wir dann den vollständigen *Peak Detection by Slope Analysis (PDSA)* - Algorithmus 8.5.5. Danach folgt

12 Diese Erkenntnis könnte z. B. in die Wahl von `max_regionsize` berücksichtigt werden.

eine Verbesserung des Verfahrens durch erweiterte Filterung des RIP, damit in dessen Schweif keine weiteren, falschen Peaks mehr erkannt werden. Die Analyse der Performanz des Algorithmus findet sich später im Kapitel der Evaluation 13.

Unser Ziel ist es, einen online-fähigen Algorithmus zu entwickeln, der schnell und speichersparend funktioniert. Daher sind die wichtigen Merkmale, die dieses Verfahren kennzeichnen sollen:

- Verzicht auf rechenintensive Filterverfahren
- Größtmöglicher Verzicht auf Konstanten oder nicht einstellbare Parameter
- Speicherung eines einzigen Spektrums für fortlaufende Berechnungen
- Lauffähigkeit auf allen bekannten Datensätzen ohne veränderte Einstellungen

8.5.1 *Gemeinsame Peak-Merkmale*

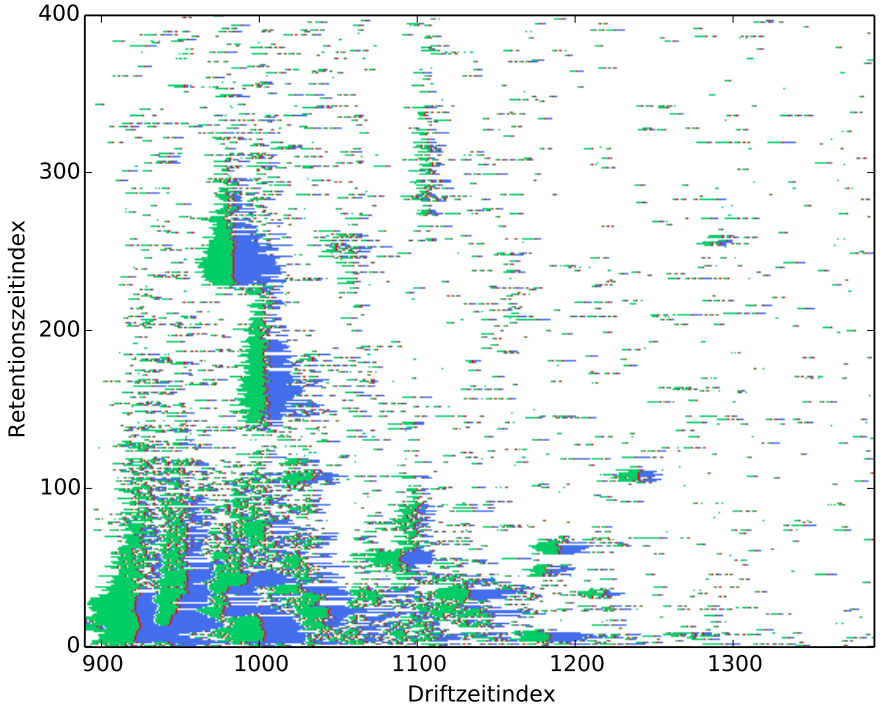
Für uns sind folgende Eigenschaften eines Peaks von Bedeutung:

1. Das Signal steigt über das typische Rauschniveau, welches eine Obergrenze für einen Großteil des normalverteilten Rauschens bildet. Die Berechnung stellen wir später im Abschnitt zur Rauschermittlung 8.5.4 vor.
2. Das Signal steigt für eine gewisse Zeit und fällt daraufhin wieder ab.
3. *Nicht zwingend gegeben*: Zwischen der steigenden und fallenden Peak-Flanke existiert ein Abschnitt, der relativ flach verläuft. Ein *Plateau* (siehe Abschnitt 8.5.2).

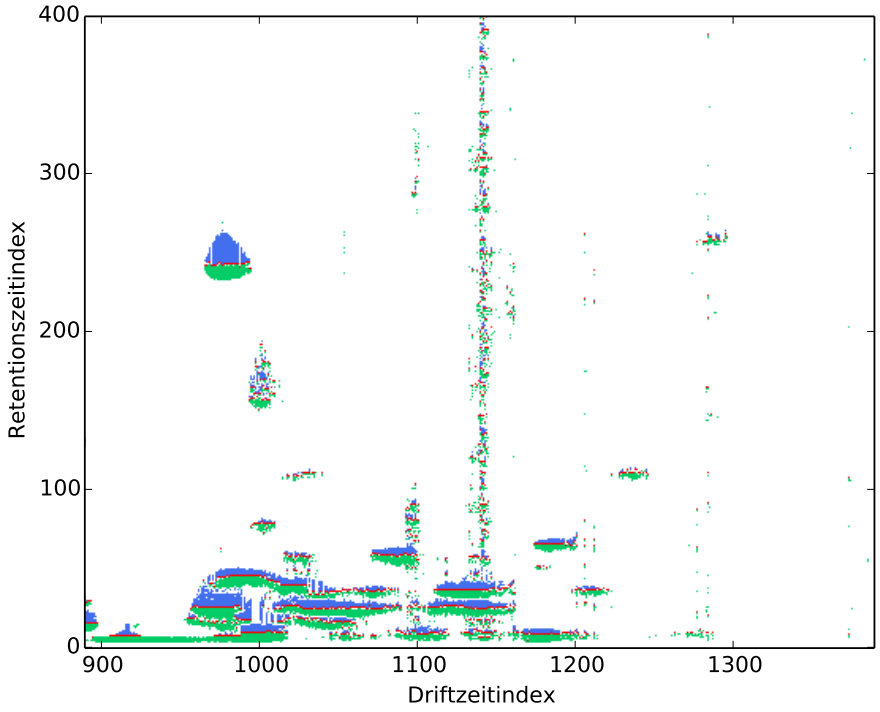
Die Messdaten, die in Abschnitt 13.2.1 unter *Vereinzelte Messdaten* vorgestellt werden, wurden genau auf diese Eigenschaften untersucht. Die ersten Ergebnisse davon basierten auf den Vergleichen zwischen den Datenpunkten, die nur durch die Basislinienkorrektur verändert wurden. Dabei entdeckten wir Plateaus, die im darauffolgenden Abschnitt dargestellt werden.

Nach dieser Feststellung entstand eine frühe Version des PDSA, in welcher die Daten mit Hilfe einer Fenstersumme geglättet werden. Auf die weiteren Details wird im Abschnitt 8.5.5 eingegangen. Abb. 46 illustriert exemplarisch die Ergebnisse. Dabei wurden die gefundenen Bereiche wie folgt farbig markiert:

- Steigende Peakflanken oberhalb des typischen Rauschens (grün)
- Lokale Maxima (rot)
- Fallende Peakflanken oberhalb des typischen Rauschens (blau)
- Bereiche unterhalb des typischen Rauschniveaus (weiß)



(a) Vergleich in Richtung der Driftzeiten



(b) Vergleich in Richtung der Retentionszeiten

Abb. 46: Gegenüberstellung der ansteigenden (grün) und fallenden (blau) Flanken, die über dem typischen Rauschniveau liegen. Lokale Maxima zwischen den Flanken sind rot. Alle anderen Werte unter dem Rauschen sind weiß markiert.

Für Abb. 46a bewegt man sich entlang den Driftzeiten. Es ist ersichtlich, dass der Verlauf erfolgreich die Peaks darstellt. Das Resultat des gleichen Vorgehens in Richtung der Retentionszeiten sieht man in

Abb. 46b. Dabei sind die Peaks weniger deutlich zu sehen und auch noch viel auf der fallenden Flanke des RIP entdeckt worden. Anhand beider Abbildungen sieht man jedoch trotzdem, dass die erwähnten Eigenschaften der Peaks zutreffend sind. Weiterhin fällt zwischen der steigenden und fallenden Flanke stets ein roter Punkt auf, der das lokale Maximum darstellt. Die Abbildungen bekräftigen unser Vorgehen, bei dem man mit einer einfachen Glättung die Struktur der Peaks verdeutlicht. Entsprechend den Darstellungen legen wir bei dem PDSA vor allem mehr Gewicht auf den Verlauf entlang den Driftzeiten.

8.5.2 *Plateaus an Peakhöhepunkten*

Ein *idealer Peak* ist im Bezug auf sein Signal in der Driftzeit durch eine Gaußverteilung genau zu beschreiben. Er besitzt eine steigende Flanke, gefolgt von einem Maximum und einer fallenden Flanke. Die bei der Ionenmobilitätsspektrometrie gemessenen Signalstärken sind jedoch Schwankungen unterworfen. Deshalb verformen sich die erkannten Peaks an ihren Flanken und es gibt nicht immer ein fehlerfrei festzustellendes, lokales Maximum.

Unsere exemplarische Analyse einiger Messdaten zeigt, dass ein Peak im Bereich des höchsten Signals durchaus ein Plateau aufweisen kann. Beim Betrachten solcher Plateaus fällt auf, dass die Änderung des Signalwertes einzelner Datenpunkte im Vergleich zu ihrem Vorgänger sehr gering ausfällt und zwischen einem Anstieg und erneutem Fallen wechseln kann. Solche Steigung von einem Datenpunkt zur Driftzeit t des r-ten Spektrums kann man wie folgt berechnen:

$$\frac{|S_{r,t} - S_{r,t-1}|}{S_{r,t}} .$$

Da die erste Ableitung und damit die Steigung an einem solchen Plateau gegen Null streben, machen Plateaus die Erkennung von den Peakmaxima schwierig. Auch die Datenpunkte, dessen Vorgänger und Nachfolger jeweils höhere Signalstärke aufweisen, machen eine korrekte Erkennung vom Ende des Plateaus und damit den Beginn der fallenden Peakflanke schwierig. Der höchste Punkt des Peaks kann nicht genau festgelegt werden, da ohne einer Glättung mehrere lokale Maxima entstehen können und somit das automatische Finden des Peakmaximums schwierig wird. Dieser Problematik erschwert die Überprüfung der Vollständigkeit eines Peak.

In Abb. 47 sind Beispiele von Peaks und ihren Signalstärken abhängig von der Driftzeit zu finden. Das von uns gewählte Kriterium für die automatische Erkennung eines Plateau ist hier, dass die Steigung im Vergleich zum vorangegangenen Datenpunkt des Spektrums nicht mehr als 2 % betragen darf und dieses Verhalten für mindestens fünf aufeinander folgende Datenpunkte anhält. An den Abbildungen erkennt man in oranger Farbe markierte Bereiche, welche diesem Kriterium entsprechen. In diesen Bereichen sind sowohl mehrere lokale Maxima als auch das erneute, schnellere Steigen oder Fallen der Intensitäten nach dem Plateau gegeben.

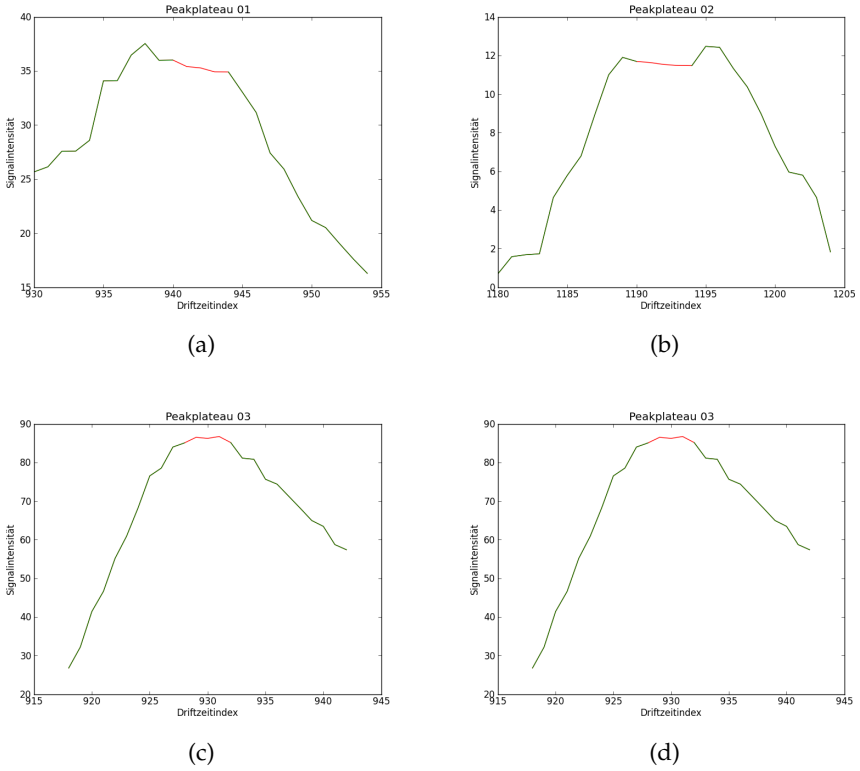


Abb. 47: Plateaus mit nicht mehr als 2 % Steigung in einem Peak

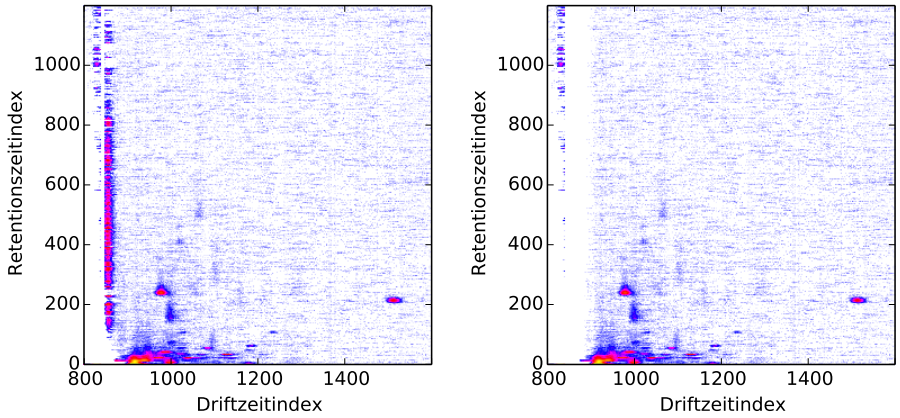
8.5.3 *Glättung der Basislinie*

Im Abschnitt 6 wurden die Messdaten analysiert und wir haben gesehen, dass man auch in einem online Fall den Mittelpunkt des RIP berechnen sowie die Basislinienkorrektur durchführen kann. Jedoch können nach der Basislinienkorrektur noch Einflüsse des Trägergases in der Messung verbleiben. Diese nehmen ab, je weiter sich die aktuelle Driftzeit von der Driftzeit des RIP-Mittelpunktes entfernt. Diese Überreste des Trägergases werden in unmittelbarer Nähe zum RIP fälschlicherweise als Peaks erkannt. Wir entscheiden uns daher dazu, die Werte der Basislinie in unmittelbar auf den RIP folgenden Driftzeiten zu modifizieren.

Sei d_t der Wert der Basisline zum Zeitpunkt t. Den Startwert von t setzen wir auf die Driftzeit des RIP-Mittelpunktes t^*. Dann verändern wir die Basislinie wie folgt:

```
int t = t_star;
while (d[t] > d[t+1]) {
    d[t+1] = 0.5 * d[t+1] + 0.5 * d[t];
}
```

Das Ende der Schleife tritt genau dann ein, wenn der RIP zu Ende ist und die Werte wieder mehr von der zufälligen Verteilung des Rauschens beeinflusst werden. Ab diesem Zeitpunkt müssen wir den RIP nicht weiter glätten. Dieses Vorgehen sorgt dafür, dass mit Abzug der Basislinie höhere Intensitäten entfernt werden, aber auch der Ausschlag des Trägergases fast vollständig gefiltert wird. Der Vergleich der normalen und der modifizierten Basislinienkorrektur wird in Abb. 48 illustriert. Die ge-



(a) Normale Basisliniekorrektur (b) Modifizierte Basisliniekorrektur

Abb. 48: Ergebnis der normalen und der modifizierten Basislinienkorrektur

glätte Basislinie schafft mehr vom RIP zu entfernen, falls es dort etwas vom Trägergas übrig gibt.

Ein Problem kann lediglich auftreten, wenn in unmittelbarer Nähe zum RIP schon erste Peaks durch die Glättung beeinflusst werden. Dort kann die Struktur so verändert werden, dass unser Algorithmus einen Peak in zwei Teile aufspaltet. Diese Teile werden aber in der Nachbearbeitung wieder zusammengefügt, was später noch erklärt wird.

8.5.4 *Approximative Rauschermittlung*

Um das Niveau des Rauschens zu approximieren, benötigen wir die Varianz $\mathrm{Var}(X) = E[X^2] - E[X]^2$ der gegebenen Daten.[13] In einem resourcen-beschränkten Online-Szenario möchte man jedoch so wenig Daten wie möglich im Speicher halten. Darum erprobten wir den von uns entwickelten Ansatz der *gleitenden Berechnung der Varianz*. Der Vorteil des Verfahrens ist es, dass man pro Spektrum den Erwartungswert und die Varianzen der einzelnen Werte speichert. Nachdem die Varianz für das Spektrum errechnet ist, können die einzelnen Varianzen der Werte verworfen werden.

GLEITENDE BERECHNUNG DER VARIANZ Gehen wir davon aus, dass uns zur Vorverarbeitung n Spektren mit t Werten zur Verfügung stehen. Für jedes Spektrum i berechnen wir den Mittelwert

$$\mu_i = \frac{1}{t} \sum_{j=1}^{t} x_{i,j} \ .$$

Ab dem Retentionszeitindex $i > 1$ berechnen wir zusätzlich

$$\sigma_{i,j}^2 = (x_{i,j} - \mu_{i-1})^2$$

13 Die Berechnung des Rauschniveaus über die Beziehung $\mathrm{Var}(X) = E[X^2] - E[X]^2$ ist möglich, aber numerisch an sich sehr instabil. Deshalb hat D. Knuth eine bessere Methode gefunden: Siehe http://www.johndcook.com/standard_deviation.html. Diesen Hinweis fanden wir erst gegen Ende der Projektzeit und blieben daher bei unserem Verfahren.

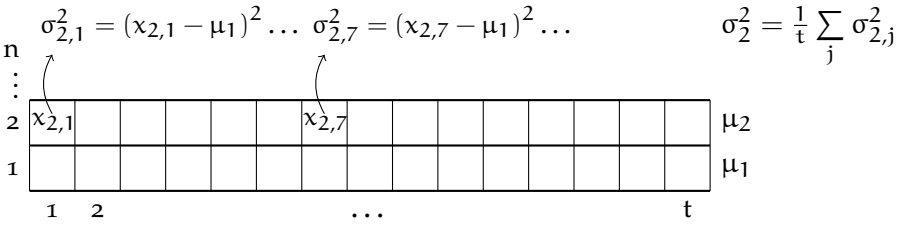


Abb. 49: Beispiel der Abschätzung von σ in einem Online-Szenario

mit $1 \leqslant j \leqslant t-1$. Da uns zu diesem Zeitpunkt der Erwartungswert des aktuellen Spektrums nicht zur Verfügung steht, benutzen wir den schon berechneten Erwartungswert μ_{i-1} des zuvor ausgewerteten Spektrums. Die durchschnittliche quadratische Abweichung pro Messpunkt im Spektrum i ergibt sich dann als

$$\sigma_i^2 = \frac{1}{t} \sum_{j=1}^{t} \sigma_{i,j}^2 \ .$$

Abb. 49 illustriert, wie die die Abschätzung von σ_i spektrenweise verläuft. Sobald alle n Spektren verarbeitet sind, bilden wir entsprechend die Mittelwerte

$$\mu = \frac{1}{n} \sum_{i=1}^{n} \mu_i \qquad\qquad \sigma^2 = \frac{1}{n-1} \sum_{i=2}^{n} \sigma_i^2$$

und berechnen anschließend die Abschätzung der Standardabweichung σ mit

$$\sigma = \sqrt{\sigma^2} \ .$$

Zur Beurteilung der Güte der beschriebenen Abschätzung vergleichen wir die resultierenden Werte für alle uns vorliegenden Datensätze mit den Werten der tatsächlichen und offline berechneten Standardabweichung über alle Spektren der Vorverarbeitung. Somit kann man den auftretenden Fehler berechnen. Wie man der Tabelle 4 entnehmen kann, weicht unsere Abschätzung um 1.16% bei den Tee-Kaffee-Daten und um 0.49% bei den Salt-Pepper-Daten von der Wahrheit ab. Eine allgemeine Aussage über die Güte der Abschätzung kann von uns nicht getroffen werden. Der berechnete, geringe Fehler auf den uns zur Verfügung stehenden Daten zeigt jedoch, dass die Standardabweichung durch unser Verfahren ausreichend approximiert wird.

ABSCHÄTZEN DES RAUSCHNIVEAUS Mit den gefundenen μ und σ lässt sich nun das typische Rauschen abschätzen. Für diese Nachweisgrenze gibt es verschiedene folgende Berechnungsansätze:

$$\begin{aligned} T_1 &= \mu + \frac{3\sigma}{2} \\ T_2 &= \mu + 3\sigma \\ T_3 &= \mu + 2\sigma \end{aligned}$$

Verwendet man die von B & S Analytik vorgeschlagene Formel T_1 als Berechnungsgrundlage, so verbleiben noch Punkte der Messung, an denen das Rauschen nicht entfernt wird. T_2 entstammt der DIN 32645,

Daten	Abweichung Minimal	Maximal	Durchschnitt
Tee-Kaffee-Daten (69 Messungen):	0.63%	1.77%	1.16%
Salt-Pepper-Daten (65 Messungen):	0.01%	2.13%	0.49%

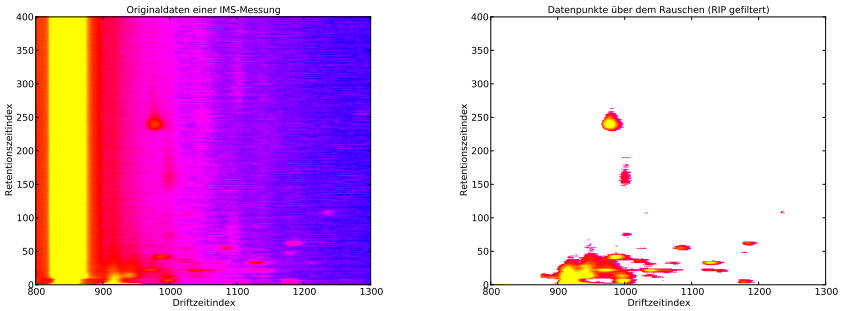
Tabelle 4: Abweichung der online berechneten Standardabweichung von der tatsächlichen Standardabweichung

eliminiert jedoch auch die Ränder einiger Peaks. Bei manchen Messungen liegen dann schwache Peaks unterhalb des errechneten Rauschwertes, so dass sie nicht mehr detektierbar sind. T_3 ist ein weniger strikter Ansatz als T_2, welcher trotzdem noch $95,4\%$ des Rauschens im Standardfall entfernt. Dieser Kompromiss erlaubt eine Erkennung nahe am Rauschen liegender Peaks zum Preis einer leicht erhöhten Fehlerquote.

8.5.5 *Das PDSA-Verfahren*

Nun besitzen wir alle erforderlichen Kenntnisse, um Peaks anhand ihrer eingangs genannten Eigenschaften (Abschnitt 8.5.1) detektieren zu können. Hierzu müssen wir die Signalintensitäten eines Spektrums bei steigenden Driftzeiten miteinander vergleichen. Während der Untersuchung der generellen Peakmerkmale in Abschnitt 8.5.1 und der Peakplateaus in Abschnitt 8.5.2 haben wir gesehen, dass die Vergleiche einzelner Datenpunkte zur Erkennung instabil sind. Wir entscheiden uns also dafür, nicht einen einzelnen Signalwert zur Betrachtung heran zu ziehen, sondern ein Fenster, welches w Datenpunkte umfasst. In diesem Fenster berechnen wir die Summe der Signalintensitäten und verschieben es im weiteren Verlauf Schritt für Schritt über das ganze Spektrum. Dies hat den Effekt, dass lokale Ausreißer innerhalb eines Fensters mit minimalem Aufwand geglättet werden. So stellen im Rauschen befindliche oder teilweise schwankende Werte keine Beeinflussung mehr dar.

Wir implementieren dieses Verfahren mit einem Fenster der Größe $w = 10$ auf Daten mit reduzierter Auflösung. Der Grund für diese Festlegung ist die maximal mögliche Breite eines Peaks. Sie beträgt 15 Datenpunkte und kann mithilfe der Formel (5) aus dem Abschnitt 2.1.1 zu den pyhsikalischen Grundlagen des MCC/IMS-Verfahrens berechnet werden. Wählt man allerdings $w = 15$, so werden die Wechsel von steigenden zu fallenden Flanken nicht mehr zuverlässig erkannt. Des Weiteren werden manche Peaks nicht mehr detektiert, da ihre Ausmaße aufgrund von Messfehlern kleiner als die angegeben 15 Datenpunkte sind. Ein Fenster der Größe $w = 5$ liefert gute Ergebnisse, ist aber nicht so robust gegen Rauscheinflüsse. Wir haben uns daher für einen Kompromiss aus guter Erkennung und Robustheit entschieden und $w = 10$ gewählt. Zu beachten ist natürlich, dass alle benutzten und über die Messung geschobenen Berechnungsfenster in einem Durchlauf die gleiche Größe besitzen müssen.



(a) (b)

Abb. 50: Vergleich zwischen der Originalmessung in (a) und den Teilen der IMS-Messung, die über dem ermittelten Rauschfenster in (b) liegen.

RAUSCHFENSTER Wie im Abschnitt zur Rauschermittlung 8.5.4 gezeigt, können wir während der Vorverarbeitung das typische Rauschen n_t approximieren. Für eine bestimmte Fenstergröße lässt sich das Rauschfenster durch das Multiplizieren der entsprechenden Werte errechnen:

$$n_w = n_t \cdot w \ .$$

Abbildung 50a und 50b zeigen einen Ausschnitt einer Originalmessung inklusive Hintergrundrauschen und zum Vergleich die Signalintensitäten, die über dem errechneten Rauschfenster liegen. Hierbei wurde ebenfalls der RIP mittels Basislinienkorrektur größtenteils entfernt.

SUCHE NACH PEAKS PRO SPEKTRUM In einem einzelnen Spektrum suchen wir nach einer Folge von Werten oberhalb des ermittelten Rauschfensters, welche die bereits erläuterten Anforderungen an einen Peak erfüllen soll. Im eindimensionalen Raum bezeichnen wir im Folgenden einen Peak als „Peakstreifen“ oder einfach „Streifen“. Bei der Suche werden Signalwerte betrachtet, von denen bereits die in der Vorverarbeitung berechnete Basislinie subtrahiert wurde. Wir iterieren über ein ankommendes Spektrum und vergleichen die errechneten Fenstersummen entlang steigender Driftzeiten miteinander. Den beispielhaften Verlauf kann man der Abbildung 51 entnehmen. Darin sieht man eine fiktive Wertereihe, die von unserem Verfahren nach Peaks untersucht wird. Das linke Fenster liegt unter dem angegeben Rauschniveau von 15, während das grüne Fenster in der Mitte dieses deutlich übersteigt und dabei im Vergleich zur vorherigen Fenstersumme trotz einzelner Schwankungen steigt. In der rechten Hälfte sieht man das Fenster an der fallenden Flanke.

Der aktuelle Wert der Fenstersumme v_{w_i} mit dem Startpunkt an der Position $0 < i < t - w$ hat das Rauschniveau überschritten, wenn gilt:

$$v_{w_i} = \sum_{j=0}^{w} S_{cur+j} > n_w \tag{25}$$

wobei cur für den aktuellen Startindex des Fensters steht. An dieser Position startet der potentielle Peak. Wir speichern den Driftzeitindex

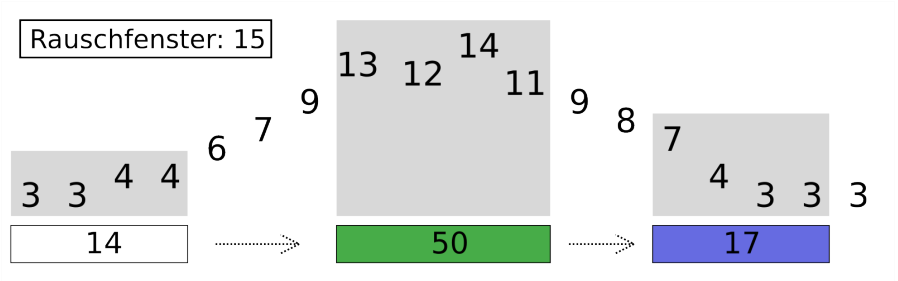


Abb. 51: Sliding Window bei dem PDSA. Das linke Fenster liegt unter dem Rauschniveau. Das grüne Fenster in der Mitte ist noch trotz kleinerer Schwankungen steigend im Vergleich zum vorherigen. Das rechte blaue Fenster gehört zur fallenden Flanke.

t als Startwert des Peaks. Gleichzeitig merken wir uns das im aktuellen Fenster vorhandene lokale Maximum und aktualisieren es fortlaufend. Für die weiterhin steigende Flanke soll gelten:

$$v_{w_i} < v_{w_{i+1}} \tag{26}$$

Sobald v_w nicht mehr steigt, sondern zu fallen beginnt, ist der höchste Punkt eines Peaks überschritten worden. Zu diesem Zeitpunkt speichern wir uns den Wert der letzten Fenstersumme als Maximum des aktuellen „Peakstreifens“. Dieser Wert dient später als Vergleichskriterium bei dem Zusammenfügen der einzelnen Streifen zu einem Peak. Zusätzlich speichern wir den Drifzeit- und Retenionsindex des Mittelpunktes des maximalen Peakstreifens. Zu beachten ist hier, dass bei $w = 10$ kein „echter Mittelpunkt“ existiert. Da wir diesen Index aber bei jedem Peakstreifen gleich setzen, ist es unerheblich, ob wir bei der Mittelpunktberechnung auf- oder abrunden, solange wir konsistent bleiben und Peakstreifen sich anhand dieses Kriteriums vergleichen lassen. Ab dem erkannten Maximum folgt man der fallenden Flanke, wobei ein direktes Abfallen ins Rauschen nicht als fallende Flanke gewertet. In solch einem Fall wird dieser Streifen nicht als vollständig angesehen, weil er nicht steigt, ein Maximum erreicht und dann wieder fällt. Somit ergibt sich folgende Bedingung:

$$v_{w_i} > v_{w_{i+1}} \geqslant n_w \quad ,$$

wobei $0 < i < t - w$ gilt. In diesem zweiten Teil der Erkennung dürfen die Fenstersummen im Vergleich zu ihren Vorgängern nur noch fallen. Zu beachten ist, dass der Streifen in zwei Fällen zu Ende sein kann:

- Die Fenstersumme v_{w_i} steigt wieder. Dies stellt den Anfang eines zweiten Peaks dar, welcher in der Driftzeit unmittelbar auf den ersten folgt. Auch überlappende Peaks können damit auseinander gehalten werden, da es wieder zum Anstieg laut der Gleichung (26) kommt. In solch einem Fall wird ab diesem Zeitpunkt der neue, zweite Streifen verfolgt.

- Der Fensterwert v_{w_i} wird kleiner als der Wert des Rauschfensters n_w. Die Intensitäten sinken unter das Rauschfenster und damit unter die Nachweisgrenze.

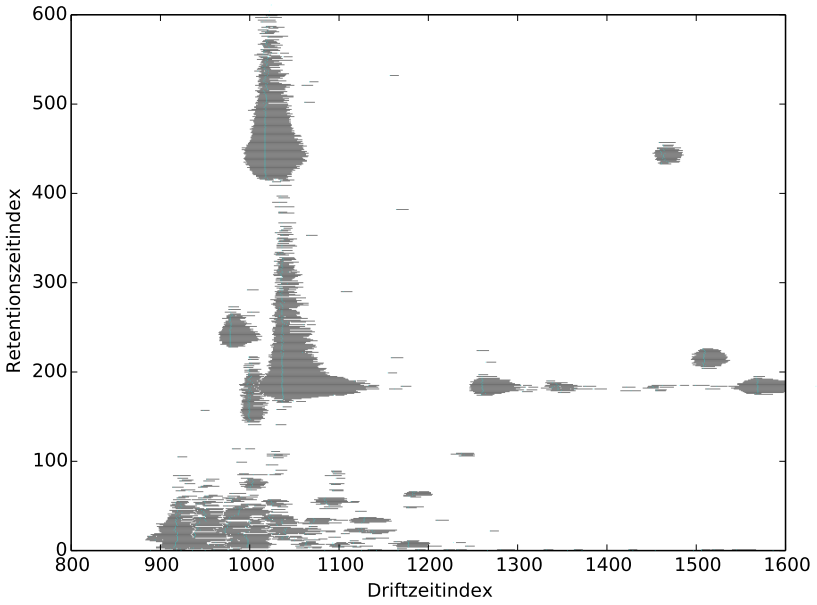


Abb. 52: Darstellung der vom PDSA entdeckten Peakstreifen

In beiden Fällen ist der Peakstreifen im aktuellen Spektrum vollständig erkannt, sein Endpunkt festgehalten und der Peakstreifen der Liste unfertiger Peaks hinzugefügt worden. Für diesen werden Start- und Endpunkt, das lokale Maximum und dessen Driftzeitindex, der zugehörige Retentionszeitpunkt und die maximale Fenstersumme gespeichert. Sollte der Peak nicht alle zuvor genannten Anforderungen erfüllen, so wird der abgearbeitete Peakstreifen verworfen. In Abb. 52 sind die Peakstreifen dargestellt, die gefunden und gemerkt wurden. Die Zusammensetzung dieser Streifen zu den Peaks wird im folgenden beschrieben.

VERGLEICH GESPEICHERTER PEAKSTREIFEN Zum Behandeln einzelner Streifen vergleichen wir die Positionen der zugehörigen Maxima. Es geht vor allem darum, einen neu gefundenen Streifen in die Liste potentieller Peaks einzuordnen. Abbildung 53 zeigt mögliche Konstellationen nach dem Vergleich der Maxima der Streifen. Das sind insgesamt vier folgende Fälle:

1. Zusammenführen aktuell untersuchter zu bereits abgespeicherten Peakstreifen
2. Beenden eines Peaks
3. Anlegen eines neuen Peaks
4. Entfernung einzelner Streifen

Anlegen des neuen Peaks mit einem einzelnen Streifen kann entweder dazu führen, dass dieser Streifen sich später als kein Peak erweist und entfernt wird, oder er wird mit weiteren Streifen zusammengeführt und irgendwann beendet. Somit interessieren uns vor allem die ersten zwei Fälle. Es soll überprüft werden, ob der aktuelle „Peakstreifen“ einem

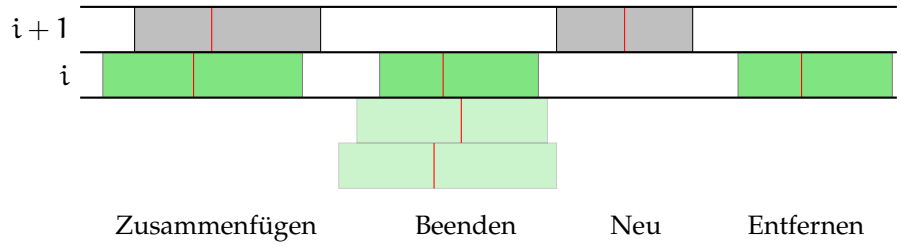


Abb. 53: Vergleich von Peakstreifen, wobei es vier unterschiedliche Fälle gibt: Zusammenführen von Streifen, Beenden eines Peaks, Anlegen eines neuen Peaks und Entfernung einzelner Streifen.

schon erkannten Peak zugeordnet werden kann, welcher in dem vorhergehenden Spektrum erkannt worden ist. Als Vergleichkriterium nutzen wir die Bedingungen von Hauschild u. a. [23, Abschnitt 2.2], laut welchen zwei Peaks zu einem Peak zusammengefasst werden können, wenn sie bezogen auf Positionen der Peakmaxima erfüllt sind:

- $|t(P_1) - t(P_2)| < 0{,}003\,Vs/cm^2$, was einer Abweichung in der Driftzeit entspricht;
- $|r(P_1) - r(P_2)| < (3\,s + 0{,}1 \cdot r(P_1))$, wobei $s(P_1) > s(P_2)$ gilt und somit entspricht es einer Abweichung der Peakzentren von P_1 und P_2 in der Retentionszeit.

Vorerst sind wir nur an dem Vergleich in der Driftzeit interessiert, da einzelne Streifen, die beim nächsten Spektrum nicht mit einem weiteren Streifen zum Peak zusammengesetzt werden, direkt entfernt werden.

Beim Fortführen des Peaks wird das Maximum auf den größeren Wert der beiden Maxima gesetzt und seine Position entsprechend aktualisiert. Die Grenzen des Peaks in Retentions- und Driftzeit werden ebenfalls angepasst. Der Startpunkt in der Retentionszeit verändert sich nicht, aber der Endpunkt wird auf die Retentionszeit des aktuellen Spektrums gesetzt, in welcher der hinzugefügte Peakstreifen gefunden wurde. In der Driftzeit überprüfen wir die Start- und Endpunkte beider Teilpeaks und bilden jeweils das arithmetische Mittel. Wenn weitere Peakstreifen zu einem bestehenden Peak hinzugefügt werden, aktualisieren wir diese Durchschnittsberechnung entsprechend. So erhalten wir am Ende der Berechnung für jeden Peak eine Breitenangabe. Die Größe dieser Region in der Retentionszeit ergibt sich durch Subtraktion der Zeit des Startspektrums von der Zeit des Endspektrums. Diese Regionen können der groben Abschätzung der Größe eines Peaks dienen, werden aber von uns zur Zeit nur für visuelle Zwecke benutzt, da auch die spätere Klassifikation keine Optionen zur Verwendung dieser Daten unterstützt.

Als Ergebnis erhält man am Ende einen Peak, der anhand der einzelnen Streifen gefunden worden ist. Ein Beispiel dafür ist in Abbildung 54a zu sehen. Mit dem PDSA werden aber auch solche Peaks wie in Abbildung 54b als getrennt erkannt, die nah aneinander liegen und zwischen denen es keine Areale gibt, die unter dem Rauschniveau liegen. Während der Entwicklung des Verfahrens bemerkten wir Schwankungen in manchen Spektren, die z.B. durch weitere starke Peaks hervorgeru-

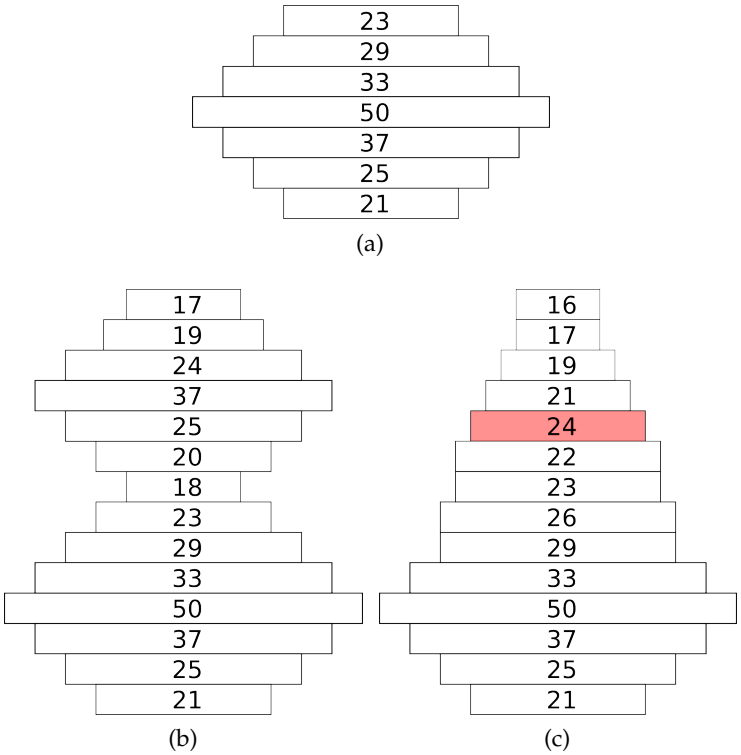


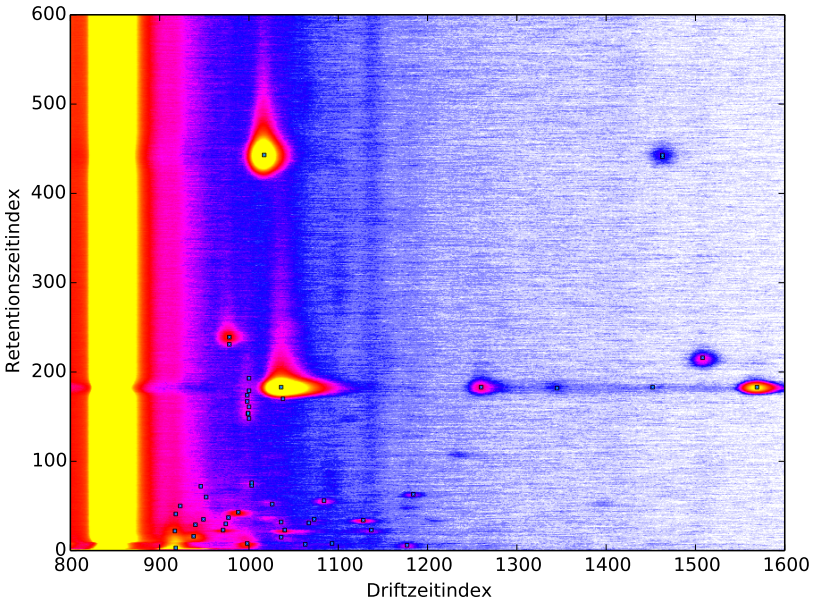
Abb. 54: (a) Zusammensetzungen von Streifen zu einem Peak. (b) Ein Peak geht in den zweiten über. (c) Toleranz in dem Schweif eines größeren Peaks.

fen wurden, so dass es zu Einschnürung anderer Peaks kam. Auch Rauscheinflüsse rufen bei schwachen Peakstreifen, die nur knapp über dem Rauschen liegen, diesen Effekt hervor. Daraus entstanden manchmal solche Fälle wie in Abbildung 54c dargestellt. Es kommt also zu einem kurzen Aufstieg der Werte im Schweif eines Peaks, während danach die in der Retentionszeit fallenden Flanke geregelt fortläuft. Um das Aufspalten eines solchen Peaks in zwei oder mehrere einzelne zu vermeiden, führten wir eine Toleranzgrenze von 10% der Peakgröße in der Retentionszeit ein. So darf ein Peak der Größe 23 bis zu zweimal ansteigen, während er sich in der fallenden Flanke befindet.

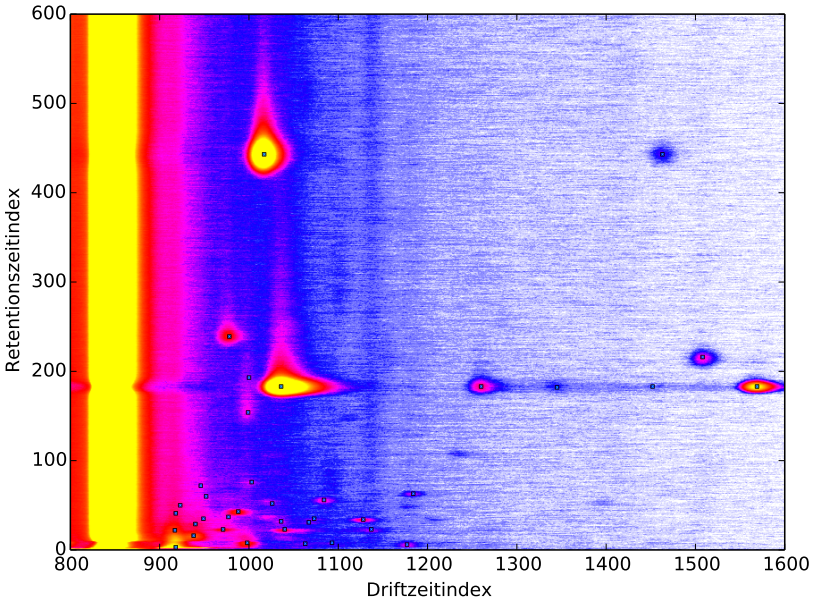
AUSGABE Vor der Ausgabe der Peaks nach der verarbeiteten Messung werden die gefundenen Peaks mit dem Kriterium, welches wir schon zum Vergleich der Peakstreifen genutzt haben, nach möglichem Zusammenführen von Peaks untersucht. Abbildung 55 soll die Wirkung dieses letzten Schrittes zeigen. Daraufhin werden alle Peaks ausgegeben.

MÖGLICHE VERBESSERUNGEN Wir sehen an vielen Stellen des PDSA noch Optimierungspotential. Im Einzelnen sind das folgende Punkte:

- Zur Zeit werden die mitberechneten Regionen der einzelnen Peaks nicht zur Klassifikation benutzt. Diese zusätzlichen Informationen könnten ebenfalls zur Abschätzung dienen, ob zwei irrtümlich als unterschiedlich erkannte Peaks zusammengehörig sind.



(a) Ohne Merge-Schritt



(b) Nach dem Merge-Schritt

Abb. 55: Vergleich zwischen allen gefundenen Peaks in (a) und der endgültigen Ausgabe nach dem Zusammenlegen einiger Peaks nach Bedingung von Hauschild u. a. [23, Abschnitt 2.2] in (b)

- Ebenfalls verbessern könnte man die Datenstruktur, in welcher die Peakstreifen abgelegt werden. Hier haben wir bis jetzt aufgrund der Schnelligkeit des Verfahrens keine Arbeit investiert, sehen aber auch noch deutliches Potential.

- Wie schon erwähnt, lässt sich eine Rauschermittlung implementieren, die ohne einen prozentual kleinen Fehler auskommt. Dies würde die Genauigkeit der Erkennung sicher verbessern.
- Die Fehlertoleranz für Peakschweife in Retentionszeit beträgt im Moment 10%. Diesen von uns festgelegten Wert könnte man genauer untersuchen und gegebenenfalls optimieren oder sogar automatisch bestimmen lassen.

AUSWERTUNG Eine detaillierte Auswertung des Verfahrens befindet sich im Kapitel der Evaluation 13. Wir stellen hier nur kurz wesentliche Vor- und Nachteile vor. Vorteile des Verfahrens:

- Gute Laufzeit durch wenige Vergleiche und Rechenoperationen
- Gute Abgrenzung und Erkennung potentiellen Peaks durch Vorberechnung des Rauschfensters in vorheriger Raumluftmessung
- Keine Erkennung des RIPs als Peak, da dieser bis zu seiner Mitte sicher abgeschnitten wird. Die eventuell noch vorhandene, fallende Flanke wird nicht als eigenständiger Peak detektiert.
- Lauffähigkeit auf unterschiedlichen Messreihen, deren Rauschen und Signalintensitäten teilweise stark voneinander abweichen. In manchen Messreihen liegen Peaks näher am Rauschen oder werden in ihrer Form stark verzerrt.

Zu den Nachteilen des Verfahrens kann man Folgendes nennen:

- Die Fenster um die Grenzen der gefundenen Peaks herum sind dank der verwendeten Durchschnittsberechnung nur eine grobe Abschätzung. Die Mittelpunkte sind hiervon allerdings unbeeinflusst.
- Eventuelle Aufspaltung eines Peaks in zwei einzelne, kleinere Peaks. Dies könnte bei Messfehlern der Signalstärken innerhalb eines Peaks entstehen, die groß genug sind, um eine fallende Flanke darzustellen. Die darauf folgende, wieder steigende Flanke wird als neuer Peak erkannt werden.

Zum Abschluss wollen wir noch kurz beispielhaft den PDSA auf übersteuerten Messungen betrachten. Diese werden bei der Auswertung von Hand meist verworfen. Abbildung 56 zeigt solch eine Messung. Auch hier erkennt unser automatisches Verfahren noch Peaks, die zwar recht zahlreich, aber nicht wahllos gesetzt werden und auch unseren eingangs definierten Merkmalen genügen.

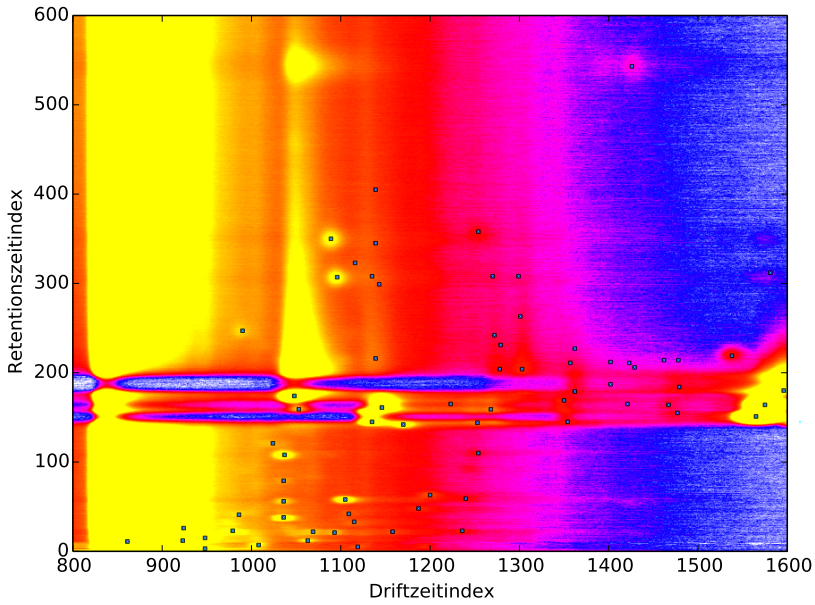


Abb. 56: Gefundene Peaks in einer übersteuerten und eigentlich unbrauchbaren Messung

9 LAUFZEITOPTIMIERUNG

In diesem Kapitel werden die Optimierungsmöglichkeiten auf der Hardware des Raspberry Pi beschrieben. Vorher ermitteln wir, welches Potential der Raspberry Pi hat und was es aus hardwaretechnischer Sicht zu beachten gibt, um tatsächlich die Daten (ggf. auch in der hohen Auflösung) in Echtzeit verarbeiten zu können. Darum werden im Nachfolgenden zuerst im Abschnitt 9.1 verschiedene Benchmarks präsentiert. Danach stellen wir in Abschnitt 9.2 die Ergebnisse der Untersuchungen diverser Caching-Effekte vor. Wie gut das Übertakten der CPU und die Programmierung auf der GPU funktionieren, wird entsprechend in den Abschnitten 9.3 und 9.4 behandelt.

9.1 BENCHMARKS

Zunächst haben wir Benchmarks auf dem Raspberry Pi ausgeführt, um die Laufzeit für verschiedene Operationen zu ermitteln. Dazu haben wir kurze Programme in C geschrieben, in denen die zu überprüfende Operation mehrfach ausgeführt wird, und ihre Laufzeit gemessen. Wir beschreiben in diesem Abschnitt die wichtigsten Benchmarks, den Vergleich vom Rechnen mit Integern und Floats, sowie die Netzwerkperformanz des Pi.

9.1.1 *Integer vs. Float*

Ein wichtiger Vergleichspunkt für die Implementierung der Algorithmen ist die Laufzeit von einfachen Operationen wie Addition und Multiplikation mit verschiedenen Datentypen.

Die wesentliche Unterscheidung wird dabei zwischen Ganzzahlen und Gleitkommazahlen vorgenommen. Während das Rechnen mit Floats aufwendiger ist, müssten für ausreichende Präzision beim Rechnen mit Integern dafür die ursprünglichen Werte mit einem ausreichend hohen Faktor skaliert werden. Bei allen betrachteten Datentypen haben wir die

	Operation		
	Addition	Multiplikation	Skalierung
`short`	12 ns	18 ns	47 ns
`int/long`	9 ns	15 ns	41 ns
`long long`	12 ns	125 ns	205 ns
`float`	27 ns	29 ns	-
`double`	27 ns	30 ns	-

Tabelle 5: Laufzeiten für den Benchmark nach Datentyp und Operation. `int` und `long` sind auf 32Bit-Systemen äquivalent

Zeit für einfache Addition und Multiplikation gemessen. Bei ganzzahligen Datentypen haben wir zusätzlich die benötigte Zeit gemessen, eine Gleitkommazahl mit einem hohen Skalierungsfaktor zu multiplizieren und den sich ergebenden Wert zum gewünschten Datentyp umzuwandeln.

Um die Zeit für die einzelnen Operationen ausreichend genau zu messen, wurden die verschiedenen Operationen jeweils eine Million Mal ausgeführt und zusätzlich die vorher gemessene Zeit für den reinen Schleifendurchlauf abgezogen. Um sicherzustellen, dass jede Operation einzeln ausgeführt wird und nicht durch Optimierung mehrere Operationen zusammengefasst werden, wurde der Benchmark ohne Optimierungen, d. h. mit `gcc`-Parameter `-O0`, kompiliert.

Die Tabelle 5 zeigt die Ergebnisse des Benchmarks. Wird der wesentlich langsamere Datentyp `long long` außer Acht gelassen, so ist für ganzzahlige Datentypen Addition etwa dreimal so schnell wie für Gleitkommazahlen und Multiplikation doppelt so schnell. Werden mehr als 5 Operationen auf einem einzelnen Datenpunkt ausgeführt, so kann der Zeitverlust für das Skalieren auf ganzzahlige Werte ausgeglichen werden und die Gesamtzeit für die Operationen ist mit ganzzahligen Datentypen schneller. Je mehr Operationen für jeden einzelnen Datenpunkt ausgeführt werden, desto besser ist die Laufzeit mit ganzen Zahlen im Vergleich zu Gleitkommazahlen.

Eine weitere interessante Beobachtung ist der geringe Unterschied der Laufzeit zwischen den Datentypen `float` und `double`. Es kann somit der Datentyp mit der größeren Präzision verwendet werden, ohne bemerkbare Unterschiede in der Laufzeit zu erhalten. Das wesentliche Ergebnis ist jedoch, dass durch die Verwendung von ganzzahligen Datentypen die Laufzeit bei ausreichend vielen Operationen verbessert werden kann.

9.1.2 *Netzwerkperformanz*

Wie in Kapitel 5 beschrieben, erfolgt die Übertragung der Messdaten an den Pi über das Netzwerk. Um sofort mit der Bearbeitung der Daten beginnen zu können, wurde anstatt auf ein gesamtes Spektrum zu warten zuerst das Empfangen einzelner Datenpunkte in Betracht gezogen. Dazu ist jedoch ein `recv`-Funktionsaufruf pro Datenpunkt notwendig, was mit erheblichem Overhead verbunden ist und zu 100% CPU-Nutzung während des Empfangens führt. Durch die Architektur mit mehreren Threads sollte es während des Wartens auf das Netzwerk möglich sein, schon empfangene Daten bearbeiten zu können. Da die CPU des Pi nur einen Kern besitzt, welcher so schon voll ausgelastet wird, ist dies mit diesem Ansatz nicht der Fall.

Zum Testen wurden Daten aus den uns zur Verfügung stehenden CSV-Dateien verwendet. Diese enthalten nur Datenpunkte aus den ersten 50 ms eines Spektrums, da in der verbleibenden Zeit laut Aussage von B & S Analytik keine nützlichen Daten mehr zu erwarten sind.

Das Empfangen eines Spektrums in geringer Auflösung mit 2500 Datenpunkten dauert so ca. 10 ms. Bei voller Auflösung mit 12500 Daten-

punkten wird mit ca. 50 ms schon die Hälfte der zur Verfügung stehenden Zeit in Anspruch genommen.

Bei einer echten Messung an einem IMS-Gerät würden Datenpunkte für die vollen 100 ms gesendet werden, womit die gesamte zur Verfügung stehende Zeit zur Bearbeitung eines Spektrums für dessen Empfang verwendet werden würde.

Wird stattdessen ein gesamtes Spektrum auf einmal empfangen, reduziert sich die benötigte Zeit auf unter 1 ms bei reduzierter Auflösung bzw. ungefähr 4 ms bei hoher Auflösung, wobei die CPU nur teilweise ausgelastet ist und noch für andere Aufgaben zur Verfügung steht. Werden Spektren so schnell wie möglich empfangen ohne diese zu verarbeiten, kann mit ca. 12 MiB/s die Netzwerkverbindung voll ausgelastet werden. Da ein vollständiges Spektrum mit 25000 Datenpunkten und 2 Byte pro Datenpunkt alle 100 ms übertragen werden muss, reicht ein Datendurchsatz von 500 kB/s für uns aus, somit stellt die Datenübertragung über das Netzwerk für uns kein Problem dar.

Der uns von B & S Analytik in Aussicht gestellte Server hätte auch gesamte Spektren auf einmal verschickt. Deshalb wurde das Empfangen einzelner Datenpunkte nicht weiter verfolgt.

9.2 CACHE-EFFEKTE

Neben der reinen Rechenzeit für die eigentlichen Operationen beeinflussen auch Speicherzugriffe entscheidend die Laufzeit. Bei größeren Programmen, bei denen die Laufzeit keinen kritischen Faktor darstellt, werden Speicherzugriffe meist erst gesondert betrachtet, wenn die zu bearbeitenden Daten nicht in den Arbeitsspeicher passen. Da wir aber auf dem Pi innerhalb von 100 ms ein Spektrum bearbeiten wollen, mussten wir hier alle möglichen Faktoren der Laufzeit behandeln.

9.2.1 *Informationen zu den Caches*

Wie in Kapitel 3 beschrieben, besitzt der Raspberry Pi einen L1- und einen L2-Cache. Dabei besitzt die CPU einen eigenen L1-Cache, während der L2-Cache zwischen GPU und CPU geteilt wird. Der L1-Cache hat eine Größe von 16 kiB, der L2-Cache eine Größe von 128 kiB. Ein Zugriff auf den L1-Cache ist am schnellsten, auf den L2-Cache ein wenig langsamer, ein Zugriff auf den Arbeitsspeicher am langsamsten. Da die zu verarbeitenden Daten ausreichend klein sind und in den Arbeitsspeicher passen, haben wir Speicherzugriffe auf die SD-Karte nicht in Betracht gezogen.

Bei einem Speicherzugriff wird normalerweise nicht nur das angeforderte Element sondern auch gleichzeitig die nachfolgenden Elemente in die Caches geladen. Wenn beispielsweise das erste Element aus einem Array angefordert wird, werden gleichzeitig die ersten 16 Elemente (je nach Größe des Datentyps) des Arrays im L1-Cache zwischengespeichert. Wird im nächsten Schritt des Programms nun das zweite Element des Arrays angefordert, so kann dieses aus dem schnellen L1-Cache geladen werden. Wird hingegen direkt danach das 17. Element geladen, so

muss dieses aus dem L2-Cache oder dem Hauptspeicher geladen werden, wodurch der Speicherzugriff hier länger dauert.

Wie viele Elemente beim Anfordern eines Elements mitgeladen werden, hängt von der Cache-Line-Size ab. Beim Laden eines Elements, das sich nicht im Cache befindet, wird genau eine Cache-Line mit den im Speicher zusammenliegenden Elementen gefüllt. Die Cache-Line-Size ist nicht dokumentiert, konnte aber mithilfe der
Benchmark-Suite lmbench bestimmt werden und beträgt sowohl für den L1-Cache als auch für den L2-Cache $32\,\mathsf{bytes}$.

Wenn also mit dem Inhalt eines Arrays gerechnet werden soll, ist es durch die Caches schneller, der Reihenfolge nach durch das Array zu gehen, statt beliebig auf das Array zuzugreifen. Problematisch wird dies beispielsweise bei mehrdimensionalen Arrays. Im Speicher liegen die Arrays der letzten Dimension hintereinander. Betrachten wir ein zweidimensionales Array als Matrix und wollen über alle Elemente iterieren, so sollten wir zunächst in einer festen Zeile alle Spalten betrachten, bevor wir mit den Elementen der zweiten Zeile rechnen.

Es müssen also zwei Optionen für schnelle Laufzeit abgewogen werden:

1. Die normale Methode ist es, Algorithmen in bekannter Form zu implementieren und möglicherweise langsamen Speicherzugriff in Kauf zu nehmen.

2. Alternativ können die Speicherzugriffe genauer betrachtet und der Algorithmus nach Möglichkeit angepasst werden, um keine unnötige Zeit für Speicherzugriffe zu verlieren.

Um diese beiden Möglichkeiten zu vergleichen, haben wir die benötigte Zeit für einen Speicherzugriff in Abhängigkeit von der Reihenfolge der Zugriffe untersucht.

9.2.2 *Methoden zum Benchmarking*

Durch den Benchmark soll untersucht werden, wie sich die Laufzeit für einen Speicherzugriff verändert, wenn ein Array in verschiedenen Reihenfolgen durchlaufen wird. Es werden verschiedene lineare Reihenfolgen (jedes Element, jedes zweite Element, jedes vierte Element usw.) sowie eine Reihenfolge, die bei jedem Elementzugriff als Index eine Zufallszahl (modulo Arraygröße) verwendet, betrachtet. Der Benchmark ist in der Programmiersprache C implementiert. Dazu wird der Compiler `gcc` mit dem Parameter `-O0` verwendet, um ungewollte Effekte durch die Optimierung zu vermeiden.

Für jede mögliche Reihenfolge der Zugriffe wird zusätzlich nach Arraygröße unterschieden. Im Folgenden wird die Größe, angefangen bei $1\,\mathsf{kiB}$, bei jeder Iteration verdoppelt. Die größte überprüfte Arraygröße ist $4\,\mathsf{MiB}$ ($4096\,\mathsf{kiB}$). Um die gewünschten Größen zu erreichen, wird ein Array, das entsprechend viele Integer-Werte aufnimmt, erzeugt. Ein einzelner Integer benötigt in C einen Speicherplatz von $4\,\mathsf{Bytes}$, somit kann mit 256 Integern eine Größe von $1\,\mathsf{kiB}$ erreicht werden.

	Arraygröße		
	Klein	Mittel	Groß
Jedes Element	15 ns	24 ns	42 ns
Jedes Zweite	15 ns	33 ns	67 ns
Jedes Vierte	15 ns	49 ns	120 ns
Jedes Achte	16 ns	84 ns	224 ns
Zufällig	16 ns	84 ns	322 ns

Tabelle 6: Laufzeiten für den Benchmark nach Arraygröße und Zugriffsreihenfolge. Kleine Arrays sind dabei Arrays bis zu einer Größe von 16 kiB, die in den L1-Cache passen. Mittlere Arrays sind Arrays bis zu einer Größe von 64 kiB, die sicher in den L2-Cache passen. Große Arrays sind Arrays ab einer Größe von 256 kiB, die in keinen der Caches passen.

Um die Zeit für einen einzelnen Zugriff hinreichend genau zu messen, werden für jede Reihenfolge und jede Arraygröße 2^{20} Zugriffe ausgeführt und schließlich die Zeit darüber gemittelt. Da bei den kleineren Arrays die Anzahl der Zugriffe die Anzahl der Elemente im Array überschreitet, werden Arrays teilweise mehrfach durchlaufen. Dazu wird der Index bei jedem Zugriff, unabhängig von der Arraygröße, zunächst mit einer einfachen Modulo-Operation aus der Iterationsnummer ermittelt.

Um sicherzustellen, dass das Element aus dem Array tatsächlich geladen wird und die Zugriffe nicht möglicherweise durch Compiler-Optimierung verworfen werden, wird auf jedem geladenen Element zusätzlich eine Multiplikation mit einer Konstanten durchgeführt. Dabei werden alle Elemente des Arrays mit einer Konstanten initialisiert.

In kompletter Abfolge wird somit eine Iteration mit 2^{20} Schritten ausgeführt, wobei in jedem Schritt neben dem zu messenden Speicherzugriff auch eine Modulo-Operation sowie eine Multiplikation ausgeführt wird. Um tatsächlich nur die Zeit für den Speicherzugriff zu messen, wird zusätzlich die Zeit für einen reinen Schleifendurchlauf ohne Speicherzugriff gemessen und jeweils von der Laufzeit für die Schleife mit Speicherzugriff abgezogen.

9.2.3 *Ergebnisse des Benchmarks*

Tabelle 6 zeigt die Ergebnisse des Benchmarks. Die Ergebnisse sind für die verschiedenen Arraygrößen innerhalb einer Gruppe zusammengefasst, da es zwischen den spezifischen Größen keine großen Unterschiede gibt. Da nicht dokumentiert ist, wie der Speicher im L2-Cache zwischen CPU und GPU aufgeteilt wird, kann der reservierte Platz für die CPU zwischen 64 und 128 kiB liegen. Daher kann hier nicht unterschieden werden, wann ein Array in den L2-Cache passt und aus diesem Grund wird die Arraygröße 128 kiB nicht berücksichtigt.

Obwohl für jeden Testfall die Anzahl der Speicherzugriffe identisch ist, fallen sowohl zeilenweise als auch spaltenweise große Unterschiede in der Tabelle auf.

Solange das Array in den L1-Cache passt, ist die Laufzeit selbst bei zufälliger Zugriffsreihenfolge kaum langsamer als bei linearer Reihenfolge. Da bei solchen kleinen Arrays stets das komplette Array im L1-Cache liegt und keine Zugriffe auf einen langsameren Speicher nötig sind, erscheint dieses Ergebnis durchaus sinnvoll.

Bei Betrachtung einer bestimmten Durchlaufreihenfolge lassen sich teilweise recht große Unterschiede zwischen den Arraygrößen feststellen. Je größer das Array ist, desto häufiger muss – unabhängig von der Zugriffsreihenfolge – ein Zugriff auf den L2-Cache bzw. den Hauptspeicher erfolgen, um weitere Elemente des Array zu laden.

Die Ergebnisse mit der höchsten Relevanz sind die Unterschiede zwischen den Zugriffsreihenfolgen bei identischer Arraygröße. Je größer die Sprünge in der Reihenfolge sind, desto größer sind die Unterschiede in der Laufzeit. Dies lässt sich damit erklären, dass bei großen Sprüngen häufig auf den Hauptspeicher zugegriffen werden muss, während bei Zugriffen entsprechend der Reihenfolge alle Elemente aus dem Cache gelesen werden, bevor ein Zugriff auf einen langsameren Speicher erfolgt. Wird auf jedes achte Element zugegriffen, so steigt die Laufzeit des Zugriffs bei mittelgroßen Arrays auf das Dreifache, bei großen Arrays auf das Fünffache.

Da die Speicherzugriffe nur einen Anteil an der Gesamtlaufzeit haben, lassen sich die Faktoren in der Gesamtlaufzeit etwas relativieren. Dennoch kann durch Beachtung dieses Aspekts die Laufzeit möglicherweise um entscheidende Millisekunden verringert werden. Bei eindimensionalen Arrays kann man noch recht einfach sicherstellen, gemäß der Reihenfolge im Speicher vorzugehen. Komplizierter wird es jedoch bei mehrdimensionalen Arrays oder Attributen von Objekten in C++. Um hier die Laufzeit der Speicherzugriffe zu verringern, müsste zunächst überprüft werden, in welcher Reihenfolge die verschiedenen Elemente oder Attribute im Speicher abgelegt werden.

Zusammenfassend lässt sich sagen, dass Speicherzugriffe aufgrund unserer Laufzeitanforderungen genauer untersucht werden sollten und sich in diesem Bereich die Laufzeit mit etwas Aufwand um einen signifikanten Anteil optimieren lässt. Tatsächlich wird bei der Implementierung des SGLTR-Verfahrens durch Berücksichtigung der Cache-Effekte die Laufzeit erheblich verbessert. Dabei wurde zwischenzeitlich über die Dimensionen eines zweidimensionalen Arrays in der falschen Reihenfolge iteriert. Durch Vertauschen der beiden Iterationen konnte die Laufzeit um den Faktor 3 verbessert werden.

9.3 ÜBERTAKTUNG

Die CPU des Raspberry Pi läuft standardmäßig mit 700 MHz. Es ist möglich, dies auf bis zu 1 GHz zu erhöhen, ein zum Testen verwendeter Pi konnte damit jedoch nicht mehr gebootet werden. Die höchste lauffähige Taktfrequenz wurde mit 950 MHz erreicht.

Um die Geschwindigkeit zu vergleichen, wurde ein Glättungsfilter auf mehrere Spektren mit jeweils 2500 Datenpunkten angewendet und die

Verwendung von `int32_t`

MHz	ms/Spektrum	rel. Geschwindigkeit
700	18,77	1,00
800	16,61	1,13
900	15,07	1,25
950	14,41	1,30

Verwendung von `float`

MHz	ms/Spektrum	rel. Geschwindigkeit
700	27,97	1,00
800	24,43	1,14
900	21,97	1,27
950	20,93	1,34

Tabelle 7: Geschwindigkeitsgewinn beim Übertakten

benötigte Zeit pro Spektrum berechnet. Diese Messungen wurden wie in Tabelle 7 dargestellt jeweils mit Integer- und Floatdaten ausgeführt.

Es kann durch Übertakten also ein Geschwindigkeitsgewinn von rund einem Drittel gegenüber der Standardfrequenz erreicht werden. Dies entspricht in etwa dem Verhältnis von 950 MHz zu 700 MHz. Damit zeigt sich, dass die notwendige Zeit für die durchgeführten Berechnungen nur von der reinen Rechenleistung, nicht von Speicherzugriffen abhängt.

Die erhöhten Taktraten wurden bis jetzt nur zum Erzeugen der Benchmarkergebnisse verwendet. Es kann also keine Aussage über die Stabilität des Systems im Dauerbetrieb gemacht werden.

Die von uns entwickelten Algorithmen haben auch bei 700 MHz ausreichend geringe Laufzeiten, sodass ein Übertakten des Pi nicht notwendig ist.

9.4 GPU-PROGRAMMIERUNG

Neben der ARM-CPU besitzt der Pi eine relativ leistungsstarke, programmierbare GPU, welche zur Berechnung der von SGLTR verwendeten Faltung benutzt werden kann. Zwei Beispielimplementationen dazu finden sich in Abschnitt A.5. Da jedoch auf der CPU mit viel geringerem Aufwand ausreichend geringe Laufzeiten erreicht werden können, wurde eine Verwendung der GPU nicht weiter verfolgt.

10 MERKMALSEXTRAKTION

Nachdem Peakinformationen mit Hilfe eines Peakerkennungsverfahrens gefunden wurden, geht es im Weiteren darum, eine Klassifikation der gefundenen Peaks durchzuführen. Dabei kann festgestellt werden, dass die Anzahl der gefundenen Peaks je Messung variiert. Das ist ein Problem, weil ein Klassifikationsverfahren (siehe Kapitel 11), das die Zuordnung von Peaks zu Klassen übernimmt und sich dabei auf Ähnlichkeitskriterien stützt, auf Eingabevektoren arbeitet, die alle die gleiche Dimension besitzen müssen. Betrachtet man eine Menge von Peaks als einen Eingabevektor, so stellt man fest, dass die Dimension der resultierenden Eingabevektoren der verschiedenen Peaklisten unterschiedlich ist. Dieses Problem wird mithilfe der Merkmalsextraktion bearbeitet und gelöst. Im nächsten Abschnitt werden Datensätze auf relevante Merkmale untersucht. Im Abschnitt 10.2 wird unser Clustering Verfahren vorgestellt und in Abschnitt 10.3 ist beschrieben, wie unsere Merkmale gebildet werden. Im letzten Abschnitt dieses Kapitels wird kurz auf die von uns verwendete Merkmalsextraktion eingegangen.

10.1 ERKENNUNG RELEVANTER MERKMALE

Um uns einen Überblick über den Tee-Kaffee-Datensatz zu verschaffen, haben wir einen Ansatz zur Visualisierung erdacht. Der Tee-Kaffee-Datensatz wird in Abschnitt 13.2.1 beschrieben. Die Idee an der gewählten Visualisierung ist, dass man um alle Peakpositionen eine Bounding Box ermittelt und diese in gleich große Bereiche unterteilt. Für jeden Bereich werden die in ihm liegenden Peaks gesucht und deren Intensität diesem zugeordnet. Da es bei dem Tee-Kaffee-Datensatz nur zwei Klassen (Tee und Kaffee) gibt, kann durch eine Betrachtung der Intensitätsdifferenz zwischen den beiden Klassen festgestellt werden, ob dieser Bereich eher Peaks mit der Klassenzuordnung Tee oder der Klassenzuordnung Kaffee enthält, oder ob kaum Unterschiede zwischen den Peakintensitäten der beiden Klassen existieren. Die resultierenden Intensitätswerte der Bereiche werden mithilfe einer Heatmap dargestellt,die die Identifizierung von Bereichen mit relevanten Intensitätsdifferenzen einfach möglich macht. Im Folgenden werden die verwendeten Verfahren definiert und verschiedene Heatmaps gezeigt.

Sei M die Menge aller Peaks der von Hand annotierten Tee-Kaffee-Messungen und $L = \{p_1, ..., p_n\} \subseteq M$ die Peaks einer Peakliste von einer Messung. Die Funktion $\mathtt{label}(L)$ gibt die Klassenzuordnung der Peakliste L an, wobei eine Peakliste L nur genau eine Klassenzugehörigkeit haben kann. Durch Betrachtung aller Peaks $p \in M$ wird eine Bounding Box R ermittelt, in welcher alle Peaks $p \in M$ mit ihrem Wert der inversen reduzierten Mobilität und ihrem Wert der Retentionszeit liegen. Sei u_t die Anzahl der gewünschten Unterteilungen der Ausdehnung auf der Achse der inversen reduzierten Mobilität von R und u_r die Anzahl der ge-

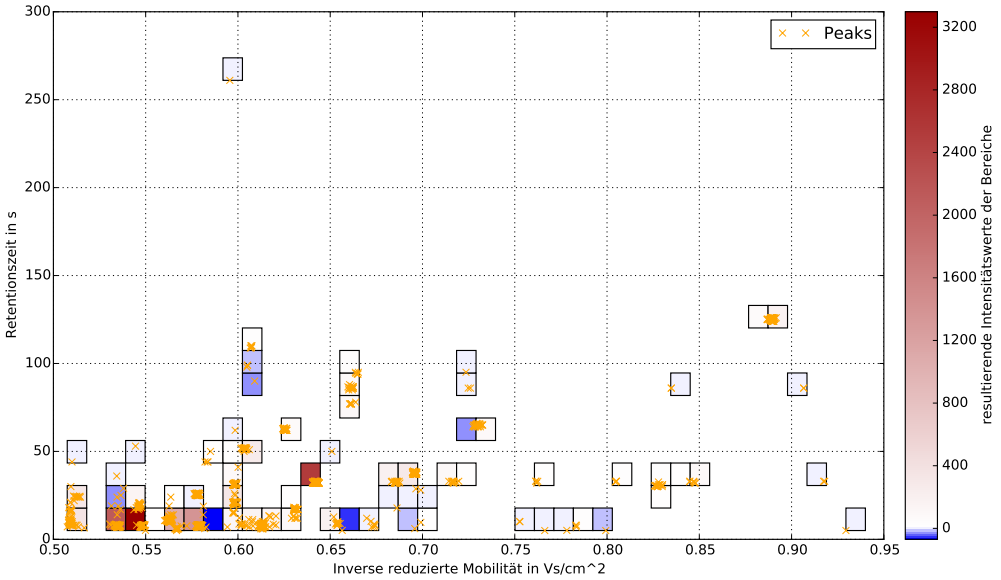


Abb. 57: Die resultierenden Peakintensitäten dieser Heatmap wurden mit Formel (27) berechnet. Rot gefärbte Bereiche zeigen eine Dominanz der Tee Peaks und blau gefärbte Bereiche eine Dominanz der Kaffee Peaks.

wünschten Unterteilungen der Ausdehnung der Achse der Rententionszeit von R. Mit u_t und u_r wird die Bounding Box R unterteilt und resultiert in der folgenden Matrix $\mathbf{Z} = (z_{ij}), \quad i \in \{1, ..., u_r\}, \quad j \in \{1, ..., u_t\}$. Sei P_C die Menge aller Peaks p aus M, wobei jeder Peak p einer Peakliste L zugeordnet ist, die die Bedingung $\mathtt{label}(L) = \textit{Coffee}$ erfüllt und P_T die Menge aller Peaks p aus M, wobei jeder Peak p einer Peakliste L zugeordnet ist, die die Bedingung $\mathtt{label}(L) = \textit{Tea}$ erfüllt. Sei $\mathtt{inZelle}(\{p_1, ..., p_n\}, z_{ij}) = \{q_1, ..., q_m\}, \quad m, n \in \mathbb{N}$ die Funktion, die für eine gegebene Menge an Peaks $\{p_1, ..., p_n\}$ und einer gegebenen Zelle $z_{ij} \in \mathbf{Z}$ eine resultierende Teilmenge an Peaks $\{q_1, ..., q_m\}$ ermittelt, welche die Eigenschaft besitzt, dass sich alle resultierenden Peaks mit ihrer inversen reduzierten Mobilität und ihrer Retentionszeit innerhalb der von Zelle z_{ij} aufgespannten Fläche befinden, welche auch Bereich $B(z_{ij})$ genannt wird. Für die folgenden Visualisierungen sei $u_t = 40$ und $u_r = 20$. In der ersten Visualisierung haben wir die Differenz der summierten Tee-Peakintensitäten und der summierten Kaffee-Peakintensitäten pro Bereich betrachtet, wobei die resultierende Peakintensität pro Bereich $B(z_{ij})$ durch die folgende Formel berechnet wird:

$$\begin{aligned} B^{(1)}(z_{ij}) &= \sum_{f \in P_T^{(1)}} s(f) - \sum_{g \in P_C^{(1)}} s(g) \quad \text{mit} \\ P_C^{(1)} &= \mathtt{inZelle}(P_C, z_{ij}), \\ P_T^{(1)} &= \mathtt{inZelle}(P_T, z_{ij}) \end{aligned} \tag{27}$$

Abbildung 57 zeigt das Ergebnis: Es ist gut erkennbar, welche Bereiche durch Kaffee-Peaks und welche Bereiche durch Tee-Peaks dominiert

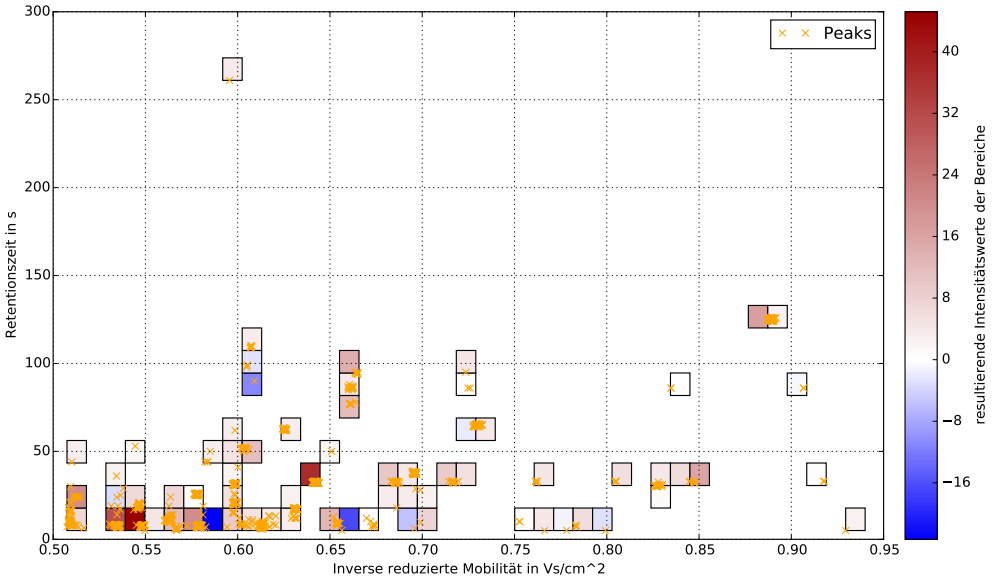


Abb. 58: Die resultierenden Peakintensitäten dieser Heatmap wurden mit Formel (28) berechnet. Im Gegensatz zu Abbildung 57 wird hier die durchschnittliche Differenz der Tee- und Kaffee-Peakintensitäten betrachtet.

werden. In einer zweiten Visualisierung haben wir die durchschnittliche Differenz der summierten Tee-Peakintensitäten und der summierten Kaffee-Peakintensitäten pro Bereich betrachtet, wobei die resultierende Peakintensität pro Bereich durch die folgende Formel berechnet wird:

$$\begin{aligned} B^{(2)}(z_{ij}) &= \frac{\sum_{f \in P_T^{(2)}} s(f) - \sum_{g \in P_C^{(2)}} s(g)}{|P_C^{(2)}| + |P_T^{(2)}|} \quad \text{mit} \\ P_C^{(2)} &= \mathit{inZelle}(P_C, z_{ij}), \\ P_T^{(2)} &= \mathit{inZelle}(P_T, z_{ij}) \end{aligned} \tag{28}$$

Das Ergebnis ist in Abbildung 58 dargestellt.

In einer dritten Visualisierung haben wir den Betrag der resultierenden Peakintensität aus Formel (28) betrachtet, welche sich pro Bereich wie folgt berechnet:

$$\begin{aligned} B^{(3)}(z_{ij}) &= \frac{\left|\sum_{f \in P_T^{(3)}} s(f) - \sum_{g \in P_C^{(3)}} s(g)\right|}{|P_C^{(3)}| + |P_T^{(3)}|} \quad \text{mit} \\ P_C^{(3)} &= \mathit{inZelle}(P_C, z_{ij}), \\ P_T^{(3)} &= \mathit{inZelle}(P_T, z_{ij}) \end{aligned} \tag{29}$$

Abbildung 59 zeigt das Ergebnis. Im Vergleich mit Abbildung 58 ist hier an der resultierenden Peakintensität zu erkennen, dass unter den 20 Bereichen mit der höchsten Peakintensität sowohl Kaffee- als auch Tee-Bereiche vorhanden sind und man diese Bereiche als Ausgangsbasis für die Merkmalsextraktion benutzen könnte. Für einen möglichen Erkenntnisgewinn haben wir verschiedene Variationen von $B(z_{ij})$ betrachtet, diese aber nicht weiter verwendet.

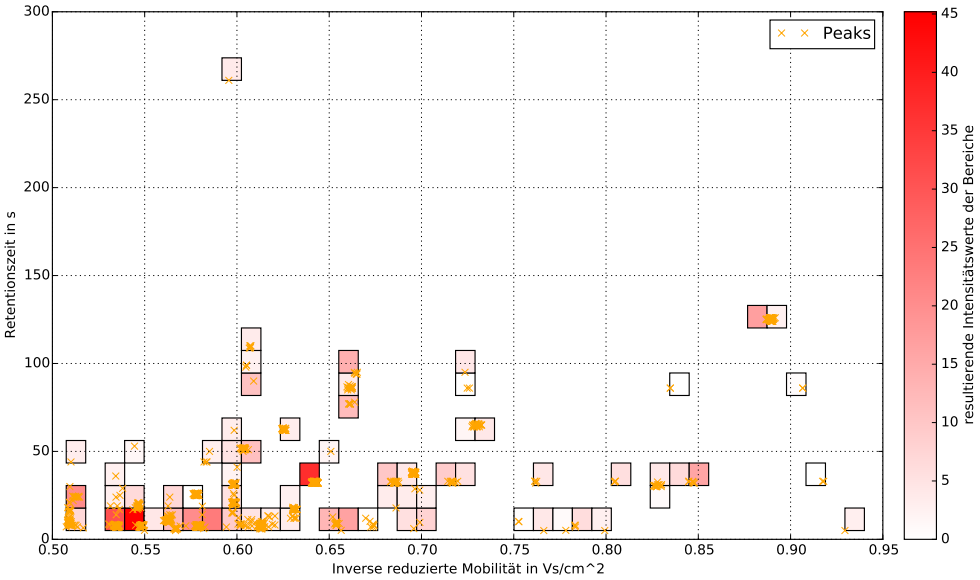


Abb. 59: Resultierende Peakintensitäten nach Formel (29)

10.2 CLUSTERING

Nachdem im vorherigen Abschnitt eine Menge von Peaklisten auf mögliche zu extrahierende Merkmale untersucht wurde, soll in diesem Abschnitt ein konkretes Verfahren zur Merkmalsextraktion vorgestellt werden.

Eine Möglichkeit zur Bildung von Eingabevektoren gleicher Dimension besteht darin, aus den Peaks verschiedener Messungen sogenannte generelle Peakbereiche zu bestimmen (vgl. Bader [4]). Zur Bestimmung der Bereiche können Clustering-Verfahren verwendet werden. Diese bilden aus Mengen von Peaks verschiedener Messungen Cluster, an deren Positionen die Signalintensitäten der Messungen betrachtet werden, um daraus gleich-dimensionale Eingabevektoren für die Klassifikation zu erhalten (vgl. Bader [3]).

Die Annahme bei der Auswahl von Merkmalen ist, dass sich Peaks, die vom gleichen Stoff der zu analysierenden Gasprobe stammen und daher gleich klassifiziert werden müssen, bei verschiedenen Messungen in das gleiche Cluster zugeordnet werden und folglich in ihrem Merkmalsvektor an der entsprechenden Stelle einen ähnlichen Wert erhalten.

In Abbildung 60 ist der Vorgang zur Berechnung der für die Merkmalsextraktion erforderlichen Komponenten, der Zentroide, veranschaulicht. Dieser Vorgang wird offline, das heißt vor der Inbetriebnahme der Klassifikation unbekannter Messungen, durchgeführt. Dabei werden die Peaks aus allen Peaklisten, die für das Training vorgesehen sind, als eine von den einzelnen Annotationen der Peaklisten unabhängige Datenmenge betrachtet. Peaks werden als Punkte in einem zweidimensionalen Raum aufgefasst. Um zu der Peakmenge die natürliche Anzahl von Clustern (das ist später die Anzahl der Merkmale) zu ermitteln, wird ein hierarchisches Clustering durchgeführt (siehe Abschnitt 10.2.2). An-

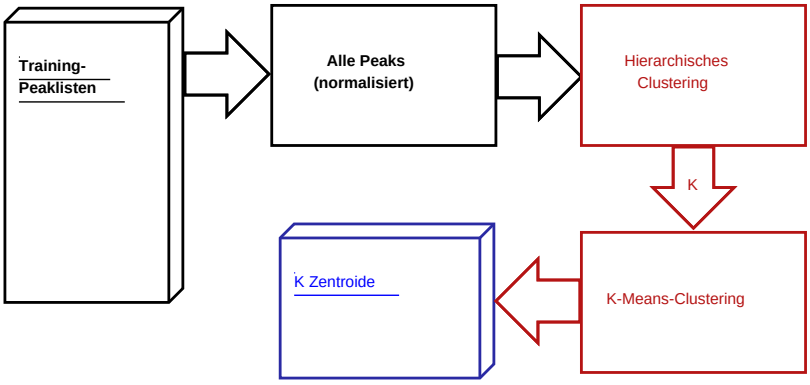


Abb. 60: Trainieren der Komponenten zur Merkmalsextraktion. Komponenten, die gespeichert werden, sind mit einer 3D-Box versehen. Clusteringverfahren sind in rot und das Ergebnis des Trainings (Zentroide) in blau dargestellt.

schließend wird ein flaches Clustering mit dem K-Means-Algorithmus durchgeführt (siehe Abschnitt 10.2.3). Die Komponenten für die Bildung von Merkmalen (siehe Abschnitt 10.3) stellen dann die Zentroide des K-Means-Clusterings dar.

10.2.1 *Normalisierung*

Um die Qualität des Clusterings zu verbessern, kann eine Vorverarbeitung der Daten durchgeführt werden. Insbesondere im Falle unterschiedlich skalierter Dimensionen der Daten, kann eine Normierung der Varianz vorteilhaft sein. Jede Dimension kann als eine Zufallsvariable aufgefasst werden. Die Varianz einer Zufallsvariable X wird auf Eins normiert, indem die Realisierungen von X durch die Standardabweichung von X dividiert werden:

$$X' = \frac{X}{\sqrt{\mathrm{Var}(X)}}.$$

Durch eine solche Normierung jeder Dimension fließen beim anschließenden Clustering alle Dimensionen mit gleichem Gewicht ein.

10.2.2 *Hierarchisches Clustering*

Beim hierarchischen Clustering wird eine hierarchische Struktur von Clustern erstellt, sodass eine Ebene innerhalb der Struktur eine eigenständige Zuordnung von Vektoren zu Clustern abbildet (vgl. Witten [39]). Die Cluster-Hierarchie wird berechnet, indem Zuordnungen von Vektoren zu Clustern anhand einer Distanzfunktion zunehmend vergröbert oder verfeinert werden. Beim *agglomerativen* Clustering beginnt man mit N Clustern, von denen jedes genau einen der N Vektoren beinhaltet. Dann wird die Zuordnung solange vergröbert, bis nur noch ein Cluster vorliegt, welches alle N Vektoren beherbergt. Ein *divisives* Clustering entspricht der umgekehrten Anwendung des agglomerativen Clusterings. Zu Beginn liegt nur ein einzelnes Cluster vor, das alle Vekto-

ren enthält. Dann werden diese groben Zuordnungen solange verfeinert, bis jeder Vektor ein eigenes Cluster bildet.

Im Folgenden wird erläutert, wie mit einem agglomerativen Clustering Peaks in Gruppen geordnet werden können. Ausgangspunkt des agglomerativen Clusterings ist eine Zerlegung der Menge der N Vektoren in N Cluster. Zur Berechnung der nächsthöheren Hierarchieebene werden die beiden Cluster, die die geringste Distanz zueinander haben, zu einem einzigen Cluster verschmolzen. Zur Bestimmung der Distanz zwischen zwei Clustern existiert eine Vielzahl von Strategien. Eine verbreitete Strategie ist das Single Linkage, bei der die Distanz zwischen zwei Clustern A und B wie folgt bestimmt wird (vgl. Morik [29]):

$$D_{SL}(A,B) = \min_{x_a \in A, x_b \in B} d(x_a, x_b).$$

Dabei ist d eine Funktion, die die Distanz zwischen zwei Vektoren angibt. Als Distanzfunktion kann z. B. die euklidische Distanz verwendet werden:

$$d(x_i, x_j) = \sqrt{\sum_{p=1}^{D} (x_{i,p} - x_{j,p})^2}.$$

Beim Single Linkage wird demnach der Abstand zweier Cluster bestimmt als der Abstand der zueinander nächsten Vektoren. Bei der Berechnung der nächsthöheren Hierarchie bei Verwendung der Single Linkage Strategie werden die beiden Cluster mit minimalem Abstand zueinander bestimmt als diejenigen Cluster, die zwei Vektoren enthalten, die unter allen Vektoren in allen Clustern am nächsten zueinander sind. Die beiden so bestimmten Cluster werden zu einem Cluster verschmolzen. Das Vorgehen wird solange wiederholt, bis nur noch ein Cluster vorliegt, dem alle Vektoren zugeordnet sind.

In Abbildung 61 ist die Baumstruktur, die beim hierarchischen Clustering von Peakmengen entsteht, als sogenanntes Dendrogramm visualisiert. Aus Gründen der Übersichtlichkeit handelt es sich bei der Abbildung um das hierarchische Clustering von 34 Peaks, denn das Dendrogramm einer tatsächlichen Anwendung des hierarchischen Clusterings auf über 1000 Peaks wäre hier nicht mehr abbildbar. An der Baumstruktur sind die verschiedenen Hierarchieebenen des Clusterings erkennbar. In der untersten Ebene bildet jeder Peak ein Cluster, in der zweit-tiefsten Ebene werden die beiden Peaks 2 und 19 gemäß der kleinsten Distanz zu einem Cluster zusammengefasst und so weiter.

BESTIMMEN DER CLUSTERANZAHL Ziel des hierarchischen Clustering ist es die Ebene im Dendrogramm zu bestimmen, die eine möglichst natürliche Clusterzuordnung der Vektoren definiert. Die Anzahl der Cluster sowie die Clusterzentren dieser Ebene können dann für die Initialisierung des K-Means-Verfahrens verwendet werden.

Eine Möglichkeit zur Bestimmung der natürlichen Anzahl von Clustern in einer Datenmenge liegt in dem sogenannten Ellenbogenkriterium. Dabei wird die Anzahl der Cluster bestimmt als der Punkt, bei dem eine Erhöhung der Clusteranzahl keine wesentlich bessere Modellierung der Datenmenge nach sich ziehen würde.

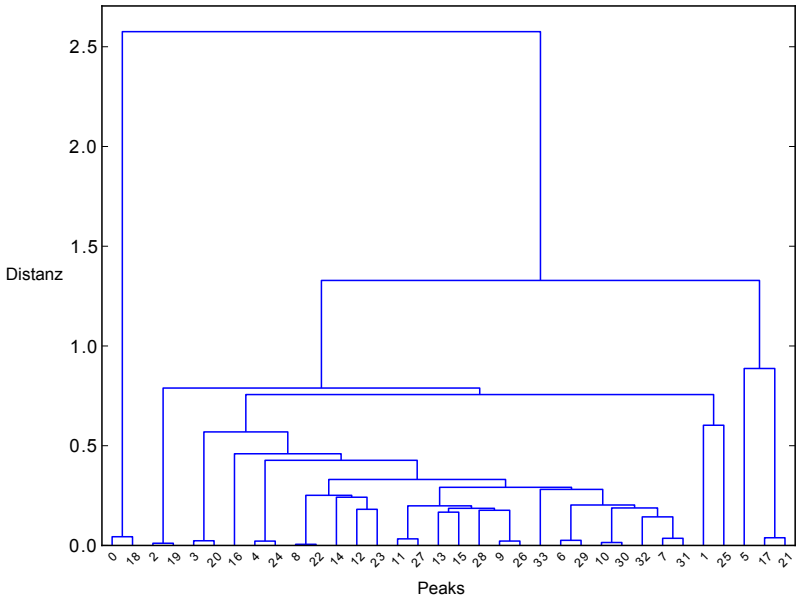


Abb. 61: Dendrogrammdarstellung des hierarchischen Clusterings einer Peakliste.

In Abbildung 62 ist das Vorgehen zur Bestimmung der Clusteranzahl mit Hilfe des Ellenbogenkriteriums geometrisch beschrieben. Dazu wird für jede mögliche Anzahl von Clustern (1 bis N) die Intra-Cluster-Varianz berechnet (blaue Kurve). Die Intra-Cluster-Varianz wird berechnet als die durchschnittliche Streuung eines Vektors von dem Clusterzentrum seines Clusters:

$$S_c = \frac{1}{N-1} \sum_{i=1}^{N} (C[x_i] - x_i)^T (C[x_i] - x_i).$$

Mit $C[x]$ ist das Clusterzentrum des Clusters, dem Vektor x zugeordnet ist, bezeichnet.

Ziel des Ellenbogenkriteriums ist es nun, den Punkt P_{opt} auf der Kurve zu bestimmen, der dem „Ellenbogengelenk“, das heißt dem Knick in der Kurve, entspricht. Dieser Punkt stellt den Kompromiss zwischen möglichst geringer Anzahl von Clustern und möglichst geringer Intra-Cluster-Varianz dar.

Mathematisch kann der Punkt P_{opt} wie folgt berechnet werden: Sei mit

$$\hat{B} = \frac{P_N - P_1}{\|P_N - P_1\|}$$

der auf Länge 1 normalisierte Vektor zwischen dem Anfangspunkt und Endpunkt der Kurve bezeichnet. Der Punkt P_{opt} ist derjenige Punkt auf der Kurve, dessen zu $\hat{B}$ orthonaler Normalenvektor unter allen anderen Punkten den größten Abstand zu $\hat{B}$ hat. Der orthogonale Abstand eines Punktes P zu $\hat{B}$ berechnet sich als euklidischer Abstand zwischen P und der Projektion von $\hat{B}$ auf P (Skalarprodukt) multipliziert mit $\hat{B}$:

$$d(P, \hat{B}) = \| P - (P \cdot \hat{B})\hat{B} \|.$$

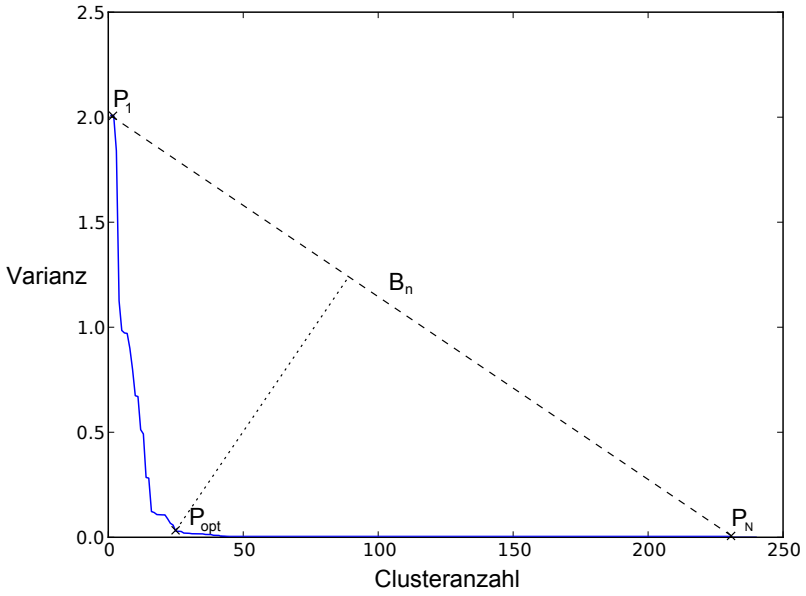


Abb. 62: Bestimmung der optimalen Clusteranzahl mit dem Ellenbogenkriterium.

Punkt P_{opt} erhält man schließlich als den Punkt mit dem größtem Abstand zu $\hat{B}$:

$$P_{opt} = \arg\max_P \quad d(P, \hat{B}).$$

Mit Hilfe von P_{opt} kann nun die Ebene in der Baumstruktur eines hierarchischen Clusterings bestimmt werden, die eine natürliche Anzahl von Clustern definiert. Außerdem können die Clusterzentren auf dieser Ebene für die Initialisierung der Zentroide beim K-Means-Verfahren benutzt werden. Das steigert die Wahrscheinlichkeit des Verfahrens nicht in einem ungünstigen lokalen Optimum zu konvergieren.

10.2.3 *K-Means*

Eine Gruppe weit verbreiteter Clustering-Verfahren wird unter der Bezeichnung K-Means-Verfahren zusammengefasst. Eine bekannte Variante der K-Means-Verfahren ist der Lloyd-Algorithmus, der auch als Voronoi-Iteration bezeichnet wird. In diesem Abschnitt wird der Lloyd-Algorithmus nach Fink [16] vorgestellt.

Der K-Means-Algorithmus erhält als Eingabe eine Menge von Vektoren (z. B. Peakparameter) $\{x_1, x_2, ..., x_N\}$ und einen festen Parameter K, der die Anzahl zu erstellender Cluster angibt. Das Verfahren unterteilt die Vektoren in K Cluster $\{C_1, C_2, ..., C_K\}$, indem eine Vektorquantisierung durchgeführt wird. Zu den Clustern werden die Mittelpunkte $\{Z_1, Z_2, ..., Z_K\}$, die Zentroide, gespeichert.

Ziel des Verfahrens ist die Minimierung des mittleren quadratischen Fehlers:

$$\min \varepsilon \qquad \text{mit} \quad \varepsilon = \sum_{c=1}^{K} \sum_{x_i \in C_c} (Z_c - x_i)^2. \tag{30}$$

Dieser Wert ist ein Maß für die Qualität der aktuellen Zuordnung der Vektoren zu den Clustern. Je niedriger der Fehler ε ist, desto besser ist die Zuordnung.

Zur Initialisierung des Verfahrens müssen eine Anfangszuordnung der Vektoren zu den Clustern und Werte für die Zentroide der Cluster vorgegeben werden. Dies kann zufallsbasiert oder mit Hilfe einer Heuristik geschehen. Der Lloyd-Algorithmus minimiert dann iterativ den Fehler ε durch abwechselnde Anwendung der folgenden beiden Optimierungsvorschriften (vgl. Fink [17]):

1. Nächster-Nachbar-Regel: Ordne Vektor x_i Cluster C_c zu, falls

$$\|Z_c - x_i\| \leqslant \|Z_d - x_i\| \quad \text{für alle } d \neq c.$$

D. h. Vektor x_i wird dem Cluster C_c zugeordnet, zu dem der geringste euklidische Abstand besteht. Bei gleichem Abstand zu zwei Clusterzentroiden kann beispielsweise der Cluster mit dem niedrigeren Index gewählt werden.

2. Zentroidbedingung: Aktualisiere Zentroide Z_c gemäß:

$$Z_c = \frac{1}{|C_c|} \sum_{x_i \in C_c} x_i.$$

Die Zentroide sind die Vektoren in einem Cluster, die den minimalen quadrierten euklidischen Abstand zu allen anderen Vektoren in dem Cluster aufweisen. Sie werden neu berechnet als Mittelvektoren der dem jeweiligen Cluster zugeordneter Vektoren.

Diese beiden Vorschriften werden solange iterativ nacheinander angewendet, bis der mittlere quadratische Fehler ε gegen ein lokales Minimum konvergiert. Bei Konvergenz terminiert das Verfahren, da die aktuelle Zuordnung der Vektoren zu den Clustern durch weitere Anwendung der beiden Vorschriften nicht mehr verbessert werden kann. Der Nachteil des K-Means-Algorithmus besteht darin, dass die gefundene Belegung der Cluster nur eine lokal optimale Lösung darstellt, welche von der Initialisierung des Verfahrens abhängt. Eine gute Initialisierung des K-Means Verfahrens kann z. B. mit den Zentroiden des hierarchischen Clusterings erreicht werden.

In Abbildung 63 ist das Resultat eines durchgeführten K-Means-Algorithmus auf einer Peakmenge zu sehen. Der Algorithmus wurde ausgeführt, um $K = 66$ Zentroide für eine Peakmenge von 1101 Peaks zu bestimmen.

10.3 MERKMALSBILDUNG

Abbildung 64 visualisiert den Vorgang zur Bildung gleich-dimensionaler Merkmalsvektoren mit Hilfe von zuvor berechneten Zentroiden. Für eine Messung, die als Peakliste vorliegt, werden paarweise die Zentroide mit den Peaks aus der Peakliste gemäß der Fängerfunktion verglichen und so ein Merkmalsvektor gebildet.

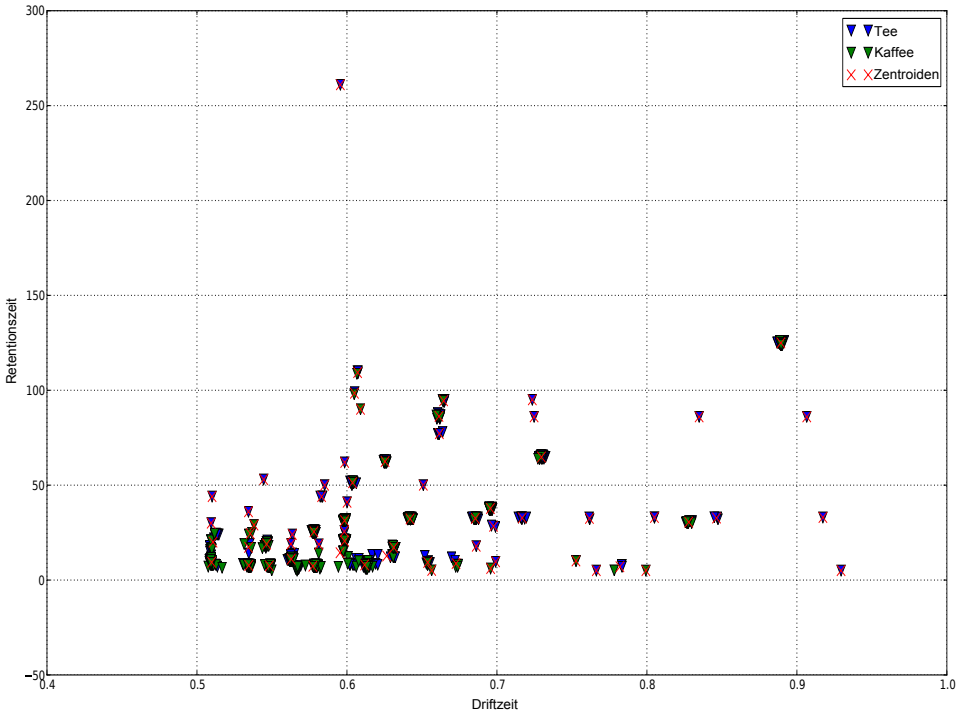


Abb. 63: Ergebnis des K-Means-Algorithmus auf einer Peakmenge von 1101 Peaks.

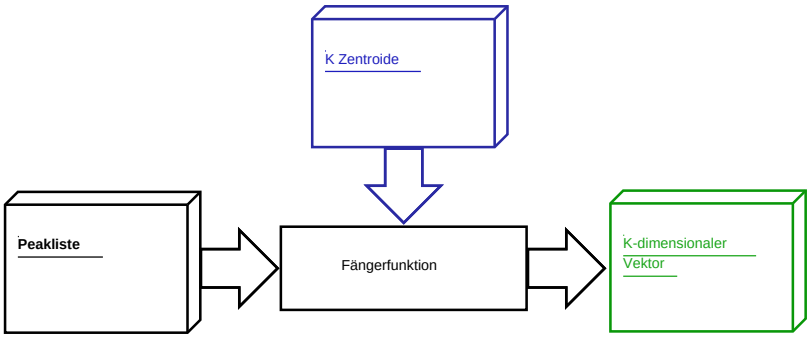


Abb. 64: Prozess zur Bildung von Merkmalsvektoren zur Klassifikation.

Für jeden interessanten Peakbereich (repräsentiert durch einen Zentroid) wird eine Dimension im Merkmalsvektor der Peakliste gebildet. Bei K Zentroiden weist der Merkmalsvektor demnach K Dimensionen auf. Der Merkmalsvektor ist initial ein Null-Vektor. Für eine Dimension p bekommt der Vektor einen Wert, falls in der Peakliste ein Peak existiert, der gemäß der Fängerfunktion nahe genug an Zentroid p ist. Der Wert des Vektors in dieser Dimension ist dann die Intensität des jeweiligen Peaks.

10.4 AUSWAHL UNSERER MERKMALSEXTRAKTION

Im Rahmen der Klassifikation wurde eine Vielzahl unterschiedlicher Methoden zur Extraktion von Merkmalen aus Peaklisten ausprobiert. Um herauszufinden, welche der getesteten Methoden sich am besten für die Klassifikation von Peaklisten eignet, wurden die Klassifikationsraten der

Methoden miteinander verglichen. Es wurde entschieden die Merkmalsextraktion mit einer Kombination von hierarchischem Clustering und K-Means-Clustering (Beschreibung in Abschnitt 10.2) durchzuführen, da diese Methode zu den besten Klassifikationsraten führte (siehe Abschnitt 13.4).

11 KLASSIFIKATION

Die von einem Clustering-Verfahren bestimmten Zentroide erlauben es, aus Peaklisten einheitliche Merkmale zu extrahieren (vgl. Kapitel 10). Diese Merkmale werden verwendet, um eine Messung in Abhängigkeit von den enthalten Stoffe des Probegases in eine Klasse (z. B. Tee oder Kaffee) zuzuordnen. In diesem Kapitel werden die getesteten Klassifikationsverfahren beschrieben. Dabei wird besonders auf das von uns gewählte Klassifikationsverfahren eingegangen.

Beim Klassifikationsproblem ist eine annotierte Stichprobe, die Trainingsmenge, von Datenpunkten gegeben, die verwendet wird, um einen Klassifikator zu trainieren. Dabei werden Muster zur Unterscheidung der Klassen der Daten erlernt, sodass nach Abschluss des Trainings der Klassifikator unbekannte Daten ihren Klassen zuordnen kann.

Unsere Wahl ist auf den Random-Forest-Klassifikator gefallen, da dieser bei der Evaluation die besten durchschnittlichen Klassifikationsraten aufwies. In Abschnitt 11.1 wird dieses Klassifikationsverfahren vorgestellt. Abschnitt 11.2 geht anschließend auf die technischen Details aller getesteten Klassifikatoren ein. Die Evaluation der Verfahren ist in Abschnitt 13.5 zu finden.

11.1 RANDOM FOREST

Der Random Forest Klassifikator ist eine Ensemble-Methode des maschinellen Lernens. Ziel ist die Verbesserung der Klassifikationsleistung eines einzelnen Entscheidungsbaum-Klassifikators durch Kombination mehrerer Entscheidungsbäume auf zufällig generierten Untermengen der Daten zu einem Wald, der als Random Forest bezeichnet wird. Dabei wird jeder Entscheidungsbaum unterschiedlich trainiert und dadurch die Generalisierungsfähigkeit gesteigert.

11.1.1 *Entscheidungsbaum*

Der Entscheidungsbaum ist ein Klassifikator, bei dem im Training ein Baum konstruiert wird. Knoten des Baums entsprechen Merkmalen der Datenpunkte, Zweige den verschiedenen Ausprägungen der Merkmale und Blätter den Klassifikationsentscheidungen des Entscheidungsbaums. Zur Klassifikation eines Vektors werden sequenziell die entsprechenden Vektorelemente mit den Knoteneinträgen des Baums verglichen und auf diese Weise der Vektor in seiner Klasse eingeschränkt.

WICHTIGKEIT VON MERKMALEN Ein Maß für die Wichtigkeit der Merkmale der Datenpunkte für die Klassifikation kann durch die Informationstheorie definiert werden. Die Grundidee ist es, die Merkmale nach ihrem jeweiligen Informationsgehalt zu ordnen. Das Merkmal mit

dem höchsten Informationsgehalt ist das für die Klassifikation wichtigste Merkmal in der Trainingsmenge. Der Informationsgehalt einer Merkmalsausprägung, die mit Wahrscheinlichkeit p in der Trainingsmenge auftritt, wird nach Shannon [34] definiert mit

$$I(p) = \log_2(1/p). \tag{31}$$

Der Wert $I(p)$ ist die Anzahl der Bits, die für die Kodierung der Information mindestens erforderlich sind. Zum Beispiel ist für eine Wahrscheinlichkeit von $\frac{1}{2}$ (z. B. Münzwurf) $I(p) = \log_2 2 = 1$ Bit für die Kodierung notwendig.

Der erwartete Informationsgehalt oder Entropie eines Merkmals Y berechnet sich durch

$$H(Y) = \sum_{a \in V(Y)} p_a I(p_a), \tag{32}$$

wobei $V(Y)$ die Menge der möglichen Ausprägungen von Y und p_a die Wahrscheinlichkeit für Ausprägung a ist. Um ein Merkmal Y zu kodieren sind mindestens $H(Y)$ Bits erforderlich.

Ist ein Merkmal gegeben, dann kann sich die Anzahl möglicher Ausprägungen eines anderen Merkmals einschränken und als Folge davon sein Informationsgehalt sinken. Dies lässt sich mit der bedingten Entropie $H(Y|X = a)$ angeben. Dabei wird die Entropie von Merkmal Y für die Datenpunkte gemessen, für die Merkmal X Ausprägung a hat. Entsprechend lässt sich die erwartete bedingte Entropie angeben:

$$H(Y|X) = \sum_{a \in V(X)} p_a H(Y|X = a). \tag{33}$$

Wird ein Merkmal X ausgewählt, d. h. die Datenmenge gemäß den Ausprägungen von X aufgeteilt, lässt sich der Informationsgewinn für ein anderes Merkmal Y berechnen:

$$IG(Y|X) = H(Y) - H(Y|X). \tag{34}$$

Das Merkmal mit dem höchsten Informationsgewinn besitzt eine höhere Relevanz für die Klassifikation, denn die Daten weisen bei diesem Merkmal die größte Streuung auf, was eine bessere Unterscheidung der Daten ermöglicht. Sollen beispielsweise für eine Datenmenge die Klassenzugehörigkeiten (Merkmal Y) vorhergesagt werden,
dann kann der Informationsgewinn eines Merkmals X als Erkenntniszunahme über die Klassenzugehörigkeiten der Daten nach Bekanntwerden der Werte von Merkmal X verstanden werden.

Um zu verhindern, dass ein stark variierendes Merkmal (z. B. PIN), welches nur einen sehr geringen Informationsgehalt für die Klassifikation besitzt, als wichtigstes Merkmal gewählt wird, führt man eine Normierung des Informationsgewinns durch und erhält den relativen Informationsgewinn

$$RIG(Y|X) = \frac{IG(Y|X)}{H(X)}. \tag{35}$$

Das Merkmal mit dem größten relativen Informationsgewinn ist das für die Klassifikation wichtigste.

AUFBAU EINES ENTSCHEIDUNGSBAUMS Bevor mit einem Entscheidungsbaum die Klassifikation von Daten durchgeführt werden kann, muss der Baum zunächst trainiert werden. Für eine gegebene Trainingsmenge von Datenpunkten kann ein Entscheidungsbaum nach folgendem Vorgehen iterativ konstruiert werden (vgl. Kern-Isberner [26]):

1. Wähle das bzgl. der Klassenzugehörigkeiten Y wichtigste Merkmal X als nächsten Knoten des Baums. Für die Initialisierung des Verfahrens wird die Wurzel des Entscheidungsbaums ebenso von dem wichtigsten Merkmal, d. h. dem Merkmal mit dem größten relativen Informationsgewinn, gebildet.

2. Erzeuge einen Zweig für jede Ausprägung des Merkmals X.

3. Führe den Algorithmus (ab Schritt 1) für jeden in Schritt 2 entstandenen Zweig rekursiv durch und verwende bei jeder Rekursion auf einem Zweig nur die Daten, die die auf dem Zweig vermerkte Ausprägung von Merkmal X erfüllen. Aus der Merkmalsmenge für die Rekursion wird Merkmal X entfernt.

Bei der Durchführung dieses Verfahrens tritt am aktuellen Knoten in Schritt 1 einer der folgenden vier Fälle auf:

(a) Die aktuell betrachtete Datenmenge enthält Datenpunkte verschiedener Klassen.

(b) Alle Datenpunkte der aktuell betrachteten Datenmenge gehören zu einer Klasse.

(c) Die aktuell betrachtete Datenmenge ist leer, d. h. sie enthält keine Datenpunkte.

(d) Die aktuelle Merkmalsmenge ist leer, aber die aktuell betrachtete Datenmenge enthält Datenpunkte aus unterschiedlichen Klassen.

In Fall a wird der Algorithmus weiter ausgeführt, da die Unterscheidung der Datenpunkte noch nicht abgeschlossen ist. Für die übrigen Fälle terminiert das Verfahren und als nächster Knoten wird ein Blatt erstellt. In Fall b steht die Klassifikationsentscheidung fest und daher wird ein Blatt mit der Klasse gebildet, zu der die Datenpunkte gehören. Tritt am aktuellen Knoten Fall c oder Fall d ein, dann wird ein Blatt-Knoten mit einer Standard-Klasse oder Rückweisungsklasse erstellt.

Nach der Erstellung kann ein Entscheidungsbaum zur Klassifikation verwendet werden, indem der zu klassifizierende Vektor als Eingabe für den Baum verwendet wird. Beginnend an der Wurzel wird an jedem Knoten des Baums entschieden, an welchen Unterknoten der Vektor weitergegeben wird. Die Entscheidungen werden dabei jeweils anhand der Ausprägung des für den aktuellen Knoten relevanten Merkmals getroffen. Sobald ein Blatt des Baums erreicht wird, ist der Klassifikationsvorgang abgeschlossen und der Datenpunkt wird der an dem Blatt vermerkten Klasse zugewiesen.

11.1.2 *Random Forest Verfahren*

Die Grundidee des Random Forests nach Breiman [8] ist es den Entscheidungsbaum-Klassifikator zu verbessern, indem mehrere Entscheidungsbäume auf variierenden Trainingsmengen (Bagging) und unterschiedlich eingeschränkten Merkmalsmengen (Random Feature Selection) trainiert werden. Die Klassifikation erfolgt dann als Mehrheitsentscheid über die Klassifikationen der einzelnen Entscheidungsbäume.

Die Verwendung identischer Entscheidungsbäume für den Random Forest würde in keiner Verbesserung der Klassifikationsleistung resultieren. Aus diesem Grund wird durch zwei zufallsbasierte Methoden das Training jedes Entscheidungsbaums variiert.

Die erste Methode ist das *Bagging* (Bootstrap aggregating), das zu einer Variation in den Trainingsdatenmengen der einzelnen Bäume führt. Dafür werden aus der insgesamt zur Verfügung stehenden N-elementigen Trainingsdatenmenge zufällig N Datenpunkte mit Zurücklegen gezogen und als Trainingsmenge zur Konstruktion eines Entscheidungsbaums verwendet.

Die zweite Methode ist *Random Feature Selection*. Bei der Erstellung jedes Knotens des Baums wird die aktuell zur Verfügung stehende Menge der Merkmale eingeschränkt, indem nur $m \ll M$ zufällig gewählte Merkmale betrachtet werden. Für die Wahl des wichtigsten Merkmals können so nur m der M Merkmale herangezogen werden.

Der Vorteil des Baggings liegt darin, dass keine separate Validierungsdatenmenge erforderlich ist, um die Modellparameter (z. B. m) anzupassen. Da bei den Trainings der einzelnen Entscheidungsbäume im Allgemeinen nicht die gesamte Trainingsdatenmenge verwendet wird, sind sogenannte Out-Of-Bag-Daten (OOB) vorhanden. Für Entscheidungsbaum k, der mit Trainingsmenge T_k konstruiert wurde, sind die OOB-Daten definiert als

$$T \setminus T_k,$$

wobei T die gesamte Trainingsmenge bezeichnet. Für einen Entscheidungsbaum sind seine OOB-Daten unbekannte Daten. OOB-Daten können zur Messung der Klassifikationsleistung des Random Forests verwendet werden. Dazu wird jeder Datenpunkt x aus T mit den Entscheidungsbäumen klassifiziert, für die x OOB-Datum ist. Die resultierende Klassifikationsrate heißt OOB-Fehlerrate und stellt eine gute Abschätzung der Generalisierungsfähigkeit des Random Forests dar. Beispielsweise kann der Parameter m, der für die Random Feature Selection benötigt wird, optimal als der Wert bestimmt werden, zu dem die OOB-Fehlerrate minimal ist. Außerdem ist es möglich, mit Hilfe der OOB-Fehlerrate die wichtigsten Merkmale der Datenpunkte zu bestimmen bzw. eine Merkmalsreduktion durchzuführen. Zu diesem Zweck wird für jedes Merkmal m die Wichtigkeit gemessen, indem die Ausprägungen von Merkmal m für die Datenpunkte zufällig permutiert werden und dann die OOB-Fehlerrate gemessen wird. Das Merkmal, dessen Permutation der Ausprägungen maximale OOB-Fehlerrate verursacht, ist für die Klassifikation am wichtigsten.

Der Random Forest besitzt durch seine Zufallskomponenten gegenüber vielen anderen Klassifikatoren einen wichtigen Vorteil. Mit dem Gesetz der großen Zahlen kann bewiesen werden, dass das Generalisierungsverhalten des Waldes mit zunehmender Baumanzahl konvergiert und deshalb keine Gefahr der Überanpassung besteht (vgl. Breiman [8]). Insbesondere müssen aus diesem Grund die Entscheidungsbäume des Random Forests nicht zur Vermeidung von Überanpassung gestutzt werden. Folglich kann der Random Forest prinzipiell aus beliebig vielen Entscheidungsbäumen gebildet werden. Ein weiterer Vorteil des Random Forests ist seine Effizienz, denn sowohl Training als auch Klassifikation mit den einzelnen Entscheidungsbäumen können parallel durchgeführt werden.

11.2 KLASSIFIKATOREN IN SCIKIT-LEARN UND PARAMETER

Zu Beginn der Merkmalsextraktions- und Klassifikationsphase musste die Entscheidung getroffen werden, ob wir eine Softwarebibliothek für die Klassifikation benutzen, oder aber sämtliche benötigte Software selber schreiben. Wir haben uns für die Benutzung einer Bibliothek entschieden. Aus einer Reihe von Softwarebibliotheken zum maschinellem Lernen mit einer Open-Source Lizenz haben wir `scikit-learn`[1] in der Version `0.14` ausgewählt. `scikit-learn` ist in Python implementiert und wird unter der BSD-Lizenz[2] bereitgestellt.
Diese Bibliothek beinhaltet sehr viele Module und Werkzeuge für das maschinelle Lernen. Für unsere Klassifikation konnten wir auf die vorhandenen Implementierungen der Random Forest-, Support Vector Machine- und k Nearest Neighbor-Algorithmen zurückgreifen. Im Folgenden wird auf die Implementationen der drei Algorithmen eingegangen.

11.2.1 *Random Forest*

Die Random Forest Implementierung basiert auf dem Paper „Random Forests“ von Leo Breiman [8]. Es werden insgesamt 11 Parameter[3] zur Verfügung gestellt und auf die von uns benutzten wird im Folgenden kurz eingegangen:

- **n_estimators:** Über diesen Parameter wird die Anzahl der Bäume für den Random Forest festgelegt. Der Standardwert ist 10.
- **criterion:** Dieser Parameter gibt die Funktion an, mit der die Qualität einer Knotenaufteilung gemessen wird. Es gibt die Optionen `„gini“` und `„entropy“`, wobei `„gini“` die Standardoption ist.
- **max_features:** Dieser Parameter legt fest, wie viele Merkmale m (siehe Abschnitt 11.1.2) berücksichtigt werden sollen, um nach

1 http://scikit-learn.org
2 https://github.com/scikit-learn/scikit-learn/blob/master/COPYING
3 http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

der besten Aufteilung eines Knotens im Random Forest zu suchen. Die folgenden Möglichkeiten stehen zur Verfügung:

- **Ganzzahl:** Die Anzahl der berücksichtigten Merkmale entspricht der Anzahl, die als Ganzzahl übergeben wurde.
- **Gleitkommazahl:** Wird hier eine Gleitkommazahl im Bereich $[0, 1]$ angegeben, gibt dieser Parameter den prozentualen Anteil der Merkmale an, der berücksichtigt wird.
- **„auto“** bzw. **„sqrt“:** Die Anzahl der berücksichtigten Merkmale entspricht der Quadratwurzel der Anzahl der Merkmale. Diese Option ist der Standardwert für diesen Parameter.
- **„log2“:** Die Anzahl der berücksichtigten Merkmale entspricht dem Logarithmus zur Basis zwei der Anzahl der Merkmale.
- **None:** Durch diese Option werden alle Merkmale berücksichtigt.

- **n_jobs:** Das Training und die Klassifizierung eines Random Forest parallelisiert werden. Die Anzahl der Threads, die parallel laufen sollen, kann angegeben werden und der Standartwert für diesen Parameter ist 1. Durch Angabe von -1 wird die Anzahl der Threads auf die Anzahl der zur Verfügung stehenden Prozessorkerne gesetzt.

- **random_state:** Dieser Parameter legt fest, wie der intern benutzte Zufallszahlengenerator parametrisiert wird. Es wird der Zufallszahlengenerator aus der Python Softwarebibliothek NumPy benutzt, welcher sich in der Klasse `numpy.random.RandomState` befindet. Es gibt die folgenden Möglichkeiten der Parametrisierung:
  - **Ganzzahl:** Die angegebene Ganzzahl wird als Seed für den Zufallszahlengenerator benutzt.
  - **RandomState Instanz:** Man kann eine eigene Implementierung der `numpy.random.RandomState` Klasse benutzen.
  - **None:** Mit `None` wird die Standardparametrisierung für den Zufallszahlengenerator benutzt. Der Seed wird dabei entweder mit Daten aus `/dev/urandom` (oder dem Windows Äquivalent) oder alternativ durch die Systemzeit gebildet.

Es werden insgesamt 10 Methoden von der Random Forest Implementierung zur Verfügung gestellt von denen wir zwei Methoden benutzen. Mit der Methode `fit(X,y)` wird ein Random Forest erstellt und trainiert. Dabei ist `X` ist ein zweidimensionales Array mit den Trainingsdaten. Jede Zeile enthält alle extrahierten Merkmale einer Stichprobe, wobei die Anzahl der Merkmale für alle Stichproben gleich sein muss. Jede Spalte repräsentiert ein Merkmal, welches sich in jeder Stichprobe an der gleichen Stelle befinden muss. Die Klassenzugehörigkeit wird über ein eindimensionales Array `y` angegeben, in welchem die Klassenzugehörigkeit einer Stichprobe angegeben werden muss. Die Position einer Zeile in `X` entspricht der Position der zugehörigen Klassenzugehörigkeit

in `y`. Die Rückgabe der Methode `fit(X,y)` ist ein trainierter Random Forest-Klassifikator.

Die Methode `predict(X)` klassifiziert eine oder mehrere Stichproben. Der Eingabeparameter `X` ist ein zweidimensionales Array mit den Testdaten. Jede Zeile enthält alle extrahierten Merkmale einer Stichprobe, wobei die Anzahl der Merkmale gleich der Anzahl der Merkmale der Trainingsmenge sein muss. Jede Spalte repräsentiert ein Merkmal, welches sich in jeder Stichprobe an der gleichen Stelle befinden muss. Für jede enthaltene Stichprobe wird dann eine Klassifikation durchgeführt. Die Rückgabe ist ein eindimensionales Array, in welchem das Ergebnis der Klassifikation an der Position der Zeile der Stichprobe in `X` steht.

11.2.2 *Support Vector Machine (SVM)*

`scikit-learn` nutzt zur Bereitstellung des SVM-Klassifikators die Softwarebibliotheken `LIBLINEAR`[4] und `LIBSVM`[5]. Die Implementation der Bibliothek `LIBLINEAR` basiert auf dem Paper von Fan u. a. [15] und die Implementation der Bibliothek `LIBSVM` auf dem Paper von Chang und Lin [11]. Es werden 13 Parameter[6] zur Verfügung gestellt, die wir alle mit ihren Standardwerten benutzen.

Es werden insgesamt acht Methoden von der SVM Implementierung zur Verfügung gestellt, von denen wir zwei benutzen. Mit der Methode `fit(X,y)` wird eine SVM erstellt und trainiert, wobei `X` ein zweidimensionales Array mit den Trainingsdaten ist. Jede Zeile enthält alle extrahierten Merkmale einer Stichprobe, wobei die Anzahl der Merkmale für alle Stichproben gleich sein muss. Jede Spalte repräsentiert ein Merkmal, welches sich in jeder Stichprobe an der gleichen Stelle befinden muss. Die Klassenzugehörigkeit wird über ein eindimensionales Array `y` angegeben, in welchem die Klassenzugehörigkeit einer Stichprobe angegeben werden muss. Die Position einer Zeile in `X` entspricht der Position der zugehörigen Klassenzugehörigkeit in `y`. Die Rückgabe der Methode `fit(X,y)` ist ein trainierter SVM-Klassifikator.

Mit der Methode `predict(X)` wird eine Stichprobe klassifiziert. Der Eingabeparameter `X` ist ein zweidimensionales Array mit den Testdaten. Jede Zeile enthält alle extrahierten Merkmale einer Stichprobe, wobei die Anzahl der Merkmale gleich der Anzahl der Merkmale der Trainingsmenge sein müssen. Jede Spalte repräsentiert ein Merkmal, welches sich in jeder Stichprobe an der gleichen Stelle befinden muss. Für jede enthaltene Stichprobe wird dann eine Klassifikation durchgeführt. Die Rückgabe ist ein eindimensionales Array, in welchem das Ergebnis der Klassifikation an der Position der Zeile der Stichprobe in `X` steht.

11.2.3 `k` *Nearest Neighbor (KNN)*

Als eine weitere Alternative haben wir die `k` Nearest Neighbor Klassifikator Implementierung der `scikit-learn` Softwarebibliothek auspro-

4 http://www.csie.ntu.edu.tw/~cjlin/liblinear/
5 http://www.csie.ntu.edu.tw/~cjlin/libsvm/
6 http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

biert. Die Details der Implementierung sind in der Dokumentation von `scikit-learn`[7] beschrieben. Um die Anzahl der benutzen Merkmale für den KNN-Klassifikator weiter einzuschränken und nach Möglichkeit nur die relevantesten Merkmale zu verwenden, projizieren wir auf die „Hauptachsen“, die über eine Principal-Componenent-Analysis (PCA) ermittelt werden. Details zu der Implementierung in Scikit-Learn finden sich in der `scikit-learn`-Dokumentation[8].

Es werden drei Parameter für die PCA Initialisierung zur Verfügung gestellt, von denen wir nur den Parameter `n_components` benutzen und für diesen den Wert 20 verwenden, um die Anzahl der Merkmale auf 20 zu reduzieren.

Die PCA Implementierung stellt sechs Methoden zur Verfügung, von denen wir nur die Methode `fit_transform(X)` verwenden, um die Anzahl der Merkmale unserer Trainings- und Testdaten `X` auf 20 zu beschränken. Die Trainings- oder Testdaten befinden sich in einem zweidimensionalem Array `X`. Jede Zeile enthält alle extrahierten Merkmale einer Stichprobe, wobei die Anzahl der Merkmale für alle Stichproben gleich sein muss. Jede Spalte repräsentiert ein Merkmal, welches sich in jeder Stichprobe an der gleichen Stelle befinden muss. Die Rückgabe der Methode `fit_transform(X)` ist ein auf 20 Merkmale beschränkter Trainigs- oder Testdatensatz.

Die KNN Implementierung[9] stellt sechs Parameter zur Verfügung, welche wir alle mit ihren Standardwerten benutzen.

Es werden insgesamt 8 Methoden von der KNN Implementierung zur Verfügung gestellt, von denen wir zwei Methoden benutzen. Mit der Methode `fit(X,y)` wird ein KNN Klassifikator erstellt und trainiert. Die Trainingsdaten befinden sich im zweidimensionalen Array `X`. Jede Zeile enthält 20 Merkmale einer Stichprobe, welche durch die PCA-Vorverarbeitung ausgewählt wurden. Jede Spalte repräsentiert ein Merkmal, welches sich in jeder Stichprobe an der gleichen Stelle befinden muss. Die Klassenzugehörigkeit einer Stichprobe muss im eindimensionalen Array `y` angegeben werden. Die Position einer Zeile in `X` entspricht der Position der zugehörigen Klassenzugehörigkeit in `y`. Die Rückgabe der Methode `fit(X,y)` ist ein trainierter KNN-Klassifikator. Mit der Methode `predict(X)` wird eine Stichprobe klassifiziert. Der Eingabeparameter `X` ist ein zweidimensionales Array mit den Testdaten. Jede Zeile enthält 20 Merkmale einer Stichprobe, welche durch die PCA-Vorverarbeitung ausgewählt wurden. Jede Spalte repräsentiert ein Merkmal, welches sich in jeder Stichprobe an der gleichen Stelle befinden muss. Für jede enthaltene Stichprobe wird dann eine Klassifikation durchgeführt. Die Rückgabe ist ein eindimensionales Array, in welchem das Ergebnis der Klassifikation an der Position der Zeile der Stichprobe in `X` steht.

7 http://scikit-learn.org/stable/modules/neighbors.html
8 http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
9 http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

12 PIPELINE

Die Algorithmen zur Peakerkennung wurden in C++11 implementiert, die Programme zur Merkmalsextraktion und Klassifikation in Python geschrieben. Um von einer Messung im CSV-Format zum Klassifikationsergebnis zu gelangen sind einige Schritte zum Installieren der benötigten Abhängigkeiten und Kompilieren der Programme notwendig. Der genaue Ablauf dazu wird nachfolgend beschrieben.

12.1 ERSTELLEN DER BENÖTIGTEN PROGRAMME

Zunächst müssen die von uns geschriebenen Programme zur Peakerkennung kompiliert werden. Der dazu benötigte Quellcode ist in einem Git-Repository verfügbar und kann mit

```
git clone git@projekte.itmc.tu-dortmund.de:in/git-pg572.git
```

heruntergeladen werden. Es wird ein Compiler mit C++11-Unterstützung benötigt, z. B. `gcc` ab Version 4.7.

Falls das in Abschnitt 4.2 beschriebene IMSH5-Format verwendet werden soll, wird `libhdf5` benötigt, unter Ubuntu etwa im Paket `libhdf5-dev` verfügbar. Diese Bibliothek wird nach Installation automatisch erkannt und verwendet.

Einige Einstellungen zum Erstellungsprozess können in der Datei `src/build_config.cmake` vorgenommen werden. Falls für den Raspberry Pi kompiliert werden soll, entweder per Cross-Compiling (falls nicht das vorgefertigte `build-cross.sh`-Script benutzt wird) oder direkt auf dem Gerät selber, muss in dieser Datei die Variable `RPI` auf `TRUE` gesetzt werden. Eine Host-spezifische Konfiguration kann mit Kopien dieser Datei mit dem Namen

```
src/build_config.${HOSTNAME}.cmake
```

erreicht werden. Dies kann nützlich sein wenn von mehreren Rechnern über das Netzwerk auf das gleiche Quellcodeverzeichnis zugegriffen wird.

Ist das Projekt wie gewünscht konfiguriert, kann der Erstellungsprozess gestartet werden. Mit `build-native.sh` wird der native Compiler des Host-Systems verwendet, dies kann ein x86-Desktop oder auch ein Raspberry Pi sein. Das Skript `build-cross.sh` verwendet den in der Datei `src/rpi-toolchain.cmake` angegebenen Cross-Compiler. Die Einrichtung einer Cross-Compiling-Umgebung wird in Abschnitt 3.3 näher erläutert.

Bei erfolgreicher Kompilation sind die resultierenden Programme in `src/build.native/` bzw. `src/build.cross/` zu finden und können direkt von dort aufgerufen werden. Ausserdem wird dort ein Debian-Paket mit dem Namen

```
imstools-${VERSION}-${ARCH}.deb
```

erstellt, das z. B. auf dem Pi unter Raspbian mittels

```
dpkg -i imstools-0.0.0-armv6l.deb
```

installiert werden kann. Eine aus dem aktuellen Zustand des Git-Repositories nach jeder Änderung neu erzeugte Version dieses Pakets ist auch von unserem PG-Server verfügbar. Nach Hinzufügen der Zeile

```
deb http://amber-v4.cs.tu-dortmund.de:8010/ wheezy pg572
```

zu `/etc/apt/sources.list` und Ausführen von `apt-get update` kann diese mit `apt-get install imstools` installiert werden.

Mit dem Script `get-datasets.sh` können die von uns verwendeten Datensätze heruntergeladen werden. Um Platz und Zeit beim Einlesen der Daten zu sparen, können Dateien im CSV-Format mit `python/dataformat/imscsv2imsh` in das IMSH5-Format konvertiert werden.

12.2 VERWENDUNG DER PROGRAMME ZUR PEAKERKENNUNG

Die Peakerkennung kann nun mit den Programmen `sgolay` und `pdsa` durchgeführt werden, die jeweils das SGLTR- und PDSA-Verfahren implementieren. Eine lokale Datei kann dazu mit dem Parameter `-q` angegeben werden. Das Format wird dabei automatisch erkannt. Die Daten müssen in der vom Algorithmus erwarteten Auflösung vorliegen, standardmäßig ist dies die geringe Auflösung. Ohne Angabe einer Datei wird auf Daten über das Netzwerk gewartet. Die Peakliste wird auf `stdout` ausgegeben und kann zur späteren Verwendung mit $>$ in eine Datei umgeleitet oder mit | direkt an das Programm zur Klassifikation weitergeleitet werden.

12.3 VERWENDUNG DES SERVERS

Um die Daten über das Netzwerk zu versenden, wird `ims_server` mit folgenden Parametern verwendet.

- `-h` Host, an den die Daten geschickt werden sollen
- `-i` Eingabedatei
- `-d` Zeit zwischen Spektren in `ms`, standardmäßig 100
- `-e` Anzahl der Spektren vom Ende der Messung für simulierte Luftmessung
- `-m` Vervielfältigungsfaktor, um Daten mit hoher Auflösung zu simulieren

Ein kompletter `ims_server`-Aufruf mit gering aufgelösten Daten könnte also wie folgt aussehen:

```
ims_server -m 5 -d 5 -e 50 -i 0.csv -h shujaa
```

12.4 MERKMALSEXTRAKTION UND KLASSIFIKATION

Im Folgenden wird die benötigte Software für die Merkmalsextraktion und Klassifikation aufgeführt und aufgezeigt, wie diese zu benutzen ist.

BENÖTIGTE SOFTWARE

Auf einem Raspberry Pi mit installiertem Raspbian-Betriebssystem kann die benötigte Software wie folgt installiert werden:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3 python3-dev python3-pip \
  python3-numpy python3-scipy libatlas-dev libatlas3-base
sudo pip-3.2 install scikit-learn
```

Auf einem Computer mit einem Ubuntu 12.04-Betriebssystem kann die benötigte Software anhand der folgenden Anweisungen installiert werden:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3 python3-dev python-pip \
  python3-numpy python3-scipy python3-setuptools \
  libopenblas-dev
sudo easy_install3 pip
sudo pip3.2 install scikit-learn
```

DURCHLAUF

Um einen Klassifikator zu trainieren und diesen später zur Klassifikation einer Peakliste benutzen zu können, wird eine Menge von Peaklisten mit ihrer Klassenzugehörigkeit benötigt. Die einzelnen Peaklisten müssen zusammen in einem Verzeichnis liegen und folgendermaßen benannt sein: *Peaklistennummer.csv* (z. B. `20.csv`). Die Klassenzugehörigkeit jeder Peakliste wird in einer separaten CSV-Datei festgehalten, welche nicht im Verzeichnis der Peaklisten liegen darf. Ein Beispiel für eine solche Datei sieht wie folgt aus:

```
file        class
0.csv        Klasse1
1.csv        Klasse2
2.csv        Klasse2
3.csv        Klasse1
```

In dieser Datei wird die Peakliste ihrer jeweiligen Klasse zugeordnet und beide Einträge sind mit einem Tabulator getrennt. Das Programm `ims_train.py` im Verzeichnis `python/classification` stellt die Funktionalität bereit, einen Klassifikator zu trainieren. Ein exemplarischer Aufruf von `ims_train.py` mit dem Peaklistenverzeichnis `peaklisten` und der Datei mit den entsprechenden Klassenzugehörigkeiten `labels.csv` sieht wie folgt aus:

```
python3 ims_train.py peaklisten/ labels.csv
```

Nach erfolgreicher Ausführung des Programms liegen die benutzten Zentroiden im Ordner `centroids` und der trainierte Klassifikator im Ordner

`models`. Mit diesen Zentroiden und dem trainierten Klassifikator kann eine Peakliste mit dem Programm `ims_classify.py` klassifiziert werden. Ein beispielhafter Aufruf von `ims_classify.py` mit einer Peakliste `single-peaklist.csv` sieht wie folgt aus:

```
python3 ims_classify.py < single-peaklist.csv
```

Die Ausgabe von `ims_classify.py` ist die Klassenzugehörigkeit der Peakliste. Beide Programme verfügen über weitere Optionen, welche mit dem Parameter `-h` ausgegeben werden.

13 EVALUATION

Wir beschreiben in diesem Kapitel, wie die von uns entworfenen Algorithmen auf verschiedenen Messdaten abschneiden. Im wesentlichen stellen sich dabei zwei Fragen: Wie gut sind die Peaklisten, die von den Algorithmen ausgegeben werden? Wie gut können Messungen mithilfe dieser Peaklisten klassifiziert werden?

Bevor wir diese Fragen beantworten, erläutern wir zunächst allgemeine Probleme bei der Evaluation (Abschnitt 13.1). In Abschnitt 13.2 beschreiben wir die Messdaten, die wir zur Evalutaiton verwendeten haben. Anschließend erklären wir in Abschnitt 13.3 das Gütekriterium für die einzelnen Peaklisten und wie unsere Algorithmen unter Verwendung des Gütekriteriums abschneiden. Klassifikationsergebnisse für verschiedene Klassifikationsverfahren und unterschiedlich erzeugte Peaklisten beschreiben wir in Abschnitt 13.5. Zusätzlich gehen wir in Abschnitt 13.6 auf Laufzeiten der Algorithmen auf dem Raspberry Pi ein.

13.1 PROBLEMATIK DER EVALUATION

Um Peakerkennung evaluieren zu können, ist eine „richtige“ Peakliste erforderlich (Gold-Standard). Für die Klassifikation ist eine Zuordnung von Messungen zu Klassen nötig, anhand derer die Klassifikationsalgorithmen arbeiten können. Wir erläutern in diesem Abschnitt generelle Probleme mit der Notwendigkeit solcher Daten.

Die Notwendigkeit einer richtigen Peakliste stellt bereits ein erstes Problem dar: Es gibt keine eindeutige Wahrheit, wann ein Peak vorliegt und wann nicht. Während bei großen Peaks mit hoher Intensität noch klar ist, dass ein Peak vorliegt, ist es bei kleinen Peaks oft nicht eindeutig, ob es sich wirklich um einen Peak handelt oder nur das Rauschen an dieser Stelle stärker ist. Außerdem ist bei großen Peaks mit mehreren lokalen Maxima nicht eindeutig, ob es sich um einen einzelnen oder mehrere kleinere Peaks handelt.

Aus diesen Gründen sind manuell von Experten erzeugte Peaklisten problematisch, da die Auswahl der Peaks teilweise subjektiv ist. Es kommt hinzu, dass bei großen Datensätzen typischerweise nicht alle Messungen von der gleichen Person annotiert wurden, sodass die subjektiven Auswahlkriterien sich je nach Messung unterscheiden. Vergleiche mit manuellen Peaklisten sind daher immer unter gewissem Vorbehalt zu betrachten. Neben diesen allgemeinen Problemen, die prinzipiell bei manueller Annotation von Messungen auftreten, gab es spezifische Probleme der uns zur Verfügung gestellten Messdaten, die im folgenden Abschnitt beschrieben werden.

13.2 VERWENDETE MESSDATEN

Um Peakerkennungs- und Klassifikations-Algorithmen evaluieren zu können, werden sinnvolle Datensätze gebraucht. Dabei sind manuelle Annotationen, Klassifikationen und eine ausreichende Größe des Datensatzes notwendig. Wir erläutern in diesem Abschnitt, welche Datensätze uns zur Verfügung standen, welche spezifischen Probleme diese Datensätze aufwiesen sowie betrachtete Möglichkeiten, die Probleme zu beheben.

13.2.1 *Verfügbare Datensätze*

Für die Projektgruppe gab es im wesentlichen zwei Datensätze, welche die grundlegenden Anforderungen nur zum Teil erfüllten. Dabei handelt sich um zwei anonymisierte Messreihen von Patienten, die wir hier als „Tee-Kaffee-Daten“ und „Salz-Pfeffer-Daten“ bezeichnen. Beide Messreihen enthalten etwas weniger als 70 Messungen, die jeweils zu zwei Klassen zugeordnet wurden.

TEE-KAFFEE-DATEN Die Tee-Kaffee-Daten enthalten 69 Messungen mit 69 manuell annotierten Peaklisten. Das erste Problem war die teilweise fragwürdige Zuordnung von Messungen in die zwei Klassen. Aufgrund einiger technischer Besonderheiten der MCC-IMS-Messgeräte ist es möglich, dass bei der Messung vorgenommene Klassifikation durch die Klassifikation der folgenden Messung überschrieben wird. Da bei der Zuordnung der Messungen die beiden Klassen alternierend war, ist es denkbar, dass Zuordnungen teilweise vertauscht wurden. Ein weiteres Problem waren vermutlich defekte Peaklisten, bei denen einige Peaks mit einer Retentionszeit von 0 eingetragen wurden.

SALZ-PFEFFER-DATEN Der zweite Datensatz besteht aus 65 Messungen, die in die Klassen „Salz“ und „Pfeffer“ eingeteilt sind. Im Gegensatz zur ersten Messreihe sind die Messungen hier anscheinend in zwei Blöcke für die zwei Klassen eingeteilt, sodass etwa die erste Hälfte der Messungen zur Klasse Salz und die zweite Hälfte zur Klasse Pfeffer gehört. Aus diesem Grund sind bei diesem Datensatz Vertauschungen der Zuordnungen unwahrscheinlicher. Allerdings waren für diesen Datensatz keine manuellen Annotationen verfügbar. Um die Peaklisten bewerten zu können, bevor diese zur Klassifikation verwendet wurden, hat die Projektgruppe selbst manuell Peaklisten für diese Messreihe erzeugt. Der Datensatz wurde von 3 Teilnehmern annotiert und daher bleibt auch hier die Frage nach der subjektiven Einschätzung der jeweiligen Person.

VEREINZELTE MESSUNGEN Neben diesen zwei Messreihen waren vereinzelte Messungen der Projektgruppe mit verschiedenen Bonbonsorten verfügbar. Diese sind bereits zu Beginn der Projektzeit durchgeführt worden. Das ursprüngliche Ziel war es, eine große Bonbon-Testreihe durchzuführen, um eigene Messungen klassifizieren zu können. Allerdings wurden für diese Messungen keine manuellen Annotationen er-

zeugt und die Anzahl der Messungen war zu gering, um sie zur Klassifikation zu verwenden.

Bis anonymisierte Daten zur Verfügung gestellt werden konnten, sind diese Messungen dennoch als Grundlage verwendet worden, um die ersten NumPy-Algorithmen SGLTR (siehe Abschnitt 8.3) und PDSA (siehe Abschnitt 8.5) zu entwickeln, zu visualisieren und zu testen. Darüber hinaus wurde mit diesen Messungen die Simulation der Client-Server-Implementierung in C/C++ zwischen dem Raspberry Pi und Laptops bzw. Desktop-Computern entwickelt. Erste Versuche zur Merkmalsextraktion und Klassifikation wurden ebenfalls anhand dieser Messungen durchgeführt.

SYNTHETISCHE DATEN Für die Evaluation der Peakerkennungsalgorithmen hat die Projektgruppe mittels des in Abschnitt 13.2.2 beschriebenen Messungsgenerators einen Datensatz von 100 synthetischen Messungen im IMS-CSV-Format erstellt. Entsprechend den Erfahrungen bei der Annotation des „Salz-Pfeffer"-Datensatzes wurden 60% der Peaks einer Messung mit niedrigen Intensitäten erzeugt und 40% mit hohen Intensitäten. Die niedrigen Intensitäten wurden gemäß einer Normalverteilung mit $\mu = 15$ und $\sigma = 4$ ermittelt, die hohen Intensitäten mit $\mu = 65$ und $\sigma = 10$. Außerdem enthält jede Messung einen Peak mit einer Intensität, die gleichverteilt aus dem Bereich von 245 bis 1237 gezogen wurde. Die Sampledaten wurden dabei als Double-Fließkommazahlen berechnet und beim Abspeichern auf vier Nachkommastellen gerundet. Dieser Datensatz eignet sich zwar nicht zum Testen der Klassifikation, bietet gegenüber den realen Datensätzen jedoch den Vorteil, dass exakt passende Peaklisten vorliegen.

13.2.2 *Messungsgenerator*

Zur Erzeugung synthetischer Messungen wurde uns freundlicherweise von Dominik Kopczynski ein Generatorprogramm zur Verfügung gestellt. Dieses liest eine IMS-CSV-Datei (siehe Abschnitt 4.1) als Vorlage ein, ersetzt die Sampledaten durch eine zufällig bestimmte Anzahl von Peaks mit ebenfalls zufälligen Formen und Intensitäten und fügt einen RIP und Rauschen hinzu. Das Resultat wird wieder in eine IMS-CSV-Datei ausgegeben.

Der Peakerzeugung liegt dabei das Modell aus Kopczynski u. a. [27] zugrunde, das Peaks als Produkt zweier verschobener inverser Gaußverteilungen darstellt (vgl. Abschnitt 8.1). Dabei beschreibt eine Verteilung die Form des Peaks in Richtung der Retentionszeit, die andere die Form in Richtung der Driftzeit.

Der RIP wird ebenfalls durch zwei solche Verteilungen modelliert. Seine Form in Driftrichtung ergibt sich hier als Konvexkombination der Verteilungen und ist für jedes Spektrum identisch. Um das Einschnüren des RIPs durch größere Peaks nachzubilden, geht der Generator allerdings von einer bis auf kleine Schwankungen konstanten Gesamtenergie pro Spektrum aus und skaliert den RIP entsprechend der Summe der im Spektrum vorliegenden Intensitäten.

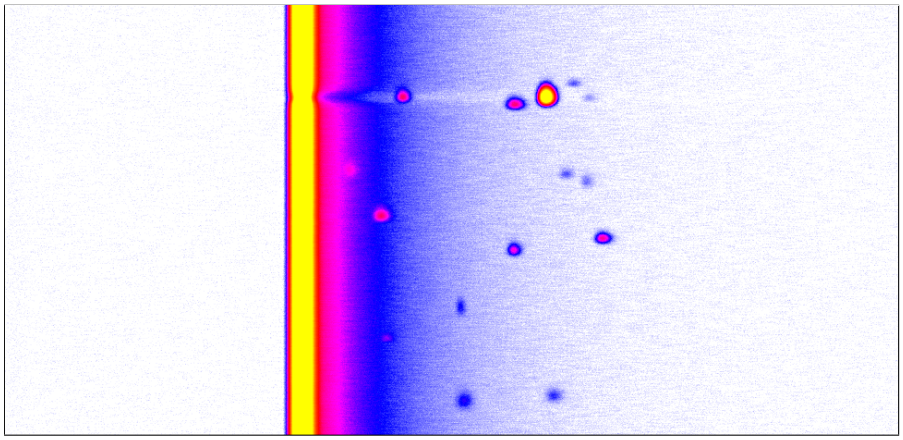


Abb. 65: Beispiel einer synthetischen Messung

Schließlich werden der Messung ein normalverteiltes Rauschen (vgl. Abschnitt 6.1) und ein niederfrequenter Sinus-Anteil in Driftrichtung beigemischt.

Den bestehenden Generator haben wir weiterentwickelt. So ist es nun möglich, zu den generierten Messungen passende Peaklisten ausgeben zu lassen und somit eine Evaluation der Peakerkennungsalgorithmen auf Messungen mit sicher bekannten Peak-Positionen durchzuführen. Umgekehrt lässt sich zu einer gegebenen Peakliste im Format aus Abschnitt 4.3 eine passende Messung erzeugen. Da solche Listen einen Peak lediglich über die Position des Maximums und die dortige Intensität beschreiben, aber keine Informationen zur Form des Peaks enthalten, wählen wir die Parameter der inversen Gaußverteilungen zur Peakmodellierung so, dass diese Normalverteilungen annähern. Zusätzlich zu IMS-CSV-Dateien liest und schreibt der Generator auch Messungen im IMSH5-Format (siehe Abschnitt 4.2), verarbeitet dabei allerdings keine Meta-Informationen.

Die im ursprünglichen Generator genutzte Parametrisierung der inversen Gaußverteilungen über Erwartungswert, Formparameter und Verschiebung hat den Nachteil, dass zwischen den Parametern und dem Modus der Verteilung, also der Position des Maximums ihrer Dichtefunktion, keine offensichtliche Beziehung besteht. Daher nutzen wir die äquivalente Beschreibung der Verteilung über Mittelwert, Standardabweichung und Modus aus Kopczynski u. a. [27, S. 1804]. Bezeichnet $\mathrm{ig}_{m,\mu,\sigma}$ die so parametrisierte Dichte, ergibt sich für einen Peak an Position $m = (m_r, m_t)$ mit Mittelwert $\mu = (\mu_r, \mu_t)$, Standardabweichung $\sigma = (\sigma_r, \sigma_t)$ und Intensität s die Darstellung

$$p_{m,\mu,\sigma}(r,t) = f_s \, \mathrm{ig}_{m_r,\mu_r,\sigma_r}(r) \, \mathrm{ig}_{m_t,\mu_t,\sigma_t}(t)$$

mit

$$f_s = \frac{s}{\mathrm{ig}_{m_r,\mu_r,\sigma_r}(m_r) \, \mathrm{ig}_{m_t,\mu_t,\sigma_t}(m_t)}.$$

Eine mit dem Generator erzeugte Messung ist in Abbildung 65 dargestellt.

13.3 PEAKERKENNUNG

In diesem Abschnitt erläutern wir die Evaluation der Peaklisten, die von unseren Algorithmen gefundenen wurden. Dazu definieren wir zunächst ein Gütekriterium, welches die Bewertung der Güte der berechneten Peaklisten im Vergleich zu den manuell annotierten Peaklisten ermöglicht. Anschließend zeigen wir die Ergebnisse der Anwendung des Gütekriteriums auf die von uns berechneten Peaklisten. Dabei trennen wir die Peaklisten, die auf den anonymisierten echten Messdaten berechnet wurden, von den Peaklisten zu den automatisch generierten, synthetischen Messungen.

Mit bloßem Hinsehen ist nicht erkennbar, ob eine ausgegebene Peakliste korrekt ist, ob Peaks übersehen wurden oder viele falsche Peaks gefunden wurden. Ein Gütekriterium kann nach abgeschlossener Messung auf die vollständig ausgegebene Peakliste angewendet werden. Wir erklären hier zunächst Grundlagen, die zur Bewertung benötigt wurden, und erläutern anschließend Anpassungen auf unseren Anwendungsfall.

13.3.1 *Grundlagen zur Peaklisten-Bewertung*

Als Grundlage für die Bewertung einer ausgegebenen Peakliste wird eine Referenz-Peakliste benötigt. Diese Referenz-Peakliste kann beispielsweise eine per Hand annotierte Messung sein. Wichtig hierbei ist, dass die Referenzliste aus genau der gleichen Messung entstammt, damit die einzelnen Peaks sinnvoll miteinander verglichen werden können. Viele von B & S Analytik erhaltenen Peaklisten basieren auf einer Menge von ähnlichen Messungen, sodass die Positionen der einzelnen Peaks nicht klar zu den Peaks der Einzelmessungen zugeordnet werden können und es kleinere Verschiebungen geben kann. Solche zusammengefassten Annotationen können nicht verwendet werden, um einzelne Peaklisten zu bewerten.

Um unsere Peakliste mit einer Referenzliste zu vergleichen, haben wir als Grundlage das Vergleichkriterium für Peaks aus einer Arbeit von Hauschild et al. [23] zu diesem Thema übernommen. Da es auch bei manueller Annotation sehr schwierig ist, die exakte Position eines Peaks anzugeben, wird hier eine kleine Abweichung zwischen den gefundenen Peaks des untersuchten Algorithmus und den Referenzpeaks erlaubt. Ein Peak gilt dann als gefunden, wenn er in der inversen reduzierten Mobilität um höchstens $0{,}003\,\mathrm{Vs/cm^2}$ und in der Retentionszeit um höchstens $3{,}0\,\mathrm{s} + r(P) \cdot 0{,}1$ von einem passenden Peak in der Referenzliste abweicht. Dabei ist $r(P)$ die Retentionszeit des Peaks in der zu überprüfenden Liste [23, 3.1.6]. Die Einbeziehung der Retentionszeit in der Formel lässt sich damit erklären, dass Peaks mit zunehmender Retentionszeit sich über immer mehr Spektren erstrecken können und es dadurch schwerer wird, die exakte Retentionszeit des Peaks anzugeben.

Zusätzlich zum Vergleichskriterium von Peaks zweier Peaklisten haben wir Begriffe von binärer Klassifikation übernommen, um eine gefundene Peakliste zu bewerten. Generell ist bei binärer Klassifikation die Problemstellung, eine Menge von Objekten mithilfe eines Klassifika-

	Peak	kein Peak	
gefunden	true positive	false positive	precision
nicht gefunden	false negative	true negative	$p = \frac{tp}{tp+fp}$
	sensitivity $s = \frac{tp}{tp+fn}$		F1-Maß $f = 2 \cdot \frac{p \cdot s}{p+s}$

Tabelle 8: Begriffe binärer Klassifikation

tors in zwei Klassen aufzuteilen. Die Aufteilung in Klassen wird dabei so gewählt, dass eine Gruppe von Objekten eine gesuchte Eigenschaft hat, während die zweite Gruppe diese Eigenschaft nicht aufweist. Die so erhaltene Klassifikation wird anschließend mit der bekannten echten Klassifikation verglichen und bewertet. In unserem Fall sind die Objekte die möglichen Positionen von Peaks, die Klassen „Peak“ und „kein Peak“, der Klassifikator ist die gegebene Peakliste.

Jedes bewertete Objekt kann in vier Gruppen fallen:

- Wird die gesuchte Eigenschaft mithilfe des Klassifikators festgestellt und stimmt die Zuordnung mit der wahren Klassifikation überein, so wurde dieses Objekt als *true positive* (kurz: tp) klassifiziert;
- Objekte, denen die Eigenschaft fälschlicherweise zugeordnet wurde, sind *false positive* (fp);
- wenn die Eigenschaft fälschlicherweise nicht zugeordnet wurde, so wurde es als *false negative* (fn) klassifiziert;
- zuletzt kann ein Objekt korrekt ohne die Eigenschaft klassifiziert werden und dann nennt man sie *true negative* (tn).

Mithilfe dieser vier Fälle können wir nun zwei Begriffe für die Gesamtbewertung definieren, die in der Tabelle 8 zusammengefasst sind. Ein Klassifikationsverfahren hat im wesentlichen zwei Ziele: Zum einen sollte möglichst bei allen Objekten, die die gesuchte Eigenschaft aufweisen, diese auch festgestellt werden; zum anderen sollte für Objekte, welche die Eigenschaft nicht aufweisen, diese auch nicht fälschlicherweise festgestellt werden. Somit sollen möglichst alle Peaks der manuellen Liste gefunden werden. Ob jedoch das Finden weiterer Peaks als negativ gewertet werden soll, ist fragwürdig. Dennoch sollen weiterhin Kriterien zur Vergleichbarkeit der Güte aufgestellt werden.

Dazu verwenden wir die Begriffe *Sensitivität* und *Genauigkeit*. Die *Sensitivität* (englisch: sensitivity) gibt an, bei wie vielen Objekten mit der Eigenschaft diese auch festgestellt wurde. Es errechnet sich wie folgt:

$$\text{sensitivity:} \quad s = \frac{tp}{tp + fn}\,.$$

Dabei entspricht $tp + fn$ genau der Anzahl der Objekte, welche die gesuchte Eigenschaft insgesamt aufweisen. Die *Genauigkeit* (englisch: precision) gibt an, wie viele der mit der Eigenschaft klassifizierten Objekte die gesuchte Eigenschaft auch tatsächlich aufweisen. Die Genauigkeit lässt sich dann wie folgt bestimmen:

$$\text{precision:} \quad p = \frac{tp}{tp + fp}\,.$$

Hier entspricht $tp + fp$ der Gesamtanzahl der Objekte, bei denen der Klassifikator die gesuchte Eigenschaft vorausgesagt hat.

Gute Klassifikatoren sollten für beide Begriffe gute Werte aufweisen. Ein Klassifikator, der nur einen der beiden Werte maximiert, ist für jeden Anwendungsfall einfach zu entwerfen: Wird für jedes beliebige Objekt die Eigenschaft vorausgesagt, so beträgt die Sensitivität immer 100%, dafür ist die Genauigkeit häufig nahe 0. Umgekehrt kann nur für sehr wenige Objekte, bei denen die Eigenschaft komplett sicher ist, die Zuordnung geschehen, sodass eine gute Genauigkeit, jedoch eine sehr geringe Sensitivität erzielt wird. Um einen einzelnen Wert zu haben, mit welchem man Klassifikatoren direkt vergleichen kann, gibt es kombinierte Maße aus beiden Größen. Für diesen Zweck verwenden wir hier das *F1-Maß*, welches als harmonisches Mittel zwischen Sensitivität und Genauigkeit bestimmt wird. Das F1-Maß berechnet sich dann mit

$$\text{F1-Maß:} \quad f = 2 \cdot \frac{p \cdot s}{p + s}\,.$$

Da je nach Anwendungsfall die Sensitivität oder Genauigkeit wichtiger als der jeweils andere Wert ist, gibt es auch weitere kombinierte Maße, die einen der beiden Werte stärker gewichten.

13.3.2 *Gütekriterium*

Der wesentliche Unterschied unseres Peaklisten-Vergleichs gegenüber der herkömmlichen binären Klassifikation ist, dass wir nicht die gesamte Menge, sondern nur eine geringe Auswahl der Objekte klassifizieren. Da die Objekte die möglichen Kombinationen aus Driftzeit und Retentionszeit sind, gibt es (bei geringer Auflösung!) 3 000 000 mögliche Punkte für Peaks, während die richtige Peakliste typischerweise Peaks im geringen zweistelligen Bereich enthält. Somit würden (bei berechneten Peaklisten ähnlicher Größe) fast sämtliche Punkte als true negative klassifiziert werden. Aus diesem Grund sind alle Größen, die die Anzahl der true negatives in der Berechnung benötigen, für uns nicht anwendbar, da sie sich nur in den hinteren Nachkommastellen unterscheiden würden.

Ein weiteres Problem ist, dass aufgrund der erlaubten Verschiebungen für einen Punkt nicht eindeutig gesagt werden kann, ob dort ein Peak vorliegt oder nicht. Theoretisch müssten für alle Punkte in erlaubtem Abstand eines Peaks der berechneten Peakliste „vielleicht“ ausgegeben werden. Bei dieser Anpassung der Zuordnung ist aber schon unklar, wie die verschiedenen Werte bestimmt werden sollten.

Um diese Probleme zu umgehen, haben wir die vier Klassifikationsfälle anders definiert, um sie sinnvoll anhand der Peaklisten bestimmen zu können, ohne jeden Punkt einzeln zu betrachten. Aufgrund der erläuterten Problematik haben wir true negatives nicht mehr betrachtet. Abb. 66 illustriert die folgenden 6 Fälle:

1. True positives sind die Peaks aus der echten Peakliste (blau), für die es innerhalb der erlaubten Abweichung einen passenden Peak der berechneten Peakliste (orange) gibt.

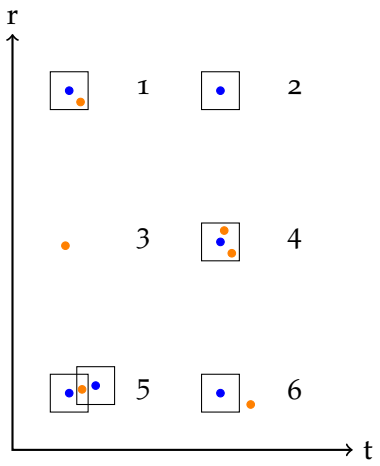


Abb. 66: Mögliche Fälle bei Peaklistenauswertung. Blau: Echter Peak. Orange: Ausgegebener Peak. Fälle 1-6: Siehe Text.

2. Die übrigen Peaks der richtigen Peakliste, für die es keinen passenden Peak in der zu überprüfenden Peakliste gibt, sind false negatives.

3. False positives sind die restlichen Peaks der berechneten Peakliste, die keinem echten Peak zugeordnet wurden.

4. Es können mehrere Peaks der berechneten Peakliste innerhalb des erlaubten Abstands zu einem einzelnen Peak der echten Peakliste sein. Dies kann beispielsweise geschehen, wenn in der Peakdetektion ein großer Peak als mehrere kleine Peaks erkannt wird. In diesem Fall wird einer der Peaks der berechneten Liste als true positive eingeordnet, während die restlichen Peaks false positives sind.

5. Umgekehrt kann ein detektierter Peak innerhalb der erlaubten Entfernung mehrerer echter Peaks sein, falls mehrere kleine, nah beieinander liegende Peaks in der Peakdetektion als einzelner großer Peak erkannt wurden. In diesem Fall wird der detektierte Peak einem (beliebigen) der echten Peaks zugeordnet und als true positive gezählt, während die restlichen echten Peaks als false negatives eingeordnet werden.

6. Ein weiterer möglicher Fall ist, wenn ein Peak in zu großer Entfernung zur tatsächlichen Position gefunden wird. In diesem Fall wird sowohl ein false negative (für den echten Peak) als auch ein false positive (für den zu falsch positionierten Peak) gezählt.

Allerdings sind false positives nicht immer zwingend falsche Peaks: Manuell annotierte Peaklisten müssen nicht der uneingeschränkten Wahrheit entsprechen. Es ist durchaus möglich, dass bei der manuellen Annotation Peaks übersehen wurden, da auch mit Hilfsmitteln bei kleineren Peaks diese nicht immer eindeutig erkennbar sind. Der umgekehrte Fall, dass Stellen manuell als Peaks markiert werden, die gar keinen Peak darstellen, tritt deutlich seltener auf. Aus diesem Grund könnte die Sensitivität etwas stärker gewichtet werden als die Genauigkeit.

Trotz der alternativen Definitionen verhalten sich die drei Größen sehr ähnlich zu den Begriffen bei normaler binärer Klassifikation. Gefundene Peaks sind auch bei unserer Definiton die Summe der true positives und false negatives. False positives sind fälschlicherweise gefundene Peaks und somit ist die Summe aus true positives und false positives die Anzahl der von der berechneten Peakliste als Peak bezeichneten Punkte. Die Werte für Genauigkeit und Sensitivität verhalten sich also genau so wie erwartet und können ohne Anpassung verwendet werden. Die Ausfallquote und andere Maße, welche von den true negatives abhängen, sind hier nicht definiert, da wir keine true negatives bestimmen (können).

Insgesamt lässt sich das F1-Maß wie bei herkömmlicher binärer Klassifikation bestimmen und kann als kombiniertes Bewertungsmaß für berechnete Peaklisten verwendet werden. Wir betrachten nun, wie sich unsere Algorithmen bei Verwendung dieses Gütekriteriums verhalten.

13.3.3 *Manuell annotierte Daten*

Wir betrachten in diesem Abschnitt, wie die Implementierungen der beiden beschriebenen Ansätze nach dem in Abschnitt 13.3.2 erklärten Gütekriterium abschneiden.

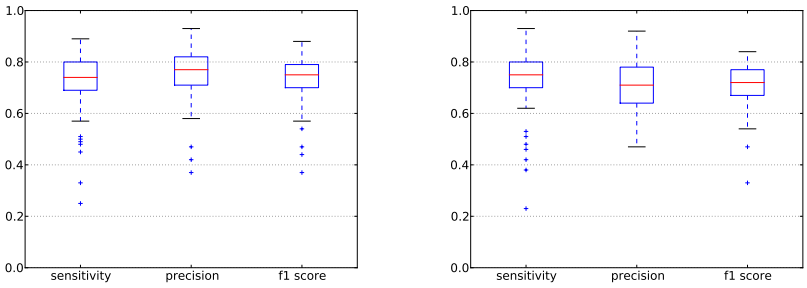
Der Anteil der gefundenen und falschen Peaks kann sich für jede Messung unterscheiden. Um aussagekräftige Ergebnisse zu erhalten, haben wir also die Ausgabeliste und Referenzliste für mehrere Messungen verglichen, statt nur für eine einzelne Messung die Ausgabe zu überprüfen. Da nur für den Tee-Kaffee-Datensatz Gold-Standard-Peaklisten verfügbar waren, beschränken wir uns hier auf die Ergebnisse auf diesem Datensatz.

Abb. 67 zeigt Boxplots der Evaluation von SGLTR und PDSA auf dem Tee-Kaffee-Datensatz. Es fällt hier auf, dass beide Algorithmen für alle drei Scores vergleichbare Ergebnisse erzielen, die durchschnittlich zwischen 70 und 80% liegen. Der PDSA schneidet bei der Sensitivität mit durchschnittlich ca. 75% knapp besser ab als der SGLTR. Dieser hat mit einem Wert von über 75% bei der Präzision jedoch Vorteile gegenüber dem PDSA, der einen Wert knapp über 70% erreicht. Der PDSA markiert also etwas häufiger falsche Peaks als der SGLTR. Dadurch fällt das F1-Maß des SGLTR mit ca. 75% besser aus als beim PDSA, der hier mit etwas über 70% abschneidet.

Insgesamt lässt sich sagen, dass beide Algorithmen auf realen Daten zufriedenstellende Peakerkennungsraten aufweisen. Es ist hier jedoch zu bedenken, dass anhand der Peakerkennungsraten nur bedingt eine Aussage über die Güte der Klassifikation getroffen werden kann. Wir evaluieren die Algorithmen in Bezug auf Klassifikationsraten in Abschnitt 13.5.

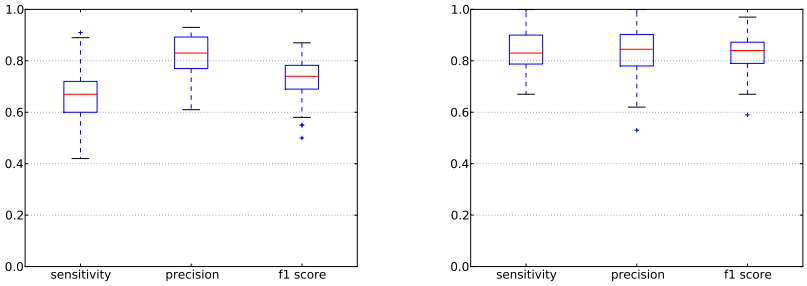
13.3.4 *Vergleich mit synthetischen Peaks*

Nachdem wir die Peakerkennung auf realen Messungen mit einer manuellen und damit potenziell ungenauen Annotation verglichen haben,



(a) SGLTR Version 5.4 (b) PDSA

Abb. 67: Vergleich der manuell annotierten Peaks gegenüber gefundenen auf den Tee-Kaffee-Daten.



(a) SGLTR Version 5.4 (b) PDSA

Abb. 68: Ergebnisse der Peakerkennungs-Evaluation auf synthetisch erzeugten Daten.

evaluieren wir in diesem Abschnitt die Peakerkennung auf dem synthetischen Datensatz. Hierbei ist eine Verfälschung der Ergebnisse durch ungenaue Peaklisten ausgeschlossen. Gleichwohl handelt es sich bei den betrachteten Messungen um Instanzen eines unter Umständen unvollständigen Messungsmodells. Insbesondere das Rausch- und Störungsmodell ist recht einfach, so dass die erzeugten Messungen für die Peakerkennung tendenziell „freundlichere“ Eigenschaften als reale Messungen aufweisen könnten. Dennoch erscheinen uns die synthetischen Messungen den realen Messungen ähnlich genug, um sie zur Evaluation zu nutzen.

Im Gegensatz zur Evaluation der Peakerkennung auf dem Tee-Kaffee-Datensatz unterscheiden sich die Ergebnisse der beiden Algorithmen auf dem synthetischen Datensatz deutlich, wie Abb. 68 zeigt. Der PDSA kann sein F1-Maß auf fast 85% steigern und schneidet nun bei der Genauigkeit mit ebenfalls knapp unter 85% sogar etwas besser ab als bei der Sensitivität. Auch der SGLTR kann seine Genauigkeit auf über 80% steigern, seine Sensitivität fällt auf diesem Datensatz jedoch auf etwa 67% ab, wodurch auch das F1-Maß leicht auf unter 75% sinkt. Der SGLTR erkennt also einen geringeren Anteil der annotierten Peaks als auf dem realen Datensatz. Dies dürfte darauf zurückzuführen sein, dass der SGLTR aus Zeitmangel nicht mehr auf den synthetischen Datensatz optimiert werden konnte und so die Intensitäten eines signifikan-

Methode	Klassifikationsrate	Durchnittliche Merkmalsanzahl
Manuelle Zentroiden	0,64	33
Hierarchisches Clustering	0.72	55
K-Means++ (K=55)	0.72	55
K-Means aus hierarch. Clustering	0.75	55
K-Means aus hierarch. Clustering und PCA	0.64	10
GMM aus hierarch. Clustering	0.74	19
Grid	0.65	400
Voronoi-Gebiete	0.58	55

Tabelle 9: Klassifikationsraten für unterschiedliche Verfahren zur Merkmalsextraktion.

ten Anteils der Peaks unterhalb der Schwellwerte des Regionfinders des SGLTR liegen.

So wird ein wesentlicher Unterschied zwischen den beiden Algorithmen deutlich: Während der PDSA ohne Anpassungen auf einen bestimmten Datensatz gute Ergebnisse erzielt, ist der SGLTR anfällig für Schwankungen in der Messungs-Charakteristik. In weiterführenden Arbeiten könnte mithilfe eines erweiterten Messungsgenerators der Einfluss verschiedener Modellparameter auf die Peakerkennungseigenschaften der Algorithmen genauer untersucht werden.

13.4 METHODEN DER MERKMALSEXTRAKTION

Es wurden die Klassifikationsraten für verschiedene Methoden zur Merkmalsextraktion gemessen, um zu ermitteln, welche Merkmalsextraktion am geeignetsten für Peaklisten ist.

In Tabelle 9 sind die Klassifikationsraten für verschiedene Herangehensweisen zur Merkmalsextraktion angegeben. Als Klassifikator wurde der Random Forest verwendet, da dieser im Durchschnitt besser als andere Klassifikatoren (siehe Abschnitt 13.5) abschneidet. Die Klassifikationsraten wurden auf dem Tee-Kaffee-Datensatz durch die Leave-One-Out-Methode gemessen und in der Tabelle 9 zusammengefasst. Die getesteten und dort abgebildeten Methoden zur Merkmalsextraktion waren:

- *Manuelle Zentroiden*: Die Peaks wurden manuell in Cluster eingeteilt, indem die PG nach eigenem Ermessen Zentroide an Peakbereichen markiert hat. Diese Methode dient nur als Vergleichsbasis für andere Merkmalsextraktionsverfahren, da es sich dabei um keine automatische Methode handelt.

- *Hierarchisches Clustering*: Durch hierarchisches Clustering (siehe Abschnitt 10.2.2) werden Zentroide ermittelt, die direkt zur Merkmalsbildung verwendet werden (siehe Abschnitt 10.3).

- *K-Means++*: Bei K-Means++ handelt es sich um eine Variante des K-Means-Algorithmus. Es wird zwar wie beim K-Means-

Algorithmus eine feste Anzahl von Zentroiden (hier: $K = 55$) berechnet (siehe Abschnitt 10.2.3), jedoch geschieht die Initialisierung der Zentroide probabilistisch in Abhängigkeit von den Abständen zu anderen Zentroiden. Dies verhindert, dass der Algorithmus in einem sehr ungünstigen lokalen Optimum konvergiert.

- *K-Means aus hierarchischem Clustering*: Diese Methode entspricht den Erläuterungen in Abschnitt 10.2.

- *K-Means aus hierarchischem Clustering kombiniert mit PCA*: Wie oben, jedoch wird zusätzlich mit der Hauptkomponentenanalyse (Principal Components Analysis, PCA) eine Dimensionsreduktion durchgeführt, um die Anzahl der Merkmale zu reduzieren. Bei der PCA werden diejenigen Merkmale ausgewählt, die die größte Varianz in ihrer Dimension aufweisen. Indem man diese am meisten streuenden Merkmale wählt, erhofft man sich eine Erhöhung der Unterscheidbarkeit zwischen den Klassen. In Tabelle 9 ist die Klassifikationsrate für eine Dimensionsreduktion auf 10 Hauptkomponenten (Merkmale) angegeben. Andere Anzahlen von Merkmalen führten in den Experimenten zu schlechteren Raten.

- *GMM aus hierarchischem Clustering*: Bei dieser Methode wird ein Mischverteilungsmodell (Gaussian Mixture Model, GMM) für die Peakmenge geschätzt. Ein GMM kombiniert mehrere Normalverteilungsmodelle und führt so ein probabilistisches („weiches“) Clustering durch. Zur Schätzung des GMM wird der Expectation-Maximization-Algorithmus ausgeführt. Hierarchisches Clustering wird verwendet, um die Anzahl der Normalverteilungen zu bestimmen und die Normalverteilungen geeignet zu initialisieren. Es ist anzumerken, dass bei dieser Methode keine Normalisierung der Peaks durchgeführt wurde, wie in Abschnitt 10.2.1 beschrieben. Aus diesem Grund besitzt das geschätzte GMM 19 Normalverteilungsmodelle (Merkmale),
während auf normalisierten Peaks bei dem hierarchischen Clustering 55 Normalverteilungsmodelle ermittelt werden konnten. Zur Bildung der Merkmale werden für jeden Peak die a-posteriori Wahrscheinlichkeiten bzgl. jeder Normalverteilungskomponente berechnet. Als Wert für Dimension k des extrahierten Vektors wird die Intensität des Peaks gesetzt, der die höchste a-posteriori Wahrscheinlichkeit für Normalverteilungskomponente k aufweist.

- *Grid*: Bei der Grid-Methode zur Merkmalsextraktion wird der Merkmalsraum der Peaks in ein 20×20-Gitter (Grid) unterteilt. Daraus wird ein 400-dimensionaler Vektor extrahiert, indem die durchschnittlichen Intensitäten der Peaks in den jeweiligen Kästchen des Gitters als Werte gesetzt werden.

- *Voronoi-Gebiete*: Als letzte Methode wurde die Merkmalsextraktion anhand der Voronoi-Gebiete aus dem K-Means-Clustering durchgeführt. Jeder Zentroid des Clusterings repräsentiert einen Cluster-Bereich (als Voronoi-Gebiet bezeichnet). Alle Peaks, die zu diesem Zentroid den geringsten Abstand besitzen, gehören zu

einem Cluster. Die Bildung der Merkmale aus einer Peakliste geschieht dann, indem für jedes Voronoi-Gebiet überprüft wird, ob ein Peak existiert, der sich in diesem Gebiet befindet. Ist dies der Fall, dann erhält der extrahierte Vektor die Intensität des Peaks als Merkmalswert (bei mehreren Peaks in einem Gebiet die durchschnittliche Intensität).

Tabelle 9 belegt, dass die Merkmalsextraktion mit der Kombination von K-Means mit hierarchischem Clustering die höchste Klassifikationsrate erzielt und für Peaklisten deshalb besser geeignet als die anderen Merkmalsextraktionsmethoden ist. Die Merkmalsextraktion durch GMM ist sicherlich auch interessant, da wenig Merkmale erzeugt werden und die Klassifikationsrate recht hoch ist.

13.5 KLASSIFIKATIONSRATEN DER ALGORITHMEN

Nach Auswahl einer Merkmalsextraktionsmethode wurde mit der Evaluation der Klassifikation begonnen. Wie bereits in Abschnitt 11.2 beschrieben, haben wir als Klassifikatoren Random Forest, Support Vector Machine (SVM) und k Nearest Neighbor (KNN) getestet. Die durchschnittliche Klassifikationsrate wurde mithilfe der Kreuzvalidierungsmethode Leave-One-Out ermittelt. Die Testläufe wurden mit der Revision `75d9d9dc4d3bc114d5c21aadd8456c7a2a75c69e` des Git-Repositories `git@projekte.itmc.tu-dortmund.de:in/git-pg572.git` ausgeführt.

Bis zur Festlegung des Seeds für den Random Forest Klassifikator wurden die Vergleichstests für die Klassifikationsrate mit zufälligen und nicht vorgegebenen Seeds ermittelt. Dadurch sind diese Testergebnisse nicht reproduzierbar und werden deshalb hier nicht gezeigt. Auf dieser fehlerhaften Grundlage wurde auch die Entscheidung für Random Forest als bevorzugtes Klassifikationsverfahren getroffen. SVM und KNN wurden im weiteren Verlauf zwar noch für Vergleiche herangezogen, aufgrund ihrer um bis zu 25% schlechteren Klassifikationsraten aber nicht eingehender untersucht. Deshalb werden sie an dieser Stelle auch nicht weiter betrachtet.

Bei der Auswahl eines Seeds für den Random Forest Klassifikator zur Reproduzierbarkeit wurde festgestellt, dass die durchschnittliche Klassifikationsrate je nach benutztem Seed sehr stark schwankt. Wenn ein Random Forest Klassifikator richtig angewendet wird, darf ein Seed sich jedoch nur minimal auf die durchschnittliche Klassifikationsrate auswirken. Um dieses Problem zu erforschen, haben wir für alle Datensätze eine Versuchsreihe aufgestellt, indem wir für jeden Datensatz die Auswirkungen der Seeds 1 bis 64 auf die durchschnittliche Klassifikationsrate betrachtet haben.

In Abb. 69 ist die durchschnittliche Klassifikationsrate in Abhängigkeit vom verwendeten Seed für den Tee-Kaffee-Datensatz aufgetragen. Die dabei verwendeten Seeds sind ganzzahlig und es wurden der Übersichtlichkeit halber nur die Seeds 1 bis 32 dargestellt.

Beim Tee-Kaffee-Datensatz beträgt die Differenz zwischen der niedrigsten und der höchsten durchschnittlichen Klassifikationsrate 14.5% (56,5% mit Seed 13 und 71,0% mit Seed 18), wenn der SGLTR zur Er-

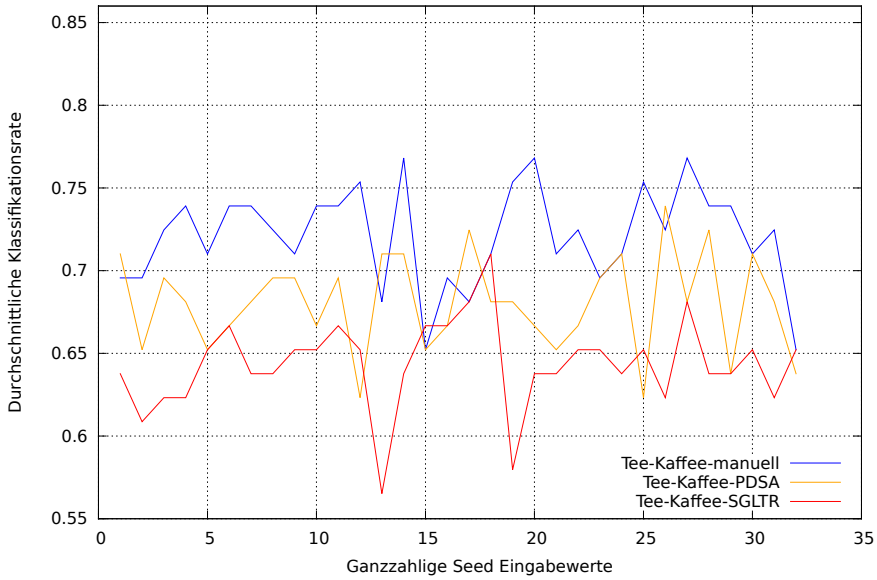


Abb. 69: Einfluss des Seeds auf die durchschnittliche Klassifikationsrate

stellung der Peaklisten benutzt wurde. Abb. 70 zeigt analog hierzu die durchschnittliche Klassifikationsrate in Abhängigkeit vom verwendeten Seed für den Salz-Pfeffer-Datensatz. Auch hier wurden nur die Seeds 1 bis 32 dargestellt. Beim Salz-Pfeffer-Datensatz beträgt die höchste Differenz der durchschnittlichen Klassifikationsraten beim SGLTR 12,3% (67,7% mit Seed 37 und 80,0% mit Seed 9).

Um diese Differenzen zu verkleinern, haben wir die benutzte Baumanzahl des Random Forest variiert und die Baumanzahlen 1, 2, 15, 60 und 100 auf den Seeds 1 bis 64 ausprobiert. Hierbei haben wir festgestellt, dass die durchschnittliche Klassifikationsrate überraschenderweise vollständig unabhängig von der jeweils verwendeten Baumanzahl ist.

Um festzustellen, ob der von uns verwendete Random Forest Klassifikator richtig arbeitet, haben wir mit ihm anschließend den bekannten und erprobten Iris-Datensatz klassifiziert. Der Iris-Datensatz wird im Paper „The use of multiple measurements in taxonomic problems“ von Ronald A. Fisher [18] beschrieben. Er enthält insgesamt 150 Stichproben, die in drei Klassen zu je 50 Stichproben unterteilt sind. Jede der Stichproben besteht aus vier Merkmalen und einer Klassenzugehörigkeit. Mit dem Iris-Datensatz haben wir ebenfalls getestet, welchen Einfluss verschiedene Baumanzahlen auf die durchschnittlichen Klassifikationsraten haben im Seed-Bereich von 1 bis 64. Wie erwartet hatte hier die Anzahl der Bäume einen großen Einfluss auf die Schwankungsbreite der durchschnittlichen Klassifikation.

In Abb. 71 sind die durchschnittlichen Klassifikationsraten in Abhängigkeit vom verwendeten Seed für verschiedene Baumanzahlen dargestellt. Dort ist zu erkennen, dass bei einem Baum (blau) die Schwankungsbreite noch ziemlich groß ist. Mit zunehmender Baumanzahl werden die Schwankungen sichtbar geringer und ab circa 30 Bäumen (orange) hat der verwendete Seed keinen wesentlichen Einfluss mehr auf die durchschnittliche Klassifikationsrate.

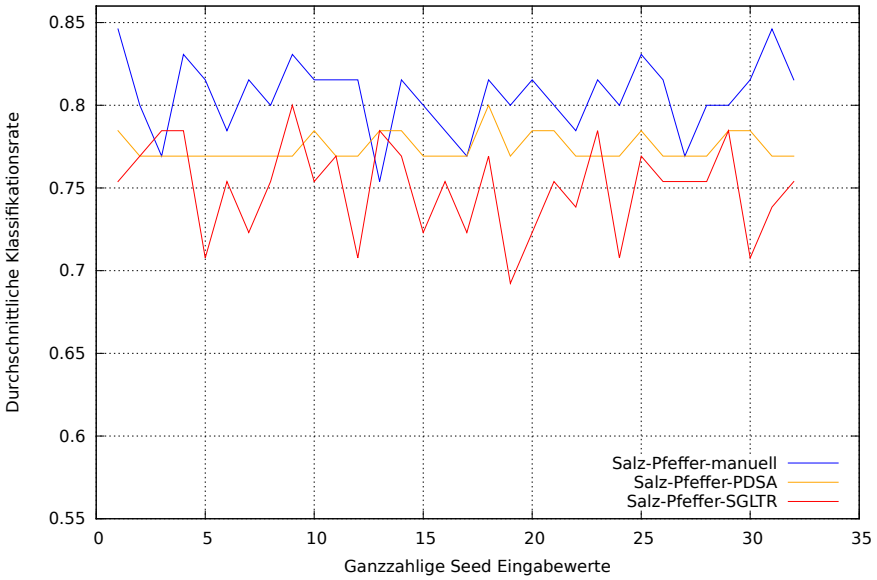


Abb. 70: Einfluss des Seeds auf die durchschnittliche Klassifikationsrate

Parameter-Satz für Random Forest									
criterion =		„entropy“				„gini“			
max_features =		„auto“		None		„auto“		None	
		ø	σ	ø	σ	ø	σ	ø	σ
Tee-Kaffee	manuell	0,719	0,028	0,707	0,026	0,707	0,030	0,700	0,028
	SGLTR	0,647	0,027	0,661	0,024	0,638	0,032	0,671	0,028
	PDSA	0,683	0,028	0,668	0,036	0,675	0,037	0,656	0,033
Salz-Pfeffer	manuell	0,803	0,022	0,778	0,022	0,800	0,022	0,785	0,013
	SGLTR	0,744	0,027	0,741	0,019	0,741	0,025	0,748	0,018
	PDSA	0,775	0,008	0,777	0,009	0,773	0,007	0,774	0,009
Spaltendurchschnitt		0,728	0,024	0,722	0,023	0,723	0,026	0,722	0,022

Tabelle 10: Ermittlung der besten Parameter für den Random Forest mittels der erzielten durchschnittlichen Klassifikationsraten (ø) und deren Standardabweichungen (σ)

Die Boxplots in Abb. 72 zeigen zunächst, dass unsere Datensätze deutlich schlechter zu klassifizieren sind als der Iris-Datensatz. Am Iris-Datensatz auf der rechten Seite ist weiterhin zu erkennen, dass dort bei einem Baum die Schwankungsbreite noch groß ist, aber mit zunehmender Baumanzahl die Schwankungen deutlich abnehmen. Dieses Verhalten hätten wir auch bei unseren Daten (links) erwartet, aber hier hat die Anzahl der verwendeten Bäume im Random Forest Klassifikator keine Auswirkungen auf die durchschnittlichen Klassifikationsraten. Möglicherweise liegt die Ursache hierfür darin, dass wir aus unseren Messdaten relativ zur Anzahl der Messungen zu viele Merkmale extrahieren. Im weiteren Verlauf der Tests haben wir uns bei der Klassifikation unserer IMS-Daten für einen Random Forest mit 30 Bäumen entschieden, da wir von einer möglichen Verringerung der extrahierten Merkmale profitieren wollten und mit den Iris-Daten 30 Bäume ein guter Kompromiss aus Laufzeit und Genauigkeit waren.

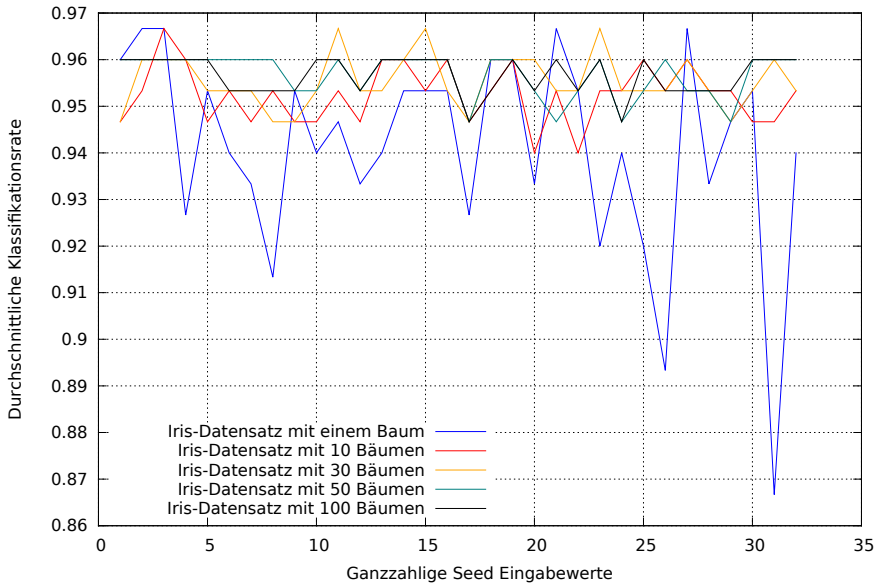


Abb. 71: Durchschnittliche Klassifikationsraten in Abhängigkeit von der Anzahl der Bäume im Random Forest

		Klassifikationsverfahren					
		Random Forest				SVM	KNN
		ø	σ	max	min		
Tee-Kaffee	manuell	0,719	0,028	0,783	0,652	0,565	0,783
	SGLTR	0,647	0,027	0,710	0,565	0,565	0,580
	PDSA	0,683	0,028	0,754	0,623	0,565	0,594
Salz-Pfeffer	manuell	0,803	0,022	0,846	0,738	0,754	0,769
	SGLTR	0,744	0,027	0,800	0,677	0,523	0,662
	PDSA	0,775	0,008	0,800	0,769	0,785	0,600
Spaltendurchschnitt		0,728	0,024	0,782	0,671	0,626	0,665

Tabelle 11: Vergleich der Klassifikationsverfahren anhand ihrer durchschnittlichen Klassifikationsraten

Um die bestmögliche Parametrisierung für den Random Forest zu finden, haben wir für unsere IMS-Daten verschiedene Parametersätze evaluiert, welche in der Tabelle 10 dargestellt sind.

Die durchschnittlich besten Klassifikationsraten (0,728) über alle Datenreihen wurden mit dem Parametersatz *criterion* = *„entropy“* und *max_features* = *„auto“* erreicht. Deshalb wird dieser Parametersatz für unsere Klassifizierung benutzt. In Tabelle 11 werden die Ergebnisse dieses Parametersatzes mit den Ergebnissen der SVM- und KNN-Klassifikatoren verglichen. Am Spaltendurchschnitt ist erkennbar, dass der Random Forest Klassifikator im Durchschnitt um circa 10% bessere Ergebnisse erzielt, als die SVM- und KNN-Klassifikatoren.

Zur Reproduzierbarkeit der Klassifikationsergebnisse haben wir den Seed 42 ausgewählt [14] und die Ergebnisse der Random Forest Klassifikation mit diesem Seed in Tabelle 12 festgehalten.

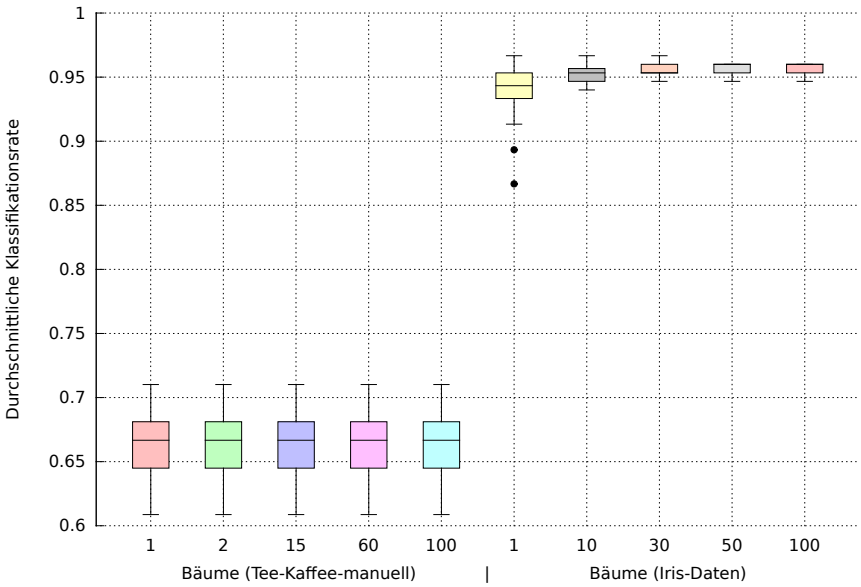


Abb. 72: Vergleich der mittels Random Forest erzielten durchschnittlichen Klassifikationsraten zwischen IMS-Daten und Iris-Daten

Datensätze		durchschnittliche Klassifikationsrate
Tee-Kaffee	manuell	0,725
	SGLTR	0,652
	PDSA	0,710
Salz-Pfeffer	manuell	0,815
	SGLTR	0,723
	PDSA	0,769

Tabelle 12: Für Random Forest mit Seed 42 ermittelte durchschnittliche Klassifikationsraten

13.6 LAUFZEITEN AUF DEM PI

Zur Bewertung der Effizienz der implementierten Algorithmen wurden Benchmarks mit verschiedenen Einstellungen durchgeführt. Zum einen wurden die Laufzeiten mit direkt vom Client geladenen Daten gemessen (Quickrun). Zum anderen wurde der Einfluss des Netzwerks untersucht, da in Abschnitt 9.1.2 eine gewisse CPU-Auslastung bei der Datenübertragung festgestellt wurde. Dabei wurde die Zeit zwischen Spektren von den üblichen 100 ms auf bis zu 5 ms verringert, womit die Netzwerkverbindung fast vollständig ausgelastet war. Die Algorithmen wurden auf Daten mit geringer Auflösung ausgeführt. Da über das Netzwerk hochaufgelöste Daten versendet werden, fand die Reduktion auf ein Fünftel der ursprünglichen Daten clientseitig statt. Somit stand zur Bearbeitung eines Spektrums das Fünffache der Wartezeit zwischen Spektren zur Verfügung. Die Ergebnisse sind in Tabelle 13 zu sehen.

Es zeigt sich, dass der mit nur einfachen Operationen arbeitende PDSA mit ohnehin sehr geringen Laufzeiten kaum vom Netzwerk beeinflusst wird. SGLTR hingegen benötigt aufgrund der komplexen Fal-

Algorithmus	ms/Spektrum
PDSA (ohne Netzwerk)	0,33
PDSA (5 ms/Spektrum)	0,38
SGLTR (ohne Netzwerk)	2,03
SGLTR (100 ms/Spektrum)	5,72
SGLTR (10 ms/Spektrum)	6,31
SGLTR (5 ms/Spektrum)	11,94

Tabelle 13: Durchschnittliche Laufzeiten der Algorithmen auf dem Pi

tungsberechnungen auch ohne Verwendung des Netzwerks ein Vielfaches an CPU-Zeit und wird durch den Netzwerk-Overhead deutlich langsamer. Trotzdem bleiben beide Algorithmen auch im langsamsten Fall des Empfangs eines Spektrums alle 5 ms weit unter dem Zeitlimit von in diesem Fall aufgrund der auf ein Fünftel verringerten Auflösung 25 ms. Bei realistischen Bedingungen stehen beim Empfang eines Spektrums mit hoher Auflösung alle 100 ms sogar 500 ms zur Verarbeitung der über fünf Spektren gemittelten Daten zur Verfügung.

Abb. 73 zeigt einen Boxplot der Laufzeiten einer Messung über 1200 Spektren. Das Auftreten der Ausreißer innerhalb einer Messung ist unregelmäßig und variiert von Durchlauf zu Durchlauf. Dies kann wahrscheinlich durch die Verwendung eines Multitasking-Betriebssystems ohne Scheduling-Garantien erklärt werden.

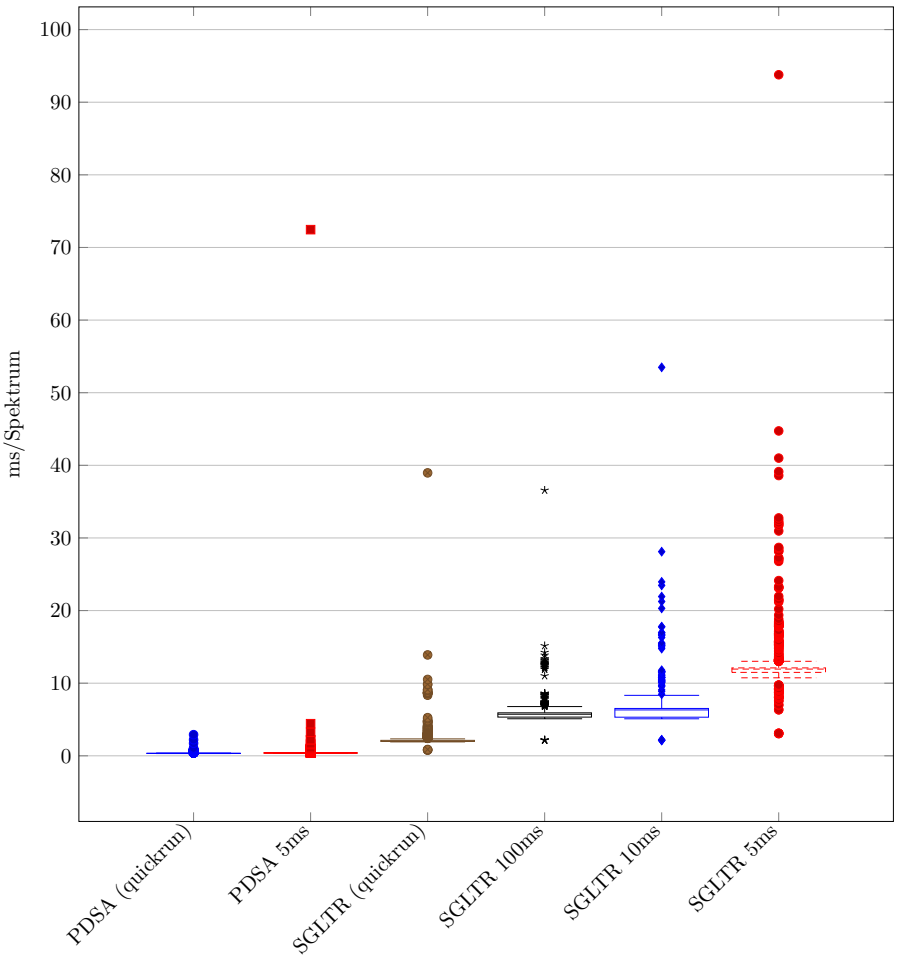


Abb. 73: Boxplot der Laufzeiten

14 FAZIT

In diesem Kapitel wird ein Fazit über die Arbeit der Projektgruppe gezogen. Dazu betrachten wir in Abschnitt 14.1 erreichte Ziele der Projektgruppe. Anschließend erläutern wir in Abschnitt 14.2, welche zusätzlichen Ergebnisse aufbauend auf dieser Arbeit erreicht werden können.

14.1 ERGEBNISSE

Die beiden wesentlichen Ziele der Projektgruppe wurden erreicht: Wir haben zwei Algorithmen entwickelt, die online unter den gegebenen Realzeitbedingungen auf dem Raspberry Pi Peaks einer hochaufgelösten Messung eines MCC/IMS-Geräts erkennen können. Zur Klassifikation können wir die ausgegebenen Peaklisten mithilfe einer angepassten Merkmalsextraktion und einem Klassifikationsverfahren klassifizieren.

Zur Peakerkennung erarbeiteten wir zwei verschiedene Konzepte: Das erste Verfahren, SGLTR, basiert auf einem angepassten Savitzky-Golay Filter, der den Laplace-Operator annähert und die Werte über einem Schwellwert zu Regionen zusammenfasst. Dieser Ansatz erkennt Peaks anhand der Krümmung im Graph der Messwerte. Das zweite Verfahren, PDSA, basiert auf Vergleich- und Summen-Operationen und ist darauf optimiert schnell ein Spektrum verarbeiten und potenzielle Peaks finden zu können.

Bei der Evaluation der Peakerkennungsleistung erzielten beide Algorithmen vergleichbare Ergebnisse. Mit einem F1-Maß von jeweils über 70% können sie als erfolgreich bezeichnet werden. Dabei ist das F1-Maß das harmonische Mittel aus Genauigkeit und Sensitivität in Bezug auf die Gold-Standard-Peaklisten.

Um Einflüsse des Rauschens in den Messwerten zu reduzieren, fließen bei beiden Verfahren die Messwerte innerhalb eines Berechnungsfensters mit ein: Beim SGLTR in Form von einer gewichteten Summe in der Faltungsberechnung, beim PDSA nur entlang des aktuellen Spektrums.

Beide Verfahren arbeiten mit Bereichen, in denen Peaks vermutet werden und die beim Eintreffen eines neuen Spektrums aktualisiert werden: Beim PDSA werden die Außengrenzen von potentiellen Peaks durch steigende und fallende Flanken innerhalb nur eines Spektrums detektiert. Der SGLTR berechnet zusammenhängende Regionen der Punkte über dem Schwellwert.

Der PDSA benötigt im Vergleich weniger Rechenoperationen und weist eine geringere Laufzeit auf. Derzeit erfolgt die Ausgabe der gefundenen Peaks jedoch erst am Ende der Messung.

Ein wesentlicher Unterschied ist, dass der SGLTR auf den Datensatz angepasst werden muss, während der PDSA universell einsetzbar ist. Dafür bietet der SGLTR mit seinen vielen Einstellmöglichkeiten mehr Potential zur gezielten Optimierung.

Beide Verfahren finden Peaks, die nahe an den von Hand annotierten Peaks liegen (F1-Maß von rund 75%). Die Unterschiede sind auch dadurch zu erklären, dass es nicht immer eindeutig ist, was ein Peak ist und was nicht.

Zur Vorbereitung der Klassifikation wurden Methoden entwickelt, um aus einer Peakliste variabler Größe einen Merkmalsvektor fester Größe zu gewinnen. Das erfolgreichste Verfahren war, Peaks mithilfe der Fängerfunktion zu zuvor gewählten Zentroiden zuzuordnen. Die Klassifikation wurde im Anschluss mit bekannten Verfahren durchgeführt, wobei der Random Forest die besten durchschnittlichen Klassifikationsraten aufwies.

Die Evaluation der Klassifikationsverfahren gestaltete sich aufgrund der geringen Menge der verfügbaren Messdaten als schwierig. Die durchschnittlichen Klassifikationsraten bewegten sich für alle Verfahren um den Bereich von 70% und sind somit nur bedingt erfolgreich. Die durchschnittlichen Klassifikationsraten für die Peaklisten, die unsere Algorithmen ausgeben, sind nicht signifikant schlechter als für die Gold-Standard-Peaklisten und schneiden teilweise gleich gut ab.

Einige ursprüngliche Ziele der Projektgruppe konnten nicht umgesetzt werden und mussten angepasst werden. So war es beispielsweise nicht möglich, den Raspberry Pi zur Auswertung direkt an ein MCC/IMS-Messgerät anzuschließen. Um die Bedingungen dennoch zu simulieren, wurde eine Server-Client-Architektur von der Projektgruppe erarbeitet.

Zusammenfassend lässt sich sagen, dass die wesentlichen Ziele der Projektgruppe erreicht wurden.

14.2 AUSBLICK

Auch nach Abschluss der Implementierungs-Phase blieb noch Raum für Optimierung der entwickelten Algorithmen. Insbesondere kann in Zukunft die Parameter-Optimierung, welche nur von Hand erfolgte, systematisch durchgeführt werden.

Bei zukünftiger Optimierung ist es sinnvoll, auf Klassifikationsraten statt auf Peakerkennungsraten zu optimieren. Dies lässt sich damit begründen, dass Peakerkennungsraten nur eine bedingte Aussagekraft über das eigentliche Ziel, der optimierten Klassifikationsrate, haben. Da das Ermitteln durchschnittlicher Klassifikationsraten eine hohe Laufzeit aufweist und die Paramater-Optimierung des SGLTR bereits bei Peaklistenbewertungen aufwendig ist, wird eine solche Optimierung einen hohen Ressourcenaufwand bedeuten.

In Zukunft wäre es interessant zu untersuchen, ob und in welcher Art sich die berechneten Peaks unterscheiden. Hierzu könnte man die errechneten nicht mit den manuell annotierten Peaks, sondern untereinander vergleichen. Sofern die Ergebnisse in bestimmten Bereichen der Messung oder unter bestimmten Umgebungsbedingungen jeweils besser sind, könnte man darüber nachdenken, wie sich die Ansätze zu einem besseren Programm kombinieren lassen. Um praxisnahe Ergebnisse der Peakerkennungsalgorithmen zu erhalten ist es erforderlich, dass

der Raspberry Pi an ein IMS-Gerät angeschlossen wird und die Algorithmen Rohdaten als Eingabemenge bekommen.

Auch bei der Merkmalsextraktion und Klassifikation besteht noch Verbesserungspotential. Mit einem überwachten oder ensemble Clustering könnte die Merkmalsextraktion verbessert werden. Bei der Klassifikation könnten zusätzliche Informationen zu den Peaks, wie z. B. die Größe der Bounding Box, berücksichtigt werden. Allein mit größeren, zuverlässigen Messreihen lassen sich vermutlich bessere Klassifikationsraten erzielen und außerdem die Verfahren besser miteinander vergleichen.

Für die Zukunft wäre es interessant, Algorithmen zu entwickeln, die auf das Erkennen von speziellen Substanzen spezialisiert sind.

Insgesamt wurden im Rahmen der Projektgruppe funktionierende Verfahren entwickelt, welche die Ziele der Projektgruppe erfüllen. Aufbauend auf diesen Ergebnissen können in zukünftigen Arbeiten noch bessere Ergebnisse für Online-Verfahren auf MCC/IMS-Messungen erzielt werden.

A

ANHANG

In diesem Anhang befinden sich technisch weiterführende Informationen, die für das Verständnis des Berichts an sich nicht erforderlich sind. Im Hauptteil des Berichts wurde an manchen Stellen nur ein ausreichender Überblick über einzelne Sachverhalte gegeben. Die hier enthaltenen Abschnitte erklären diese Sachverhalte genauer und befinden sich in analoger Reihenfolge zur Struktur des Hauptteils. Diese Abschnitte sind nicht als zusammenhängende Lektüre gedacht, sondern als Nachschlagewerk für den Leser, der weitere Details zu den einzelnen Abschnitten erfahren möchte.

A.1 IMS-CSV HEADER

Im Folgenden werden tabellarisch alle Metadaten aufgeführt, welche im Header des IMS-CSV Dateiformats enthalten sind. In der ersten Spalte der Tabelle steht die Zeilennummer in der IMS-CSV Datei. Ungenutzte Zeilen sind dementsprechend nicht aufgeführt.

#	Bezeichnung	Datentyp	Erklärung
1	data type	Text	Datentyp je nach Anwendung, z.B. IMS raw data, exsc (external sampling control), ...
2	version	Text	Eingesetzte Messsoftware z.B. VOCan - v2.7 (2013-04-10)
3	template version	Text	Version des Dateiformats, z.B. 0.3
4	AD-board type	Text	Typ des AD-Wandlers, z.B. usbADC3
5	ser.-no.	Text	Seriennummer des AD-Wandlers, z.B. BD01
7	date	Datum	MM/DD/YYYY, Datum der Messung
8	time	Uhrzeit	hh:mm:ss, Startzeit der Messung
9	file	NNNN_YYMM DDhhmm_ims.csv	Dateiname, wobei NNNN die Seriennummer ist
12	SAMPLE INFORMATION	—	Titelzeile
14	sample type	sample, reference, ublank, test signal, series	Art der Messanwendung
15	sample ID	20 char	Sample ID, wird manuell vergeben, z.B. PG_AL_1
16	comment	Text	Kommentar, wird manuell vergeben, z.B. EM-eukal
17	location	4 char	Ort der Messung (kurz), z.B. B&S
18	location name	Text	Ort der Messung, z.B. B&S
19	height ASL / m	Integer	Höhe in Metern über dem Meeresspiegel, z.B. 90
20	total data acquisition time / s	Real	Dauer der Messung in Sekunden, z.B. 621.567
23	IMS - INFORMATION	—	Titelzeile
25	operator	2 char	Operator Kürzel, z.B. PG
26	operator name	Text	Operator, z.B. PG Demo
27	IMS	4 char	Gerätename, z.B. BD01 (offenbar identisch zu Zeile 5)
29	K0 RIP positive / cm^2/Vs	Real	Ionenmobilität K_0 in $\frac{cm^2}{Vs}$, bei der im positiven Messmodus der Reaktionsionenpeak (RIP) auftritt, z.B. 2.02

#	Bezeichnung	Datentyp	Erklärung
30	K0 RIP negative / cm^2/Vs	Real	Ionenmobilität K_0 in $\frac{cm^2}{Vs}$, bei der im negativen Messmodus der Reaktionsionenpeak (RIP) auftritt, z.B. 2.22
31	polarity	positive, negative	Messmodus
32	grid opening time / us	Integer	Öffnungzeit t_{grid} des Ionengitters in µs, z.B. 300
34	pause / s	Integer	zeitlicher Abstand zwischen zwei Spektren in Sekunden, z.B. 0
35	tD interval (corr.) / ms from	Real	Start des Aufzeichnungsintervalls bei korrigierter Driftzeit t_d in Millisekunden, z.B. -0.150
36	tD interval (corr.) / ms to	Real	Ende des Aufzeichnungsintervalls bei korrigierter Driftzeit t_d in Millisekunden, z.B. 49.846
37	1/K0 interval / Vs/cm^2 from	Real	Start des Aufzeichnungsintervalls bei inverser Ionenmobilität $\frac{1}{K_0}$ in $\frac{Vs}{cm^2}$, z.B. -0.00454
38	1/K0 interval / Vs/cm^2 to	Real	Ende des Aufzeichnungsintervalls bei inverser Ionenmobilität $\frac{1}{K_0}$ in $\frac{Vs}{cm^2}$, z.B. 1.50962
39	no. of data points per spectra	Integer	Anzahl der Datenpunkte pro Spektrum, z.B. 12500
40	no. of spectra	Integer	Anzahl der Spektren, z.B. 6000
41	no. averaged spectra	Integer	Anzahl der Spektren, über die gemittelt wird, z.B. 1
42	baseline / signal units	Integer	Basislinie (Offset) in Spannungsstufen, z.B. 2011 Der Verwendete AD-Wandler hat eine Auflösung von 4096 Spannungsstufen zwischen 0 V und 10 V. In diesem Bereich liegen die real gemessenen Werte. Die verwendete Messsoftware bestimmt während der Messung die sog. Basislinie, einen Wert, der vor als Offset vor dem Speichernvon jedem gemessenen Wert abgezogen wird.
43	baseline / V	Real	Basislinie in Volt (Produkt aus Spannungsstufe und Spannungsdifferenz zwischen zwei Stufen), z.B. 4.9505816
44	V / signal unit	Real	Differenz zwischen zwei Spannungsstufen des AD-Wandlers in Volt, z.B. 0.0024617512
46	drift length / mm	Integer	Länge der Driftkammer in mm, z.B. 120.0
47	HV / kV	Real	Hochspannung, die an der Driftkammer anliegt in kV, z.B. 4.38
48	amplification / V/nA	Real	Verstärkung in $\frac{V}{nA}$, z.B. 0.0
50	drift gas	Text	Driftgas, z.B. N2
51	drift gas flow / mL/min	Integer	Durchflussrate des Driftgases in $\frac{ml}{min}$, z.B. 100
52	sample gas	Text	Probengas, z.B. N2
53	sample gas flow / mL/min	Integer	Durchflussrate des Probengases in $\frac{ml}{min}$, z.B. 101
54	carrier gas	Text	Trägergas, z.B. N2
55	carrier gas flow / mL/min	Integer	Durchflussrate des Trägergases in $\frac{ml}{min}$, z.B. 149
56	pre-separation type	Text	Vorseparationsverfahren (MCC, GC, ...), bei unseren Messungen: MCC OV5 (Multikapillarsäule mit OV5-Gel beschichtet)
57	pre-separation T / deg C	Real	Temperatur der Multikapillarsäule in °C, z.B. 40.0; OK
58	sample loop T / deg C	Real	Temperatur des Probenkreislaufs in °C, falls - wie bei unseren Messungen - ein solcher verwendet wird z.B. 40.0; OK
59	sample loop volume / mL	Real	Volumen des Probenkreislaufs in ml, z.B. •
61	ambient T source	sensor, manual	Datenquelle für Umgebungstemperatur, bei unseren Messungen: manual
62	ambient T / deg C	Real	Umgebungstemperatur in °C, bei unseren Messungen ist dieser Wert offensichtlich falsch: -9999.9
63	ambient T x^2	Real	Umrechnungsfaktor für Temperatursensor
64	ambient T x^1	Real	Umrechnungsfaktor für Temperatursensor
65	ambient T x^0	Real	Umrechnungsfaktor für Temperatursensor
66	ambient T x^-1	Real	Umrechnungsfaktor für Temperatursensor
67	ambient T x^-2	Real	Umrechnungsfaktor für Temperatursensor

#	Bezeichnung	Datentyp	Erklärung
68	ambient p source	sensor, manual	Datenquelle für Umgebungsdruck, bei unseren Messungen: manual
69	ambient p / hPa	Real	Umgebungsdruck in hPa, bei unseren Messungen ist dieser Wert offensichtlich falsch: -9999.9
70	ambient p x^2	Real	Umrechnungsfaktor für Drucksensor
71	ambient p x^1	Real	Umrechnungsfaktor für Drucksensor
72	ambient p x^0	Real	Umrechnungsfaktor für Drucksensor
73	ambient p x^-1	Real	Umrechnungsfaktor für Drucksensor
74	ambient p x^-2	Real	Umrechnungsfaktor für Drucksensor
76	6-way valve	manual, auto	Steuerung des Probenkreislaufs, falls - wie bei unseren Messungen - ein solcher verwendet wird, bei uns: auto
79	EXTERNAL SAMPLING CONTROL	—	Titelzeile
81	control status	off, on	Externe Kontrolle der Messung z.B. durch SpiroScout, bei unseren Messungen i.d.R: off Dieser Abschnitt ist für uns somit nicht relevant.
82	control zero / signal units	Integer	Basislinie, siehe Zeile 42!
83	control zero / V	Real	Basislinie in Volt, siehe Zeile 43!
84	control threshold / signal units	Integer	Grenzwert für Start der Messung in Spannungsstufen
85	control threshold / V	Real	Grenzwert für Start der Messung in Volt
86	control threshold2 / signal units	Integer	Grenzwert für Ende der Messung in Spannungsstufen
87	control threshold2 / V	Real	Grenzwert für Ende der Messung in Volt
88	control sampling time / s	Integer	Messdauer(?)
89	control variable	Text	Kontrollvariable(?), z.B. Flow
90	control dimension	Text	Dimension der Kontrollvariablen(?)
91	control x^2	Real	Umrechnungsfaktor für Kontrollvariable
92	control x^1	Real	Umrechnungsfaktor für Kontrollvariable
93	control x^0	Real	Umrechnungsfaktor für Kontrollvariable
94	control x^-1	Real	Umrechnungsfaktor für Kontrollvariable
95	control x^-2	Real	Umrechnungsfaktor für Kontrollvariable
98	STATISTICS	—	Titelzeile
100	RIP detection	enable, disbale	Automatische Erkennung des RIPs, bei unseren Messungen: enabled
101	tD (RIP corr.) / ms	Real	korrigierte Driftzeit t_d in ms, bei der der RIP auftritt, z.B. 16.346
102	1/K0 (RIP) / Vs/cm^2	Real	Inverse Ionenmobilität $\frac{1}{K_0}$ in $\frac{Vs}{cm^2}$, bei der der RIP auftritt, z.B. 0.4950495
103	K0 (RIP) / cm^2/Vs	Real	Ionenmobilität K_0 in $\frac{cm^2}{Vs}$, bei der der RIP auftritt, z.B. 2.0199999939799307
104	SNR (RIP)	Real	Signal-Rausch-Verhältnis des RIPs, z.B. 200.83836
105	WHM (RIP) / Vs/cm^2	Real	„Breite“ des RIPs auf halber Höhe des Maximums in $\frac{Vs}{cm^2}$, z.B. 0.014557707
106	res. power (RIP)	Real	Auflösungsvermögen(?) des RIPs (Inverse Ionenmobilität des RIPs / Breite des RIPs), z.B. 34.00365
108	tD (preRIP corr.) / ms	Real	korrigierte Driftzeit t_d in ms, bei der der Pre RIP auftritt, z.B. 16.346
109	1/K0 (preRIP) / Vs/cm^2	Real	Inverse Ionenmobilität $\frac{1}{K_0}$ in $\frac{Vs}{cm^2}$, bei der der Pre RIP auftritt, z.B. 0.49504948
110	K0 (preRIP) / cm^2/Vs	Real	Ionenmobilität K_0 in $\frac{cm^2}{Vs}$, bei der der Pre RIP auftritt, z.B. 2.0200001155853338
111	SNR (preRIP)	Real	Signal-Rausch-Verhältnis des Pre RIPs, z.B. 0.1339761
112	WHM (preRIP) / Vs/cm^2	Real	„Breite“ des Pre RIPs auf halber Höhe des Maximums in $\frac{Vs}{cm^2}$, z.B. 0.11715223
113	res. power (preRIP)	Real	Auflösungsvermögen(?) des Pre RIPs (Inverse Ionenmobilität des Pre RIPs / Breite des Pre RIPs), z.B. 4.2603364
115	signal RIP / V	Real	„Höhe“ des RIPs in Volt, z.B. -9.950582
116	signal preRIP / V	Real	„Höhe“ des Pre RIPs in Volt, z.B. 0.0
117	RIP / preRIP	Real	Verhältnis von RIP zu Pre RIP, z.B. -Infinity

#	Bezeichnung	Datentyp	Erklärung
120	Fims / cm^2/kV	Real	Konstante des Messgeräts $F_{ims} = K_0 \cdot t_d$ in $\frac{cm^2}{kV}$, z.B. 33.01892

Tabelle 14: Header des IMS-CSV Dateiformats, basierend auf den Arbeiten von Vautz et al. [37] und Baumbach [6].

A.2 HARDWAREAUSSTATTUNG DES RASPBERRY PI

Der Raspberry Pi basiert auf einem System on a Chip, dem Broadcom BCM2835. Darin enthalten sind ein mit 700 MHz getakteter ARMv6-Kern und eine VideoCore 4 GPU mit 250 MHz. Der ARM-Kern ist mit einem jeweils 16 KiB großen L1-Daten- und Instruktions-Cache ausgestattet. Ein eigentlich zur GPU gehörender 128 KiB großer L2-Cache lässt sich dem ARM-Kern zuordnen. Das von uns genutzte Modell B des Raspberry Pi verfügt über 512 MiB RAM, dessen Aufteilung zwischen CPU und GPU konfigurierbar ist.

Das Gerät bietet zwei USB 2.0-Schnittstellen, einen 10/100 MBit Ethernet-Port und Video-Ausgabe über HDMI und Composite. Eine Audio-Ausgabe ist sowohl über HDMI als auch über eine Stereo-Klinken-Buchse möglich. Die Stromversorgung erfolgt über einen Micro-USB-Port, an den ein USB-Netzteil angeschlossen werden kann. Außerdem soll darüber eine Spannungsversorgung mit Batterien möglich sein.

Neben diesen Standard-Ports verfügt der Raspberry Pi über zwei Anschlüsse, an denen ein spezielles Kamera-Modul und ein DSI-Display betrieben werden können. Zum Hardware-Debugging sind die JTAG-Anschlüsse des VideoCore und des LAN-Chips über eine unbestückte Stiftleiste erreichbar. Für die Ansteuerung externer Hardware bietet der Raspberry Pi zudem einen 26-poligen General Purpose IO-Anschluss. Firmware und Betriebssystem lädt der Raspberry Pi beim Einschalten von einer SD-Karte.

A.3 ARM-CHIPSATZ

Dieser Abschnitt stellt den ARM-Chipsatz vor, dabei basieren alle Informationenen in diesem Abschnitt, soweit diese nicht anders gekennzeichnet sind, auf der ARM Ltd. Webseite (http://www.arm.com) und Rick Murrays ARMwiki (http://www.heyrick.co.uk/armwiki/The_ARM_family). Die ARM-Architektur wurde 1983 von der Acorn Computers Ltd. entwickelt. ARM steht dabei für Acorn RISC Machine. Am 26. April 1985 wurde der erste funktionierende Prototyp der ARMv1 Architektur entwickelt. Der erste Prozessor für den produktiven Einsatz wurde ab 1986 hergestellt und basierte auf der ARMv2 Architektur. Stand der Technik ist die ARMv7 Architektur und die ARMv8 Architektur ist für 2014 angekündigt.

A.3.1 *RISC*

Die Abkürzung RISC steht für Reduced Instruction Set Computers. Alle Informationen in diesem Abschnitt basieren auf dem Buch „Computer

architecture a quantitative approach“ [24]. Das Ziel einer RISC-Architektur ist eine hohe Ausführungsgeschwindigkeit, welche durch die folgenden Maßnahmen erreicht wird: Eine geringe Anzahl interner Zyklen pro Befehl, einfache Befehle, Fließbandverarbeitung und ein CPI-Wert der kleiner oder gleich eins ist. Unter dem CPI-Wert (cycles per instruction) einer Menge von Maschinenbefehlen versteht man die mittlere Anzahl interner Bus-Zyklen pro Maschinenbefehl.

Eine RISC-Architektur besitzt die folgenden Eigenschaften: Eine feste Befehlswortlänge, eine LOAD/STORE-Architektur, einfache Adressierungsarten, eine „semantische Lücke“ zwischen den Hochsprachen und den Assemblerbefehlen und sie sind rein in Hardware realisierbar. Das Gegenstück zu RISC sind CISC-Architekturen, welche meistens einen CPI-Wert von zwei oder höher besitzen. Auf die CISC-Architekturen wird hier nicht näher eingegangen, da diese für uns nicht relevant sind.

A.3.2 *Raspberry Pi ARM-Prozessor*

Der Prozessor des Raspberry Pis gehört zur 32-Bit ARMv6 Architektur. Alle Informationen in diesem Abschnitt basieren auf dem ARM Architecture Reference Manual [1] und dem ARM1176JZF-S Technical Reference Manual [2].

Im Raspberry Pi ist der SoC (System on a Chip) Broadcom BCM2835 verbaut. Dieser enthält den ARM-Prozessor ARM1176JZF-S aus der ARM11 Familie.

Auf einige wichtige Funktionsmerkmale des Prozessors wird etwas ausführlicher eingegangen. Der ARM1176JZF-S Prozessor enthält drei verschiedene Befehlssätze, den 32-Bit ARM Befehlssatz, den 32/16-Bit Thumb(2) Befehlssatz und den 8-Bit Jazelle Befehlssatz.

Der Thumb(2) Befehlssatz besitzt weniger Funktionalität als der ARM Befehlssatz, arbeitet aber auf den gleichen Registern. Der Vorteil von Thumb(2) liegt in kompakterem Quelltext und einer besseren Leistung bei Bussystemen mit weniger als 32-Bit Busbreite. Durch die von Thumb(2) eingeführten 32-Bit Befehle wird eine variable Befehlslänge ermöglicht. Zusätzlich enthält Thumb(2) Befehle zur direkten Bit-Manipulation und ermöglicht die bedingte Ausführung von Befehlen.

Der Jazelle Befehlssatz ermöglicht die Ausführung von Java Bytecode, dabei können stackbasierte Operationen direkt in Hardware ausgeführt werden und komplexer Java Bytecode in Software. Jazelle hat die Nachteile, dass es proprietär ist und die Dokumentation nicht frei verfügbar ist. Außerdem ist Jazelle in neueren ARM Architekturen nicht mehr verfügbar.

SIMD steht für Single Instruction Multiple Data. Die Idee der SIMD Befehle besteht darin, dass mehrere Daten mit einem Befehl bearbeitet werden können. Viele Multimediadaten benötigen eine geringe Bitbreite z.B. 8-Bit bei R/G/B. Viele Prozessoren haben eine große ALU und Registerbreite. Bei der Verarbeitung von Multimediadaten besteht eine hohe Leistungsanforderung und mit den SIMD Befehlen können die obigen Anforderungen gut erfüllt werden. Der

ARM1176JZFS Prozessor kann entweder vier 8-Bit Integer oder zwei 16-Bit Integer parallel verarbeiten.

Die VFPv2 (Vector Floating Point Version 2) Funktionsmerkmale ermöglichen es, dass Gleitkommazahloperationen direkt in Hardware ausgeführt werden und nicht in Software emuliert werden. VFPv2 enthält zwei Berechnungsmodi um Gleitkommazahlen nach IEEE 754 zu berechnen. Der full-compliance mode hält alle IEEE 754 Spezifikationen ein. Der run-fast mode hält sich an die IEEE 754 Spezifikationen, lässt dabei aber einige vorgeschriebene Überprüfungen der Zwischenwerte aus. ARM VFPv2 unterstützt die IEEE 754 Datenformate Float und Double, aber nicht die extended Formate. Außerdem werden die unsigned 32-Bit Integer und 32-Bit Integer im Zweierkomplement unterstützt.

Im 32-Bit ARM Befehlssatz sind fast alle Befehle bedingt ausführbar. Der ARM Befehlssatz besteht aus den folgenden Befehlsarten: Sprungbefehle, Datenverarbeitungsbefehlen, Befehle für
LOAD/STORE, Befehle für Exceptionhandling und Koprozessorbefehle. Weitere Informationen können im ARM Architecture Reference Manual [1] und ARM1176JZF-S Technical Reference Manual [2] gefunden werden.

A.4 VORVERARBEITUNG DER DATEN

In diesem Abschnitt werden verschiedene offline Verfahren zur Bereinigung der Daten vorgestellt. Diese Verfahren wurden in der Arbeit von Kopczynski [28] beschrieben. Im Rahmen der Projektgruppe wurden sie ausprobiert. Nach den Versuchen sie online-fähig zu machen, stellten wir jedoch fest, dass sie entweder problematisch zu berechnen oder gar unnötig waren. Wir beschränkten uns also nur auf die Basislinienkorrektur, die wir in Abschnitt 6.3 erklärten und mit der wir schon recht gute Ergebnisse erzielen konnten.

A.4.1 *Niveaukorrektur*

Um das Hintergrundrauschen der gesamten Messung zu senken, kann die Niveaukorrektur angewandt werden. Um dies zu erreichen, wird in der Arbeit von Kopczynski [28] vorgeschlagen, den Median d über alle Signalwerte zu berechnen, da er verglichen mit dem arithmetischen Mittel resistenter gegen Ausreißer wie Peaks ist. Zieht man diesen Wert überall ab, so wird ein großer Teil des Rauschens dadurch eliminiert werden. Zusätzlich kann man die wenigen übrigen negativen Werte auf 0 setzen. Somit sieht die Berechnung für den Signalwert $S_{r,t}$ wie folgt aus:

$$S_{r,t} := \max(S_{r,t} - d, 0).$$

für alle Retentionszeiten r und alle Driftzeiten t.

ONLINE-LÖSUNG Um d zu berechnen, braucht man die ganzen Messdaten, was im Online-Fall nicht gegeben ist. Ein mögliches Vorgehen

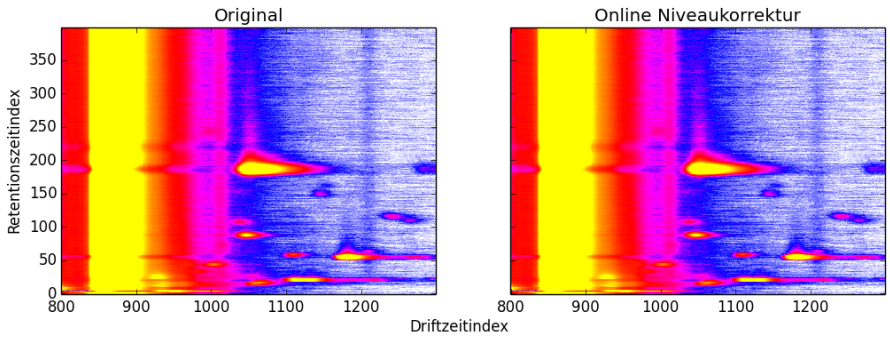


Abb. 74: In sieht man die unverfälschte MCC/IMS Messung. Das Ergebnis der Online-Variante der Niveaukorrektur ist in der zu sehen, wobei für die Berechnung des Niveaus nur 20 Spektren benutzt werden.

wäre, wie bei der Basislinienkorrektur eine gewisse Anzahl an Spektren aus der letzten Messung zu betrachten und die benötigten Daten, in diesem Fall den Median $\mathbf{d}$, daraus zu gewinnen.

A.4.2 *Feuchtekorrektur*

Schaut man sich die Daten nach der Niveaukorrektur an, so sieht man immer noch recht auffallende Verschiebungen von Spektren am Anfang der Messung. Diesen Effekt kann man korrigieren. Dazu sind folgende Schritte notwendig:

- Finde den Index $\mathbf{t}^*$ des Chromatogramms mit der höchsten Wertesumme.
- Verschiebe alle Spektren so, dass deren höchster Wert in der $\mathbf{t}^*$-ten Spalte steht.

ONLINE-LÖSUNG Für die Online-Lösung können die Daten aus der vorhergehenden Messung bezogen werden. Jedoch schwankt der Wert von Messung zur Messung, wie es in Abb. 18 dargestellt wird. Darum kann es passieren, dass die bereits verschobenen Daten erneut verschoben werden müssen, da man einen neuen Wert für $\mathbf{t}^*$ gefunden hat.

Um das Problem der vielen Verschiebungen zu vermeiden, kann man einen zusätzlichen Offset-Array benutzen. Dann geht man wie folgt vor:

- Finde für jede Zeile den höchsten Wert und seine Position.
- Simuliere das Verschieben der Signalwerte in einem Spektrum, indem man sich einen Offset-Wert für jede Zeile abspeichert.
- Greife auf die Werte des Spektrums entsprechend deren Verschiebung um den Offset-Wert zu.

Diese Korrektur wurde nur angedacht, aber dann nicht mehr umgesetzt.

A.4.3 *Ionenkompensation*

Eine weitere Auffälligkeit an der visualisierten und unbearbeiteten Messung sind Stellen, an denen der RIP etwas eingeschnürt erscheint. Das

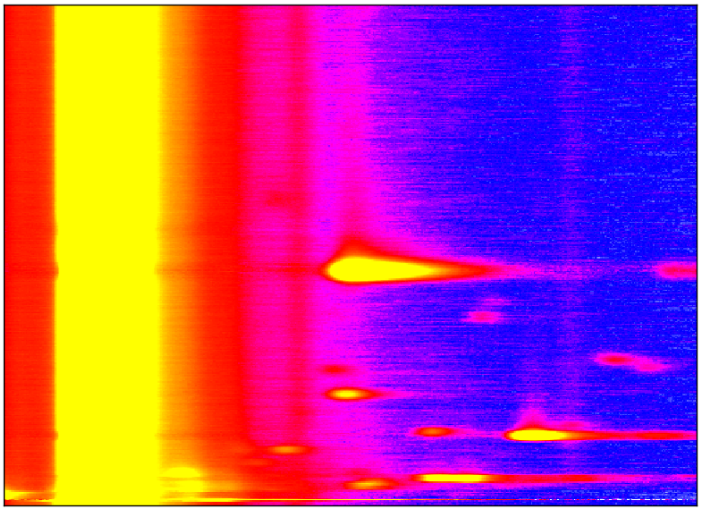


Abb. 75: Eine Ionenkompensation ohne Normalisierung, wobei die Einschnürrungen im Vergleich zu Abb. 17 etwas kleiner ausfallen.

sind meist die Stellen, an denen mehrere zeitlich versetzte bzw. sehr stark ausgeprägte Peaks erscheinen, wie es im Kapitel 2.1 erläutert wird. Diesem Effekt kann man mit wenigen Schritten entgegenwirken, wobei die Summe aller Werte in der Messung vor der Korrektur dem Wert $u = \sum_{r,t} S_{r,t}$ entspricht:

1. Finde in der t^*-ten Spalte den höchsten Wert h.
2. Berechne den Faktor $f_r = \frac{h}{S_{r,t^*}}$, der beschreibt, um wie viel h größer als S_{r,t^*} in dem entsprechenden Spektrum r ist.
3. Multipliziere jeden Wert in der Zeile mit dem Faktor f_r, um den Wert S_{r,t^*} dem höchsten Wert h des Chromatograms an der Position t^* anzugleichen: $S^*_{r,t} = f_r S_{r,t}$ für alle Driftzeiten t.
4. Normalisiere die Werte durch das Multiplizieren mit $\frac{u}{u^*}$, wobei $u^* = \sum_{r,t} S^*_{r,t}$ die Summe aller Werte in der Messung nach der Ionenkompensation im Schritt 3 ist.

ONLINE-LÖSUNG Auch in dieser Situation kann man die Werte aus den letzten Zeilen der vorherigen Messung beziehen. Die vorgeschlagene Normalisierung ist jedoch nicht genau so durchführbar. Abb. 75 zeigt das Ergebniss der Ionenkompensation ohne Normalisierung. Die erhaltenen Änderungen sind nur gering und daher kann die Erkennung der Peaks schon nach der einfachen Niveau- und ggf. Basislinienkorrektur erfolgen.

A.5 GPU-PROGRAMMIERUNG

Moderne Grafikkarten wurden für Berechnungen von zweidimensionalen Arrays von Pixeln in Echtzeit (Computerspiele, hoch auflösende Videos)

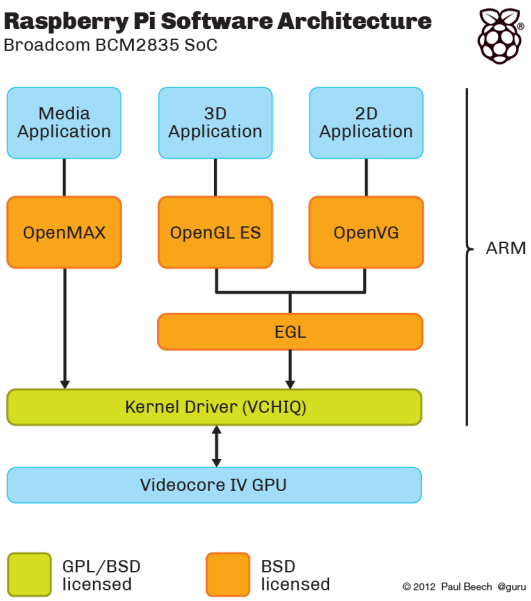


Abb. 76: Architektur zur GPU-Programmierung (mit freundlicher Genehmigung von Paul Beech).[4]

gebaut bzw. daraufhin optimiert. Die dazu auszuführenden Operationen sind hochgradig parallelisierbar: so können z. B. die eine Szene darstellenden Vertices (Eckpunkte von Polygonen) unabhängig voneinander transformiert und daraus resultierende Farbwerte von mehreren Pixeln gleichzeitig berechnet werden. Außer für Grafikausgabe kann die zur Verfügung stehende Rechenleistung auch für andere Zwecke verwendet werden. Dies ist aufgrund der hochgradig parallelen Architektur des Grafikprozessors natürlich nur für bestimmte Probleme sinnvoll nutzbar. Ein solches Problem ist die Berechnung der Faltung, die wir bis jetzt auf der CPU durchgeführt haben. Die Grafikkarte des Pi ist mit 24 GFLOPS[1] viel leistungsfähiger als die CPU, die bei Benchmarks mit Linpack ohne Übertaktung maximal 40 MFLOPS[2] erreicht. Es muss nur ein Weg gefunden werden, diese Rechenleistung für unsere Zwecke zu verwenden. Dazu werden im Folgenden einige Optionen betrachtet.

A.5.1 *Möglichkeiten auf dem Pi*

Für Berechnungen auf der Grafikkarte, die nicht dem Erzeugen von Computergrafiken dienen (bekannt als GPGPU – General Purpose Computing on Graphics Processing Units), ist OpenCL (Open Computing Language) als offener Standard weit verbreitet. Auf dem Pi ist eine Unterstützung für OpenCL jedoch weder verfügbar noch in Zukunft zu erwarten[3], sodass die GPU nur mit Bibliotheken angesprochen werden kann, die für andere Zwecke vorgesehen sind.

1 http://elinux.org/RPi_VideoCore_APIs
2 http://elinux.org/RPi_Performance#Results
3 Laut Aussage von Eben Upton auf der Bay Area Maker Faire 2012. http://www.youtube.com/watch?v=WIf4Fk2252A&t=21m20s

Auf dem Pi sind folgende Bibliotheken verfügbar, welche die GPU verwenden:

- Open Media Acceleration (OpenMAX)
  Bibiliothek zur Audio- und Video(de)kodierung
- Open Graphics Library for Embedded Systems (OpenGL ES) 2.0
  3D-Grafik-Bibliothek für eingebettete Systeme
- OpenVG
  2D-Vektorgrafik-Bibiliothek

Die proprietären Treiber dazu werden als Bibliotheken im Raspbian mitgeliefert und liegen unter `/opt/vc`.

OpenGL ES und OpenVG erscheinen für unsere Zwecke möglicherweise erfolgsversprechend und werden daher im Folgenden genauer dargestellt.

A.5.2 *OpenGL ES*

Die Erzeugung von Grafik erfolgt bei OpenGL in einem mehrschrittigen Verfahren, wobei die Ausgabe von einem Schritt als Eingabe für den nächsten verwendet wird. Diese Grafikpipeline ist bei OpenGL ES 2.0 an zwei Stellen programmierbar: zur Berechnug von Vertex-Positionen und Farbwerten von Pixeln. Die dazu verwendeten Programme, die auf der GPU ausgeführt werden, heißen Shader. Shader werden auf ihren Eingabedaten in so vielen Threads wie von der Hardware maximal unterstützt parallel ausgeführt. Jeder dieser gleichzeitig laufenden Threads führt auf seinen Daten jeweils die gleiche Operation aus. Dadurch wird ein hoher Durchsatz bei geringer Flexibilität erlangt.

Um eine darzustellende Szene zu beschreiben, übergibt das Anwendungsprogramm eine Liste von Attributen für mehrere Vertices an den Treiber. Dieser führt für jeden angegebenen Vertex mit den ihm zugewiesenen Attributen den vorher festgelegten Vertex-Shader aus. Der Shader kann die übergebenen Attribute beliebig interpretieren. Als einzige Anforderung muss dem Vertex mit der Variablen `gl_Position` eine Position zugewiesen werden, welche im nicht-trivialen Fall von übergebenen Attributen abhängen wird. Ausserdem können beliebige andere Ausgaben zur weiteren Verwendung im Fragment-Shader erzeugt werden. Die zur Berechnung der Pixelfarbe benötigten Daten heißen Fragments und bestehen unter anderem aus im Vertex-Shader berechneten und zwischen Vertices interpolierten Werten. Der Fragment-Shader wird nun für jedes generierte Fragment aufgerufen und kann durch Zuweisen von `gl_FragColor` eine Ausgabe im Framebuffer erzeugen. Den Shadern stehen als zusätzliche Eingaben vom Anwendungsprogramm festgelegte Daten wie Skalare, Vektoren und Matritzen mit ganzen oder Gleitkommazahlen sowie Texturen (nur im Fragment-Shader) zur Verfügung.

Wie können wir dies nun zur Durchführung einer Faltung verwenden? Zur Ausgabe benutzen wir einen Offscreen-Buffer in der Größe

4 http://www.raspberrypi.org/wp-content/uploads/2012/10/Architecture-and-Source.png

```
for (int y = 0; y < num_spectra; y++) {
    for (int x = 0; x < num_chromatograms; x++) {
        texture[(y * num_chromatograms + x) * 3]
            = measurement[y][x] >> 4 & 0xff;
        texture[(y * num_chromatograms + x) * 3 + 1]
            = measurement[y][x]      & 0x0f;
    }
}
```

Listing 3: Aufspalten der Messwerte auf zwei Farbkomponenten einer Textur. Rot enthält die höherwertigen 8 Bit, grün die niederwertigen 4 Bit.

der verwendeten Eingabedaten. Diese Größe ist in einer Richtung ein Pixel für jeden Datenpunkt eines Spektrums, in der anderen Richtung bei Verwendung eines Offlineverfahrens ein Pixel pro Spektrum, bei einem Onlineverfahren die Auflösung des Filterkerns in Richtung der Retentionszeit. Um für jeden Pixel des Ausgabebuffers ein Fragment mit den später benötigten Texturkoordinaten zu generieren, wird der Vertex-Shader mit als Position interpretierten Attributen für Vertices aufgerufen, welche die Ecken des Buffers darstellen. Der verwendete Vertex-Shader berechnet mit Hilfe dieser Attribute passende Texturkoordinaten im Bereich von $(0, 0)$ bis $(1, 1)$. Diese Werte werden zwischen den gezeichneten Pixeln interpoliert. Auf den resultierenden Fragments wird der Fragment-Shader aufgerufen. Diesem werden zusätzlich die Eingabedaten und die beiden in SGLTR (siehe Abschnitt 8.3) verwendeten Filterkerne als Texturen zur Verfügung gestellt. Hierbei wird angenommen, dass alle verwendeten Kerne die gleiche Auflösung besitzen. Dies vermeidet zahlreiche Fallunterscheidungen und ist für unsere Anwendung ausreichend.

Die Texturen bestehen aus linear in der Reihenfolge rot, grün, blau angeordneten Farbwerten für jeden Pixel. Die einzelnen Komponenten der Texturen können in verschiedenen Farbtiefen angegeben werden, jedoch beim Pi maximal mit 8 Bit. Jeder ganzzahlige Wert einer Komponente aus dem Bereich $[0, 255]$ wird vom Treiber in eine Gleitkommazahl im Bereich von $[0, 1]$ konvertiert. Es existiert zwar eine Erweiterung, um direkt Gleitkommadaten angeben zu können (`GL_OES_texture_float`), jedoch wird sie vom Pi nicht unterstützt[5]. Da unsere Eingabedaten 12 Bit pro Datenpunkt besitzen, müssen diese auf zwei Farbkomponenten aufgeteilt und im Shader wieder kombiniert werden. Der Code dazu ist in Listing 3 zu sehen.

Die im Vertex-Shader generierten Texturkoordinaten werden nun im Fragment-Shader zur Adressierung der Textur mit den Eingabedaten benutzt. Zu Testzwecken kann direkt mit dem so gewonnenen Wert eine Zuweisung an die Ausgabevariable `gl_FragColor` erfolgen. Die erzeugten Daten können mit `glReadPixels` ausgelesen und von der CPU weiter verarbeitet werden, um z. B. gleichfarbige Pixel zu Regionen zusammenzufassen. Als ersten Test kann man sich z. B. mit

5 Erscheint nicht in der Ausgabe von `glGetString(GL_EXTENSIONS)`.

```
float val_at(vec2 pos)
{
    vec2 val = texture2D(tex_data, pos).rg;
    return val.r * 16.0 * 255.0 + val.g * 255.0;
}
```

Listing 4: Hilsfunktion zum Zusammenfügen der Farbkomponenten zum ursprünglichen Wert im Fragment-Shader.

```
gl_FragColor = vec4(1.0,
                    1.0 - val_at(out_tex_coord) / 32.0,
                    1.0,
                    1.0);
```

überzeugen, dass die Eingabedaten tatsächlich dort ankommen, wo sie erwartet werden. Dafür wird die Funktion `val_at` genutzt, dessen Code in Listing 4 zu finden ist. Dies resultiert in Abb. 77a.

Einen Schritt weiter geht man mit dem folgenden Code:

```
if (val_at(out_tex_coord) > 8.0) gl_FragColor = red;
else gl_FragColor = white;
```

Damit wird (mit entsprechenden Definitionen von `red` und `white`) Abb. 77b generiert.

Die tatsächliche Faltung wird nun durchgeführt, indem wie gewohnt über die Filterkerne in beide Dimensionen iteriert wird, die Filterkerntexturen und die Textur mit den Eingabedaten mit entsprechendem Offset adressiert werden, die resultierenden Werte multipliziert, addiert und mit einem Schwellenwert verglichen werden. Das Ergebnis davon ist in Abb. 77c zu sehen.

Da kein Zwischenspeichern und Wiederverwenden von Werten in einem Shader möglich ist, wird bei diesem Ansatz die Separierbarkeit der verwendeten Kerne nicht ausgenutzt. Obwohl so erheblich mehr Berechnungen pro Pixel ausgeführt werden müssen, erscheint bei ca. 500 Mal mehr FLOPS der GPU verglichen mit der CPU des Pi ein Geschwindigkeitsgewinn als möglich. Allerdings stellte sich heraus, dass dieser Ansatz ungeeignet ist, da die GPU des Pi nur eine begrenzte Anzahl an Texturzugriffen im Fragment-Shader unterstützt, die für die von uns verwendeten Filterkerne zu klein ist[6]. So können zwar die ersten beiden präsentierten Ausgaben auf dem Pi erzeugt werden, das dritte Bild stammt jedoch von einem Desktop-Rechner.

Eine Möglichkeit, um die Separierbarkeit der Kerne auszunutzen und die Anzahl der Texturzugriffe pro Fragment stark zu verringern, wäre die Verwendung zwei verschiedener Fragment-Shader. Damit wären zwei Renderdurchgänge erforderlich: im Ersten würde ein Fragment-Shader als Ausgabe eine Textur mit der Faltung in einer Richtung erzeugen, diese dann in einem zweiten Durchgang als Eingabe des zweiten Shaders verwendet werden, um die Faltung in der anderen Richtung durchzuführen. Da jedoch auf der CPU mit viel geringerem Aufwand ausreichend

6 ERROR:CUSTOM-8 (fragment shader, line -1) Too many texture accesses

Operation	Laufzeit
Initialisierung des Bildpuffers	7,327 ms
Laden der Messung in den Puffer	274,412 ms
Berechnung der Faltung	23,332 ms
Auslesen des Grafikspeichers	448,259 ms

Tabelle 15: Laufzeiten von Teiloperationen bei einer Faltung mit OpenVG

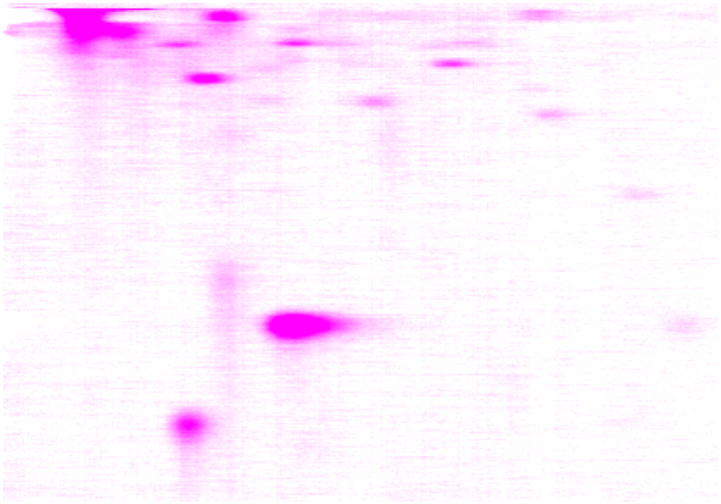
geringe Laufzeiten erreicht werden konnten, wurde dieser Ansatz nicht weiter betrachtet.

A.5.3 *OpenVG*

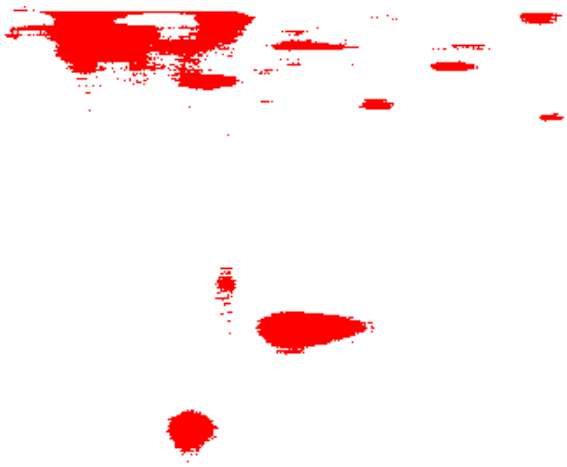
OpenVG bietet als Bibliothek für 2D-Vektor- und Rastergrafik die Funktionen `vgConvolve` und `vgSeparableConvolve` an, mit deren Hilfe Filterkerne bzw. separable Filterkerne auf Bilddaten angewandt werden können.

Dies haben wir mit den SGLTR-Filterkernen auf einer Messung mit 1200 Spektren mit jeweils 2499 Datenpunkten auf dem Raspberry Pi getestet. Die Faltung wird dabei für die komplette Messung berechnet. Die Laufzeiten der Teiloperationen der Berechnung finden sich in Tabelle 15.

Die eigentliche Faltung für eine komplette Messung ist im Vergleich zu unserer spektrenweisen Implementation auf der CPU deutlich schneller. Problematisch für die Nutzung als Online-Algorithmus sind der Overhead durch die nötigen Transformationen der Messdaten in ein durch die GPU nutzbares Format sowie die Transferzeiten in und aus dem GPU-Speicher.



(a) Rohdaten



(b) einfacher Schwellwert

(c) Faltung mit SGLTR

Abb. 77: Ausgaben der OpenGL ES-Filterimplementation. Positive Driftzeit nach rechts, positive Retentionszeit nach unten.

LITERATURVERZEICHNIS

[1] ARM Ltd.: *ARM Architecture Reference Manual*. Juli 2013. – URL https://silver.arm.com/download/ARM_and_AMBA_Architecture/AR570-DA-70000-r0p0-00rel1/DDI0406C_b_arm_architecture_reference_manual.pdf

[2] ARM Ltd.: *ARM1176JZF-S Technical Reference Manual*. Juli 2013. – URL http://infocenter.arm.com/help/topic/com.arm.doc.ddi0301h/DDI0301H_arm1176jzfs_r0p7_trm.pdf

[3] Bader, S. ; Urfer, W. ; Baumbach, J. I.: Processing ion mobility spectrometry data to characterize group differences in a multiple class comparison. In: *International Journal for Ion Mobility Spectrometry* 8 (2005), S. 1–4

[4] Bader, Sabine: *Identification and quantification of peaks in spectrometric data*, TU Dortmund, Fakultät Statistik, Dissertation, 2008. – URL https://eldorado.tu-dortmund.de/bitstream/2003/25833/1/DissertationSabineBader.pdf

[5] Baumbach, J. I. ; Maddula, S. ; Bödeker, B. ; Litterst, P. ; Westhoff, M.: Differenzierung von COPD-Patienten einschließlich Bronchialkarzinompatienten von Gesunden mittels Ionenmobilitätsspektrometrie – Einsatz statistischer und bioinformatischer Methoden. (2013). – URL http://www.bs-analytik.de/pdf/ANAKON_2013_COPD.pdf

[6] Baumbach, Jörg Ingo: Ion mobility spectrometry coupled with multi-capillary columns for metabolic profiling of human breath. In: *Journal of Breath Research* 3 (2009), Nr. 3, S. 034001. – URL http://stacks.iop.org/1752-7163/3/i=3/a=034001

[7] Bödeker, Bertram ; Baumbach, Jörg Ingo: Analytical description of IMS-signals. In: *International Journal for Ion Mobility Spectrometry* 12 (2009), Nr. 3, S. 103–108. – URL http://dx.doi.org/10.1007/s12127-009-0024-y. – ISSN 1435-6163

[8] Breiman, Leo: Random Forests. In: *Mach. Learn.* 45 (2001), Oktober, Nr. 1, S. 5–32

[9] Bunkowski, Alexander: *MCC-IMS data analysis using automated spectra processing and explorative visualisation methods*, Bielefeld University, Dissertation, 2012

[10] Burger, Wilhelm ; Burge, Mark J.: *Digital image processing*. 1st edition. New York [u.a.] : Springer, 2008. – ISBN 978-1-8462-8379-6

[11] Chang, Chih-Chung ; Lin, Chih-Jen: LIBSVM: a library for support vector machines. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 2 (2011), Nr. 3, S. 27

[12] Chinrungrueng, C. ; Toonkum, P.: Directional Savitzky-Golay filters for real-time speckle reduction and coherence enhancement of medical ultrasound images. In: *TENCON 2004. 2004 IEEE Region 10 Conference* Bd. A, 2004, S. 163–166 Vol. 1

[13] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Introduction to Algorithms*. 3rd edition. MIT Press, 2009. – ISBN 978-0-262-03384-8

[14] Douglas, Adams: *The Hitchhiker's Guide to the Galaxy*. Pan Books, 1979. – ISBN 0-330-25864-8

[15] Fan, Rong-En ; Chang, Kai-Wei ; Hsieh, Cho-Jui ; Wang, Xiang-Rui ; Lin, Chih-Jen: LIBLINEAR: A library for large linear classification. In: *The Journal of Machine Learning Research* 9 (2008), S. 1871–1874

[16] Fink, G. A.: *Mustererkennung Mit Markov-Modellen: Theorie – Praxis – Anwendungsgebiete*. Wiesbaden : Teubner, 2003 (Leitfäden der Informatik). – ISBN 978-3-519-00453-0

[17] Fink, G. A.: *Mustererkennung*. Vorlesungsskript. 2012. – Technische Universität Dortmund

[18] Fisher, Ronald A.: The use of multiple measurements in taxonomic problems. In: *Annals of eugenics* 7 (1936), Nr. 2, S. 179–188

[19] Gonzalez, Rafael C. ; Woods, Richard E.: *Digital image processing*. 3rd edition. Upper Saddle River, NJ : Pearson Prentice Hall, 2008. – ISBN 978-0-13-168728-8

[20] Greensted, Andrew: *FIR Filters by Windowing*. März 2010. – URL http://www.labbookpages.co.uk/index.html. – Zugriffsdatum: 2013-08

[21] Hain, Johannes: *Comparison of Common Tests for Normality*, Julius-Maximilians-Universität Würzburg, Lehrstuhl für Mathematik VIII (Statistik), Diplomarbeit, August 2010. – URL http://www.statistik-mathematik.uni-wuerzburg.de/fileadmin/10040800/user_upload/hain/da_hain_final.pdf

[22] Handa, H. ; Inou, N. ; Yonemura, Y. ; Maddula, S. ; Baumbach, J. I. ; Miyazawa, T.: Volatile metabolites arising from lung cancer in Japan. In: *Breath Analysis Summit 2013* (2013). – URL http://www.bs-analytik.de/pdf/breath2013/Handa.pdf

[23] Hauschild, Anne-Christin ; Kopczynski, Dominik ; D'Addario, Marianna ; Baumbach, Jörg Ingo ; Rahmann, Sven ; Baumbach, Jan: Peak Detection Method Evaluation for Ion Mobility Spectrometry by Using Machine Learning Approaches. In: *Metabolites* 3 (2013), Nr. 2, S. 277–293. – URL http://www.mdpi.com/2218-1989/3/2/277. – ISSN 2218-1989

[24] Hennessy, Patterson: *Computer architecture a quantitative approach*. Kaufmann, 2007. – ISBN 9780123704900

[25] JÄHNE, Bernd: *Kanten, Linien und Ecken*. S. 367–394. In: *Digitale Bildverarbeitung*, Springer Berlin Heidelberg, 2012. – ISBN 978-3-642-04951-4

[26] KERN-ISBERNER, G.: *Darstellung, Verarbeitung und Erwerb von Wissen, Kapitel 5*. Vorlesungsfolien. 2011. – URL http://ls1-www.cs.uni-dortmund.de/ie/lehre/ws1112/dvew/. – Zugriffsdatum: 2013-03-31. – Technische Universität Dortmund

[27] KOPCZYNSKI, D. ; BAUMBACH, J. I. ; RAHMANN, S.: Peak modeling for Ion mobility spectrometry measurements. In: *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, IEEE, Aug 2012, S. 1801–1805. – ISSN 2219-5491

[28] KOPCZYNSKI, Dominik: *Datenreduktion und Merkmalsextraktion bei Ionen-Mobilitäts-Spektrometrie-Messungen*, Technische Universität Dortmund, Fakultät für Informatik, Diplomarbeit, 2010. – URL http://ls11-www.cs.uni-dortmund.de/_media/techreports/tr11-04.pdf

[29] MORIK, K.: *Maschinelles Lernen*. Vorlesungsskript. 2013. – Technische Universität Dortmund

[30] OPPENHEIM, Alan V. ; SCHAFER, Ronald W.: *Zeitdiskrete Signalverarbeitung*. 2., neu übers. und überarb. Aufl. München [u.a.] : Oldenbourg, 1995 (Grundlagen der Schaltungstechnik). – ISBN 3-486-22948-6

[31] PRECHT, Manfred ; KRAFT, Roland ; BACHMAIER, Martin: *Angewandte Statistik 1*. 6., vollst. überarb. Oldenbourg, 1999. – ISBN 3-486-24416-7

[32] PRESS, William H. ; TEUKOLSKY, Saul A. ; VETTERLING, William T. ; FLANNERY, Brian P.: *Numerical Recipes: The Art of Scientific Computing*. 3rd edition. New York : Cambridge University Press, 2007. – ISBN 978-0-5218-8068-8

[33] SAVITZKY, Abraham ; GOLAY, M. J. E.: Smoothing and Differentiation of Data by Simplified Least Squares Procedures. In: *Analytical Chemistry* 36 (1964), Nr. 8, S. 1627–1639. – URL http://pubs.acs.org/doi/abs/10.1021/ac60214a047

[34] SHANNON, C. E.: A mathematical theory of communication. In: *SIGMOBILE Mob. Comput. Commun. Rev.* 5 (2001), Januar, Nr. 1, S. 3–55

[35] SPANGLER, Glenn E. ; COLLINS, Charles I.: Peak shape analysis and plate theory for plasma chromatography. In: *Analytical Chemistry* 47 (1975), Nr. 3, S. 403–407. – URL http://pubs.acs.org/doi/abs/10.1021/ac60353a013

[36] THORNLEY, David: Novel Anisotropic Multidimensional Convolutional Filters for Derivative Estimation and Reconstruction. In: *Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on*, Nov 2007, S. 253–256

[37] Vautz, Wolfgang ; Bödeker, Bertram ; Bader, Sabine ; Baumbach, Jörg Ingo: Recommendation of a standard format for data sets from GC/IMS with sensor-controlled sampling. In: *International Journal for Ion Mobility Spectrometry* 11 (2008), Nr. 1-4, S. 71–76

[38] Wang, B. H. ; Binford, T. O.: Generic, Model-Based Estimation and Detection of Peaks Image Surfaces. In: *ARPA96*, 1996, S. 913–922

[39] Witten, Ian H. ; Frank, Eibe: *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., 2005