Effiziente und robuste, feature-basierte Wiedererkennung langer Zeitreihen in großen Datenbanken

Materialverfolgung im Walzprozess anhand von Dicken- bzw. Breitenprofilen

Dissertation

zur Erlangung des Grades
"Doktor der Naturwissenschaften"
am Fachbereich Physik, Mathematik und Informatik
der Johannes Gutenberg-Universität in Mainz

vorgelegt von
Egor Dranischnikow
geboren in Nishnij Tagil

Mainz, Januar 2014

Berichterstatter:

Mündliche Prüfung am 20.03.2014

Abstract

Material tracking is becoming increasingly important in the metal industry: It is necessary that a metal coil passes through a fixed program during its manufacturing process - only then the quality of the product is guaranteed. The current practice is to assign each metal coil a number with which it is labeled. This method proves to be prone to errors: During the day-long storage of the metal coils between two production steps the labels may be lost, swapped, incorrectly read or become unreadable.

In 2007 iba AG filed a patent for the identification of the metal coils by their thickness profile (Anhaus [3]). Thus, the identity of a metal coil can be definitely established and a reliable material tracking becomes possible.

However, it turned out that the measured thickness profiles, which can be considered to be long time series, are notoriously error-prone due to difficult circumstances of the measurement process. The length of the time series and the multitude of different measurement errors make it impossible for established well-known methods (such as $\mathcal{L}^2$ distance minimization or *Dynamic Time Warping*) to be successfully applied.

This work presents an efficient feature-based algorithm for comparison of two time series. It is robust to noise and measurement failures as well as invariant with respect to coordinate transformations of the time series such as scaling and translation. In adition, comparison of subsequences is also possible.

Our framework is characterized by its high accuracy as well as by its high speed: More than 99.5% of the queries to our database with real world data are answered correctly. With several hundred time series comparisons per second, it is approximately 10 times faster than other tested methods, which are not able to process more than 90% of the queries correctly.

This performance of the algorithm enabled the iba AG to build a unique thickness profile based verification system that is already successfully applied in several steel and aluminum rolling mills.

Zusammenfassung

Die Materialverfolgung gewinnt in der Metallindustrie immer mehr an Bedeutung: Es ist notwendig, dass ein Metallband im Fertigungsprozess ein festgelegtes Programm durchläuft - erst dann ist die Qualität des Endprodukts garantiert. Die bisherige Praxis besteht darin, jedem Metallband eine Nummer zuzuordnen, mit der dieses Band beschriftet wird. Bei einer tagelangen Lagerung der Bänder zwischen zwei Produktionsschritten erweist sich diese Methode als fehleranfällig: Die Beschriftungen können z.B. verloren gehen, verwechselt, falsch ausgelesen oder unleserlich werden.

2007 meldete die iba AG das Patent zur Identifikation der Metallbänder anhand ihres Dickenprofils an (Anhaus [3]) - damit kann die Identität des Metallbandes zweifelsfrei nachgewiesen werden, eine zuverlässige Materialverfolgung wurde möglich.

Es stellte sich jedoch heraus, dass die messfehlerbehafteten Dickenprofile, die als lange Zeitreihen aufgefasst werden können, mit Hilfe von bisherigen Verfahren (z.B. $\mathcal{L}^2$-Abstandsminimierung oder *Dynamic Time Warping*) nicht erfolgreich verglichen werden können.

Diese Arbeit stellt einen effizienten feature-basierten Algorithmus zum Vergleich zweier Zeitreihen vor. Er ist sowohl robust gegenüber Rauschen und Messausfällen als auch invariant gegenüber solchen Koordinatentransformationen der Zeitreihen wie Skalierung und Translation. Des Weiteren sind auch Vergleiche mit Teilzeitreihen möglich.

Unser Framework zeichnet sich sowohl durch seine hohe Genauigkeit als auch durch seine hohe Geschwindigkeit aus: Mehr als 99.5% der Anfragen an unsere aus realen Profilen bestehende Testdatenbank werden richtig beantwortet. Mit mehreren hundert Zeitreihen-Vergleichen pro Sekunde ist es etwa um den Faktor 10 schneller als die auf dem Gebiet der Zeitreihenanalyse etablierten Verfahren, die jedoch nicht im Stande sind, mehr als 90% der Anfragen korrekt zu verarbeiten.

Der Algorithmus hat sich als industrietauglich erwiesen. Die iba AG setzt ihn in einem weltweit einzigartigen dickenprofilbasierten Überwachungssystem zur Materialverfolgung ein, das in ersten Stahl- und Aluminiumwalzwerken bereits erfolgreich zum Einsatz kommt.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Zertifizierung einer Firma gemäß den Vorgaben der ISO 9000-Familie verlangt unter anderem eine lückenlose Materialverfolgung. Dies gilt im gleichen Maße auch für die Produzenten in der Metallindustrie. Ihre Kunden, z.B. Flugzeug- und Autobauer, erwarten von den Herstellern, dass das Durchlaufen aller Produktionsschritte gewährleistet und belegt ist. Auch Walzwerke, die kilometerlange Metallbänder herstellen, aus denen z.B. Autoteile geformt werden, sind daran interessiert, jegliche Fehler auszuschließen: Sollte festgestellt werden, dass die Ursache für einen Unfall im minderwertigen Material liegt, können schnell Schadenersatzforderungen in Millionenhöhe entstehen.

Die bisherige Praxis der Materialverfolgung besteht oft darin, jedem Metallband eine Nummer zuzuordnen, mit der dieses Band beschriftet wird. Was nach einer akzeptablen Lösung klingt, erweist sich jedoch als recht fehleranfällig: Die Beschriftungen können während der tagelangen Zwischenlagerung verloren gehen, verwechselt, falsch ausgelesen, manipuliert oder unleserlich werden.

Die iba AG, Fürth, hat eine Spitzenposition im Bereich der Datenerfassung in der Walzwerkautomatisierung. Seit Jahrzehnten gab es dort die Idee, ob man nicht die Dickenprofile von Metallbändern für eine eindeutige Identifikation verwenden könnte. Als die iba AG dann 2007 das Patent dafür anmeldete (Anhaus [3]), wurde der Grundstein für diese Arbeit gelegt. Die Vorteile einer Benutzung der Dickenprofile zur Materialverfolgung liegen auf der Hand: Das Dickenprofil (für ein Beispiel siehe Abbildung 1.1) kann als eine Art Fingerabdruck zu einer eindeutigen Wiedererkennung der gewalzten Metallbänder dienen. Dieser “Fingerprint” bleibt bis zum nächsten Walzvorgang erhalten. Sogar ein Teilstück des Profils reicht aus, um ein Metallband identifizieren zu können. Auch wenn ein Metallband in mehrere Teile zerschnitten wird, geht die Identität nicht verloren!

Es stellte sich jedoch schnell heraus, dass die Benutzung der Dickenprofile

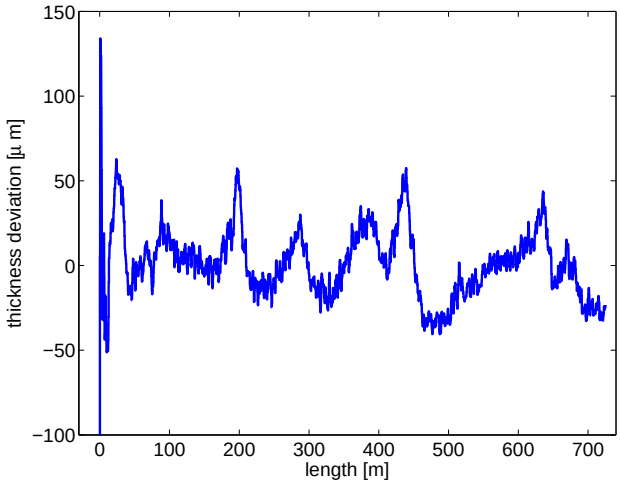

Abbildung 1.1: Verlauf eines Dickenprofils für ein eher kurzes (ca. 700 *Meter) Stahlband. Die Abweichungen befinden sich im Bereich* $[-50, 50]$ *μm, während das Band ca.* 5 *mm dick ist.*

für die Materialverfolgung mit einigen Schwierigkeiten verbunden ist: Bevor zwei Dickenprofile - eines könnte z.B. beim Auslauf aus dem Walzgerüst entstehen, das andere beim anschließenden Einlauf in die Beizanlage - verglichen werden können, müssen diese zuerst aligniert werden. Die Messfehler bzw. die Messqualität, die oft die höchsten Anforderungen an die Genauigkeit nicht erfüllt, stellen die größte Herausforderung dar. Es ist auch nicht verwunderlich: Die Abweichungen vom Dickesollwert, der üblicherweise mehrere Millimeter beträgt, belaufen sich auf einige Mikrometer und sind für das wiedererkennbare Dickenprofil verantwortlich. Der relative Messfehler darf also nicht 10^{-3} übersteigen. Bedenkt man, dass die Messung an einem Metallband stattfindet, das sich mit einer Geschwindigkeit von bis zu 10 m/s bewegt und mehrere hundert Grad heiß ist, muss mit verrauschten Messungen, Trends, Offsets und Ausreißern gerechnet werden.

Die vermessenen Profile stellen hohe Anforderungen an die Algorithmen. Sie müssen

1. invariant gegenüber Koordinatentransformationen sein. Insbesondere gegenüber Translation, Spiegelung und Skalierung
2. robust gegenüber unterschiedlichen Rauscharten, Ausreißern, Messausfällen sein
3. speicher- und laufzeiteffizient sein: Die Suche nach dem passenden Dickenprofil, welches aus bis zu 10^5 Werten bestehen kann, soll letztendlich in einer Datenbank mit mehreren tausend Einträgen stattfinden.

Dickenprofile können als Mess- bzw. Zeitreihen aufgefasst werden - dafür

wird die Längenkoordinate als Zeit interpretiert. Das Alignment von Zeitreihen ist ein bekanntes Problem. In den letzten Jahren wurden mehrere Lösungsansätze entwickelt (z.B. $\mathcal{L}^2$-Abstandsminimierung, *Mutual Information* Maximierung oder *Dynamic Time Warping*. Für eine Übersicht siehe Kapitel 3.). Unsere Experimente zeigten jedoch, dass keines dieser Verfahren den hohen Ansprüchen genügt.

Diese Arbeit stellt einen Algorithmus vor, der die obigen Anforderungen an Invarianz, Robustheit und Effizienz erfüllt. Seine Hauptidee klingt zunächst recht einfach: Aus dem Verlauf der Zeitreihen werden besonders markante, stabile und wiederfindbare Abschnitte, die wir als *Features* bezeichnen, ausgewählt und in eine leicht vergleichbare Gestalt umgewandelt, in die sogenannten *Deskriptoren*. Die Menge der Features/Deskriptoren-Paare einer Zeitreihe wird zu einem *Fingerprint* zusammengefasst, der für einen schnellen Vergleich verwendet werden kann.

Unser Algorithmus zeichnet sich sowohl durch seine hohe Genauigkeit als auch durch seine hohe Geschwindigkeit aus: Mehr als 99.5% der Anfragen werden richtig beantwortet. Mit mehreren hundert Zeitreihen-Vergleichen pro Sekunde ist er etwa um den Faktor 10 schneller als die auf dem Gebiet der Zeitreihenanalyse etablierten Verfahren, die außerdem nicht im Stande sind, mehr als 90% der Anfragen korrekt zu verarbeiten.

Auch wenn die zu unserem Framework verwandten Vorgehensweisen, solche wie der Shazam-Algorithmus (Wang [63]) oder Lowes SIFT-Algorithmus[1] (Lowe [48]), seit einiger Zeit im Einsatz sind, ist unser Algorithmus unseres Wissens nach die erste Weiterentwicklung, die eine stabile Wiedererkennung von stark verrauschten, mehrere tausende Messpunkte langen Zeitreihen ermöglicht.

In den letzten Monaten installierte die iba AG in ersten Stahl- und Aluminiumwalzwerken dickenprofilbasierte Überwachungssysteme zur Materialverfolgung, die auf dem in dieser Arbeit vorgestellten und entwickelten Algorithmus basieren. Zu den positiven Auswirkungen dieser Systeme zählt die Tatsache, dass das Finden verwechselter Metallbänder die Suche nach Fehlerquellen und infolgedessen eine Qualitätsverbesserung der fehlerhaften Abläufe ermöglicht.

Im weiteren Verlauf dieser Arbeit werden die Einzelheiten und die Performanz des Algorithmus beleuchtet.

Im Kapitel 2 werden Definitionen eingeführt, mit deren Hilfe die bearbeiteten Probleme

A Suche des passenden Dickenprofils in einer Datenbank.

B Bestimmung, ob zwei vorliegende Dickenprofile aus den Vermessungen

[1] Scale Invariant Feature Transform

des gleichen Metallbandes entstanden sind.

formal beschrieben werden.

Im Kapitel 3 erläutern wir die etablierten Lösungsansätze. Des Weiteren werden die Vorläufer-Verfahren vorgestellt, die unsere Vorgehensweise inspiriert haben: Shazam-Algorithmus, Shotgun (Venter u. a. [61]), Scale-Space-Analyse (Witkin [64]) und insbesondere das SIFT-Verfahren.

Das SIFT-Verfahren, von dem unser Framework am meisten beeinflusst wurde, wird vor allem in der 2D-Bildverarbeitung eingesetzt. Dabei wird ein sogenannter *DoG-Detektor* benutzt, um besonders interessante Abschnitte/Features im Bild zu detektieren.

Im Abschnitt 3.11 betrachten wir Wavelets bzw. die kontinuierliche Wavelettransformation und stellen fest, dass es sich um eine Verallgemeinerung der DoG- und anderer sich im Umlauf befindlicher Feature-Detektoren handelt. Jedes Wavelet kann also als ein Feature-Detektor eingesetzt werden! Damit haben wir eine Vielzahl von möglichen Feature-Detektoren zur Verfügung, die je nach Art der Zeitreihen eingesetzt werden können.

Im Kapitel 4 wird unser Framework vorgestellt, dessen Implementierungseinzelheiten in Kapitel 5 besprochen werden. Dabei liegt unser Augenmerk vor allem auf der (Zeit-)Effizienz.

Besonders wichtig für die vorliegende Arbeit ist das Kapitel 6. In diesem Abschnitt wird systematisch die Performanz der möglichen Feature-Detektoren, Deskriptoren und Fingerprint-Datenstrukturen untersucht. Neben theoretischen Überlegungen wurde großer Wert auf aussagekräftige Tests mit realen Daten gelegt: Jedes Diagramm ist das Ergebnis tagelanger Berechnungen mit tausenden von Zeitreihen.

Dabei erweisen sich der von uns vorgeschlagene x^2Gauß-Detektor (Abschnitt 6.5), die SIFT-Deskriptoren (Abschnitt 6.1.3) und erstaunlicherweise die naive lineare Suche (Abschnitt 6.7.5) als die besten Alternativen.

Die Erkenntnisse dieses Kapitels ermöglichten uns, eine optimale Konfiguration zu finden, den Einsatz des Frameworks mit realen Daten zu testen und die Ergebnisse mit den etablierten Verfahren zu vergleichen. Diese Resultate werden im Kapitel 7 zusammengefasst und präsentiert.

Im Anhang befinden sich neben einigen Ideen für weitere Einsatzmöglichkeiten des Frameworks vor allem die notwendigen mathematischen Grundlagen, die im Verlauf dieser Arbeit benutzt werden.

Kapitel 2

Formale Definitionen

2.1 Allgemeine Grundlagen

Als Erstes beschäftigen wir uns mit einer formalen Beschreibung unserer Fragestellungen. Dabei werden solche grundlegende Begriffe wie Koordinatentransformationen, Metriken und ihr Zusammenspiel beleuchtet.

Die Anwendung der entwickelten abstrakten Begriffe auf die in der Einleitung vorgestellten Probleme findet im Abschnitt 2.2.3 statt.

Weitere mathematische Details können im Anhang 10 eingesehen werden.

2.1.1 Observablen

Im Laufe dieser Arbeit betrachten wir Funktionen, die wir als *Observablen* bezeichnen:

$$\{f : \mathbb{R}^n \to \mathbb{R}^m\} =: Abb\left(\mathbb{R}^n, \mathbb{R}^m\right),$$

die Werte von Messgrößen in Abhängigkeit von den räumlichen Koordinaten bzw. der Zeit darstellen. Solche Größen können z.B.

- Dichte in einer CT-Rekonstruktion ($n = 3,\ m = 1$)
- Dickenprofil eines Walzbands ($n = 1$, $m = 1$)
- Überwachung vieler Eigenschaften (Temperatur, Feuchtigkeit usw.) eines Prozesses in Abhängigkeit von der Zeit ($n = 1$, $m \geq 1$)

sein.

Im weiteren Verlauf konzentrieren wir uns auf den Fall $n = 1$, $m = 1$.

Es kann davon ausgegangen werden, dass der Träger der Funktionen, für die wir uns interessieren, kompakt (beschränkt und abgeschlossen) ist. Des

Weiteren nehmen wir einfachheitshalber an, dass diese Funktionen stetig (und wegen des kompakten Trägers auch beschränkt) sind.
Die Menge dieser Funktionen bezeichnen wir mit

$$Obs(\mathbb{R},\mathbb{R}) \subset Abb\,(\mathbb{R},\mathbb{R})\,.$$

Da die stetigen Funktionen Borel-messbar sind, gilt auch:

$$Obs(\mathbb{R},\mathbb{R}) \subset \mathcal{L}^p\,(\mathbb{R}) := \mathcal{L}^p\,(\mathbb{R},\mathcal{B}(\mathbb{R}),\lambda)\,.$$

Damit können auch die $\mathcal{L}^p\,(\mathbb{R})$ Metriken

$$d_p(f,g) := \|f-g\|_p := \left(\int_{\mathbb{R}} |f(x)-g(x)|^p\,dx\right)^{\frac{1}{p}} \quad p = 1,2\ldots$$

benutzt werden.
Auch $p=\infty$ ist möglich, wird von uns aber nicht genauer untersucht.

2.1.2 Eindimensionale Koordinatentransformationen

Eine Messgröße kann aus unterschiedlichen Koordinatensystemen beobachtet werden. Im eindimensionalen Fall interessieren wir uns für folgende Koordinatentransformationen:

1. Spiegelung: $x \mapsto -x$
2. Skalierung: $x \mapsto \alpha \cdot x$, $\alpha > 0$
3. Translation: $x \mapsto x+\beta$, $\beta \in \mathbb{R}$
4. alle oberen drei gleichzeitig: $x \mapsto \alpha \cdot x + \beta$, $\alpha \in \mathbb{R}\setminus\{0\}$, $\beta \in \mathbb{R}$.

Die Menge der Koordinatentransformationen $(\alpha,\beta) \in \mathbb{R}\setminus\{0\}\times\mathbb{R} =: T_{1d}$ zusammen mit der Verknüpfung

$$\begin{aligned} \circ : T_{1d}\times T_{1d} &\to T_{1d} \\ (\alpha_1,\,\beta_1)\circ(\alpha_2,\,\beta_2) &:= (\alpha_1\cdot\alpha_2,\,\alpha_1\cdot\beta_1+\beta_2). \end{aligned}$$

bilden eine Gruppe.
Es lässt sich leicht nachrechnen, dass diese Verknüpfung assoziativ, jedoch nicht kommutativ ist, dass $(1,0)$ das Einselement ist und dass die Inverse $(\alpha,\,\beta)^{-1} = \left(\frac{1}{\alpha},\,-\frac{\beta}{\alpha}\right)$ lautet.
Die oben erwähnten Koordinatentransformationen bilden Untergruppen:

1. Spiegelung: $T_m := \{(1,\, 0),\, (-1,\, 0)\}$
2. Skalierung: $T_s := \{(\alpha,\, 0) \,|\, \alpha > 0\}$
3. Spiegelung und Skalierung: $T_{sm} := \{(\alpha,\, 0) \,|\, \alpha \neq 0\}$
4. Translation: $T_t := \{(0,\, \beta) \,|\, \beta \in \mathbb{R}\}$
5. Translation und Spiegelung: $T_{tm} := \{(s,\, \beta) \,|\, s \in \{-1, 1\},\, \beta \in \mathbb{R}\}$

Die Gruppenhomomorphismen ϕ

$$\begin{aligned} &\phi : T_{d1} \to (\mathbb{R} \to \mathbb{R}) \\ &\phi(\alpha,\, \beta) \mapsto T_{(\alpha,\, \beta)}(x) := \alpha \cdot x + \beta \quad \forall x \in \mathbb{R}\ \forall (\alpha, \beta) \in T_{1d} \end{aligned}$$

und ϕ'

$$\begin{aligned} &\phi' : T_{d1} \to ((\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})) \\ &\phi'(\alpha,\, \beta) \mapsto T_{(\alpha,\, \beta)}(f) := T_{(\alpha,\beta)} f := f \circ \phi((\alpha,\, \beta)^{-1}) \end{aligned}$$

erzeugen zwei Darstellungen der Koordinatentransformationen.

Im weiteren Verlauf möchten wir nicht zwischen T_{1d} und den Darstellungen $\phi(T_{1d})$ und $\phi'(T_{1d})$ unterscheiden und bezeichnen sie alle als Koordinatentransformationen bzw. T_{1d}.

2.1.3 Observablen und T_{1d}

Die Menge der Observablen $Obs(\mathbb{R}, \mathbb{R})$ ist unter den Koordinatentransformationen aus T_{1d} abgeschlossen, d.h.:

$$\forall T \in T_{1d} \qquad T\,(Obs(\mathbb{R}, \mathbb{R})) \subset Obs(\mathbb{R}, \mathbb{R}).$$

Wir definieren folgende Relation:

$$\forall f, g \in Obs(\mathbb{R}, \mathbb{R}) : \quad f \propto_{T_{1d}} g \;\Leftrightarrow\; \exists T \in T_{1d} \,:\, f = Tg.$$

Man überzeugt sich leicht, dass es sich um eine Äquivalenzrelation handelt. Damit ergeben sich die Äquivalenzklassen

$$[f]_{T_{d1}} = \{g \in Obs(\mathbb{R},\, \mathbb{R}) \,|\, \exists T \in T_{1d} \,:\, f = Tg\}\,.$$

Analog kann man die Äquivalenzrelationen $\propto_{T_{tm}}$ oder $\propto_{T_{sm}}$ definieren.

2.1.4 Alignierende Metrik

Es liegt nahe zu versuchen, für die Menge der Äquivalenzklassen eine "alignierende Metrik" mittels

$$\delta_p^A(f,g) := \inf_{T\in T_{1d}} d_p(f,Tg) \quad f,g \in Obs(\mathbb{R},\mathbb{R}) \tag{2.1}$$

zu definieren. Sie hätte den Vorteil, dass sie den Abstand der ausgerichteten (daher auch *alignierend*) Funktionen messen würde. Doch wie wir später sehen werden, ist dieser Ausdruck nicht einmal wohldefiniert.

Falls man die Koordinatentransformationen auf Spiegelungen und Translationen beschränkt, so ist δ_p^A eine Metrik.

Alignierende Metrik für Translationen

Zuerst stellen wir fest, dass

$$\delta_p^A(f,g) = \inf_{T\in T_{tm}} d_p(f,Tg)$$

für die Äquivalenzklassen der Äquivalenzrelation $\propto_{T_{tm}}$ wohldefiniert ist. Um dies zu zeigen, betrachten wir folgenden Zusammenhang zwischen den transformierten Observablen $T_{(\alpha,\beta)}f$ und der Norm $\|\cdot\|_p$:

$$\begin{aligned}\|T_{(\alpha,\beta)}f\|_p = \left(\int_{\mathbb{R}} |(T_{(\alpha,\beta)}f)(x)|^p dx\right)^{\frac{1}{p}} &= \left(\int_{\mathbb{R}} |f(T^{-1}_{(\alpha,\beta)}(x))|^p dx\right)^{\frac{1}{p}} \\ &= \left(\int_{\mathbb{R}} |f(y)|^p |\alpha| dy\right)^{\frac{1}{p}} = \sqrt[p]{|\alpha|}\|f\|_p.\end{aligned}$$

Oder auch:

$$\begin{aligned}d_p(T_{(\alpha,\beta)}f,g) = \|T_{(\alpha,\beta)}f - g\|_p &= \|T_{(\alpha,\beta)}(f - T^{-1}_{(\alpha,\beta)}g)\| \\ &= \sqrt[p]{|\alpha|}\|f - T^{-1}_{(\alpha,\beta)}g\|_p = \sqrt[p]{|\alpha|}d_p(f,T^{-1}g).\end{aligned}$$

Man sieht also, dass die Translationen T_t und Spiegelungen T_m die Norm einer Messung invariant lassen, da für sie $|\alpha| = 1$ gilt.
Und damit ist $\delta_p^A(f,g)$

$$\begin{aligned}\delta_p^A(T_f f, T_g g) = \inf_{T\in T_{tm}} d_p(T_f f, TT_g g) &= \inf_{T\in T_{tm}} \sqrt[p]{|\alpha|}d_p(f,T_f^{-1}TT_g g) \\ = \inf_{T'\in T_{tm}} \sqrt[p]{|\alpha|}d_p(f,T'g) &= \delta_p^A(f,g) \qquad \forall T_f, T_g \in T_{tm}\end{aligned}$$

Außerdem wird das Minimum angenommen: Wegen der Kompaktheit der Träger kann man eine Konstante R angeben, sodass die Träger von f und $T_{(\alpha,\beta)}g$ vollständig separiert werden und damit $\forall\beta > R$ ($|\alpha| = 1$):

$$\delta_p(f, Tg) = \sqrt[p]{\|f\|_p^p + \|g\|_p^p}.$$

Es reicht also, nur $\beta \in [-R, R]$ zu betrachten und damit ist auch klar, dass das Minimum angenommen wird. Ohne diese Eigenschaft wären die unteren Beweisführungen nicht möglich!

- δ_p^A ist definit: Sei $d_p^A(f, g) = 0$. Nun, weil das Minimum angenommen wird, $\exists T$ so dass $\|f - Tg\|_p = 0$. Wegen der Stetigkeit von f, Tg folgt $f = Tg$ und damit $f \propto_{T_{1d}} g$. Die Rückrichtung ist offensichtlich.
- Auch Symmetrie ist für δ_p^A gegeben:

$$\begin{aligned}\delta_p^A(f, g) &= \min_{T\in T_{tm}} d_p(f, Tg) \\ &= \min_{T\in T_{tm}} \sqrt[p]{|\alpha|} d_p(g, T^{-1}f) \overset{|\alpha|=1}{=} \delta_p^A(g, f)\end{aligned}$$

- Die Dreiecksungleichung gilt, weil das Minimum angenommen wird:

$$\begin{aligned}\delta_p^A(h, f) + \delta_p^A(h, g) &= \|h - T_f f\|_p + \|h - T_g g\|_p \geq \|T_f f - T_g g\|_p \\ &\geq \delta_p^A(T_f f, T_g g) = \delta_p^A(f, g)\end{aligned}$$

Alignierende Metrik für allgemeine Transformationen

Wie schon erwähnt wurde, ist δ_p^A aus (2.1) für allgemeine Koordinatentransformation nicht wohldefiniert. Betrachte dazu

$$\begin{aligned}\delta_p^A(T_{(\alpha,\beta)}f,\, T_{(\alpha,\beta)}g) &= \sqrt[p]{|\alpha|}\delta_p^A(f, g) \\ &\neq \delta_p^A(f, g) \quad \text{wenn } |\alpha| \neq 1 \wedge \delta_p^A(f, g) \neq 0.\end{aligned}$$

Ein besserer Versuch eine (Semi-)Metrik für allgemeine Transformationen T_{1d} zu definieren, besteht in

$$rd_p(f, g) := \begin{cases} 0 & f \equiv 0 \wedge g \equiv 0, \\ \|g\|_p & f \equiv 0, \\ \|f\|_p & g \equiv 0, \\ \frac{\|f-g\|_p}{\sqrt[p]{\min\{\lambda(support(f)), \lambda(support(g))\}}} & \text{sonst.} \end{cases}$$

Dabei wird das Ergebnis durch das Lebesque-Maß für den Träger von f ($\lambda(support(f))$) normiert. Dies führt zu

$$\delta_p^A(f,g) := \inf_{T\in T_{1d}} rd_p(f,Tg) = \inf_{T_1\in T_{sm}} \inf_{T_2\in T_t} rd_p(f,T_1T_2g).$$

Die zweite Gleichung sieht man ein, wenn man sich vor Augen führt, dass $\forall T \in T_{1d}$:

$$\inf_{T_1\in T_{sm}} \inf_{T_2\in T_t} rd_p(f,T_1T_2g) \leq rd_p(f,Tg)$$

und damit $\inf_{T\in T_{1d}} rd_p(f,Tg)$ auch nicht kleiner werden kann.
Man stellt auch leicht fest:

$$\lambda(support(T_{(\alpha,\beta)}f)) = \lambda(support(f\circ T^{-1}_{(\alpha,\beta)}) = |\alpha|\lambda(support(f)).$$

Analog zum "einfachen" translatorischen Fall kann man Folgendes feststellen (wir verzichten hier auf einen Beweis der Aussagen):

1. Das Minimum wird angenommen.
2. Die Definition ist wohldefiniert für Äquivalenzklassen von $\propto_{T_{1d}}$.
3. $\delta_p^A(f,g)$ ist definit.
4. $\delta_p^A(f,g)$ ist symmetrisch.

Leider gilt das nicht für die Dreiecksungleichung. Für unser Gegenbeispiel lassen wir einfachheitshalber die Bedingung der Stetigkeit fallen. Doch dieses Beispiel kann leicht abgeändert werden, sodass auch diese Bedingung erfüllt ist. So können wir die Indikator-Funktion für $B \subset \mathbb{R}$

$$\mathbb{I}_B(x) := \begin{cases} 1 & x \in B \\ 0 & x \notin B \end{cases}$$

benutzen. Und wählen (siehe Abbildung 2.1)

$$\begin{aligned} f &:= 3\mathbb{I}_{[0,1]}(x) \\ g &:= \mathbb{I}_{[0,1]}(x) + \varepsilon\cdot\mathbb{I}_{[1,2]}(x) \\ h &:= 2\mathbb{I}_{[0,1]}(x) + \varepsilon\cdot\mathbb{I}_{[1,2]}(x) \end{aligned}$$

Damit gilt für die alignierten Abstände, falls ε genügend klein ist:

$$\begin{aligned} \delta_p^A(f,g) &= \sqrt[p]{2^p+\varepsilon^p} \\ \delta_p^A(f,h) &= \sqrt[p]{1+\varepsilon^p} \\ \delta_p^A(g,h) &= \sqrt[p]{\frac{1}{2}} \end{aligned}$$

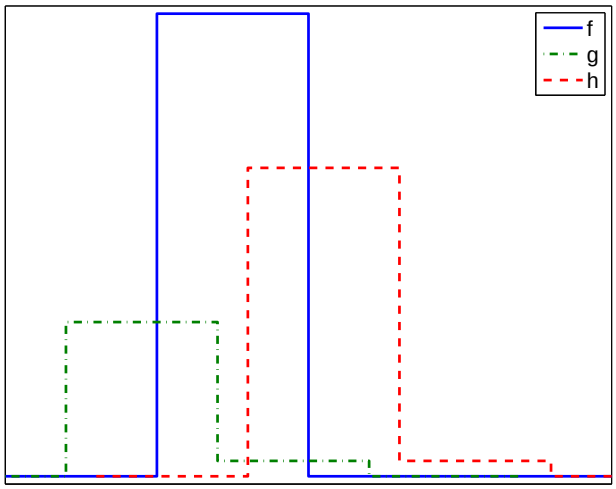

Abbildung 2.1: Für diese Funktionen gilt nicht die Dreiecksungleichung, denn $\delta_p^A(f,g) > \delta_p^A(f,h) + \delta_p^A(h,g)$

was zum Widerspruch mit der Dreiecksungleichung (dazu $\varepsilon \to 0$):

$$2 \overset{0 \leftarrow \varepsilon}{\longleftarrow} \delta_p^A(f,g) \leq \delta_p^A(f,h) + \delta_p^A(g,h) \overset{\varepsilon \to 0}{\longrightarrow} 1 + \sqrt[p]{\frac{1}{2}}$$

für alle $p \geq 1$ führt.

Ähnliche Probleme mussten wir auch für andere Definitionen von δ_p^A feststellen, wenn man z.B. statt $\min\{\lambda(support(f)), \lambda(support(g))\}$ das Produkt $\lambda(support(f)) \cdot \lambda(support(g))$ benutzt.

2.1.5 Messungen

Eine Messung kann nur mit einer endlichen Frequenz erfolgen - damit wird die ursprünglich kontinuierliche Observable diskretisiert. Als Messungen bezeichnen wir folgende Funktionen:

$$Mess(\mathbb{Z}, \mathbb{R}) := \{f : \mathbb{Z} \to \mathbb{R} \,|\, \mu(support(f)) := |support(f)| < \infty\}.$$

Der Messprozess wird durch den Diskretisierungsoperator D

$$D : Obs(\mathbb{R}, \mathbb{R}) \to Mess(\mathbb{Z}, \mathbb{R})$$

modelliert, welcher punktweise definiert wird:

$$[D(f)](z) := f(z) \quad \forall f \in Obs(\mathbb{R}, \mathbb{R}),\ \forall z \in \mathbb{Z}.$$

D ist nicht bijektiv, wir definieren jedoch mit

$$D^{-1} : Mess(\mathbb{Z}, \mathbb{R}) \to Obs(\mathbb{R}, \mathbb{R})$$
$$\left[D^{-1} g\right](x) := (1 - (x - x_{fl}))g(x_{fl}) + (x - x_{fl})g(x_{fl} + 1)$$

eine "Quasi-Inverse". Dabei ist $x_{fl} := \lfloor x \rfloor = \max\{z \leq x \mid z \in \mathbb{Z}\}$. Diese Quasi-Inverse ermöglicht uns die originale Observable aus der Messung (fehlerbehaftet) zu rekonstruieren.

Die von uns benutzte lineare Interpolation ist nur exemplarisch und kann z.B. durch die Nearest-Neighbor-Interpolation oder eine kompliziertere Interpolation ersetzt werden (für mehr Informationen zur Interpolation siehe Anhang 10.2.1).

Koordinatentransformationen induzieren Operatoren auf $Mess(\mathbb{Z}, \mathbb{R})$

$$T^d_{(\alpha,\beta)} := D \circ T_{(\alpha,\beta)} \circ D^{-1} \quad \forall T_{(\alpha,\beta)} \in T_{1d}.$$

Wir benutzen die vereinfachte Schreibweise $T_{(\alpha,\beta)}$ für $T^d_{(\alpha,\beta)}$. T und D kommutieren nicht, d.h. im Allgemeinen gilt

$$TD(f) = DT(f)$$

nicht, wobei auf der linken Seite natürlich T^d gemeint ist. Ausnahmen bilden die Koordinatentransformationen $T_{(\alpha,\beta)}$ mit $\alpha = \frac{1}{z}$, $z \in \mathbb{Z} \setminus \{0\}$ und $\beta \in \mathbb{Z}$. Diese Transformationen bilden jedoch keine Untergruppe, da die Inversen fehlen.

Anders sieht es bei den ganzzahligen Translationen $T^d_{tm} := \{(\alpha, \beta) \mid \alpha \in \{-1, 1\} \wedge \beta \in \mathbb{Z}\}$ aus. Offensichtlich handelt es sich dabei um eine Untergruppe.

Analog zum kontinuierlichen Fall gilt

$$Mess(\mathbb{Z}, \mathbb{R}) \subset \mathcal{L}^p(\mathbb{Z}, \mathcal{A}, \mu)$$

mit dem Zahlmaß $\mu(M) := |M| \; \forall M \subset \mathbb{Z}$.

Damit werden Metriken

$$d_p(f, g) := \sum_{z \in \mathbb{Z}} |f(z) - g(z)|^p \quad p = 1, 2 \ldots$$

vererbt.

Man sieht analog ein, dass

$$\delta^A_1(f, g) = \inf_{T \in T^d_{tm}} d_1(f, Tg)$$

eine Metrik auf den Äquivalenzklassen von $\propto_{T^d_{tm}}$ darstellt.

Man stößt auf ähnliche Probleme wie im kontinuierlichen Fall, wenn man eine Metrik für allgemeine Koordinatentransformationen finden möchte.

2.1.6 Messprozess

Die im obigen Abschnitt vorgestellte Messung ist eine Idealisierung. Klar ist, dass sich bei einer Messung Messfehler einschleichen. Es gibt unzählige Fehlerquellen, mit deren Auswirkungen unser Algorithmus klarkommen müsste. Wir möchten nun einige davon modellieren.

So ist die Vermessung der x-Achse bzw. der Zeit fehleranfällig. Bei unserer Modellierung werden nur eine globale Skalierung und die Verschiebung des Koordinatenursprungs berücksichtigt. Diese Fehler können durch die schon betrachteten globalen Koordinatentransformationen abgedeckt werden. Neben einem globalen Skalierungsfaktor kann es z.B. auch lokale Variationen geben, die zu lokalen Verzerrungen führen. Diese werden von dem Modell jedoch nicht abgedeckt.

Eine große Anzahl von Fehlerarten sind für die Messungen möglich (siehe auch Abschnitt 2.2.4 für Beispiele):

- Rauschen/Messfehler
- Messausfälle
- Durch eine fehlerhafte Kalibrierung von Geschwindigkeits- oder Längenmesseinrichtungen bedingte falsche Skalierung der Werte
- Vorliegende Trends, die z.B. auf Fehler im Produktionsprozess zurückzuführen sind. Es kann sich um einen konstanten Offset handeln oder viele andere “Grundfunktionen”
- Ausreißer, die z.B. durch Verunreinigungen zustande kommen.

Abgesehen von der Skalierung können diese Fehler als eine additive Funktion

$$err : \mathbb{Z} \times Obs(\mathbb{R}, \mathbb{R}) \to \mathbb{R}$$

dargestellt werden. $err(z, f(z))$ mit $z \in \mathbb{Z}$, $f \in Mess(\mathbb{Z}, \mathbb{R})$ sind Zufallsvariablen, die nicht unbedingt gleichverteilt oder unabhängig sind.

Für den normalverteilten Fehler könnten wir $err(z, f(z))$ wie i.i.d. (*independent and identically distributed*) behandeln: $err(z, f(z)) \stackrel{d}{=} N(0, \sigma)$ oder falls der Fehler linear mit dem Wert skaliert $err(z, f(z)) \stackrel{d}{=} N(0, \sigma_0 \cdot f(z))$. Doch bei einem Messausfall, der z.B. als

$$err(z, f) = -f(z)$$

modelliert werden kann, ist offensichtlich, dass die Fehler nicht mehr als unabhängig angesehen werden können: Ein Ausfall ist wahrscheinlicher, wenn die Messung auch beim vorherigen Messvorgang schon ausfiel.

Die durch die fehlerhafte Kalibrierung bedingte Skalierung der Werte kann durch die Multiplikation mit einem globalen Vorfaktor s_y realisiert werden. Mit Hilfe der folgenden punktweise definierten Fehleroperatoren

$$Mess(\mathbb{Z}, \mathbb{R}) \to Mess(\mathbb{Z}, \mathbb{R})$$

können wir unseren Überlegungen einen formalen Rahmen geben:

$$\begin{aligned} [S_s f](z) &= s \cdot f(z) \qquad (Skalierungsfehler), \\ [Ef](z) &= f(z) + err(z, f(z) \qquad (andereFehler) \end{aligned}$$

Messprozesse sind eine Erweiterung des Diskretisierungsoperators, der auch Fehler "kennt":

$$\begin{aligned} M &: Obs(\mathbb{R}, \mathbb{R}) \to Mess(\mathbb{Z}, \mathbb{R}) \\ M &:= ES_s DT_{(\alpha,\beta)} \\ [M(f)](z) &= s \cdot f\left(\frac{z-\beta}{\alpha}\right) + err\left(z, s \cdot f\left(\frac{z-\beta}{\alpha}\right)\right) \end{aligned}$$

Diese Zusammensetzung der Operatoren lässt sich wie folgt erklären:

- $T_{(\alpha,\,\beta)}$ ist verantwortlich für die Fehler der Zeitmessung.
- S_s steht für den globalen Skalierungsfehler der y-Achse.
- E modelliert weitere Fehler: Messausfälle, Ausreißer, Trends und Rauschen.

Es soll klargestellt werden, dass für eine Messanordnung der Messprozessoperator keine Konstante ist. Viel eher kann einer Messanordnung eine Verteilung der Messprozesse zugeordnet werden und das bezieht sich nicht nur auf den Fehleroperator E sondern auch auf $T_{(\alpha,\,\beta)}$: In einer Messanordnung ändern sich die Parameter α und β leicht mit jeder Messung.

2.2 Fragestellungen

2.2.1 Problem A

Wir betrachten nun eine Menge aus n Observablen

$$O := \{o_1, \ldots, o_n\} \subset Obs(\mathbb{R}, \mathbb{R}^m),$$

welche schon einmal vermessen wurden. Damit kennen wir

$$\mathcal{M} := \{M_i o_i \,|\, o_i \in O \text{ für } i = 1, \ldots n\}$$

dabei sind $M_i \in \{Obs(\mathbb{R},\mathbb{R}) \to Mess(\mathbb{Z},\mathbb{R})\}$ unabhängige (so ist unsere Annahme) identisch verteilte Zufallsvariablen, deren Verteilung charakteristisch für die Messanlage sind, mit deren Hilfe die Vermessung stattgefunden hat.

Des Weiteren wird ein $i^* \in \{1,\ldots,n\}$ gewählt und nochmal vermessen, so dass sich die Messung

$$\hat{m} = \hat{M}o_{i^*}$$

ergibt.

Problem A: Gegeben $\hat{m}$ und $\mathcal{M}$, finde heraus, welches o_i bei der Vermessung $\hat{m}$ ergab bzw. das dazu passende $M_i o_i \in \mathcal{M}$.

Im Allgemeinen, wenn man die Information über die Verteilung der Messungen

$$M : Obs(\mathbb{R},\mathbb{R}^m) \to Mess(\mathbb{Z},\mathbb{R}^m)$$

besitzen würde, wäre dies mit der Maximum-Likelihood-Methode (Gelb [30]) zu lösen.

So ist das Minimieren der L^2-Norm nichts anderes als das Maximieren der Likelihood-Funktion unter der Annahme, dass der Fehler normalverteilt ist.

Das Problem wird jedoch dadurch erschwert, dass wir keine Information über die Messungen bzw. deren Verteilung haben.

Obwohl die Lage aussichtslos erscheint, schaffen es Experten, dieses Problem zu lösen und zwar mit folgender Strategie:

- Bereiche mit Ausreißern werden ignoriert, da man daraus nicht viel Information gewinnen kann.
- Bei den normalen Bereichen (eventuell nach einer Trendbereinigung) wird der L^2-Abstand oder eine andere Norm als Maß für die Ähnlichkeit der Abschnitte herangezogen. Dabei können die Bereiche skaliert werden, damit sie besser zueinander passen.

Unsere Aufgabe ist es eine Heuristik zu finden, die auf realen Daten den menschlichen Experten in nichts nachstehen würde.

2.2.2 Problem B

Problem B: Gegeben zwei Messungen m und $\hat{m}$ soll bestimmt werden, ob sie durch das Vermessen der gleichen Observablen entstanden sind.

Obwohl diese Aufgabenstellung einfacher klingt, ist sie jedoch viel schwieriger als das Problem `A`, weil uns ein Vergleich mit anderen Messungen fehlt.

Beim Problem `A` handelt es sich um eine Clusteringanalyse: Wir teilen die Elemente aus $\mathcal{M}$ in zwei Cluster auf: die passende Messung (eine einelementige Menge) und die Menge der nicht passenden Messungen. Die passende Messung unterscheidet sich von allen anderen und stellt einen Ausreißer dar.

Die Idee, wie das Problem `B` trotzdem gelöst werden kann, wird im Abschnitt 3.5.1 vorgestellt. Sie bildet in einer leicht abgeänderten Form die Grundlage für die von uns vorgestellte Heuristik, deren Ergebnisse im Kapitel 7 zusammengefasst werden.

2.2.3 Anwendungsbeispiele

Für die Problemstellung A

Diese Arbeit beschäftigt sich vor allem mit dem Suchen und Finden von Dickenprofilen der Walzbänder aus Stahl oder Aluminium.

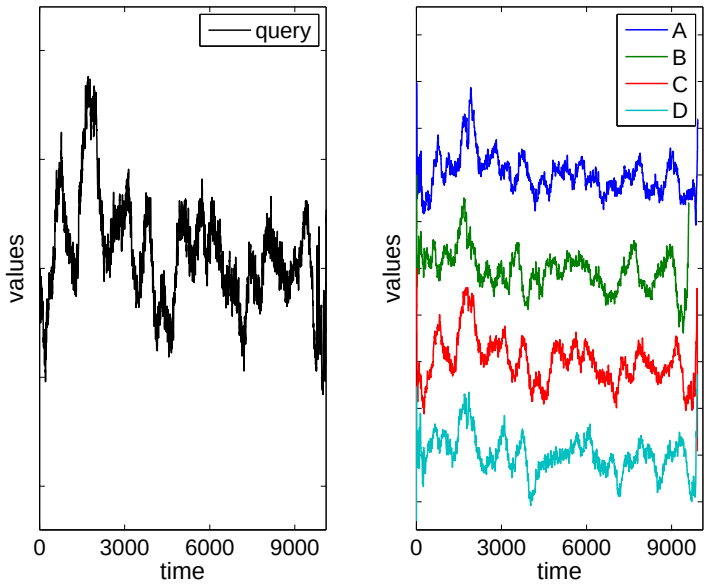

Abbildung 2.2: Ein Beispiel für die Problemstellung `A`: Sogar für diese sehr kleine Datenbank ist die Anfrage (schwarz, im rechten Bild) nicht leicht zu beantworten. Die richtige Antwort ist C (rot), auch wenn die Zeitreihe A (blau) einen ähnlichen groben Verlauf aufweist.

Das Problem, mit dem wir als erstes konfrontiert wurden, sieht wie folgt aus:

Die Dickenprofile der Metallbänder werden in jedem Produktionsschritt ver-

messen, da man sicher gehen möchte, dass die Toleranzen eingehalten werden. Am Ende jedes Produktionsschrittes werden die Bänder dann in sogenannten Coils aufgewickelt und zwischengelagert, bevor es mit der Verarbeitungskette weiter geht. Bei der Zwischenlagerung können Coils "verloren" gehen: Es lässt sich manchmal nicht mehr feststellen, welche "Identität" das verlorene Coil hat, sodass es nicht mehr klar ist, welche Schritte genau das Metallband schon durchlaufen und welchen Ursprung (chemische Komposition, Behandlungen usw.) es hat.

Nun stehen dem Betreiber zwei Wege zur Verfügung:

- Das Band völlig zu verwerfen, einzuschmelzen, neu zu walzen und den Verlust von einigen zig-tausend Euro in Kauf zu nehmen.

- Die Dicke des Bandes neu zu vermessen und mit den Einträgen in der Datenbank zu vergleichen, um die ID des "Findlings" zu bestimmen.

Bei der zweiten Möglichkeit handelt es sich um das Problem `A`: Es ist notwendig, tausende von Coils in einer Datenbank möglichst schnell anzuschauen, um den passenden Eintrag zu finden. Für ein kleines Beispiel siehe die Abbildung 2.2.

Für die Problemstellung B

Auch das zweite Problem spielt eine Rolle im Umfeld des Walzens, wenn man eine Online-Materialflusskontrolle implementieren möchte.

Die Vorteile der Materialflusskontrolle liegen auf der Hand: Der Verlust eines Coils ist nicht das schlimmste Szenario - eine Verwechslung kann zu wesentlich größeren Risiken führen, weil dabei unter Umständen Endprodukte ausgeliefert werden, die z.B. nicht die erwarteten mechanischen Eigenschaften besitzen. Schlimmstenfalls können solche vertauschte Coils später für Unfälle und andere Probleme verantwortlich sein.

Deswegen wäre es erstrebenswert, sich nicht nur auf fehleranfällige Bezeichnungen zu verlassen, sondern bei jedem Coil das Dickenprofil vor dem Produktionsschritt zu vermessen und mit der vorangegangenen Messung zu vergleichen (Abbildung 2.3) und diese so eindeutig zu identifizieren. Dieser Vergleich darf nicht mehrere Minuten lang dauern, weil der eigentliche Produktinsprozess selbst unter Umständen nur wenige Minuten in Anspruch nimmt.

Und so ist es notwendig, nur anhand der zwei vorliegenden Messungen die Frage nach der Zusammengehörigkeit zu beantworten (Problem `B`). Wird eine Verwechslung festgestellt, so muss die richtige ID gesucht werden, was jedoch wiederum das Problem `A` ist.

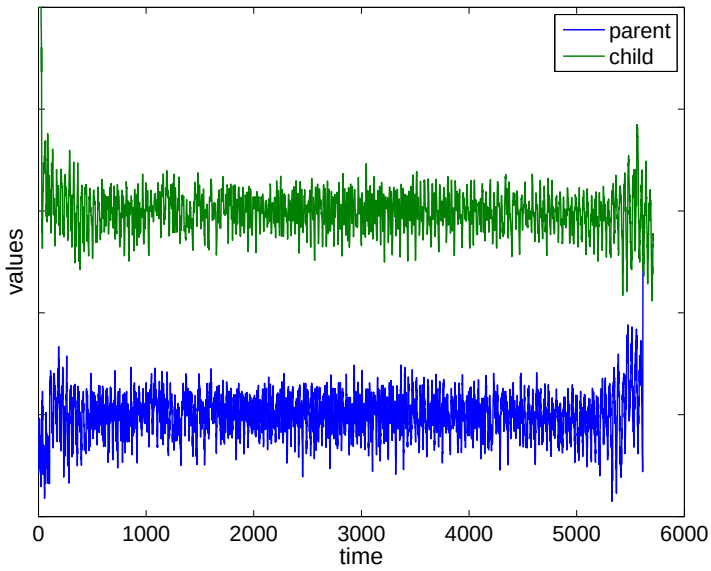

Abbildung 2.3: Ein Beispiel für die Problemstellung B: Ohne irgendwelche Anhaltspunkte zu haben, ist es nicht einfach zu sagen, ob die obigen zwei Messreihen zueinander passen oder nicht. Wie wir sehen werden, wird unser Framework im Stande sein, die Frage hier richtig mit "Ja" zu beantworten.

2.2.4 Zeitreihenbeispiele

In diesem Abschnitt möchten wir einige Zeitreihenpaare zeigen, die dem Leser zeigen sollen, mit welchen Problemen es ein funktionierender Algorithmus aufnehmen muss. Dabei werden die in die Datenbank $\mathcal{M}$ aufgenommenen ersten Messungen durchgehend mit blau (Legendenbezeichnung *parent*) dargestellt, während die zweite Messung $\hat{m}$ mit grün (Legendenbezeichnung *child*) gezeichnet wird.

Es soll beachtet werden, dass die dargestellten Zeitreihen schon aligniert sind, damit ein Vergleich dem ungeübten Leser leichter bzw. erst möglich gemacht wird.

Folgende interessante, oft vorkommende Fälle werden illustriert:

1. Gut passende Zeitreihen (Abbildung 2.4)
2. Gut passende, lange (ca. 10^5 Einträge) Zeitreihen (Abbildung 2.5)
3. Gut passende Zeitreihen, bei denen jedoch die y-Achse skaliert ist (Abbildung 2.5, links)
4. Trends und Offsets (Abbildung 2.6)
5. Ausreißer und Messausfälle (Abbildung 2.7)

6. Unterschiedliche Störungen mechanischer und elektronischer Herkunft (Abbildung 2.8)
7. Nur Teilstücke der Zeitreihen vorhanden (Abbildung 2.9)
8. Kopf/Fußschrott (Abbildung 2.10, links)
9. Veränderungen zwischen den Messungen (Abbildung 2.10, rechts)

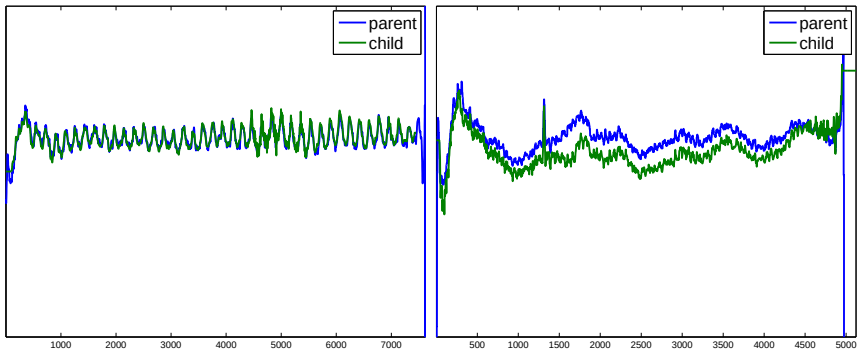

Abbildung 2.4: Links: *Ein Beispiel für zwei gut zusammenpassende Zeitreihen.* Rechts: *Obwohl die Zeitreihen in der Mitte auseinander driften, hat niemand einen Zweifel daran, dass diese beide Zeitreihen zusammen gehören - der Hauptgrund dafür ist der markante Peak bei* 1300.

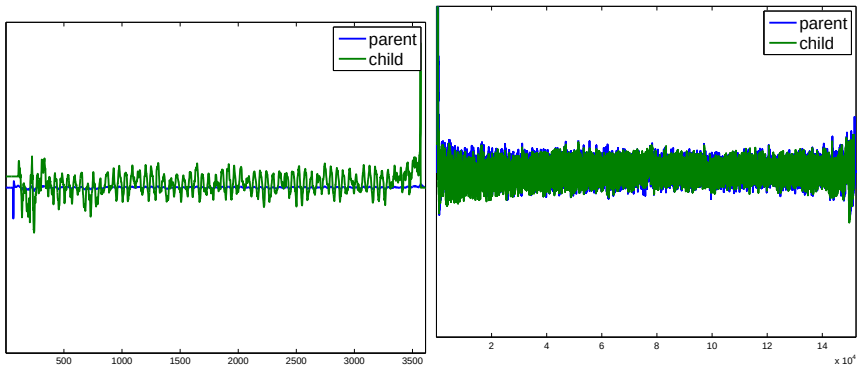

Abbildung 2.5: Links: *Hier ist die y-Achse bei der* child*-Zeitreihe ungefähr um den Faktor* 20 *skaliert.* Rechts: *Ein Beispiel für zwei gut zusammenpassende extrem lange Zeitreihen. Möchten man das Skalierungsproblem lösen, indem man alle Zeitreihen auf eine feste Länge (z.B.* 2000*) skaliert, so würde man bei diesen Zeitreihen nach der Skalierung keine Features mehr erkennen - damit wäre eine Zuordnung unmöglich.*

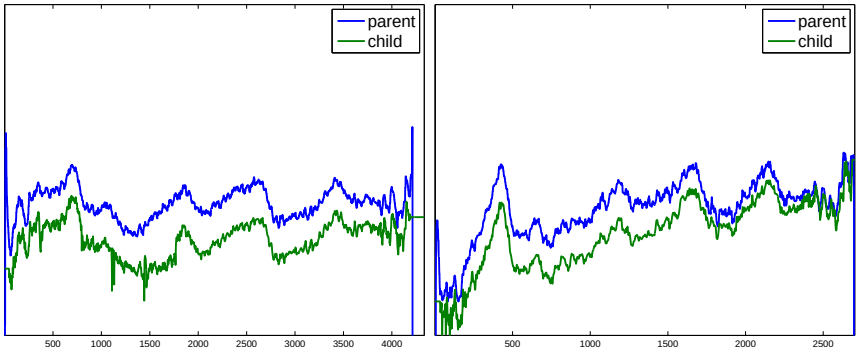

Abbildung 2.6: Links: *Eine additive Konstante lässt sich leicht durch die Normalisierung bereinigen. Die Normalisierung hat jedoch auch ihre "Nebenwirkungen" - z.B. einen negativen Effekt für die (sonst richtigen) Zeitreihen mit einem großen Ausreißer oder Messausfall (Abbildung 2.7).* Rechts*: Ein Trend lässt sich nicht so leicht bereinigen: Ohne eine Bezugszeitreihe ist es oft unklar, ob es sich um einen Trend oder wirklich eine abfallende/steigende Zeitreihe handelt.*

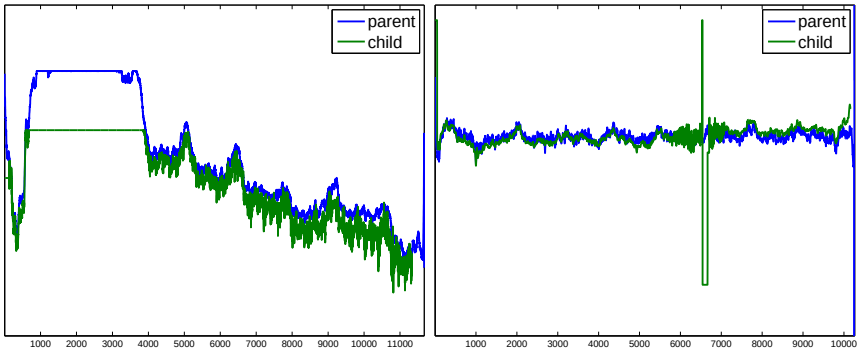

Abbildung 2.7: Links: *Ein Messausfall über eine längere Zeit.* Rechts: *Ein kurzer Messausfall bzw. eine kurze Messstörung.*

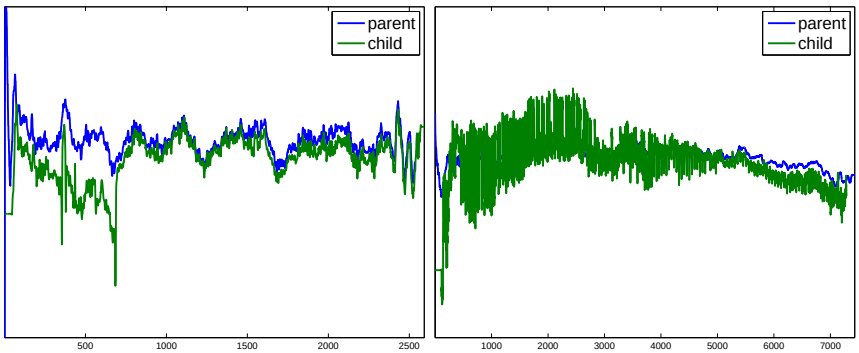

Abbildung 2.8: Links: *Die Ursache für den Anfangstrend ist nicht klar.* Rechts: *Durch Interferenzen in der Elektronik kann es zu Störungen kommen, die auf den ersten Blick einem heftigen Gaußrauschen ähneln.*

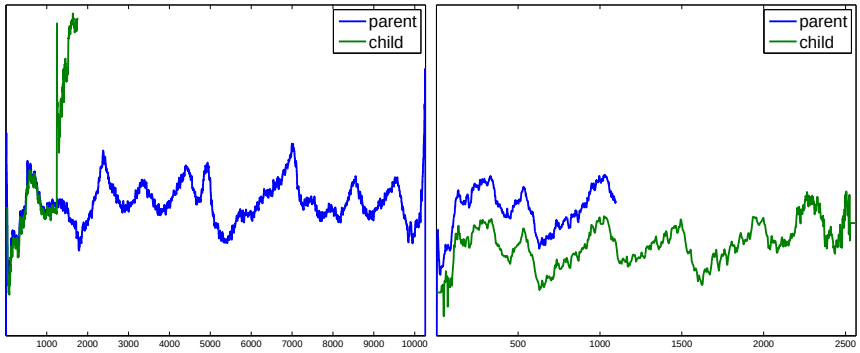

Abbildung 2.9: Links: *Bedingt durch einen Riss des Stahlbandes wurde nur ein Teil der* child-*Zeitreihe aufgezeichnet.* Rechts: *Die Heuristik zum Bestimmen der Ränder des Bandes schlug fehl, so dass nur ein Teil der* parent-*Zeitreihe erfasst wurde.*

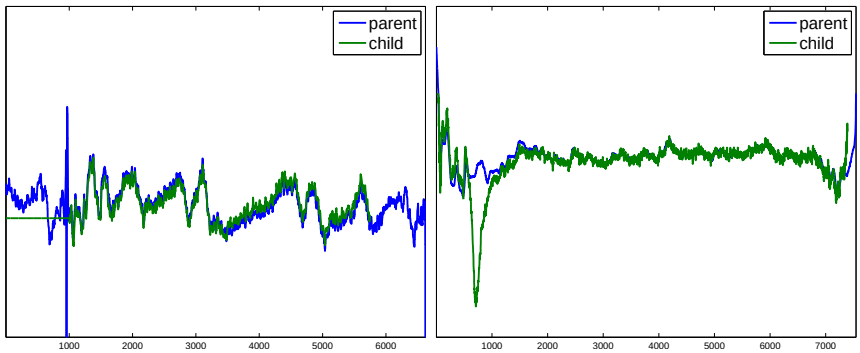

Abbildung 2.10: Links: *Das Gegenstück zum Fall in der Abbildung 2.9, rechts: Die Erfassung startete zu früh.* Rechts: *Eine Profiländerung durch ein mechanisches Einwirken zwischen den beiden Messungen.*

Kapitel 3

Frühere Arbeiten

Die Untersuchung der Zeitreihen blickt auf eine lange Geschichte zurück. In letzter Zeit gewinnt die Fähigkeit mit Zeitreihen umzugehen immer mehr an Bedeutung, weil immer mehr Prozesse vermessen und automatisch ausgewertet werden.

In diesem Abschnitt möchten wir jedoch nicht eine komplette historische Übersicht anbieten, sondern konzentrieren uns auf diejenige Ansätze, die entweder als *state-of-the-art* gelten oder schon für die Dickenprofile der Metallbänder angewendet wurden oder als Vorläufer unseres Frameworks angesehen werden können.

3.1 Abstandsbestimmung durch Alignieren

In diesem Abschnitt betrachten wir Methoden, die die Messreihen global vergleichen. Eine naheliegende Vorgehensweise besteht darin, aus allen zugelassenen Koordinatentransformationen diejenige zu wählen, die zum kleinsten Fehler in einer Norm der Wahl führt. Im Abschnitt 2.1.4 wurde schon ohne Erfolg versucht, solche alignierende Metriken zu entwerfen.

3.1.1 $\mathcal{L}^2$-Abstand

Kemeter [41] untersuchte in seiner Diplomarbeit den Ansatz, bei dem der $\mathcal{L}^2$-Abstand minimiert wurde. Die Ergebnisse wurden auch in Lipowsky u. a. [47] präsentiert.

Im ersten Schritt wurde das translatorische Alignment-Problem gelöst, d.h. für zwei Funktionen $f, g \in Mess(\mathbb{Z}, \mathbb{R})$

$$\min_{T_{(1,\beta)} \in T_t} z(\beta) := \min_{T_{(1,\beta)} \in T_t} d_2(f, Tg)$$

bestimmt.

Unsere Zielfunktion $z(\beta)$, die minimiert werden soll, lässt sich als eine Faltung aufschreiben:

$$\begin{aligned} z(\beta) &= \sum_x (f(x) - g(x-\beta))^2 \\ &= \sum_x f^2(x)dx + \sum_x g^2(x) - 2\sum_x f(x) \cdot g(x-\beta) \\ &= -2[f * \hat{S}g](\beta) + const \end{aligned}$$

mit

$$[\hat{S}g](x) = g(-x).$$

Damit ist das Minimieren von $z(\beta)$ äquivalent zum Maximieren von

$$[f * \hat{S}g](\beta).$$

Vernachlässigt man die Randeffekte, so kann man f und g als periodische Funktionen auffassen und die Faltung mit Hilfe von FFT in $O(n \log n)$ ausrechnen (für mathematische Grundlagen siehe Anhang 10.2).

Die möglichen Skalierungen α bzw. Spiegelungen müssen jedoch abgetastet werden, nur das translatorische Problem konnte mit FFT beschleunigt werden. Der damit verbundene zeitliche Aufwand macht dieses Verfahren jedoch für eine Suche in einer großen Datenbank (Fragenstellung `A`) ungeeignet.

Auch eine hierarchische Vorgehensweise wurde untersucht. Dabei wurde die Laufzeit des Suchvorgangs dadurch reduziert, dass man sich alle Daten zuerst in der gröbsten Auflösung anschaute und die vielversprechenden Zeitreihen auswählte bzw. die unpassenden Zeitreihen mit wenig Aufwand aussortierte. Mit jeder Erhöhung der Auflösung reduzierte sich die Anzahl der Kandidaten, sodass am Ende nur sehr wenige Zeitreihen bei der vollen Auflösung untersucht werden mussten. Als ein Nachteil dieser Methode stellt sich heraus, dass es nicht ganz klar ist, wie viele und welche Zeitreihen auf jeder Auflösungsstufe ausgesiebt werden dürfen - in der Diplomarbeit wurde dies vor allem mit Hilfe von Experimenten festgestellt.

3.1.2 Cross-Correlation

In der Statistik lässt sich mit Hilfe von Cross-Correlation bestimmen, ob zwei Zeitreihen korreliert sind:

$$\rho(f,g) = \frac{1}{n}\sum_{i=1}^{n} \frac{(f(i) - \mu_f) \cdot (g(i) - \mu_g)}{\sigma_f \cdot \sigma_g}$$

mit μ_f, σ_f - der Mittelwert und die Standardabweichung der Zeitreihe f und μ_g, σ_g entsprechend für die Zeitreihe g.

Ist $\rho(f,g) \approx 0$, so haben die Reihen wenig miteinander zu tun. $|\rho(f,g)| \approx 1$ ist ein Zeichen für einen Zusammenhang: So ist z.B. $\rho(f,f) = 1$ und $\rho(f,-f) = -1$.

Es liegt also nahe, die Zielfunktion

$$z_{CC}(\beta) = \rho(f(x), g(x-\beta))$$

zu maximieren. Wir verzichten auf die Betragsbildung, weil wir davon ausgehen, dass die Zeitreihen nicht an der x-Achse gespiegelt werden.

Es lässt sich leicht zeigen, dass für die periodischen Funktionen f und g das Maximieren der Zielfunktion z_{CC} äquivalent zum Minimieren der Zielfunktion z ist:

$$\begin{aligned}
z_{CC}(\beta) &= \frac{1}{n}\sum_{i=1}^{n} \frac{(f(i)-\mu_f)\cdot(g(i-\beta)-\mu_g)}{\sigma_f \cdot \sigma_g} \\
&= \frac{1}{\sigma_f \sigma_y}\left(\frac{1}{n}\sum_{i=1}^{n} f(i)g(i-\beta)\right. \\
&\qquad \left. -\mu_g \frac{1}{n}\sum_{i=1}^{n} f(i) - \mu_f \frac{1}{n}\sum_{i=1}^{n} g(i) + \frac{1}{n}\sum_{i=1}^{n} \mu_f \mu_g\right) \\
&= \frac{1}{\sigma_f \sigma_y}\left(\frac{1}{n}\sum_{i=1}^{n} f(i)g(i-\beta) - \mu_f\mu_g\right) \\
&= c\sum_{i=1}^{n} f(i)g(i-\beta) - \tilde{c} \\
z(\beta) &= \sum_{i=1}^{n} (f(i) - g(i-\beta))^2 \\
&= \sum_{i=1}^{n} \left(f(i) - 2f(i)g(i-\beta) + g(i-\beta)^2\right) \\
&= -2\sum_{i=1}^{n} f(i)g(i-\beta) + \hat{c}
\end{aligned}$$

mit passenden Konstanten $c > 0$, $\tilde{c}$ und $\hat{c}$.

Es macht also im Prinzip keinen Unterschied, ob man den $\mathcal{L}^2$-Abstand oder die Cross-Correlation zum Lösen des translatorischen Alignments verwendet - es kommt die gleiche Transformation heraus. Für die Suche der Transformation sind auch hier mit Hilfe von FFT nur $O(n \log n)$ Operationen notwendig.

Es macht jedoch einen Unterschied, welches Maß man verwendet, um den nächsten Nachbarn zu bestimmen: Während eine additive Konstante die Nachbarschaftsbeziehungen bei der $\mathcal{L}^2$-Norm durcheinander bringt, wird sie keinen Einfluss auf die mit Hilfe der Cross-Correlation bestimmten Nachbarschaften haben.

Benutzt man jedoch normalisierte Zeitreihen

$$\hat{f}(x) := \frac{f(x) - \mu_f}{\sigma_f},$$

so gilt zum einen wegen $\mu_{\hat{f}} = 0$ und $\sigma_{\hat{f}} = 1$

$$\rho(\hat{f}, \hat{g}) = \rho(f, g),$$

zum anderen

$$\left\|\hat{f}\right\|_2^2 = \frac{1}{\sigma_f^2} \sum_{i=1}^{n} (f(x) - \mu_f)^2 = \frac{1}{\sigma_f^2} n \cdot \sigma_f^2 = n$$

und schließlich folgt daraus, dass die Nachbarschaftsbeziehung zwischen normalisierten Zeitreihen identisch für beide Verfahren ist:

$$\begin{gathered}
\rho(\hat{f}, \hat{g}) \geq \rho(\hat{f}, \hat{g}') \Leftrightarrow \\
\sum_{i=1}^{n} \hat{f}(i)\hat{g}(i) \geq \sum_{i=1}^{n} \hat{f}(i)\hat{g}'(i) \Leftrightarrow \\
n - 2\sum_{i=1}^{n} \hat{f}(i)\hat{g}(i) + n \leq n - 2\sum_{i=1}^{n} \hat{f}(i)\hat{g}'(i) + n \Leftrightarrow \\
\sum (\hat{f}(i) - \hat{g}(i))^2 \leq \sum (\hat{f}(i) - \hat{g}'(i))^2 \Leftrightarrow \\
\left\|\hat{f} - \hat{g}\right\|_2 \leq \left\|\hat{f} - \hat{g}'\right\|_2 .
\end{gathered}$$

Damit stellt die Cross-Correlation nur eine spezielle Vorgehensweise der Minimierung des $\mathcal{L}^2$-Abstands dar. Man kann sich z.B. vorstellen, die Zeitreihe nicht mit dem Mittelwert, sondern mit dem Median zu bereinigen, um den Vorgang robuster gegenüber Ausreißern zu machen.

3.1.3 Mutual Information

Die Mutual Information (MI) für zwei Zufallsvariablen X und Y - im Deutschen oft als *gegenseitige Information* oder *Transinformation* bezeichnet - ist definiert als:

$$I(X;Y) := \int_{\mathcal{S}} f(x,y) \log_2 \frac{f(x,y)}{f(x) \cdot f(y)} \, dxdy = E\left(\log_2 \frac{f(x,y)}{f(x) \cdot f(y)}\right).$$

Dabei folgen wir Ausführungen und Definitionen aus Cover und Thomas [21]:

- $f(x,y)$ - gemeinsame Verteilungsdichte von X und Y.

- $f(x) := \int_{\mathbb{R}} f(x,y)dy$ - die Verteilungsdichte von X.
- $f(y) := \int_{\mathbb{R}} f(x,y)dx$ - die Verteilungsdichte von Y.
- $S = support(f(x,y))$.

Sind X und Y stochastisch unabhängig, so gilt:

$$f(x,y) = f(x) \cdot f(y) \Rightarrow$$
$$E\left(\log_2 \frac{f(x,y)}{f(x) \cdot f(y)}\right) = E\left(\log_2 1\right) = 0.$$

MI $I(X;Y)$ ist am größten für $Y = X$ (genauer für $Y = b(X)$ mit einer bijektiven Abbildung b), wie man es auch intuitiv erwarten würde. Wir sparen uns die formalen Details des Beweises an dieser Stelle.

In Viola und Wells [62] wurde vorgeschlagen, das Alignment-Problem zu lösen, indem man die Mutual Information der beiden Datensätze durch das richtige Alignment maximiert. Diese Vorgehensweise wurde mehrfach erfolgreich bei der Fusion von multimodalen Daten angewandt, z.B. bei der CT-PET-Fusion (siehe Maes u. a. [49]).

Angepasst an unsere Problemstellung mit zwei Zeitreihen $X = (x_1, \ldots, x_{N_X})$ und $Y = (y_1, \ldots, y_{N_Y})$ sieht das Verfahren wie folgt aus:

1. Schritt: *Schätzung der Verteilungen $f(x)$ und $f(y)$ aus den Werten der Zeitreihen X und Y*: Dafür ist es möglich, die als *parzen window* bekannte Methode einzusetzen (siehe Duda und Hart [24]):

$$f(x) \approx f_s(x) := \frac{1}{|I_h|} \sum_{i \in I_h} G(x - x_i, \sigma_x) \qquad I_h \subset \{1, \ldots, N_X\},$$
$$f(y) \approx f_s(y) := \frac{1}{|I_g|} \sum_{i \in I_g} G(y - y_i, \sigma_y) \qquad I_g \subset \{1, \ldots, N_Y\}.$$

Dabei werden I_h und I_g zufällig gewählt. Mit $G(\cdot, \sigma)$ wird wie gewohnt die Gaußfunktion bezeichnet. Die Parameter σ_x und σ_y sind passend gewählt. Es ist allerdings nicht notwendig, die Normalverteilung zu benutzen - andere Verteilungen sind durchaus denkbar. Das genauste Ergebnis bekommt man, wenn alle Elemente aus X und Y gewählt werden.

2. Schritt: *Bewertung einer Transformation $T_{(\alpha\,,\beta)}$*

1. *Schätzung der gemeinsamen Verteilung $f(x,y)$*: Durch $T_{(\alpha\,,\beta)}$ ergeben sich die Zuordnungen

$$t : \{1, \ldots, N_X\} \to \{1, \ldots, N_Y\}$$

und damit die zueinander gehörenden Werte-Paare

$$Common := \left((x_1, y_{t(1)}), \ldots, (x_{N_X}, y_{t(N_X)})\right).$$

Von diesen werden nun einige zufällig gewählt

$$I_s \subset \{1, \ldots, N_X\} \quad \text{mit } |I_s| =: N_s,$$

um die gemeinsame Verteilung

$$f(x,y) \approx f_s(x,y) = \frac{1}{N_s} \sum_{i \in I_s} G(x - x_i, \sigma_x) \cdot G(y - y_{t(i)}, \sigma_y)$$

zu schätzen.

2. *Berechnung von* $I(X;Y)$: Wähle zufällig

$$I_a \subset \{1, \ldots, N_h\} \quad \text{mit } |I_a| =: N_a$$

aus und berechne:

$$I(X;Y) = E\left(\log_2 \frac{f(x,y)}{f(x) \cdot f(y)}\right) \approx$$
$$I_s(X;Y) := \frac{1}{N_a} \log_2 \sum_{i \in I_a} \frac{f_s(x_i, y_{t(i)})}{f_s(x_i) \cdot f_s(y_{t(i)})}.$$

3. Schritt: *Maximierung von* $I_s(X;Y)$: Finde diejenige Koordinatentransformation $T_{(\alpha^*, \beta^*)}$, die $I_s(X;Y)$ maximiert.

Der Rechenaufwand verbunden mit der Schätzung der Wahrscheinlichkeiten und der Berechnung der Mutual Information, zahlt sich beim multimodalen Alignment aus, bei dem der Zusammenhang zwischen den Werten beider Aufnahmen bzw. Zeitreihen nicht offensichtlich oder gar unbekannt ist.

Was passiert, falls wir annehmen, dass sich die beiden Zufallsvariablen nur durch einen (Mess-)Fehler unterscheiden, d.h. $X = Y + \xi$ mit ξ- eine Zufallsvariable mit der Wahrscheinlichkeitsdichte $f_\xi(\xi)$, die wir kennen?

Zuerst sei bemerkt, dass in dem Fall die folgende Identität gilt:

$$\frac{f(x,y)}{f(y)} = f(x|y) = f_\xi(x - y)$$

und damit

$$I(X;Y) = E\left(\log_2 \frac{f(x,y)}{f(x) \cdot f(y)}\right) = E\left(\log_2 \frac{1}{f(x)}\right) + E\left(\log_2 f_\xi(x - y)\right).$$

Da der erste Summand einen von der Koordinatentransformation unabhängigen Wert darstellt, ist die Maximierung der Mutual Information äquivalent zur Maximierung des zweiten Summanden. Dies führt uns zu:

$$S_2 := E\left(\log_2 f_\xi(x-y)\right) \approx \frac{1}{N_X}\sum_{i=1}^{N_X} \log_2 f_\xi(x_i - y_{t(i)})$$

Nehmen wir für den Fehler ξ die Normalverteilung $N(0,\sigma)$ an, so gilt $f_\xi(\xi) = G(\xi,\sigma)$ und damit wird die Maximierung der Mutual Information äquivalent zur Minimierung des $\mathcal{L}^2$-Abstands:

$$\begin{aligned} S_2 &\approx \frac{1}{N_X}\sum_{i=1}^{N_X} \log_2 \left(C \cdot e^{-\frac{(x_i - y_{t(i)})^2}{2\sigma^2}} \right) \\ &= -C' \frac{1}{N_X}\sum_{i=1}^{N_X} (x_i - y_{t(i)})^2 + \tilde{C} \end{aligned}$$

mit passenden Konstanten C, C' und $\tilde{C}$.

Damit steht fest: Der zusätzliche Rechenaufwand lohnt sich nicht, falls wir davon ausgehen, dass die Fehler normal verteilt sind.

Ist die Verteilung des Fehlers ξ unbekannt, so muss diese zuerst aus den vorliegenden Zuordnungen *Common* geschätzt werden, was den Rechenaufwand deutlich nach oben treibt.

Auch wenn die Maximierung der Mutual Information in den meisten Fällen eine robustere Alternative darstellt, die vor allem mit den Ausreißern besser als $\mathcal{L}^2$-Abstand zurechtkommt (für ein Beispiel siehe Abbildung 3.1), wird dieser durch die zusätzliche Laufzeit teuer bezahlt. Erschwerend kommt hinzu, dass man nicht mehr die Möglichkeit hat, das translatorische Problem mit Hilfe von FFT zu beschleunigen.

3.1.4 Richtig aligniert $\neq$ richtig zugeordnet

An dieser Stelle möchten wir einen wichtigen Punkt betonen: Der Fakt, dass man zwei Reihen richtig aligniert, bedeutet noch nicht, dass man in einer Datenbank die richtige dazu passende Zeitreihe findet.

Um dies zu illustrieren betrachten wir ein kleines Beispiel (visualisiert in Abbildung 3.2): Es wird eine Anfrage (in den Abbildungen durch die blaue Zeitreihe dargestellt) an eine Datenbank gestellt. Diese Datenbank besteht aus zwei Zeitreihen: Die richtige (grüne) Zeitreihe, die jedoch einen Messausfall und einen Trend aufweist und eine falsche (rote) Zeitreihe, die mit der Anfrage nichts zu tun hat.

Wir bestimmen das bestmögliche Alignment mit Hilfe der oben besprochenen Verfahren: $\mathcal{L}^2$-Abstand, Cross-Correlation und Mutual Information. Die nachfolgenden Abbildungen stellen die Ergebnisse dar.

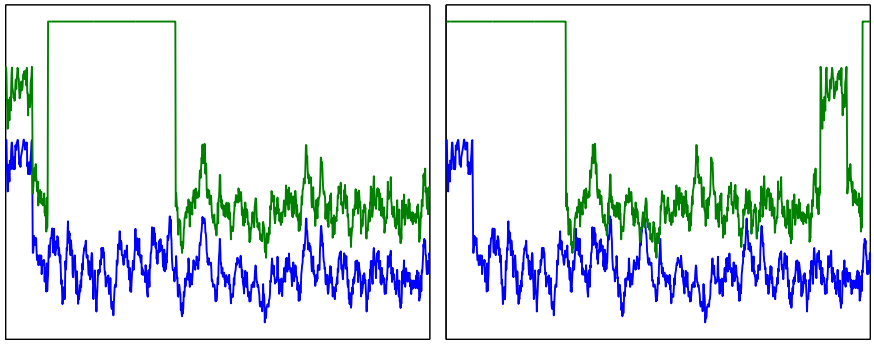

Abbildung 3.1: Das linke Diagramm zeigt das richtige Alignment, das auch trotz eines konstanten Offsets und eines Messausfalls durch die Maximierung der Mutual Information gefunden wurde. Das rechte Diagramm zeigt das falsche Alignment, welches durch die Minimierung des $\mathcal{L}^2$-Abstands bestimmt wurde.

3.2 Globale Deskriptoren

Eine Möglichkeit die Laufzeit zu reduzieren, ist das Benutzen von sogenannten Deskriptoren. Die Deskriptoren erfüllen zwei Voraussetzungen:

1. Sie sind invariant unter einer großen Anzahl von Transformationen (am besten unter T_{1d}).

2. Sie haben eine kleinere Dimension als die Zeitreihe, welche durch sie beschrieben wird.

Diskrete Fourier Transformation-Deskriptoren

Einer der ersten Versuchen die Dimension zu reduzieren, wurde in Agrawal u. a. [1] unternommen. Dabei wurden nur die ersten d Koeffizienten der Fourier-Transformierten betrachtet: In den Zeitreihen tragen oft nur die Messfehler zu den höheren Frequenzen bei. So können zwei Zeitreihen in $O(d)$ mit $d << n$ nach einem Vorverarbeitungsschritt in $O(n \log n)$ verglichen werden. Dies ermöglicht eine schnelle Suche in großen Datenbanken.

Im Unterschied zu den in Agrawal u. a. [1] betrachteten Problemen, bei denen es nur auf die Berechnung der $\mathcal{L}^2$-Norm ankam, muss in unserem Fall auch noch das Alignmentproblem gelöst werden. Es lässt sich ein translationsinvarianter Deskriptor erstellen, indem man die Beträge der Fourier-Koeffizienten betrachtet, denn eine Translation führt nur zur Änderung der Phase der Fourier-Koeffizienten:

$$\hat{\mathcal{F}}\left(f(\cdot + \beta)\right)(\omega) = \hat{\mathcal{F}}\left(f\right)(\omega) \cdot e^{-i2\pi\beta\omega}$$

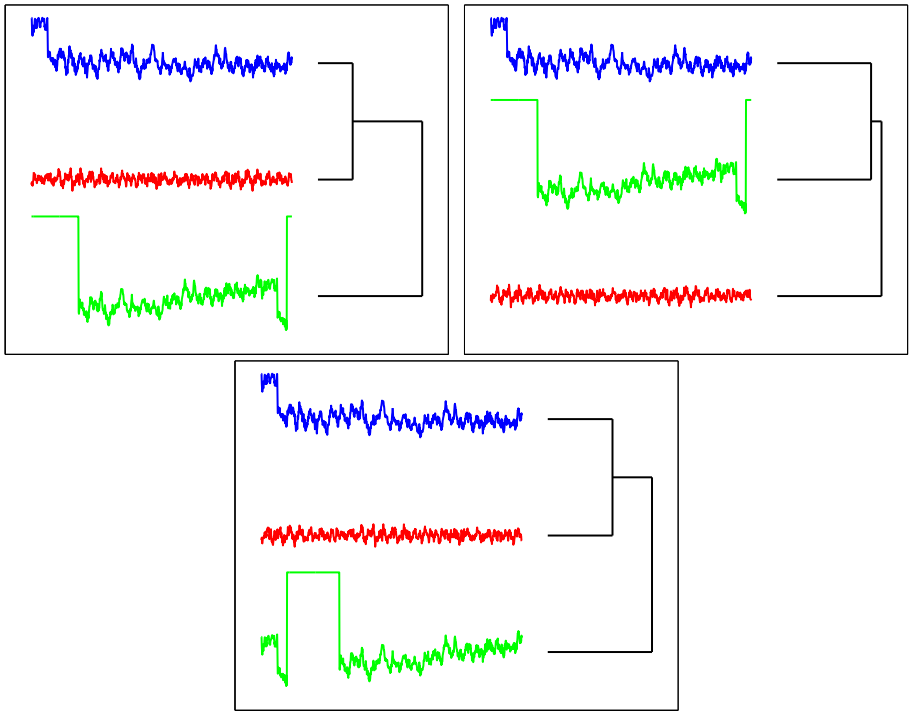

Abbildung 3.2: Das erste Dendrogramm stellt die Ergebnisse mit dem $\mathcal{L}^2$-Abstand dar: Das richtige blau-grüne Alignment wurde nicht gefunden und als Folge erscheinen die blaue und die rote Zeitreihe näher aneinander zu sein, obwohl sie nicht zusammenpassen. Im zweiten Dendrogramm ist der Ausgang der Anfrage vorgestellt, falls die Cross-Correlation benutzt wurde: Das richtige Alignment wurde nicht gefunden, durch Zufall haben die blaue und die grüne Zeitreihe einen kleineren Abstand: Bei mehreren Zeitreihen in der Datenbank wäre grün wohl nicht mehr die Top 1 *Antwort der Anfrage. Im letzten Dendrogramm sieht man, dass, obwohl das blau-grüne Aligment mit Hilfe von Mutual Information richtig bestimmt wurde, die falsche Zeitreihe Top* 1 *der Anfrage ist.*

Die Translationsinvarianz reicht im Allgemeinen nicht, weil Skalierungen oft nicht vernachlässigbar sind.

Ein skalierungsinvarianter DFT-Deskriptor

Es existiert auch eine Möglichkeit, skalierungs- und translationsinvariante DFT-Deskriptoren zu erzeugen. Betrachte dazu zwei Funktionen f und $f'(x) = f(\alpha x + \beta)$ mit (einfachheitshalber) $\alpha > 0$. Betrachtet man die Be-

träge der Fourier-Transformierten M und M', so stellt man fest:

$$\begin{aligned} M(\omega) &= \left|\hat{\mathcal{F}}\left(f\right)\left(\omega\right)\right| \\ M'(\omega) &= \left|\frac{1}{\alpha}\hat{\mathcal{F}}\left(f\right)\left(\frac{\omega}{\alpha}\right)\cdot e^{-i2\pi\beta\alpha^{-1}\omega}\right| \Rightarrow \\ M'(\omega) &= \frac{1}{\alpha}M\left(\frac{\omega}{\alpha}\right) \end{aligned}$$

Der Koordinatenwechsel in ein logarithmisches Koordinatensystem ergibt ($y := \log\omega$):

$$M'(y) = \frac{1}{\alpha}M(y-s) \quad s := \log\alpha.$$

Eine erneute Fouriertransformation liefert dann den bis auf eine Normierung invarianten Deskriptor:

$$\begin{aligned} D(\omega) &= |\hat{\mathcal{F}}\left(M\right)\left(\omega\right)| \\ D'(\omega) &= \frac{1}{\alpha}|\hat{\mathcal{F}}\left(M\right)\left(\omega\right)\cdot e^{i2\pi s\omega}| \Rightarrow \\ D'(\omega) &= \frac{1}{\alpha}D(\omega) \end{aligned}$$

Dieser Deskriptor kann für einen schnellen Vergleich benutzt werden. Es ist durchaus denkbar, nur die niedrigeren Frequenzen beim Vergleich zu beachten oder $\log M(y)$ statt $M(y)$ zu benutzen. Mit dieser Methode kann auch das beste Alignment bestimmt werden, wie man z.B. Lipowsky u. a. [47] entnehmen kann.

PAA

Die *Piecewise Aggregate Approximation* (PAA, Keogh u. a. [43]) bietet noch eine Möglichkeit, die Laufzeit zu reduzieren: Mehrere Datenpunkte werden zusammengefasst und durch ihren Mittelwert repräsentiert. Damit lassen sich unterschiedliche Auflösungen erstellen (Abbildung 3.3).

Man macht sich zunutze, dass das Alignieren für grobe Auflösungen deutlich weniger Aufwand benötigt als für die volle Auflösung. Trotzdem lassen sich schon viele Bänder auf Grund dieser groben Betrachtung ausschließen und nur ganz wenige müssen mit der maximalen Auflösung verglichen werden.

Diese Idee des *early exits* ist universal und nicht nur auf PAA anwendbar. Sie wurde z.B. eindrucksvoll in Keogh und Ratanamahatana [44] in Verbindung mit DTW (Abschnitt 3.3) eingesetzt.

Es gibt neben PAA auch weitere Methoden, um die Dimension zu reduzieren: die schon erwähnte DFT (Agrawal u. a. [1]), *Singular Value Decomposition* (SVD, Keogh u. a. [43]), *Diskrete Wavelet Transformation* (DWT) und *Adaptive Piecewise Constant Approximation* (APCA, Chakrabarti u. a. [16]).

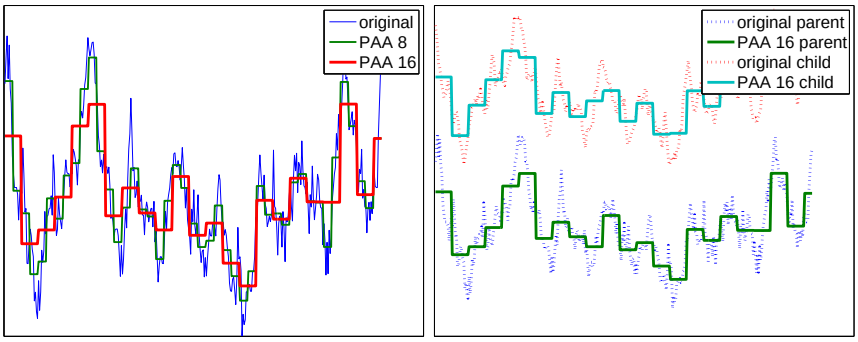

Abbildung 3.3: Im linken Bild: Das Original (kann als PAA 1 aufgefasst werden), PAA 8 (acht Datenpunkte werden zusammengefasst) und PAA 16. Im rechten Bild sieht man, dass PAA 16 mehr als ausreichend ist, um die Ähnlichkeit dieser zwei Zeitreihen festzustellen.

Jedoch ist keine von ihnen so leicht zu berechnen und so intuitiv zu behandeln wie PAA.

Bei der Benutzung von PAA sollte man Folgendes beachten: Eine zu große Reduktion kann dazu führen, dass die Merkmale, anhand welcher die Zeitreihen unterschieden werden können, durch die Mittelung eliminiert werden. Die Stufe des eingesetzten PAAs muss daher experimentell anhand der Daten bestimmt werden.

Die Methoden zur Reduktion der Dimensionalität zielen jedoch alle auf die Reduktion der Laufzeit ab und von keiner von ihnen kann man eine höhere Genauigkeit erwarten als vom Alignment der Zeitreihen mit maximaler Auflösung. Deswegen werden wir im weiteren Verlauf vor allem die Güte beim Aligment mit der maximalen Auflösung beurteilen und erst im nächsten Schritt entscheiden, ob diese Idee weiter verfolgt und deren Ausführung beschleunigt werden soll.

3.3 DTW

Eine weitere Standardmethode der Messreihen-Untersuchung ist *Dynamic Time Warping* (DTW) bzw. die schnellere Version *Bounded Dynamic Time Warping* (BDTW).

Wie der Name schon teilweise verrät, handelt es sich dabei um ein dynamisches Programm. DTW wurde zuerst in der Spracherkennung angewandt (Sakoe und Chiba [58]), fand jedoch schnell seine Anwendung für allgemeine Zeitreihen (Berndt und Clifford [8]). Die Laufzeit des DTW-Algorithmus ist $O(n^2)$, falls n die Länge der beiden Zeitreihen ist, was sich zumindest für unsere Problemstellungen (vor allem die Problemstellung `A`) als zu langsam

erweisen wird. Beschränkt man den Suchraum für die optimale Anordnung (siehe z.B. Berndt und Clifford [8], Keogh und Ratanamahatana [44]), so kann die Laufzeit auf $O(w \cdot n)$ reduziert werden. Dabei ist w - die Breite des Fensters, in dem sich der sogenannte *wrapping path* bewegen kann. Für diesen als BDTW bekannten Algorithmus kann w kleiner als 10% von n gewählt werden (Keogh und Ratanamahatana [42]), vergleiche Abbildung 3.4.

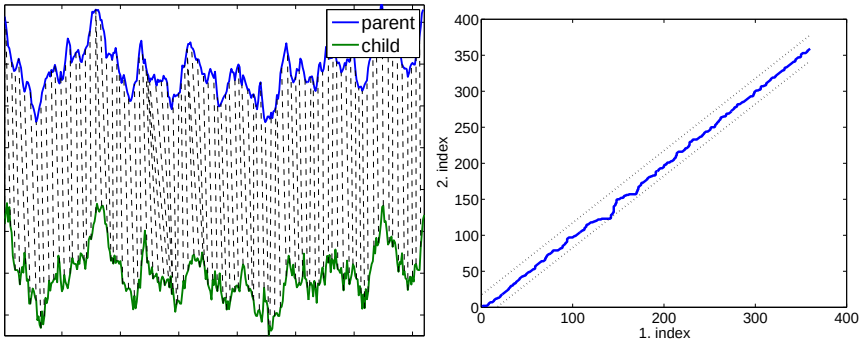

Abbildung 3.4: Das linke Bild zeigt das mit Hilfe von DTW ermittelte Alignment zweier Zeitreihen. Dabei werden die zusammengehörenden Stellen durch die vertikalen gestrichelten Linien markiert. Das rechte Bild zeigt die Zuordnungen der Koordinaten beider Zeitreihen. Es ist offensichtlich, dass man bei der Suche nicht den ganzen Raum erforschen muss. Es reicht auch, wenn man nur ein schmales Band um die Diagonale betrachtet (begrenzt durch zwei gestrichelte Linien). Es fällt weiter auf, dass die gefundene Koordinatenzuordnung nicht unbedingt die wirklich zu Grunde liegende Koordinatentransformation beschreibt: Ein Knick im Bereich von $x \approx 150$ ist offensichtlich auf eine Störung durch Messfehler zurückzuführen und nicht auf eine lokale Änderung der Skalierung.

In letzter Zeit gewann DTW immer mehr an Bedeutung, nachdem gezeigt wurde, dass DTW auch für eine schnelle exakte Suche in großen Datenbanken geeignet ist (Keogh und Ratanamahatana [44] bzw. Rakthanmanon u. a. [56]). Während die vorgeschlagene Indexierung das Problem der langen Laufzeit weitgehend entschärft hat, gibt es aber noch folgende Mankos:

- Die vorliegenden Datensätze sollten normalisiert (d.h. mittelwertbereinigt und durch die Standardabweichung geteilt) und trendbereinigt sein.
- Die Datensätze sollten am besten nur aus einem gemeinsamen Bereich bestehen, sonst kann die richtige Zuordnung u.U. nicht gefunden werden (siehe Abbildung 3.5).

In für uns interessanten Anwendungsszenarien kann jedoch nicht gewährleistet werden, dass die obigen Anforderungen zutreffen.

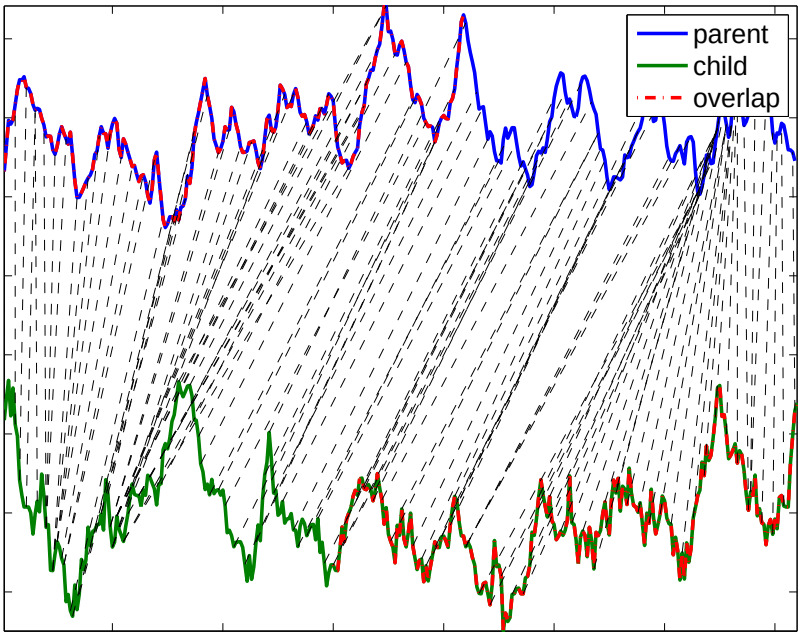

Abbildung 3.5: Ein Beispiel dafür, dass DTW falsche Ergebnisse liefert, falls die "Randbereiche" nicht mehr vernachlässigbar sind und die beiden Zeitreihen nicht nur aus dem überlappenden Bereich bestehen. Der durch dieses Alignment gelieferte Abstand kann keine Aussagekraft darüber haben, ob die beiden Zeitreihen zumindest teilweise zueinander passen.

Die DTW-Methode wurde auch in Lipowsky u. a. [47] untersucht. Man stellte fest, dass das im vorherigen Abschnitt vorgestellte Minimieren der $\mathcal{L}^2$-Norm für die uns interessierenden Dickenprofile zuverlässigere Ergebnisse liefert. Vor allem falls die Zeitreihen durch Ausreißer bzw. Messausfälle verunreinigt sind.

3.4 SAX

Eine weitere Idee besteht darin, die Zeitreihen zu diskretisieren und sie in Zeichenketten (*Strings*) umzuwandeln, um anschließend die Fülle von Algorithmen zur Suche in den Zeichenketten zu verwenden. So sind z.B. mit Boyer-Moor (Boyer und Moore [10], Cole u. a. [18]) Algorithmen bekannt, die eine genaue Stringsuche in linearer Zeit ($O(n + m)$) erledigen. Möchte man jedoch Fehler zulassen, so ist mit einer quadratischen Laufzeit ($O(n \cdot m)$) zu rechnen, wie z.B. beim Smith-Waterman-Algorithmus (Smith und Waterman [60]).

Die wahrscheinlich bekannteste Vorgehensweise ist *Symbolic Aggregate approXimation* (SAX, Lin u. a. [46]). Dabei wird wie folgt vorgegangen (siehe

Abbildung 3.6 für ein Beispiel):

1. Die Zeitreihe wird normalisiert, d.h. mittelwertbereinigt und normiert.

2. Es wird die Mächtigkeit des Alphabets gewählt. Je kleiner das Alphabet, desto resistenter ist die Darstellung gegenüber Messfehlern. Dafür bezahlt man damit, dass manche Zeitreihen nicht mehr von einander unterschieden werden können. Es ist durchaus denkbar, nur zwei Buchstaben im Alphabet zuzulassen und die Zeitreihe als einen binären String zu kodieren.

3. Man versucht die Grenzen der Quantisierung so zu wählen, dass jeder Buchstabe gleich oft vorkommt. Dies kann auch mit einer statischen Aufteilung der Quantisierungsgrenzen erreicht werden: Die normalisierten Werte sind in erster Näherung normalverteilt, deswegen reicht es, die Grenzen so zu wählen, dass die Wahrscheinlichkeiten für die Bins gleich sind. Für nur zwei Buchstaben liegt die Aufteilung auf der Hand: kleiner und größer 0.

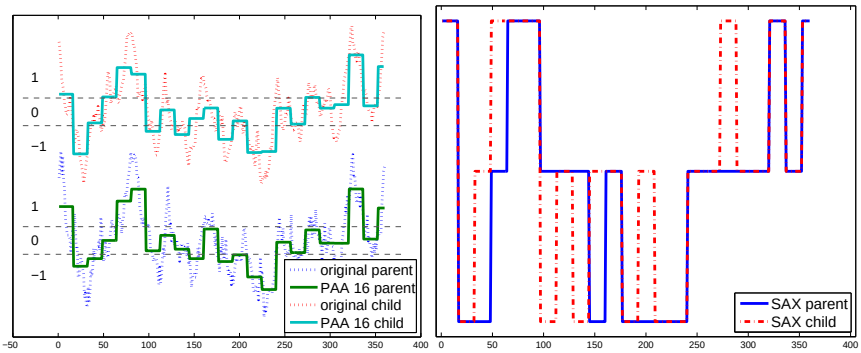

Abbildung 3.6: Im linken Bild: Die uns schon bekannten (Abschnitt 3.2) PAA 16 *der beiden Zeitreihen werden in SAX-Strings überführt. Im rechten Bild: Die jeweilige Repräsentation als SAX-String über dem Alphabet* $\{-1, 0, 1\}$*, wobei* 16 *der* 23 *Buchstaben übereinstimmen.*

Ein großer Vorteil dieser Methode ist die Reduktion der Dimensionen und des Speicherbedarfs und die damit verbundene gestiegene Geschwindigkeit der Suche. Zu den Nachteilen gehört die Notwendigkeit der Normalisierung der Zeitreihen, wobei auch Trends, Messausfälle und Ausreißer im Laufe einer Vorverarbeitung korrigiert werden müssen, was nicht immer robust und ohne größeren Aufwand zu erledigen ist.

3.5 Weiterentwicklung von Shotgun

Die Idee des Shotgun-Algorithmus stammt aus dem Feld der Bioinformatik. Dabei wird das Alignment kleiner Schnipsel dazu genutzt, DNS-Strukturen zu rekonstruieren (siehe Venter u. a. [61]). Der originale Shotgun-Algorithmus arbeitet mit Schnipseln, die aus diskreten Werten (A, C, G und T) bestehen.

Lipowsky u. a. [47] entwickelten diese Idee weiter, um Vergleich und Alignment von kontinuierlichen Zeitreihen durchzuführen. Dieser Algorithmus sieht wie folgt aus:

1. Die Zeitreihe c wird in kleine Schnipsel zerschnitten. Diese werden möglichst passend in der Zeitreihe p platziert. Dabei muss eventuell eine lokale Mittelwertbereinigung und Normierung durchgeführt werden.

2. Für jeden Schnipsel i existiert damit eine Koordinaten-Zuordnung

   $$x_c^i \mapsto x_p^i.$$

   Aus der Menge dieser Zuordnungen lässt sich nun die Koordinatentransformation $T_{(\alpha,\beta)}$ bestimmen.

3. Bei einem "perfekten" Lauf wären alle Schnipsel richtig zugeordnet und würden auf der Geraden

   $$x_p = \alpha x_c + \beta$$

   liegen. Normalerweise ist dies jedoch nicht der Fall: Bei jeder Zuordnung kann es sowohl zu kleinen Fehlern/Abweichungen von der Transformationsvorschrift als auch zu großen Ausreißern kommen.

   Um die richtigen Koordinatenzuordnungen $x_c^i \mapsto x_p^i$ zu finden bzw. die Ausreißer zu verwerfen, kann man das RANSAC-Verfahren oder die Hough-Transformation (siehe Duda und Hart [25] oder Abschnitt 5.6.1) anwenden.

4. Die Anzahl der "richtigen" (wie im vorletzten Schritt bestimmt) Koordinaten-Zuordnungen ist letztendlich das Maß dafür, dass zwei Messungen zusammengehören.

Die Vorteile dieser Vorgehensweise sind unter anderem:

- Der Einfluss der Ausreißer/Messausfälle bleibt lokal beschränkt.
- Kleine Skalierungen der x-Achse können für die kurzen Schnipsel vernachlässigt werden.

- Probleme mit Trends, Offsets und Skalierung der Werte können durch (lokale) Normalisierung entschärft werden.

Der Shotgun-Algorithmus hat noch einen weiteren "versteckten" Vorteil: Mit seiner Hilfe kann man nicht nur das Problem `A` lösen, sondern er liefert auch einen Ansatz, um das Problem `B` zu bearbeiten, wie es im nächsten Abschnitt gezeigt wird.

3.5.1 Stochastische Sicht auf Shotgun

In diesem Abschnitt betrachten wir das Problem B und unsere Hypothese ist, dass die in Frage kommenden Messungen nichts miteinander zu tun haben. Diese Annahme führt zu folgender Überlegung:

Während die Koordinaten x_c^i durch die Wahl der Schnipsel fest vorgegeben sind, sind die n zugeordneten x_p^i-Koordinaten uniform und voneinander unabhängig verteilt (i.i.d. bzw. *independent and identically distributed*):

$$x_p^i \stackrel{d}{=} U([0,1]) \quad x_p^i,\ i.i.d. \text{ für } i = 1 \ldots n.$$

Einfachheitshalber skalieren wir den Zahlenbereich für x_p^i auf $[0,1]$.

Mit Hilfe des Kolmogorow-Smirnow-Tests (für theoretische Grundlagen siehe Feller [26]) kann man überprüfen, ob die Beobachtungen $(x^1, \ldots, x^n)$ die Annahme $x^i = F_0(x)$ *i.i.d.* für eine angenommene Verteilung F_0 bestätigen oder diese unplausibel/unwahrscheinlich erscheinen lassen.

Dafür erstellt man die empirische Verteilungsfunktion

$$S(x) := \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}_{[x^i, \infty)}(x)$$

und berechnet anschließend

$$d_{max} := \max \{|S(x_i) - F_0(x_i)|\} \cup \{|S(x_{i-1}) - F_0(x_i)|\} \quad i = 1, \ldots, n$$

mit $x_0 := -\infty$. Übersteigt d_{max} einen Wert d_α, der vom Signifikanzniveau α und der Anzahl der Beobachtungen n abhängt, so wird die Hypothese abgelehnt. d_α kann für große n näherungsweise via

$$d_\alpha := \frac{\sqrt{\ln(\frac{2}{\alpha})}}{\sqrt{2n}}$$

berechnet werden.

Das heißt allerdings nicht, dass, falls die Beobachtungen den Test bestehen, es sich um zwei unabhängige/unpassende Messungen handelt. Um sich davon

zu überzeugen, kann man zwei Messungen betrachten, die perfekt zueinander passen. Die Beobachtungen wären dann

$$(x_p^1, \ldots, x_p^n) = (0, \frac{1}{n-1}, \ldots, \frac{n-2}{n-1}, 1),$$

dabei haben wir die Reihenfolge der Schnipsel so gewählt, dass die Koordinaten x_p^i aufsteigend sortiert sind.

Die Verteilungsfunktion $F_0(x)$ für die Gleichverteilung

$$F_0(x) = x \cdot \mathbb{I}_{[0,1]}(x) + \mathbb{I}_{[1,\infty)}$$

wird sehr gut mit der empirischen Verteilungsfunktion $S(x)$ angenähert. Es resultiert in $d_{max} = \frac{1}{n}$ und somit wird die Hypothese auch für kleine α's angenommen, solange es viele Beobachtungen gibt und n groß genug ist.

Es bietet sich jedoch an, $\Delta^i := x_p^{i+1} - x_p^i$ zu betrachten. Sind x_p^i wieder uniform nach $U([0,1])$ verteilt, wie oben angenommen, so gilt für die Verteilungsdichte bzw. Verteilungsfunktion von Δ^i

$$\begin{aligned} f(\Delta) &= \int_{\mathbb{R}} \mathbb{I}_{[0,1]}(x) \cdot \mathbb{I}_{[0,1]}(x+\Delta) dx \\ &= (1 - |\Delta|)\mathbb{I}_{[-1,1]} \end{aligned}$$

$$F_0(\Delta) = \begin{cases} 0 & \Delta < -1 \\ \frac{1}{2}(1+\Delta)^2 & -1 \leq \Delta < 0 \\ 1 - \frac{1}{2}(1-\Delta)^2 & 0 \leq \Delta < 1 \\ 1 & \Delta \geq 1. \end{cases}$$

Man muss jedoch beachten, dass Δ^i und Δ^{i+1} keine unabhängigen Variablen mehr sind und man deswegen den Kolmogorow-Smirnow-Test für $\Delta^1, \ldots, \Delta^{n-1}$ nicht anwenden kann. Die Abhängigkeit von Δ^i und Δ^{i+1} liegt auf der Hand. Es reicht dazu z.B. die Verteilungsdichte für Δ^{i+1} unter der Bedingung $\{\Delta^i = -1\}$ zu betrachten. Diese Bedingung bedeutet, dass $x_p^{i+1} = 0$ und $x_p^i = 1$ gelten (die einzige Möglichkeit den Abstand $\Delta^i = -1$ zu erreichen) und damit

$$\Delta^{i+1} = x_p^{i+2} - x_p^{i+1} = x_p^{i+2} \stackrel{d}{=} U([0,1]) \stackrel{d}{\neq} F_0(\Delta).$$

Das Problem lässt sich lösen, indem man jede zweite Differenz in die Beobachtungsreihe aufnimmt, d.h. die Beobachtungen $(\Delta^1, \Delta^3, \ldots, \Delta^{n-1})$ sind i.i.d.

Für unser obiges Beispiel mit zwei perfekt passenden Messungen hätten wir die "langweilige Beobachtungsreihe" $(\frac{1}{n-1}, \frac{1}{n-1}, \ldots, \frac{1}{n-1})$ und damit die empirische Verteilungsfunktion

$$S(x) = \mathbb{I}_{[\frac{1}{n-1}, \infty)}(x).$$

Daraus ergibt sich für d_{max}:

$$d_{max} = |F_0(\frac{1}{n-1}) - S(-\infty)| = 1 - \frac{1}{2}(1 - \frac{1}{n-1})^2 > \frac{1}{2},$$

was zur Ablehnung der Hypothese führt.

Es sind auch andere Tests denkbar. Man kann sich z.B. fragen, wie wahrscheinlich es ist, dass für zwei unpassende Messungen k von n Koordinatenzuordnungen für dieselbe Koordinatentransformation "abstimmen".

Eine analytische Berechnung erscheint sehr langwierig, falls überhaupt möglich. Deswegen präsentieren wir hier Ergebnisse einer Simulation.

Bei der Simulation werden n Zufallsvariablen $y_i \stackrel{d}{=} U([0,1])$ generiert. Danach wird der RANSAC-Algorithmus zum vorgegebenen Parameter δ für Punkte

$$\left(\frac{i-1}{n-1}, y_i\right) \in [0,1] \times [0,1] \quad i = 1, \ldots, n$$

durchgeführt. Der RANSAC-Algorithmus bestimmt eine Gerade, für die die Anzahl der Punkte in der δ-Umgebung am größten ist.

Die in der Abbildung 3.7 vorgestellte Ergebnisse wurden anhand von $m = 10^5$ Läufen bestimmt.

Interessant ist die Frage, wie viele Läufe notwendig sind , um zu gewährleisten, dass ein Ereignis mit Auftrittswahrscheinlichkeit p mit 99% Wahrscheinlichkeit mindestens einmal bei m Versuchen auftritt. Dies führt zur Gleichung

$$\begin{aligned} (1-p)^m < \alpha = 1 - 0.99 \Rightarrow \\ m > \frac{\log \alpha}{\log(1-p)} \quad \text{oder} \\ p > 1 - \exp\left(\frac{\log \alpha}{m}\right) \approx -\frac{\log \alpha}{m}. \end{aligned} \tag{3.1}$$

Die letzte Aussage ist deswegen interessant, weil sie eine Abschätzung für die Wahrscheinlichkeit der Ereignisse ermöglicht, die bei der Simulation nicht aufgetreten sind. So würde bei $m = 10^5$ ein Ereignis mit $p > 4.6 \cdot 10^{-5}$ mit Wahrscheinlichkeit von 99% mindestens einmal auftauchen.

Für die in der Abbildung 3.7 dargestellten Ergebnisse wurden für jeden Durchlauf zufällig $y^i \stackrel{d}{=} U([0,1])$ $i = 1, \ldots, 100$ generiert und anschließend der RANSAC-Algorithmus für die Punkte

$$\left(0, y^1\right), \left(\frac{1}{99}, y^2\right), \ldots, \left(\frac{98}{99}, y^{n-1}\right), \left(1, y^n\right)$$

und für Parameter $\delta = 0.01$ bzw. $\delta = 0.02$ angewendet.

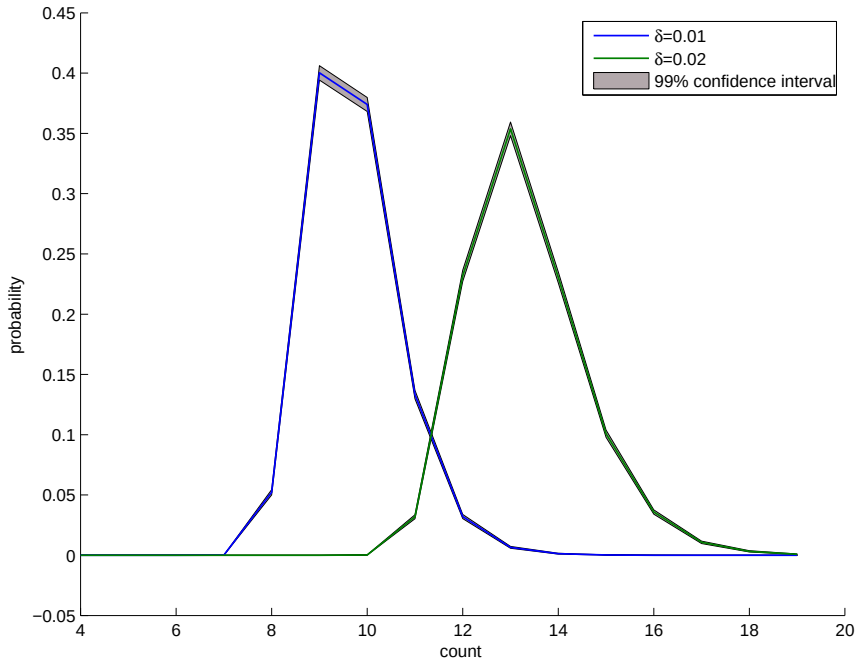

Abbildung 3.7: Diese Abbildung zeigt die Wahrscheinlichkeit, dass sich count von 100 *zufälligen Koordinatenzuordnungen beim Durchlauf des RANSAC-Algorithmus in Abstand von weniger als* δ *von der besten Geraden befinden. Für* $\delta = 0.01$ *war* 18 *der maximale aufgetretene count-Wert, für* $\delta = 0.02$ *betrug dieser Wert* 21*.*

An dieser Stelle möchten wir direkt einen etwas allgemeineren Fall abhandeln: Die x-Koordinaten werden nun nicht mehr fest vorgeschrieben, sondern werden auch zufällig generiert. Dies würde passieren, wenn man die Zeitreihen nicht in die Schnipsel zerschneidet, sondern die Lage der Schnipsel zufällig wählen würde. Anhand der Simulationen stellt man fest, dass es zwar gewisse Unterschiede gibt, doch die Verteilungen qualitativ ähnliches Verhalten aufweisen, wie hier in der Abbildung 3.8 anhand der Fallstudie $\delta = 0.01$ und $n = 100$ illustriert wird:

Diese Simulationen zeigen einen großen Vorteil des Shotgun-Algorithmus auf, der beim folgenden Anwendungsbeispiel besonders deutlich wird:

Angenommen, wir suchen ein Band in einer Datenbank mit 1000 Bändern. Beim richtigen Paar gelingt es uns, bei $\delta = 0.01$ nur 20 von 100 Schnipseln richtig zuzuordnen. Wie man bei der obigen Simulation gesehen hat, liegt die Wahrscheinlichkeit, dass man bei einem nicht zusammenpassenden Band zufällig auf nicht weniger als 20 “Treffer” kommt, unter $5 \cdot 10^{-3}\%$. Also besteht immer noch 99.995%-Chance, dass das richtige Paar Top 1 der Abfrage sein wird. Und das obwohl ca. 80% der Schnipsel bzw. Messwerte nicht brauchbar waren!

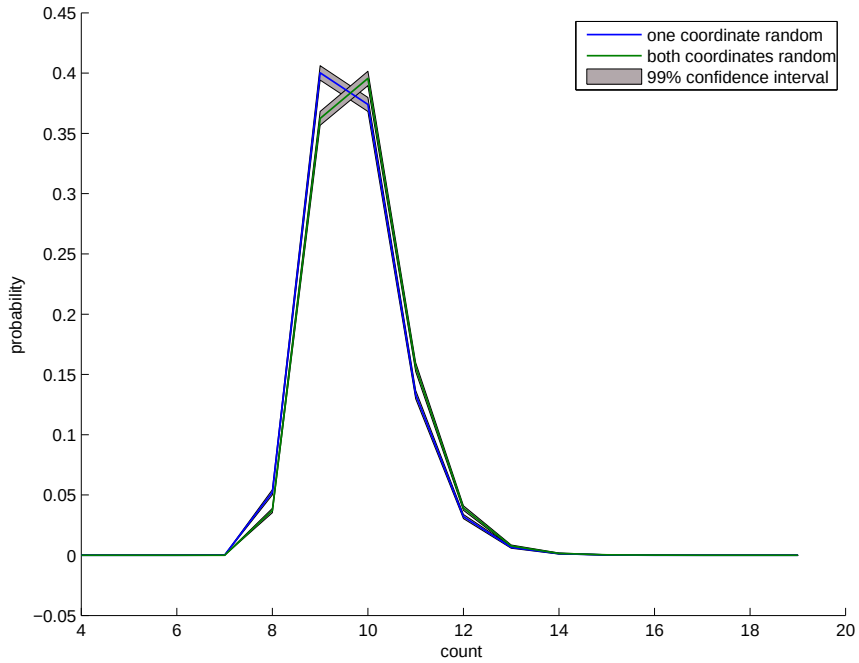

Abbildung 3.8: Unsere Experimente ergaben qualitativ ähnliche Verteilungsdichten, unabhängig davon ob die x-Koordinate zufällig gewählt wurde oder nicht.

Fazit: Nur anhand des Vergleiches zweier Bänder kann man mit Hilfe des Shotgun-Algorithmus gut abschätzen, ob diese zueinander gehören, ohne weitere Informationen über Fehler der Messung zur Verfügung zu haben. Dagegen hängen die von globalen Deskriptoren gelieferten Werte/Abstände von den Messfehlern ab. Ohne die Verteilung dieser Fehler zu kennen, lässt sich keine Aussage über die Zugehörigkeit dieser Bänder treffen.

3.6 Shazam-Algorithmus

Bei Shazam[1] können kurze, wenige Sekunden lange Musikstücke eingesandt werden, für die nach einer Analyse der Name des Songs/des Interpreten ermittelt werden.

Der Algorithmus, auf dem der Online-Dienst Shazam basiert, hat viele Ähnlichkeiten mit dem zuvor beschriebenen Shotgun. Der Hauptunterschied besteht darin, dass die Schnipsel nicht willkürlich gewählt werden, sondern dass man nur besonders interessante (und stabile) Abschnitte betrachtet. Diese Abschnitte werden bei einem Vorverarbeitungsschritt gefunden und werden als *Features* bezeichnet. Alle Features bilden den sogenannten *Fingerprint*

[1] `http://www.shazam.com/`

einer Zeitreihe.

Nach dem Vergleich der Features bekommt man die Koordinatenzuordnungen $x_p^i \mapsto x_c^i$ und kann damit identisch zum Shotgun-Algorithmus verfahren.

Eine interessante Frage ist: Wie werden diese Features bestimmt? In Wang [63] wird die folgende Vorgehensweise beschrieben:

- Im ersten Schritt wird das Spektrogramm

$$S[f](t,\omega) = \left| \int_{t-\frac{\Delta}{2}}^{t+\frac{\Delta}{2}} f(x) e^{-2\pi i x \omega}\, dx \right|$$

  des Musikstücks $f : \mathbb{R} \to \mathbb{R}$ mit einem passenden Fenster Δ erstellt.

- Als Nächstes werden die höchsten Peaks im Spektrogramm bestimmt - diese haben sich als besonders robust und spezifisch herausgestellt. Zwei solche Peaks im Zeit-Frequenz-Raum werden zu einem Feature zusammengefasst. Ein Feature besteht aus dem *Frame*, das die Information zur Lage des Features beinhaltet, und dem *Deskriptor*, der die Zuordnung der Features zueinander erleichtert. Im Falle des Shazam-Algorithmus besteht das Frame nur aus dem Zeitstempel des ersten (auf der Zeitachse früheren) Peaks (von Wang als *anchor point* bezeichnet), während der Deskriptor durch die Frequenz der Peaks und die Zeitdifferenz dazwischen gegeben ist.

- Die Features werden anhand ihrer Deskriptoren gehasht und können daher schnell verglichen werden.

Dieser Algorithmus scheint für Musik bestens geeignet zu sein. Einer der Vorteile ist, dass er gegen lokale Störungen wie Hintergrundgeräusche immun zu sein scheint. Auch die Fähigkeit, die Features schnell vergleichen zu können, wofür man nur eine handvoll Operationen braucht, ist beeindruckend. Diese hohe Geschwindigkeit ist der große Vorteil des Shazam-Algorithmus im Vergleich zum Shotgun.

Der Hauptnachteil dieser Technik besteht darin, dass eine mögliche Skalierung der Zeitachse nicht berücksichtigt wird - diese Schwäche wurde mehrfach auf youtube benutzt, indem die Benutzer die Zeitachse leicht manipulierten und damit die automatische Detektion geschützter Inhalte austricksten. Außerdem gibt es noch folgende Probleme:

1. Die Peaks des Spektrogramms als Features sind besonders für Musik gut geeignet: Bei den Musikstücken wird die Energie von bestimmten Frequenzen getragen. In anderen Arten von Zeitreihen muss dies nicht der Fall sein, so dass die Peaks nicht robust bzw. nicht aussagekräftig genug sind.

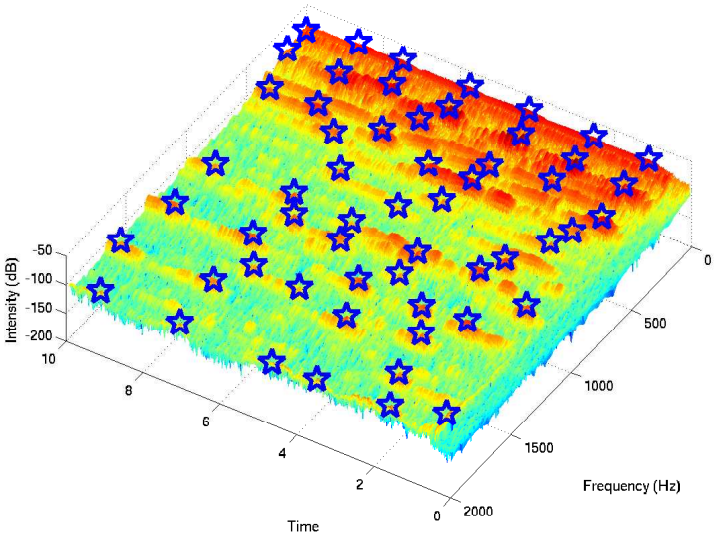

Abbildung 3.9: Das Spektrogramm der ersten 10 *Sekunden von Led Zeppelins "Stairway to Heaven", wobei die Sterne die höchsten Peaks (*anchor points*) der Umgebung markieren.*

2. Man braucht mindestens 5-sekundige Aufnahmen, um verlässliche Ergebnisse zu erhalten. Bei der Sampling-Rate einer CD von 44.100 Hz wären dies ca. 220.000 Werte - nicht jede Zeitreihe ist so lang!

3.7 Scale-Space Filtering

Ein Spektrogramm scheint nicht für alle Zeitreihen gut geeignet zu sein. Im Scale-Space-Filtering, vorgestellt in Witkin [64], wird die Frequenz-Achse durch die Skalierungsachse ersetzt (was im Groben als $\frac{1}{\text{Frequenz}}$ verstanden werden kann). Dabei wird die Scale-Space-Funktion als Faltung der Zeitreihe mit Gaußkerneln $G(x, \sigma)$ definiert (siehe Abbildung 3.10, mehr über Faltung und Gaußfunktion $G(x, \sigma)$ kann im Anhang 10.1 bzw. 10.3 gefunden werden):

$$F(x, \sigma) := F[f](x, \sigma) := f(x) * G(x, \sigma).$$

Witkin konzentrierte sich bei seinen Untersuchungen auf F_{xx} und zeigte, dass der Gaußkernel der einzige ist, für den gilt:

- "Gutartigkeit": symmetrisch, strickt fallend zu den Flanken. Für $\sigma \to 0$ geht die Faltung $f(x) * G(x, \sigma)$ gegen f und für $s \to \infty$ gegen den Mittelwert der Zeitreihe.

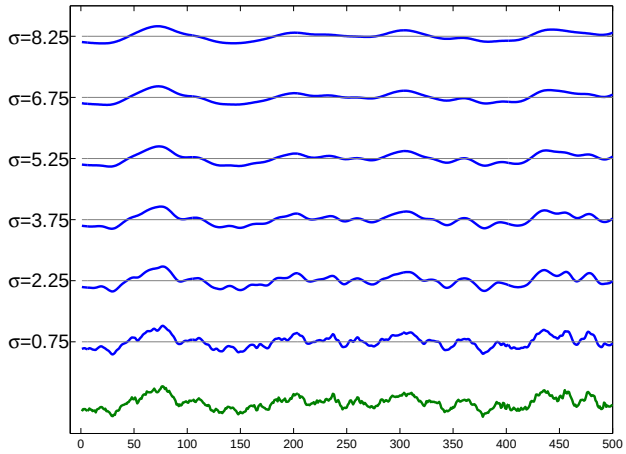

Abbildung 3.10: Die Zeitreihe f (grün) gefaltet mit unterschiedlichen Gaußkerneln. Die Faltungen nähern sich mit wachsendem σ einer Konstanten. Für kleine Werte von σ ähneln sich die Faltungen der Originalzeitreihe f.

- Erhaltung der Nullstellen und die daraus folgende Aussage:

$$\left|\left\{\frac{d^k}{dx^k}F(x,\,\sigma)\right\}\right| \geq \left|\left\{\frac{d^k}{dx^k}F(x,\,\sigma')\right\}\right| \Leftrightarrow \sigma \leq \sigma' \quad \forall k \geq 0.$$

  Die Nullstellen verschwinden nicht mit dem fallenden σ, sondern "wandern" und bleiben erhalten (siehe Abbildung 3.11).

Witkin versuchte die stabilsten Nullstellen von $F_{xx}(x,\,\sigma)$ zu bestimmen und benutzt diese, um eine vereinfachte Beschreibung der Zeitreihen zu erzeugen.

Um bei der "Sprache" des Shazam-Algorithmus zu bleiben, können die ausgezeichneten Punkte - die Ursprünge der Nullstellen als *Feature-Punkte* aufgefasst und zu einem Fingerprint der Reihe zusammengefasst werden.

Im weiteren Verlauf der Arbeit werden wir robustere Feature-Punkte kennenlernen bzw. entwickeln, doch sie alle werden ihre Koordinatentransformation-Invarianz mit diesen "einfachen" Witkin-Feature-Punkten gemein haben.

Wie verhalten sich also die Nullstellen von $F_{xx}(x,\,\sigma)$ unter Koordinaten-

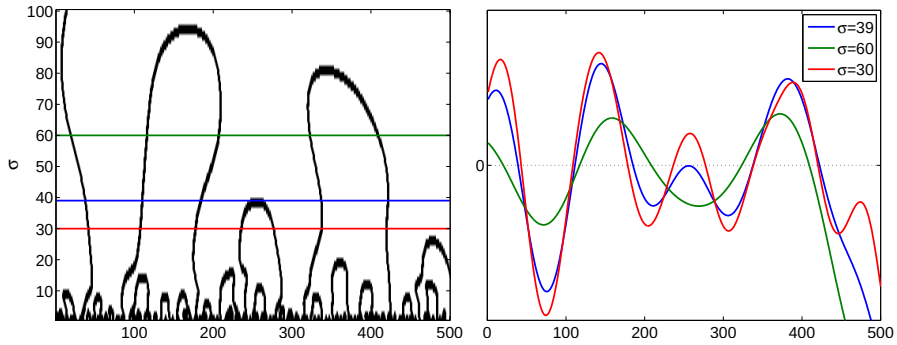

Abbildung 3.11: Links: Die Konturen von $F_{xx}(x,\sigma) = 0$. Man kann beobachten, dass die Nullstellen nicht verschwinden, sondern, einmal entstanden (z.B. für $x \approx 250$, $\sigma \approx 40$), wandern sie nach unten zu den kleineren σ's. Rechts: Schnitte durch $F_{xx}(x,\sigma)$ für $\sigma = 30, 39$ und 60. Besonders interessant ist das Verhalten in der Nähe von $x \approx 250$. Für $\sigma = 60$ gibt es keine Nullstelle, für $\sigma = 39$ entsteht eine (doppelte) Nullstelle, die für $\sigma = 30$ in zwei einfache Nullstellen transformiert wird.

transformationen T_{1d}?

$$\begin{aligned}
F_{xx}[T_{(\alpha,\beta)}f](x,\sigma) &= f(T^{-1}_{(\alpha,\beta)}(x)) * G_{xx}(x,\sigma) \\
&= \int_{\mathbb{R}} f(T^{-1}_{(\alpha,\beta)}(y)) G_{xx}(x-y,\sigma)dy \\
&= |\alpha| \int_{\mathbb{R}} f(z) G_{xx}(x - T_{(\alpha,\beta)}z, \sigma)dz \\
&= \frac{|\alpha|}{|\alpha|^3} \int_{\mathbb{R}} f(z) G_{xx}(\frac{1}{\alpha}(x - T_{(\alpha,\beta)}z), \frac{1}{|\alpha|}\sigma)dz \\
&= \frac{1}{|\alpha|^2} \int_{\mathbb{R}} f(z) G_{xx}((\frac{x}{\alpha} - \frac{\beta}{\alpha}) - z), \frac{1}{|\alpha|}\sigma)dz \\
&= \frac{1}{|\alpha|^2} F_{xx}[f](T^{-1}_{(\alpha,\beta)}(x), \frac{\sigma}{|\alpha|}).
\end{aligned}$$

Also

$$F_{xx}[Tf](x,\sigma) = \frac{1}{|\alpha|^2} F_{xx}[f](T^{-1}(x), \frac{\sigma}{|\alpha|}) \tag{3.2}$$

bzw.

$$F_{xx}[Tf](x,\sigma) = 0 \Leftrightarrow F_{xx}(T^{-1}x, \frac{\sigma}{|\alpha|}) = 0$$

und damit wären die Ursprünge der Nullstellen der zweiten Ableitung auch unter Koordinatentransformationen erhalten. Das heißt, falls (x_0, σ_0) ein Nullstellenursprung von $F(x,\sigma)$ ist, so ist $(T^{-1}_{(\alpha,\beta)}(x_0), \frac{\sigma_0}{|\alpha|})$ auch ein Nullstellenursprung von $F[T_{(\alpha,\beta)}f](x,\sigma)$ (Abbildung 3.12).

Im nächsten Abschnitt lernen wir weitere Möglichkeiten kennen, Feature-Punkte in einer Zeitreihe zu detektieren.

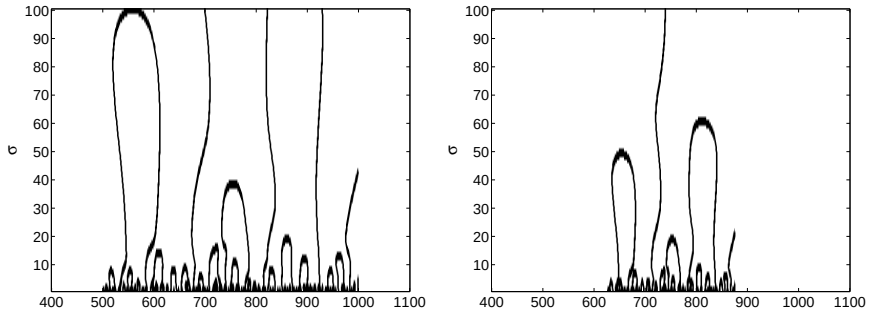

Abbildung 3.12: Das linke Bild zeigt die Konturen von $F_{xx}(x,\sigma)=0$, das rechte die Konturen von $F_{xx}[T_{(\frac{1}{2},\beta)}f](x,\sigma)=0$. Man kann sehen, dass nicht nur die x-Achse um den Faktor 2 gestaucht wird, sondern auch die σ-Achse.

3.8 LoG und DoG - Filter

Ein Vergleich der Bilder anhand der während einer Vorverarbeitungsphase gefundenen Features ist in der Bildverarbeitung weit verbreitet. *Corner Detektor* (Harris und Stephens [35]), *Canny-Detektor* (Canny [15]), LoG, DoG- und DoH-Filter - um nur die bekanntesten Feature-Detektoren zu nennen. In den letzten Jahren wurden auch erste interessante Versuche unternommen, diese im eindimensionalen Fall einzusetzen (z.B. Xie und Beigi [65]).

Die Detektion der Features basierend auf dem *Laplacian-of-Gaussian*-Filter (LoG-Filter) ist ein zum Scale-Space-Filtering sehr verwandtes Verfahren. Hier stellen wir die eindimensionale Version des Verfahrens vor.

Statt Scale-Space-Funktion betrachtet man die Faltungen mit der skalennormierten zweiten Ableitung der Gaußfunktion

$$L[f](x,\sigma) := \left[f(\cdot) * \sigma^2 G_{xx}(\cdot,\sigma)\right](x) = \frac{\partial^2}{\partial^2 x}\sigma^2 F(x,\sigma).$$

Die Skalennormierung dient vor allem der Invarianz unter Koordinatentransformationen. Wie in der Gleichung 3.2 schon gesehen:

$$\begin{aligned} L[Tf](x,\sigma) &= \sigma^2 F_{xx}[Tf])(x,\sigma) \\ &= \frac{\sigma^2}{|\alpha|^2} F_{xx}[f](T^{-1}x, \frac{\sigma}{|\alpha|}) \\ &= L[f](T^{-1}x, \frac{\sigma}{|\alpha|}). \end{aligned} \tag{3.3}$$

Natürlich gilt auch

$$\int_{\mathbb{R}} |\sigma^2 G_{xx}(x,\sigma)| = const \quad \forall \sigma > 0.$$

Nun sucht man lokale Maxima bzw. Minima der Funktion $L(x, \sigma)$ und betrachtet diese als mögliche Feature-Punkte.

Was ist hier der Hauptunterschied zum naiven Wählen der lokalen Minima und Maxima direkt in der Zeitreihe? Offensichtlich wird auch die "intristische Skalierung" der Punkte mitbestimmt, anhand der man die passend große Umgebung zur Erstellung der Deskriptoren wählen kann. Und zwar so, dass diese Umgebung und damit auch die Deskriptoren invariant unter den Koordinatentransformationen $T(\alpha, \beta)$ und insbesondere skalierungsinvariant sind:

Sei (x^*, σ^*) ein Feature-Punkt von f, dann nennen wir die Funktion

$$u[f](\cdot; x^*, \sigma^*) := T^{-1}_{(\sigma^*, x^*)} \left[G(\cdot, \sigma^*) * f \right]$$

die Umgebung des Feature-Punkts.

Für $T_{(\alpha,\beta)} f$ lauten die Koordinaten des Feature-Punkts $\left(T_{(\alpha,\beta)} x^*, |\alpha| \cdot \sigma^* \right)$ und die Umgebung:

$$u[Tf](\cdot; T_{(\alpha,\beta)} x, |\alpha| \cdot \sigma^*) = T^{-1}_{(\sigma^*|\alpha|, T_{(\alpha,\beta)} x^*)} \left[G(\cdot, \sigma^*|\alpha|) * T_{(\alpha,\beta)} f \right].$$

Zuerst zwei Nebenrechnungen:

$$\begin{aligned}
\left[G(\cdot, \sigma^*|\alpha|) * T_{(\alpha,\beta)} f \right](x) &= \int_{\mathbb{R}} \frac{1}{\sqrt{2\pi}\sigma^*|\alpha|} G(x - y, |\alpha|\sigma^*) \cdot f(T^{-1}_{(\alpha,\beta)} y)\, dy \\
&\overset{y=Tz}{=} \int_{\mathbb{R}} \frac{1}{\sqrt{2\pi}\sigma^*|\alpha|} G(x - \alpha z - \beta, |\alpha|\sigma^*) \cdot f(z)\, |\alpha| dz \\
&= \int_{\mathbb{R}} \frac{1}{\sqrt{2\pi}\sigma^*} e^{\frac{-\alpha^2 (\frac{x-\beta}{\alpha} - z)^2}{2\sigma^{*2}|\alpha|^2}} \cdot f(z)\, dz \\
&= \left[G(\cdot, \sigma^*) * f \right] (T^{-1}_{(\alpha,\beta)} x) \\
&= T_{(\alpha,\beta)} \left[G(\cdot, \sigma^*) * f \right] (x)
\end{aligned}$$

und

$$\begin{aligned}
T^{-1}_{(\alpha,\beta)} T_{(\sigma^*|\alpha|, T_{(\alpha,\beta)} x^*)} x &= T^{-1}_{(\alpha,\beta)} (\sigma^*|\alpha| x + T_{(\alpha,\beta)} x^*) \\
&= T^{-1}_{(\alpha,\beta)} (\sigma^*|\alpha| x + \alpha x^* + \beta) \\
&= \frac{\sigma^*|\alpha| x + \alpha x^* + \beta}{\alpha} - \frac{\beta}{\alpha} \\
&= sign(\alpha)\sigma^* x + x^* = T_{(sign(\alpha)\sigma^*, x^*)} x.
\end{aligned}$$

Damit gilt:

$$u[Tf](\cdot; T_{(\alpha,\beta)} x, |\alpha| \cdot \sigma^*) = T^{-1}_{(sign(\alpha)\sigma^*, x^*)} \left[G(\cdot, \sigma^*) * f \right](x).$$

bzw.

$$u[Tf](\cdot; T_{(\alpha,\beta)}x, |\alpha| \cdot \sigma^*) = T^{-1}_{(sign(\alpha)\sigma^*, x^*)} T_{\sigma^*, x^*)} u[f](\cdot; x^*, \sigma^*)$$

und für $sign(\alpha) = 1$ sogar

$$u[Tf](\cdot; T_{(\alpha,\beta)}x, |\alpha| \cdot \sigma^*) = u[f](\cdot; x^*, \sigma^*).$$

Dies ermöglichst es uns, die Umgebung $U \in \mathbb{R}^{2\cdot d+1}$ eines Extremums in (x^*, σ^*) via

$$U_i = u[f](i, x^*, \sigma^*) \quad -d \leq i \leq d$$

skalierungsinvariant abzutasten.

Doch welche Punkte werden durch den LoG-Filter gewählt?

Feature-Punkte des LoG-Filters

In diesem Abschnitt untersuchen wir das Verhalten des LoG-Filters angewandt auf eine Gaußfunktion. Zur Erinnerung: LoG einer Funktion wird berechnet als

$$LoG[f](x, \sigma) = \sigma^2 f(\cdot) * G_{xx}(\cdot, \sigma)(x).$$

Mit $f(x) = G(x, \sigma_0)$ können wir die Eigenschaften der Faltung (Abschnitte 10.1 und 10.3.2 im Anhang) benutzen, um auf das folgende Ergebnis zu kommen:

$$\begin{aligned} LoG[G(\cdot, \sigma_0)](x, \sigma) &= \sigma^2 (G(\cdot, \sigma_0) * G_{xx}(\cdot, \sigma)(x) \\ &= \sigma^2 \partial_x \partial_x (G(\cdot, \sigma_0) * G(\cdot, \sigma)(x) \\ &= \sigma^2 \partial_x \partial_x G(x, \sigma^*) \quad \sigma^* := \sqrt{\sigma_0^2 + \sigma^2} \\ &= \sigma^2 G_{xx}(x, \sigma^*). \end{aligned}$$

LoG sieht dann qualitativ aus, wie dargestellt in der Abbildung 3.13.

Wo befinden sich die Extrema dieser Funktion?

$$\begin{aligned} \partial_x (LoG[G(\cdot, \sigma_0)](x, \sigma)) &= \sigma^2 G_{xxx}(x, \sigma^*) \\ &= \frac{1}{\sqrt{2\pi}\sigma^*} e^{-\frac{x^2}{2\sigma^{*2}}} \left(\frac{3\sigma^{*2}x - x^3}{\sigma^{*6}} \right) \end{aligned}$$

In Frage für Extrema kommen $x = 0$ und $x = \pm\sqrt{3}\sigma^*$.

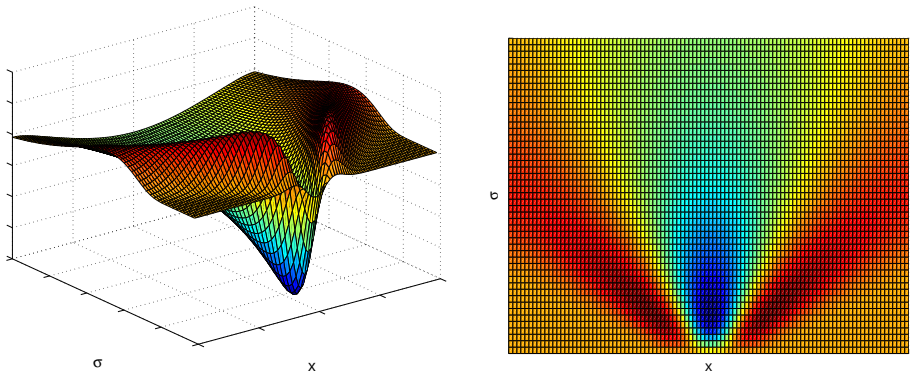

Abbildung 3.13: LoG einer Gaußfunktion.

$$\begin{aligned}\partial_\sigma LoG[G(\cdot,\sigma_0)](x,\sigma) &= 2\sigma G_{xx}(x,\sigma^*) + \partial_{\sigma^*} G_{xx}(x,\sigma^*)\frac{\partial\sigma^*}{\partial\sigma} \\ &= \frac{\sigma}{\sqrt{2\pi}\sigma^{*9}} e^{-\frac{x^2}{2\sigma^{*2}}} \cdot \\ &\quad \left(2x^2\sigma^{*4} - 2\sigma^{*6} + x^4\sigma^2 - 6x^2\sigma^{*2}\sigma^2 + 3\sigma^{*4}\sigma^2\right)\end{aligned}$$

- $x = 0$ führt zu

  $$\partial_\sigma [G(\cdot,\sigma_0)](x,\sigma) = \frac{\sigma}{\sqrt{2\pi}\sigma^{*5}}\left(-2\sigma^{*2} + 3\sigma^2\right) \stackrel{!}{=} 0$$

  und damit $\sigma = 0$ (uninteressant) und $\sigma = \sqrt{2}\sigma_0$.

- $x = \pm\sqrt{3}\sigma^*$ führt zu ($\sigma = 0$ uninteressant):

  $$\begin{aligned} 4\sigma^{*2} - 6\sigma^2 &\stackrel{!}{=} 0 \stackrel{\sigma^{*2}=\sigma_0^2+\sigma^2}{\Longrightarrow} \\ \sigma &= \sqrt{2}\sigma_0. \end{aligned}$$

Damit befinden sich die detektierten Extrema bei Koordinaten $(0, \sqrt{2}\sigma_0)$ und $(\pm 3\sigma_0, \sqrt{2}\sigma_0)$.

Von einem besonderen Interesse ist das "Hauptextremum" in $(0, \sqrt{2}\sigma_0)$. Für $x = 0$ sieht LoG entlang der σ-Achse wie folgt aus (Abbildung 3.14):

$$LoG[G(\cdot,\sigma_0)](0,\sigma) = -\frac{\sigma^2}{\sqrt{2\pi}\sigma^{*3}}.$$

Im Bereich des Minimums kann der Verlauf mit einer Parabel angenähert werden.

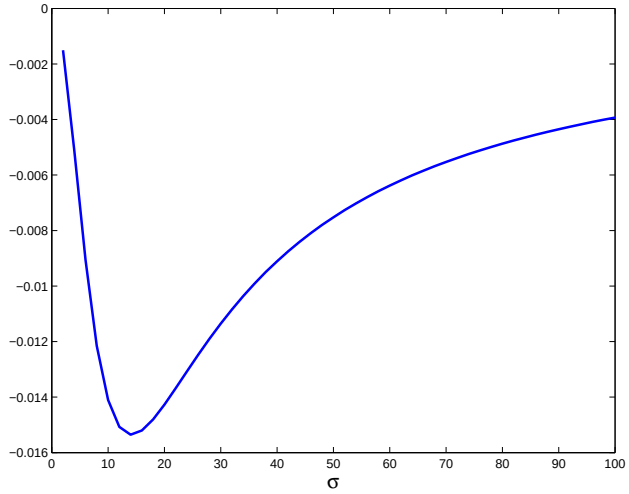

Abbildung 3.14: LoG einer Gaußfunktion für $x = 0$. Der Verlauf in der Umgebung des Minimums könnte mit einer Parabel approximiert werden.

Während sich die *LoG* der Gaußfunktion leicht analytisch berechnen lässt, ist es für andere Funktionen nicht mehr der Fall. Von Interesse kann auch das Verhalten einer Cosinus-Funktion sein. Betrachtet man z.B.

$$cosf(\omega x) := \begin{cases} 0 & \omega x < -\frac{\pi}{2} \\ 1 + \cos(\omega x) & -\frac{\pi}{2} \leq \omega x \leq \frac{\pi}{2} \\ 0 & \omega x > \frac{\pi}{2}, \end{cases}$$

so ist der Unterschied zur (skalierten) Gaußfunktion nicht all zu groß, den man numerisch berechnen kann (siehe Abbildung 3.15).

Zusammenfassend stellt man fest, dass ein Extremum einer Funktion f zu drei Extrema im *LoG* (und damit drei Feature-Punkten) führt, deren Lage auf der σ-Achse von der Breite des Funktionsextremums abhängt.

3.8.1 DoG-Filter

Natürlich ist es möglich, die Funktion f direkt mit $\sigma^2 G_{xx}$ zu falten. Für die Erstellung der Deskriptoren werden jedoch die Faltungen mit der Gaußfunktion $G(x, \sigma)$ gebraucht und deswegen kann man gewisse Synergieeffekte ausnutzen, indem man eine Näherung an den LoG-Filter, den sogenannten *Difference-of-Gaussian*-Filter (DoG-Filter bzw. DoG-Detektor), benutzt:

$$\begin{aligned} D[f](x, \sigma_1, \sigma_2) &:= \left[(G(\cdot, \sigma_2) - G(\cdot, \sigma_1)) * f(\cdot) \right](x) \\ &= \left(F(x, \sigma_2) - F(x, \sigma_1) \right). \end{aligned}$$

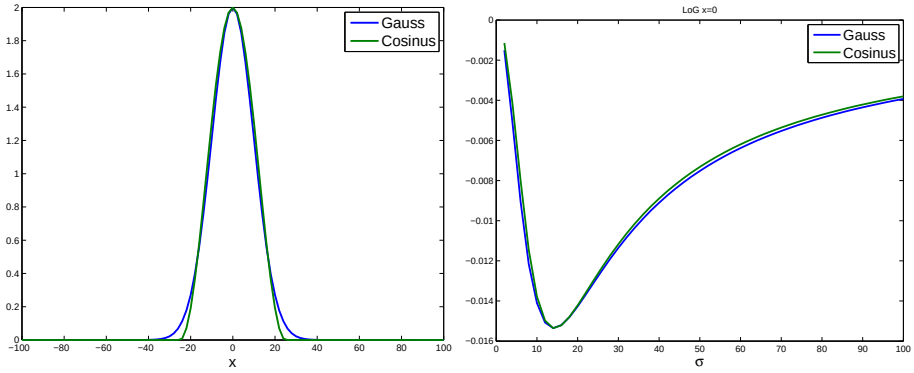

Abbildung 3.15: Vergleich zwischen einer Gaußfunktion und cosf.

Der Zusammenhang wird durch die folgende Betrachtung klar:

$$\begin{aligned} D[f](x,\sigma_0,\sigma_0+\Delta\sigma) &= F(x,\sigma_0+\Delta\sigma) - F(x,\sigma_0) \\ &\approx \frac{\partial F(x,\sigma_0)}{\partial\sigma}\Delta\sigma \\ &= [f(\cdot) * G_\sigma(\cdot,\sigma_0)](x)\Delta\sigma \\ &\overset{(10.1)}{=} [f(\cdot) * \sigma_0 G_{xx}(\cdot,\sigma_0)](x)\Delta\sigma \\ &= [f(\cdot) * \sigma_0^2 G_{xx}(\cdot,\sigma_0)](x)\frac{\Delta\sigma}{\sigma_0} \\ &= L[f](x,\sigma_0)\frac{\Delta\sigma}{\sigma_0}. \end{aligned}$$

Wählt man $\Delta\sigma := a\sigma_0$, so gilt die Näherung:

$$D[f](x,\sigma_0,(1+a)\sigma_0) \approx a \cdot L[f](x,\sigma_0).$$

3.9 Determinant-of-Hessian-Filter

In der Bildverarbeitung wird oft ein etwas anderer Filter angewandt - die skalierungsnormierte Determinante der Hessematrix der zweidimensionalen Gaußfunktion $G(x,y,\sigma) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$:

$$DoH(x,\sigma) := \sigma^2 det\begin{pmatrix} G_{xx}(x,\sigma) & G_{xy}(x,\sigma) \\ G_{yx}(x,\sigma) & G_{yy}(x,\sigma) \end{pmatrix}.$$

Es ist auch möglich, die Gewichtungen zu verändern, so wie es z.B. bei SURF-Detektoren der Fall ist:

$$DoH_\omega(x,\sigma) := \sigma^2\left(G_{xx}(x,\sigma)\cdot G_{yy}(x,\sigma) - \omega G_{xy}(x,\sigma)\cdot G_{yx}(x,\sigma)\right)$$

mit dem Gewicht $\omega \geq 0$.

Wie auch immer, im eindimensionalen Fall gibt es keinen Unterschied zwischen DoH und LoG, und deswegen ist dieser Ansatz nicht länger von Interesse für uns.

3.10 Box-Filters

Die Approximation des DoG-Filters durch die Box-Filters im zweidimensionalen Fall gelang durch die SURF-Features zur Prominenz in der Bildverarbeitung. Im eindimensional Pendant dazu könnte der LoG-Filter z.B. durch die Boxfunktion

$$DD(x,\Delta) := \begin{cases} 0 & x < -3\cdot\Delta \\ \frac{1}{\Delta} & -3\cdot\Delta \leq x \leq -\Delta \\ -\frac{2}{\Delta} & -\Delta < x < \Delta \\ \frac{1}{\Delta} & \Delta \leq x \leq 3\cdot\Delta \\ 0 & 3\cdot\Delta < x \end{cases} \approx L(x, \frac{5}{6}\Delta)$$

angenähert werden. Die Wahl von $\sigma = \frac{5}{6}\Delta$ wird im Abschnitt 10.3.3 im Anhang begründet. Diese Approximation reduziert die zum Auffinden der Feature-Punkte notwendige Zeit, weil Faltung mit einer Box in $O(n)$ berechnet werden kann, und damit unabhängig von der Breite der Box (für mehr Details siehe Abschnitt 5.1.1).

3.11 Wavelets

Mittlerweile gehören Wavelet-Transformationen zu Standardhandwerkzeugen bei der Scale-Space-Analyse. Von Interesse für uns ist vor allem die *continuous wavelet transform* (CWT), mit deren Hilfe man Funktionen ähnlich wie bei einer Fourier-Transformation in Wavelets zerlegen kann. Der Vorteil dieser Zerlegung besteht darin, dass man die Lokalität nicht gänzlich verliert, wie es bei einer Fourier-Transformation der Fall wäre.

Ein Standardwerk (vor allem für diskrete) Wavelets ist “Ten Lectures on wavelets” von Ingrid Daubechies (Daubechies [23]), wir folgen in unseren Ausführungen jedoch “Wavelets. An analysis Tool” von Matthias Holschneider (Holschneider [37]), weil seine (zu der von Daubieche äquivalente) Sichtweise sich leichter mit unserem bisherigen Framework vereinigen lässt.

Wir betrachten

$$\phi \in C^\infty(\mathbb{R}) \cap L^1(\mathbb{R}).$$

Des Weiteren nehmen wir an, dass $\phi(x) \xrightarrow{|x|\to\infty} 0$ schnell genug und $\phi(x) \geq 0$ $\forall x \in \mathbb{R}$.

O.B.d.A. $\int_{\mathbb{R}} \phi(x)dx = \|\phi\|_1 = 1$, sonst betrachte $\tilde{\phi} := \frac{\phi}{\|\phi\|_1}$.
Betrachten wir $\phi_a(x) = \frac{1}{a}\phi\left(\frac{x}{a}\right)$, so ist klar, dass

$$\lim_{k\to\infty} \phi_{\frac{1}{k}}(x) = \delta(x)$$

mit $\delta(x)$ - Dirac-Delta-Funktion bzw.

$$\|\phi_a * s - s\|_p \xrightarrow{a\to 0} 0$$

für $\forall s \in L^p(\mathbb{R})$ mit $1 \leq p < \infty$ gilt.
Die Funktion

$$g(x) := (x\partial_x + 1)\phi(x)$$

wird als *Wavelet* bezeichnet und besitzt folgende Eigenschaften:

- $g(x)$ ist mittelwertbereinigt

$$\begin{aligned}\int_{\mathbb{R}} g(x)dx &= \int_{\mathbb{R}} x\partial_x\phi(x)dx + \int_{\mathbb{R}} \phi(x)dx \\ &= -\int_{\mathbb{R}} \phi(x)dx + \int_{\mathbb{R}} \phi(x)dx = 0\end{aligned}$$

- ist $\phi(x)$ gerade, so ist auch $g(x)$ gerade:

$$g(-x) = -x\partial_x\phi(-x) + \phi(-x) = x\partial_x\phi(x) + \phi(x) = g(x).$$

  Damit gilt unter anderem für gerade $\phi(x)$:

$$\int_{\mathbb{R}} xg(x) = 0,$$

  da x eine ungerade Funktion ist.

- Die skalierten Versionen $g_a(x) := \frac{1}{a}g\left(\frac{x}{a}\right)$ $a > 0$ sind nicht in $\|\cdot\|_2$, sondern in $\|\cdot\|_1$ invariant:

$$\begin{aligned}\int_{\mathbb{R}} |g_a(x)|dx &= \int_{\mathbb{R}} |g\left(\frac{x}{a}\right)|d\frac{x}{a} \\ &= \int_{\mathbb{R}} |g(x)|dx = \|g\|_1\end{aligned}$$

Die Wavelet-Transformierte

$$\mathcal{W}[s](x,a) := [g_a * s](x)$$

lässt sich wieder zurücktransformieren via

$$s = \int_{\mathbb{R}_{>0}} \frac{da}{a}\mathcal{W}[s](x,a) = \int_{\mathbb{R}_{>0}} \frac{da}{a}[g_a * s](x),$$

falls $s \in L^1(\mathbb{R}) \cap L^\infty(\mathbb{R})$ und s gleichmäßig stetig. Und zwar in dem Sinne, dass

$$\int_\varepsilon^\rho \mathcal{W}[s](\cdot, a) \xrightarrow{\varepsilon\to 0,\ \rho\to\infty} s.$$

Für den Beweis wird auf Holschneider [37, Seite 7ff.] verwiesen.

3.11.1 LoG als Wavelet

Betrachtet man $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, so ergibt sich das Wavelet

$$g(x) = \frac{1}{\sqrt{2\pi}} \left(1 - x^2\right) e^{-\frac{x^2}{2}} = -\partial_{xx} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

und es wird klar, dass der LoG-Filter nichts anderes als ein in der Literatur als "Mexican Hat-" bzw. "Ricker-" bekanntes Wavelet ist.

Nun liegt es auf der Hand, weitere Wavelets zu erzeugen und diese als Filter zu benutzen. Im Prinzip kann man (fast) jede Verteilungsdichte nehmen und daraus ein Wavelet basteln. Hier dient uns die Cauchy-Verteilung als Ausgangspunkt für das Cauchy-Wavelet:

$$\begin{aligned} \phi(x) &= \frac{1}{\pi(1+x^2)} \Rightarrow \\ g(x) &= \frac{-2x^2}{\pi(1+x^2)^2} + \frac{1}{\pi(1+x^2)}. \end{aligned}$$

Es ist auch denkbar, ein Polynom $p(x) = p(-x)$ mit $e^{-\frac{x^2}{2}}$ zu multiplizieren, wie z.B. $x^2 \cdot e^{-\frac{x^2}{2}}$, und daraus ein Wavelet abzuleiten.

Für unsere Zwecke beschränken wir uns auf die geraden ($\phi(x) = \phi(-x)$), genügend glatte Dichten $\phi(x)$. Für die entstandenen Wavelets $g(x)$ und Wavelet-Transformierten gilt dann:

- Invarianz gegenüber Trends:

$$\begin{aligned} \mathcal{W}[\alpha x + \beta](x, a) &= \int_{\mathbb{R}} g_a(y) \cdot (\alpha(x - y) + \beta) dy \\ &= -\alpha \int_{\mathbb{R}} g_a(y) \cdot y dy + (\alpha x + \beta) \int_{\mathbb{R}} g_a(y) dy \\ &= 0 + 0 = 0, \end{aligned}$$

  weil $g_a(x)$ für geraden $\phi(x)$ gerade und mittelwertbereinigt ist.

- Das zur Gleichung (3.3) analoge Verhalten unter Koordinatentransformationen:

$$\begin{aligned}\mathcal{W}[Tf](x,a) &= \int_{\mathbb{R}} \frac{1}{a} g\left(\frac{x}{a}\right) f\left(\frac{x-y}{\alpha} - \frac{\beta}{\alpha}\right) dy \\ &\overset{y:=x-T(z)}{=} \int_{\mathbb{R}} \frac{|\alpha|}{a} g\left(\frac{x-(\alpha z+\beta)}{a}\right) f(z) dz \\ &= \int_{\mathbb{R}} \frac{1}{a/|\alpha|} g\left(\frac{T^{-1}x - z}{a/\alpha}\right) f(z) \\ &\overset{g(x)=g(-x)}{=} W[f]\left(T^{-1}x, \frac{a}{|\alpha|}\right)\end{aligned}$$

D.h. der LoG-Filter kann durch ein beliebiges Wavelet ersetzt werden, das aus einer geraden Dichte entstanden ist.

Was macht ein "gutes" Wavelet aus? Man möchte für ein Ereignis möglichst genaue zeitliche und spektrale Auflösungen erreichen. Doch ähnlich wie in der Quantenmechanik gilt hier die Heisenbergsche Unschärferelation:

Man kann nicht beliebig genaue Auflösung der Zeit und der Frequenz gleichzeitig erreichen.

Diese Aussage lässt sich präzisieren. Fasst man, ähnlich wie in der Quantenmechanik, $\frac{|g_a(x)|^2}{\|g\|_2^2}$ als eine Wahrscheinlichkeitsverteilung auf, so lässt sich die Varianz bestimmen (einfachheitshalber setzen wir voraus, dass

$$\int_{\mathbb{R}} x|g(x)|^2 dx = 0$$

ist, was insbesondere für gerade $\phi(x)$ der Fall ist):

$$\sigma^2[g] := \frac{1}{\|g\|_2^2} \int_{\mathbb{R}} x^2 |g(x)|^2 dx.$$

Die Ortsauflösung kann verbessert werden, indem man das Wavelet mit $a < 1$ skaliert:

$$\sigma^2[g_a] = a^2 \sigma^2[g].$$

Dafür bezahlt man jedoch mit der Auflösung bei der Frequenz. Betrachtet man die Fouriertransformierte

$$\hat{g} := \hat{\mathcal{F}}(g) = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} g(x) e^{-i\omega x} dx,$$

so gilt zum einem die Parcelsche Gleichung

$$\|g\|_2^2 = \left\|\hat{\mathcal{F}}(g)\right\|_2^2$$

zum anderen, falls $g(x)$ reellwertig, so ist $|\hat{\mathcal{F}}(g)(x)|^2$ bekanntermaßen gerade und damit $\int_{\mathbb{R}} x|\hat{\mathcal{F}}(g)(x)|^2 dx = 0$.

$$\sigma^2[\hat{g}] := \frac{1}{\|g\|_2^2} \int_{\mathbb{R}} \omega^2 |\hat{g}(\omega)|^2 d\omega$$

Bei der Betrachtung des skalierten Wavelets g_a ergibt sich dann

$$\hat{g}_a = \hat{g}(a\omega)$$

und

$$\sigma^2[\hat{g}_a] := \frac{1}{a^2}\sigma^2[\hat{g}].$$

Damit ist das Produkt $\sigma[\hat{g}_a] \cdot \sigma[g_a]$ eine Konstante und hängt nur von der Wahl von g ab. Die Heisenbergsche Ungleichung besagt (Beweis z.B. in Mallat [50, Theorem 2.6]):

$$\sigma[\hat{g}_a] \cdot \sigma[g_a] \geq \frac{1}{2}$$

mit der Gleichheit nur für $g(x) = a \cdot e^{-bx^2}$ mit $a, b \in \mathbb{C} \setminus \{0\}$. Dabei handelt es sich jedoch nicht um ein Wavelet, denn $\int_{\mathbb{R}} e^{-bx^2} dt \neq 0$.

Als Beispiel betrachten wir die Verteilungen (siehe Abbildung 3.16)

$$\begin{aligned}
\phi_1(x) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \quad \text{(Gaußfunktion)} \\
\phi_2(x) &= \frac{1}{\pi(1+x^2)} \quad \text{(Cauchy-Verteilung)} \\
\phi_3(x) &= \frac{1}{\sqrt{2\pi}} x^2 e^{-\frac{x^2}{2}},
\end{aligned}$$

so ergeben sich Wavelets

$$\begin{aligned}
g_1(x) &= \frac{1}{\sqrt{2\pi}}(1-x^2)e^{-\frac{x^2}{2}} \\
g_2(x) &= \frac{1-x^2}{\pi(1+x^2)^2} \\
g_3(x) &= \frac{1}{\sqrt{2\pi}}(3x^2-x^4)e^{-\frac{x^2}{2}}
\end{aligned}$$

mit Auflösungen

$$\begin{aligned}
\sigma[\hat{g}_1] \cdot \sigma[g_1] &= \sqrt{\frac{35}{12}} \\
\sigma[\hat{g}_2] \cdot \sigma[g_2] &= \sqrt{3} \\
\sigma[\hat{g}_3] \cdot \sigma[g_3] &= \frac{125}{22}.
\end{aligned}$$

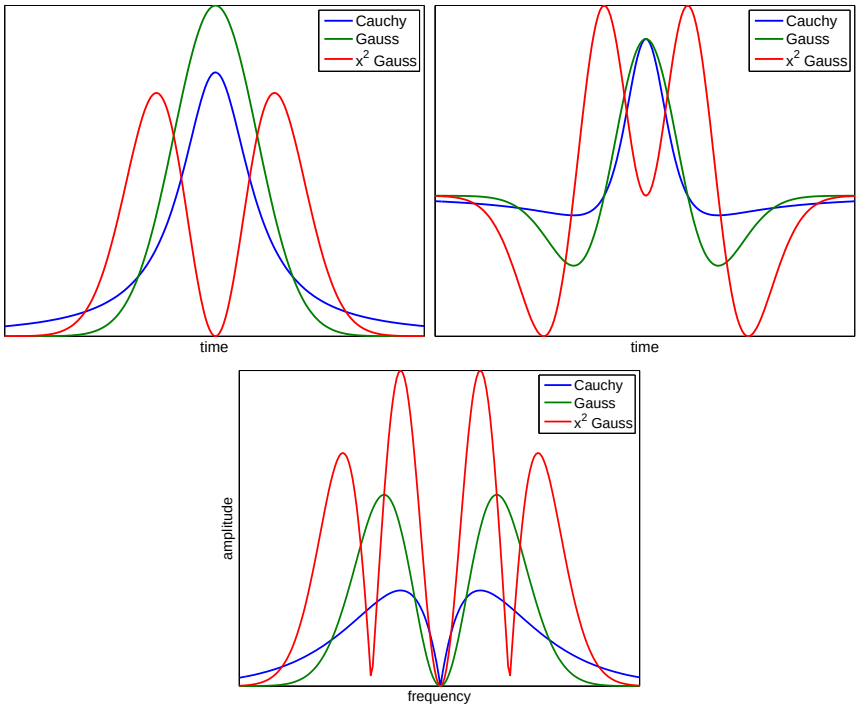

Abbildung 3.16: Oben links: Verteilungen ϕ_1, ϕ_2, ϕ_3. Oben rechts: auf den Verteilungen basierte Wavelets. Unten: Beträge der Fourier-Transformierten der Wavelets.

Man sieht, dass der LoG-Filter die beste Auflösung aufweist, obwohl die Cauchy-Verteilung kaum in etwas nachsteht.

Es ist jedoch nicht gänzlich klar, ob die bessere Auflösung letztendlich eine Rolle bei der Stabilität der detektierten Feature-Punkte spielt oder nicht. Die Performance der Detektoren wird von uns im Abschnitt 6.5 untersucht.

3.12 Lokale Deskriptoren

Theoretisch wäre es möglich, die Bänder nur anhand der Lage der Feature-Punkte mit Hilfe von *Iterative-Closest-Point-Algorithm* (Besl und McKay [9]) zu bestimmen. Unsere Experimente haben jedoch gezeigt, dass dieser Ansatz für unsere Datensätze nicht stabil genug ist. Ähnliche Probleme gibt es auch für andere Datensätze, so dass es eine große Anzahl von Feature-Deskriptoren gibt, die benutzt werden, um Zuordnungen zwischen den Features zu bestimmen.

Dieses Problem wird in der Bildverarbeitung dadurch gelöst, dass zu jedem Feature-Punkt auch ein Deskriptor gehört, mit dessen Hilfe man die Feature-Punkte einander zuordnen kann.

Hier sollen nun einige aus der Bildverarbeitung bekannte Deskriptoren benannt werden. Diese haben alle eine Gemeinsamkeit: Sie werden alle aus der lokalen Umgebung der Feature-Punkte berechnet. Wie schon oben gesehen kann diese Umgebung skalierungsinvariant bestimmt werden. Für die Anpassung dieser normalerweise 2D-Deskriptoren auf den eindimensionalen Fall und deren Performance wird an der Stelle auf den Abschnitt 6.1 verwiesen.

An der Stelle seien vollständigkeitshalber nur die Name der von uns untersuchten Deskriptoren genannt:

1. Cross-Correlation-Deskriptoren
2. SIFT-Deskriptor (Lowe [48])
3. PCA-Deskriptoren (Ke und Sukthankar [40])
4. Momentinvarianten (Gool u. a. [32])
5. Ein SURF-Deskriptor (Bay u. a. [5])
6. BRIEF (Calonder u. a. [14]), BRISC (Leutenegger u. a. [45]) und FREAK (Ortiz [55]) Deskriptoren.

Kapitel 4

Generisches Framework

Unser generisches Framework zur Lösung der im Kapitel 2 vorgestellten Probleme besteht aus zwei Teilen:

- Vorverarbeitung bzw. Erstellung der sogenannten *Fingerprints*, die aus den wesentlichen Abschnitten einer Zeitreihe bestehen. In diesem Teil werden die Zeitreihen analysiert, besonders interessante Abschnitte (auch als Feature-Punkte bezeichnet) gefunden und zu den "Fingerprints" zusammengefasst.
- Abgleich der Fingerprints, woraus man auf das Alignment der Zeitreihen bzw. deren Zusammengehörigkeit zurückschließen kann.

4.1 Erstellung der Fingerprints

Unsere Fingerprints bestehen aus mehreren Feature-Punkten. Das sind Abschnitte der Zeitreihe, die sich leicht wiederfinden lassen, trotz möglicher Verzerrungen, - solchen wie Skalierungen der Achsen oder Gaußrauschen. Solche Feature-Punkte könnten z.B. Maxima, Minima, Wendepunkte oder auch viele andere denkbare Muster sein.

Ein Feature-Punkt beinhaltet sowohl die Information über seine Lage bzw. Koordinaten in der Messreihe (diese Information wird von uns als *Frame* bezeichnet) als auch die Information über seine unmittelbare Umgebung. Diese Umgebungsinformation nennen wir *Deskriptor* - sie ist notwendig, um Feature-Punkte zweier Fingerprints einander zuordnen zu können.

Es werden folgende Anforderungen an die Feature-Punkte gestellt:

1. schnell zu berechnen
2. störungsresistent

3. skalierungsinvariant (sowohl x- als auch y-Achsen)
4. translationsinvariant
5. spiegelungsinvariant.

Von allen geforderten Eigenschaften ist die Spiegelungsinvarianz im eindimensionalen Fall am wenigsten kritisch: Es gibt nur zwei mögliche Orientierungen, die auch einzeln getestet werden können, was für höhere Dimensionen nicht der Fall wäre.

Um solche Punkte zu finden, benutzen wir, wie in Abschnitt 3.11 vorgestellt, die kontinuierliche Wavelettransformierte

$$\mathcal{W}[f](x,\sigma) = \int_{\mathbb{R}} f(x-y)\cdot w(y,\sigma)\,dy,$$

hier mit dem Wavelet $w(\cdot,\sigma)$ oder deren Näherungen z.B. DoG.

Die Feature-Punkte sind dann ähnlich wie im Shazam-Ansatz (Abschnitt 3.6) die lokalen Extrema von $\mathcal{W}[f](x,\sigma)$. Das Frame des Feature-Punktes wird durch die Koordinaten des lokalen Extremums (x^*, σ^*) und die Orientierung festgelegt.

Die Koordinaten des Extremums werden benutzt, um eine skalierungsinvarinate Umgebung festzulegen (siehe Abschnitt 3.8)

$$u[f](\cdot; x^*, \sigma^*) := T^{-1}_{(\sigma^*, x^*)}\left[G(\cdot,\sigma^*) * f\right],$$

aus welcher ein skalierungsinvarinater Deskriptor berechnet werden kann.

Wir betrachten zwei Möglichkeiten, Spiegelungsinvarianz zu erreichen bzw. zu umgehen, je nach der Art des ausgewählten Deskriptors.

1. Es wird eine lokale Orientierung ς festgelegt, z.B. anhand der Umgebung oder des Deskriptors.
2. Die beiden möglichen Orientierungen werden einzeln getestet.

Anschließend können die besonders stabilen Features gewählt werden, um die Größe des Fingerprints zu reduzieren.

Im letzten Schritt werden die Deskriptoren in passende Datenstrukturen eingeordnet, die eine schnelle Suche ermöglichen.

4.2 Abgleich der Fingerprints

Der Abgleich der Fingerprints passiert in zwei Phasen.

In der ersten Phase des Abgleichs werden die Features anhand ihrer Deskriptoren verglichen. Dabei werden sie jeweils ihrem nächsten Nachbarn in der

L^2-Norm unter Berücksichtigung der Orientierung zugeordnet. Aus diesem Vorgang ergeben sich n Koordinatenzuordnungen

$$z_i := \Big((x^i,\, \sigma_x^i) \mapsto (y^i,\, \sigma_y^i),\, \varsigma_i \Big)$$

mit jeweils einer Bewertung e_i, die im Prinzip aus dem hinzugehörigen L^2-Abstand besteht. Diese Menge bezeichnen wir mit $\mathcal{Z} := \{z_i \,|\, i = 1, \cdot, n\}$. Das Element ς gibt an, ob die verglichenen Deskriptoren die gleiche Orientierung ($\varsigma = 1$) hatten oder nicht ($\varsigma = -1$).

$\varsigma = -1$ bedeutet, dass die Koordinatentransformation zwischen den Koordinatensystemen der Fingerprints eine Spiegelung beinhaltet.

An der Stelle möchten wir einen Sprachgebrauch einführen, der in der ganzen Arbeit verwendet wird. Zu einer Zuordnung und einer Koordinatentransformation $T_{(\alpha\,,\,\beta)}$ sagen wir, dass die Zuordnung die Koordinatentransformation *untermauert*, falls gilt

$$y_i \approx T_{(\alpha\,,\,\beta)}(x_i).$$

Man muss beachten, dass $\approx$ hier im generischen Sinne gebraucht wird und die genaue Bedeutung erst später festgelegt wird.

Mit *Inliers* einer Koordinatentransformation bezeichnen wir alle Zuordnungen, die diese Transformation untermauern:

$$In(T_{(\alpha\,,\,\beta)}) := \{z_i \,|\, y_i \approx T_{(\alpha\,,\,\beta)}(x_i)\} \subset \mathcal{Z}$$

In der zweiten Phase des Abgleichs werden die richtigen Zuordnungen von den falschen getrennt.

Wir gehen dabei von der Annahme aus, dass die richtige Koordinatentransformation $T^r_{(\alpha\,,\,\beta)}$ die größte Anzahl der Inliers aufweist:

$$\left| In\left(T^r_{(\alpha\,,\,\beta)}\right) \right| \geq \left| In\left(T_{(\alpha\,,\,\beta)}\right) \right| \quad \forall T_{(\alpha\,,\,\beta)}.$$

Und so kann man mit Hilfe von Hough-Transformation (Duda und Hart [25]) bzw. RANSAC-Verfahren (Fischler und Bolles [27]) die richtige Transformation T^r und die richtigen Zuordnungen $In\,(T^r)$ bestimmen.

Die Anzahl der richtigen Zuordnungen dient letztendlich als Grundlage für die Entscheidung, ob die beiden Fingerprints zueinander passen oder nicht.

4.3 Ein Beispiel

Nun möchten wir uns ein kleines Beispiel anschauen, um die oben beschriebene Vorgehensweise zu illustrieren. Um die Übersichtlichkeit nicht zu ver-

lieren, beschränken wir uns auf zwei synthetische Messreihen

$$m_1(x) = \frac{x-1}{5} e^{-|\frac{x-1}{5}|} + \xi_1,$$
$$m_2(x) = -4\frac{x-26}{5} e^{-|2\frac{x-26}{5}|} + \xi_2,$$

die bis auf moderate Störungen ξ_1 und ξ_2 gespiegelte (x-Achse), skalierte (x- und y -Achsen) und verschobene Versionen voneinander sind (Abbildung 4.1). Das durch ξ_1 und ξ_2 erzeugte Rauschen übersteigt nicht die 10% des maximalen Werts der Zeitreihe.

Wir benutzen den DoG-Detector (Abbildungen 4.2 und 4.3), um die Feature-Punkte zu bestimmen.

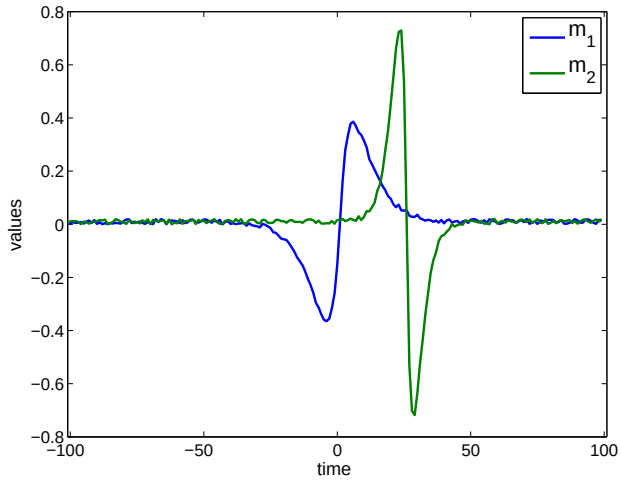

Abbildung 4.1: Wenn auf den ersten Blick auch nicht klar, handelt es sich dabei um zwei Messungen der gleichen 0bservablen. Was ist nun die Koordinatentransformation $T_{(\alpha,\beta)}$?

Die Koordinaten dieser Extrema bzw. der Feature-Punkte lauten:

Deskriptornummer	(x, σ) von m_1	(x, σ) von m_2
1	(−27, 9.0)	(12, 4.5)
2	(−5, 6.0)	(23, 3.0)
3	(8, 6.5)	(29, 3.0)
4	(29, 9.0)	(40, 4.5)

Den Deskriptor für ein in (x_e, σ_e) lokalisiertes Feature der Zeitreihe m berechnen wir dabei wie folgt:

- Die Zeitreihe m wird mit der passenden Gaußfunktion geglättet:

$$M(x) := m * G(\cdot, \sigma_e).$$

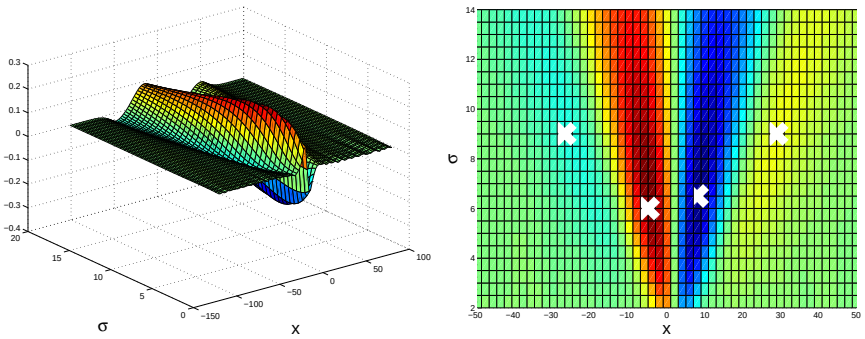

Abbildung 4.2: Der zur Detektion der Feature-Punkte der Zeitreihe m_1 benutzte DoG-Detektor ($DoG * m_1$). Links als 3D-Gebirge und rechts als Sicht von oben mit markierten 4 Extrema. Die sehr kleinen, durch Rauschen hervorgerufenen Extrema werden von uns vernachlässigt.

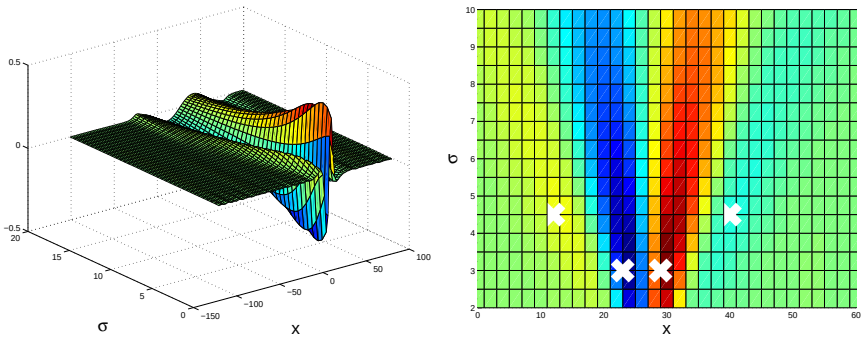

Abbildung 4.3: Der zur Detektion der Feature-Punkte der Zeitreihe m_2 benutzte DoG-Detektor ($DoG * m_2$). Links als 3D-Gebirge und rechts als Sicht von oben mit 4 markierten Extrema. Die sehr kleinen, durch Rauschen hervorgerufenen Extrema werden von uns vernachlässigt.

.

- Die geglättete Zeitreihe $M(x)$ wird in der Umgebung von x_e mit einer passenden Schrittweite ($\propto \sigma_e$) abgetastet (dabei kann z.B. lineare Interpolation verwendet werden):

$$U_e^j := M(x_e - \sigma_e \cdot j) \quad -8 \leq j \leq 8.$$

- Als Deskriptor d_e wird die normalisierte Umgebung U_e benutzt:

$$\begin{aligned}\hat{U_e} &:= \frac{1}{17}\sum_{j=-8}^{8} U_e^j \\ \tilde{U}_e^j &:= U_e^j - \hat{U_e} \quad -8 \le j \le 8 \\ d_e^j &:= \frac{\tilde{U}_e^j}{\left\|\tilde{U}_e\right\|_2} \quad -8 \le j \le 8.\end{aligned}$$

  Es handelt sich dabei um einen Cross-Correlation-Deskriptor, der von uns in Abschnitt 6.1.1 genauer untersucht wird.

- Außerdem wird die lokale Orientierung festgelegt: die linke Seite des Deskriptors d_e sollte "schwerer" sein als die rechte, d.h.

$$\sum_{j<0} |d_e^j| \ge \sum_{j>0} |d_e^j|.$$

  Ist dies nicht der Fall, dann wird der Deskriptor einfach umgedreht/-gespiegelt mit Hilfe der Operation S:

$$S(d_e)^j := d_e^{-j}.$$

  Es lässt sich leicht einsehen, dass solche Deskriptoren spiegelungsinvariant sind (bis auf den unwahrscheinlichen, vernachlässigbaren Fall $\sum_{j<0} |d_e^j| = \sum_{j>0} |d_e^j|$). Dies kann in Abschnitt 6.4 detaillierter nachgelesen werden.

Die Abbildungen 4.4 und 4.5 illustrieren die obige Vorgehensweise und präsentieren die entstandenen Deskriptoren für die vorgegebenen Zeitreihen m_1 und m_2.

Die Fingerprints der Messreihen m_1 und m_2 bestehen also aus jeweils 4 Features. Jedes Feature besteht aus einem Deskriptor und einem Frame, welches seinerseits aus x-, σ-Koordinaten und der lokalen Orientierung besteht. Die lokale Orientierung hat den Wert 1, falls die Umgebung nicht umgedreht werden musste, und -1 sonst. Also:

$$\begin{aligned}Feature_e &:= (Deskriptor_e, Frame_e) \\ &= (d_e, (x_e, \sigma_e, orientation_e))\,.\end{aligned}$$

Die Fingerprints der beiden Zeitreihen sehen explizit wie folgt aus:

$$\begin{aligned}FP(m_1) = \{&(d_1, (-27, 9.0, -1)\,), \\ &(d_2, (\ -5, 6.0, -1)\,), \\ &(d_3, (\ \ 8, 6.5,\ \ 1)\,), \\ &(d_4, (\ 29, 9.0,\ \ 1)\,)\}\end{aligned}$$

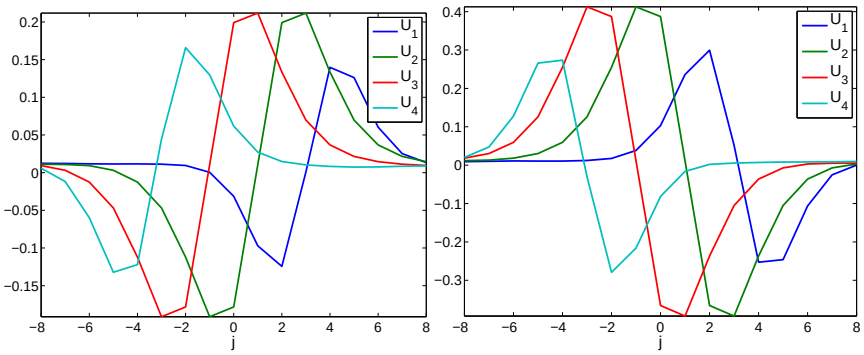

Abbildung 4.4: Links befinden sich die Umgebungen der Extrema von m_1 und rechts die Umgebungen der Extrema von m_2. Die Umgebungen 1 und 2 sowohl von m_1 als auch von m_2 müssen nach unseren Regeln (denn die rechte Seite ist "schwerer" als die linke) umgedreht werden. Damit ergeben sich nach der Normalisierung die in der Abbildung 4.5 abgebildeten Deskriptoren. Man beachte: Die Umgebungen sind skalierungsinvariant. Obwohl die Zeitreihen m_1 und m_2 unterschiedliche x-Achsenskalierungen aufweisen, ist es nicht mehr der Fall für die Umgebungen. Dafür ist die Wahl der Schrittweite $\propto \sigma_e$ verantwortlich.

und

$$\begin{aligned} FP(m_2) = \{&(d_1, (12, 4.5, -1)\,),\\ &(d_2, (23, 3.0, -1)\,),\\ &(d_3, (29, 3.0,\ \ 1)\,),\\ &(d_4, (40, 4.5,\ \ 1)\,)\}. \end{aligned}$$

Dabei handelt sich bei den Deskriptoren d_i in $FP(m_1)$ um die Deskriptoren der Zeitreihe m_1 (links in der Abbildung 4.5) und bei den d_i in $FP(m_2)$ um die Deskriptoren der Zeitreihe m_2 (rechts in der Abbildung 4.5).

Sucht man den nächsten Nachbarn (in der $\mathcal{L}^2$-Norm) für jeden Deskriptor, so stellt man folgende Zuordnungen fest, wovon man sich leicht anhand der Abbildung 4.5 überzeugen kann:

Feature-Nummer $FP(m_1)$	Feature-Nummer $FP(m_2)$
1	4
2	3
3	2
4	1

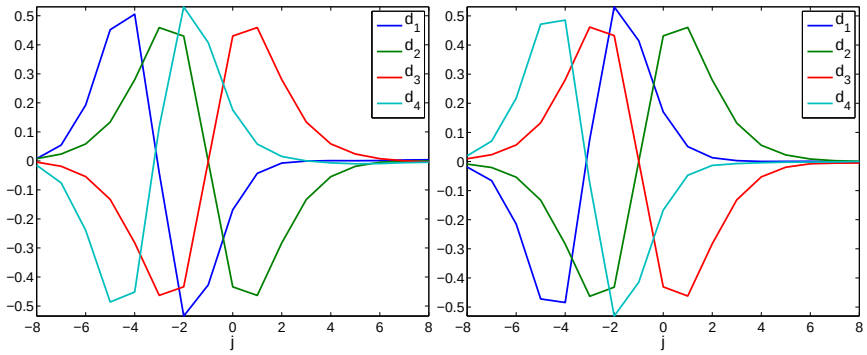

Abbildung 4.5: Links befinden sich die Deskriptoren der Features von m_1 und rechts die Deskriptoren der Extrema von m_2. Beachte: Durch die Normalisierung können die Deskriptoren von m_1 mit den Deskriptoren von m_2 verglichen werden, obwohl die Umgebungen sich noch um den Faktor 2 unterschieden haben.

Daraus ergeben sich folgende Koordinatenzuordnungen:

$$
\begin{aligned}
z_1 &= ((-27, 9.0) \mapsto (40, 4.5),\ -1)\,, \\
z_2 &= ((-5, 6.0) \mapsto (29, 3.0),\ -1)\,, \\
z_3 &= ((8, 6.5) \mapsto (23, 3.0),\ -1)\,, \\
z_4 &= ((29, 9.0) \mapsto (12, 4.5),\ -1)\,.
\end{aligned}
$$

Die Orientierung ergibt sich aus dem Produkt der lokalen Orientierungen der beiden zugeordneten Features:

- Ist das Produkt 1, so heißt es, dass entweder beide Features umgedreht (gespiegelt) oder beide nicht umgedreht wurden. Die Ausrichtung der Zeitreihen relativ zueinander beinhaltet also keine Spiegelung.

- Ist das Produkt -1, so bedeutet es, dass genau ein Feature umgedreht wurde und genau eines nicht. Die Ausrichtung der Zeitreihen relativ zueinander beinhalten damit eine Spiegelung.

Aus den Zuordnungen der Frames gewinnt man folgende Erkenntnisse:

1. Die Orientierung aller vier Zuordnungen beträgt -1, was darauf hindeutet, dass die Koordinatentransformation eine Spiegelung beinhaltet.

2. Betrachtet man das Verhältnis der zugeordneten σ-Koordinaten, so stellt man fest, dass der Skalierungsfaktor 2 betragen muss (wenn man von dem kleinen Ausreißer bei der Zuordnung z_3 absieht).

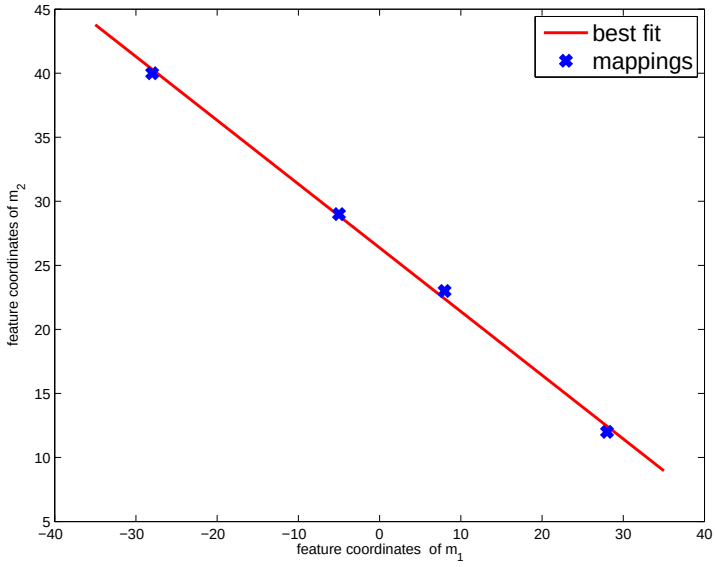

Abbildung 4.6: Die Ausgleichsgerade für die Koordinatentransformation ergibt $\alpha = -0.498$ *und* $\beta = 26.62$*. Die leichten Abweichungen von den tatsächlichen Werten* $(-0.5, 26.5)$ *sind durch den Einfluss der Störungen* ξ_1 *und* ξ_2 *zu erklären. Übrigens wurde die Spiegelung der Zeitreihen relativ zu einander auch richtig erkannt (Steigung* < 0*).*

Die genaue Koordinatentransformation $x_{m_2} = \alpha x_{m_1} + \beta$ bekommt man, indem man die Koordinatenzuordnungen auswertet, z.B mit Hilfe einer Ausgleichsgeraden (Abbildung 4.6).

Es ist offensichtlich, dass alle vier Zuordnungen die gefundene Transformation untermauern, falls wir alle Punkte "anerkennen", deren Abstand zur Ausgleichsgerade nicht größer als 1 beträgt. Damit wäre der *Score* für das Zueinandergehören der beiden Messungen 4 aus 4.

Natürlich reichen vier Zuordnungen nicht aus, um die Frage nach der Zugehörigkeit der beiden Messungen zum gleichen Band mit großer Sicherheit zu beantworten, doch für die Zwecke der Illustrierung akzeptieren wir hier die Annahme, dass m_1 und m_2 zueinander passen. Es ist wichtig zu beachten, dass diese Entscheidung von uns nur aufgrund der vorliegenden Fingerprints getroffen wurde, ohne auf die Messdaten wieder zuzugreifen.

Kapitel 5

Details der Implementierung

In diesem Kapitel besprechen wir die Details unserer Implementierung: Angefangen mit der Berechnung von Fingerprints aus den Zeitreihen bis hin zur Bestimmung der Anzahl von sogenannten *Inliers* (siehe Abschnitt 4.2 für die Definition), die als ein Maß für die Ähnlichkeit zweier Zeitreihen dient.

Zuerst sei nochmals zur Erinnerung ein allgemeiner Überblick über unsere Methode angeführt:

1. Berechnung des Fingerprints:
   (a) Berechnung der kontinuierlichen Wavelet-Transformation beziehungsweise deren Näherung.
   (b) Bestimmung von lokalen Extrema der Wavelet-Transformierten.
   (c) Eine passende Abtastung der Umgebung der gefundenen Extrema. Dafür werden die Faltungen der Zeitreihe mit den passenden Gaußfunktionen benutzt.
   (d) Erstellung der Deskriptoren aus den abgetasteten Umgebungen und die damit verbundene Reduktion der Dimensionalität.
   (e) Abspeichern der Features (der berechnete Deskriptor und die dazugehörigen Koordinaten des Extremums) in einer passenden Datenstruktur zur Suche des nächsten Nachbarn.
2. Abgleich zweier Fingerprints:
   (a) Bestimmung der Zuordnung zwischen den Feature-Punkten beider Fingerprints aufgrund ihrer Deskriptoren.
   (b) Aussortierung der falsch bestimmten Zuordnungen.
   (c) Bestimmung des Alignments der beiden Fingerprints bzw. der zugrunde liegenden Koordinatentransformation.
   (d) Berechnung des Maßes für die Ähnlichkeit der beiden Fingerprints mit Hilfe des gefundenen Alignments.

5.1 Darstellung von Faltungen als Pyramide

Im Laufe der vorliegenden Arbeit betrachten wir folgende Feature-Detektoren:

- DoG (siehe Abschnitt 3.8).
- Boxes (siehe Abschnitt 3.10).
- Wavelets (z.B. Cauchy, Mexican Hat; mehr in Abschnitt 3.11).

Es handelt sich dabei um die Faltung zweier Funktionen, die im Falle von *Boxes* mit Hilfe der Präfixsumme effizient erledigt werden kann. Deswegen betrachten wir als erstes Möglichkeiten, eine Faltung zu berechnen.

5.1.1 Berechnung einer Faltung

Die naive Berechnung der Faltung zweier Zeitreihen der Länge n würde eine $O(n^2)$ Laufzeit in Anspruch nehmen, wodurch sie für die Untersuchung der Zeitreihen mit mehreren tausend Einträgen unbrauchbar wäre. Es gibt jedoch (mindestens) zwei Möglichkeiten die Rechenzeit zu reduzieren.

Die erste beruht auf dem Zusammenhang zwischen der Faltung und der Fouriertransformation, dem sogenannten Faltungstheorem (Theorem 2 in Anhang 10.2): Mit Hilfe von FFT kann die Faltung in $O(n \log n)$ berechnet werden.

In unserer Implementierung nutzen wir die Tatsache aus, dass die Zeitreihen mit einem Wavelet oder einem Gaußkernel gefaltet werden müssen. Diese haben zwar einen unbegrenzten Träger, es reicht jedoch, nur wenige Einträge zu betrachten, um den entstehenden Fehler vernachlässigbar zu machen. Im Falle einer Gaußfunktion $G(x, \sigma)$ reicht die Breite von $8 \cdot \sigma$, um mehr als 99.99% Genauigkeit in der $\mathcal{L}^1$-Norm zu erreichen (für weitere Einzelheiten siehe Anhang 10.3.3). So liegt es nahe mit abgeschnittenen Kernels zu arbeiten, deren Länge m nicht mehr als 30 Werte enthält. Die asymptotische Laufzeit der Berechnung beträgt dann $O(m \cdot n)$ und ist in der Praxis deutlich schneller als die Anwendung der FFT für lange Zeitreihen.

Bei der Berechnung der Scale-Space-Funktion ist es jedoch notwendig, die Breite des Kernels ständig bis hin zu den Größen zu erhöhen, die mit der Länge der Zeitreihe vergleichbar sind. Für diesen Bereich müsste auch m - die Länge des abgeschnittenen Kernels - wachsen, was die Laufzeit letztendlich auf die unerwünschten $O(n^2)$ anwachsen ließe. Zu Hilfe kommen die Low-Pass-Filter Eigenschaften der uns interessierenden Kernel: Nach der damit durchgeführten Faltung werden die hohen Frequenzen ausgelöscht und das Ergebnis kann in einer niedrigeren Auflösung abgetastet werden, ohne dabei

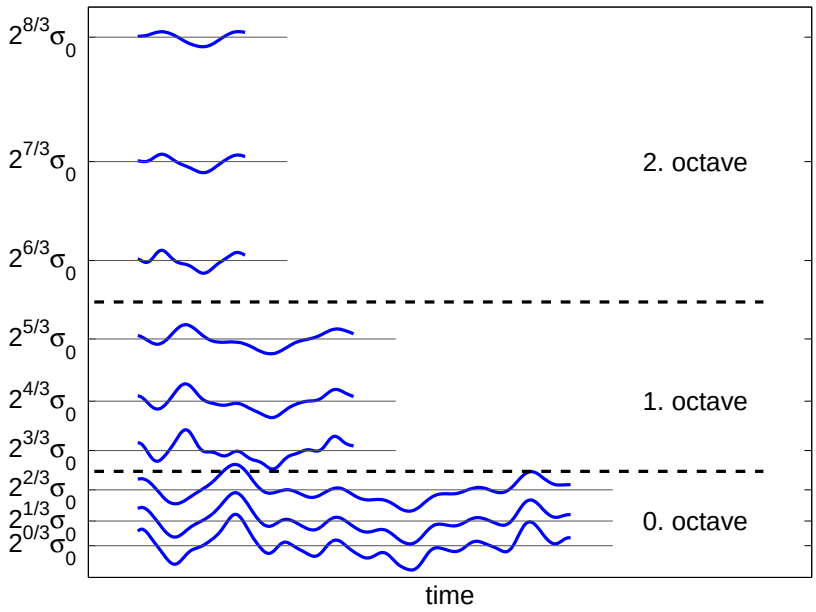

Abbildung 5.1: Diese Abbildung zeigt die ersten drei Oktaven einer Gauß-Pyramide mit 3 *Skalierungen pro Oktave und dem Skalierungsfaktor* $f = \sqrt[3]{2}$ *zwischen zwei benachbarten Niveaus. Mit jeder Oktave wird die Auflösung und damit die Länge der Zeitreihe und der Rechenaufwand halbiert. Die* 0. *Oktave rechnet mit der originalen Auflösung,* 1. *Oktave mit der halbierten Auflösung,* 2. *Oktave mit einem Viertel der ursprünglichen Auflösung usw.*

Information zu verlieren, wie es das Abtasttheorem (Anhang 10.3.5) besagt. Nun liegt es auf der Hand: Statt die Länge des Kernels zu erhöhen, reduziert man die Länge der Zeitreihe!

Die auf dieser Idee basierende zuerst von P. Burt (Burt [12]) vorgeschlagene Vorgehensweise, die Faltungen in einer Pyramide anzuordnen, wurde zur Standardmethode und unter anderem in Lowe [48] oder Burt u. a. [13] angewandt.

Die Scale-Space-Funktion als Pyramide

In diesem Abschnitt sollen die Einzelheiten der von uns benutzten Methode zur Berechnung von Faltungen in Form einer Pyramide (Abbildung 5.1) besprochen werden. Die von uns berechnete Scale-Space-Funktion der originalen Zeitreihe h ist in mehrere sogenannte *Oktaven* unterteilt, denn wir wählen die folgende Vorgehensweise (vergleiche Abbildung 5.2):

1. Wähle die Anzahl der Oktaven o, anfängliche Skalierung σ_0 und die Anzahl der Skalierungen pro Oktave k.

2. Erzeuge Faltungskernel g_j der festen Länge m zu den Skalierungen

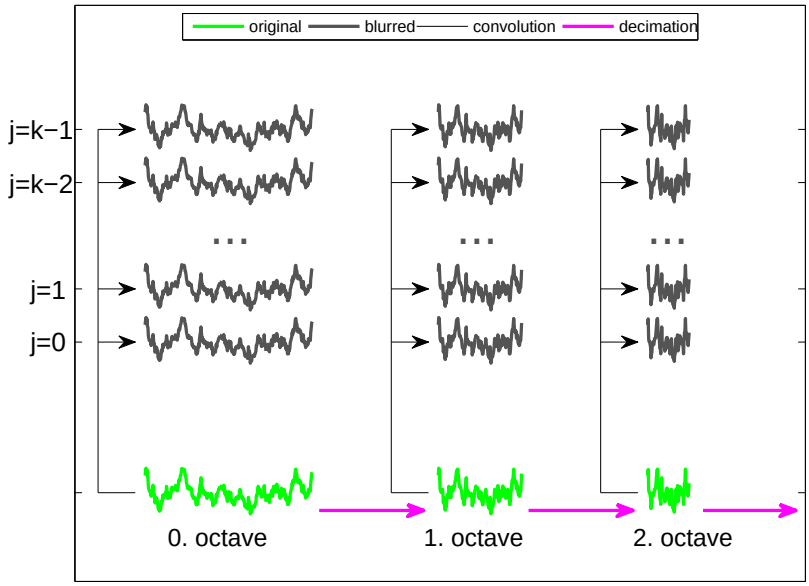

Abbildung 5.2: Unsere Vorgehensweise bei der Berechnung der Gauß-Pyramide: Das Original (grün) wird mit den Gaußkerneln für $j = 0, \ldots, k-1$ gefaltet (schwarze Pfeile), womit eine Oktave in der Pyramide erstellt (graue geglätteten Kurven) wird. Anschließend wird die Auflösung mit Hilfe von Decimation halbiert (magentafarbene Pfeile) und nach der Faltung mit den gleichen Gaußkerneln wird eine weitere Oktave der Pyramide hinzugefügt. Der Vorgang wiederholt sich mehrfach nach weiteren Halbierungen der originalen Auflösung.

$f^j \cdot \sigma_0$ mit $0 \leq j \leq k-1$ und $f := \sqrt[k]{2}$.

3. Berechne die Faltungen $h * g_0$, ..., $h * g_{k-1}$. Die dazu benötigte Zeit: $O(m \cdot n \cdot k)$.

4. Wir benutzen die *Decimation* (Anhang 10.4) - ein vor allem aus der Bildverarbeitung bekanntes Verfahren - um die Auflösung der Zeitreihe zu halbieren. Dadurch halbiert sich auch die Dimension n.

5. Wiederhole Schritte 3. und 4. insgesamt o mal.

Damit wird die Scale-Space-Funktion für die Skalierungen $2^{i+j/k} \cdot \sigma_0$ mit $0 \leq i \leq o-1$ und $0 \leq j \leq k-1$ abgetastet. Dabei wird jedoch nur für $i = 0$ mit der vollen Zeitauflösung gearbeitet.

Man sollte jedoch $\sigma_0 > 1$ wählen: Wird nur bei den ganzzahligen x abgetastet, so ähnelt $G(x, 1)$ schon fast der $\delta(x)$-Funktion

$$\delta(x) = \begin{cases} 1 & x = 0 \\ 0 & \text{sonst} \end{cases}$$

und die berechnete Faltung liefert nicht das erwünschte Ergebnis. Um mit solchen kleinen Skalierungen rechnen zu können, müsste man beide Funktionen feiner als mit Abstand 1 abtasten.

Wegen der Halbierung der Auflösung bzw. der Länge der Zeitreihe h ist es offensichtlich, dass die Berechnung der ersten Oktave mehr als die Hälfte der Gesamtrechenzeit beansprucht. Das legt eine Möglichkeit zur Beschleunigung nahe: Man kann die für die Berechnungen notwendige Zeit halbieren, indem man die erste Oktave auslässt und die Betrachtung mit der zweiten Oktave startet.

LoG und andere Wavelets

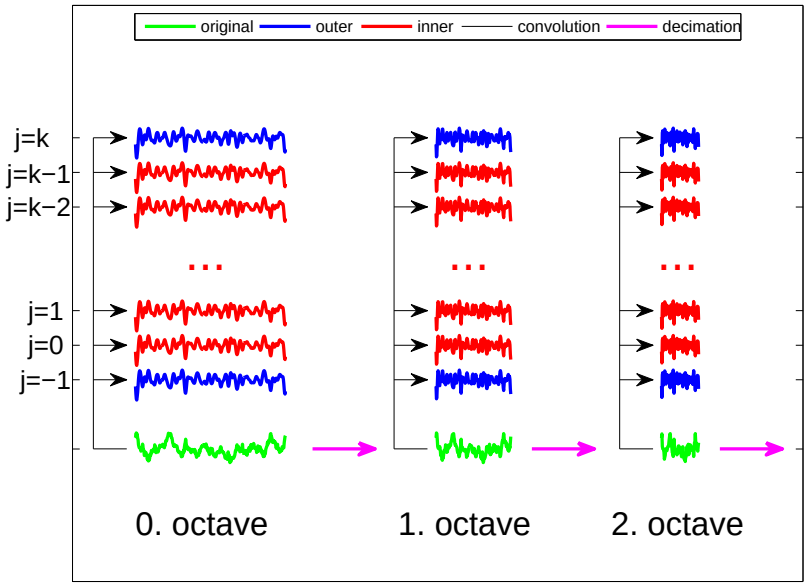

Abbildung 5.3: Unsere Vorgehensweise bei der Berechnung der LoG-Pyramide: Durch die Faltung (schwarze Pfeile) werden aus der originalen Zeitreihe (grün) neben den Hauptskalierungen (rot, $j = 0, \ldots, k-1$) auch die LoGs für die Skalierungen $j = -1$ und $j = k$ (blau) berechnet. Die blauen LoGs sind notwendig, um die lokalen Extrema für alle Hauptskalierungen zu finden. Beim Übergang zur nächsten Oktave wird die Auflösung mit Hilfe von Decimation halbiert. Der Vorgang wiederholt sich mit jeder Halbierung der Auflösung der originalen Zeitreihe.

Benutzten wir LoG oder andere Wavelets als Feature-Detektor, so wird zum einen die Gauß-Pyramide gebraucht, um die Umgebungen der Feature-Punkte skalierungsinvariant abtasten zu können, zum anderen die dazu passende Wavelet-Pyramide, um die Feature-Punkte zu finden.

Zur Berechnung der Wavelet-Pyramide benutzen wir die gleiche Vorgehensweise wie oben, mit dem Unterschied, dass statt der Gaußkernel die (belie-

bigen) Wavelet-Funktionen benutzt werden (Abbildung 5.3).

Ein weiterer Unterschied besteht darin, dass zusätzlich noch zwei weitere Levels pro Oktave berechnet werden. Und zwar: $j = -1$ und $j = k$. Dies ist notwendig, um die lokalen Extrema für $j = 0$ und $j = k - 1$ detektieren zu können. Zwar liegt diese Information in den Nachbaroktaven vor, doch in einer anderen Auflösung.

Die Levels $j = -1$ und $j = 0$ könnten durch das Downsampling von $j = k-1$ und $j = k$ aus der vorherigen Oktave gewonnen werden. Wie im Anhang 10.3.5 illustriert wurde, besitzt der LoG-Kernel weniger gute Low-Pass-Filter-Eigenschaften als der Gaußkernel zum gleichen Parameter σ. Deswegen ist auch eine Faltung mit einem Low-Pass-Filter vor dem Downsampling notwendig. Diese zusätzlichen Faltungen führen dazu, dass es keine Vorteile gegenüber der naiven Vorgehensweise gibt, bei der die Levels $j = -1$ und $j = 0$ durch eine Faltung berechnet werden.

DoG

Wie im Abschnitt 3.8.1 schon besprochen, stellt der DoG-Filter eine Approximation des LoG-Filters dar, welcher seinerseits ein Wavelet (Mexican Hat) ist. Der Vorteil des DoG-Filters besteht darin, dass man die Berechnungszeit (fast) halbieren kann, weil man neben den Faltungen mit den Gaußkerneln keine weiteren Faltungsvorgänge braucht. Dagegen sind beim Berechnen von LoG noch weitere Faltungen mit dem Mexican-Hat-Wavelet notwendig.

Pro Oktave werden hier jedoch nicht $k + 2$ sondern $k + 3$ Faltungen mit Gaußkerneln berechnet und zwar für $j = -1, \ldots, k + 1$, damit man $k + 1$ Differenzen via

$$DoG_j^i = G_{j+1}^i - G_j^i \qquad 0 \leq i \leq o - 1 \text{ und } 0 \leq j \leq k \tag{5.1}$$

bilden kann. G_j^i sind die Levels aus der Gauß-Pyramide in passender Auflösung.

Die Abbildung 5.4 fasst den Algorithmus zur Berechnung des DoG-Detektors zusammen. Dabei sollte Folgendes berücksichtigt werden:

- Die Halbierung der Auflösung mit Downsamling ist nur deswegen möglich, weil nach der Faltung mit dem Gaußkernel wegen dessen Low-Pass-Filter-Eigenschaften die hohen Frequenzen ausgelöscht werden und man nach dem Abtasttheorem die Auflösung reduzieren kann. Dies ist jedoch nur dann zulässig, wenn der $\sigma = 2\sigma_0$ - Parameter von $G(x, \sigma)$ (Kernel zu $j = 0$) groß genug ist. Im Anhang 10.3.5 wurde gezeigt, dass $\sigma > 2$ ausreichend ist, deswegen sollte man $\sigma_0 > 1$ wählen.

- Es ist nicht notwendig immer die originale Zeitreihe s zu falten, denn

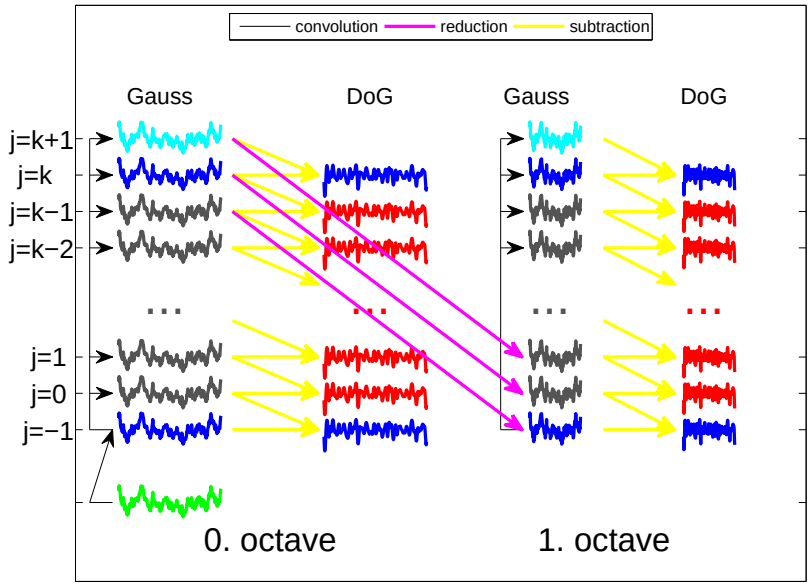

Abbildung 5.4: Unsere Vorgehensweise bei der Berechnung der DoG-Pyramide: Dabei werden nur die Faltungen mit den Gaußkerneln bzw. eine leicht erweiterte Gauß-Pyramide benötigt. Da die Reduktion der Auflösung mit Downsamling (magenta Pfeile) "billiger" ist als die Faltung (schwarze Pfeile), lässt sich der Rechenaufwand verringern, indem man keine Faltungen mit Gaußkerneln für $j = 0, 1, 2$ ab der 1. Oktave berechnen muss. Die Subtraktion via Formel 5.1 (gelbe Pfeile) ist deutlich schneller als eine Faltung mit einem LoG-Filter.

wegen der Assoziativität der Faltung und der Eigenschaften der Gaußfunktion gilt:

$$\begin{aligned} s * G(\cdot, \sigma_0 \cdot f^k) &= s * \left(G(\cdot, \sigma_0) * G(\cdot, \sigma_0 \sqrt{f^k - 1}) \right) \\ &= (s * G(\cdot, \sigma_0)) * G(\cdot, \sigma_0 \sqrt{f^k - 1}) \end{aligned}$$

Diese Aufteilung führt dazu, dass der σ-Parameter des Gaußkernels $\sigma_0 \sqrt{f^k - 1}$ im Vergleich zu $2\sigma_0$ im "Normalfall" klein bleibt und man damit bei der Faltung eine kleinere Nachbarschaft mitberücksichtigen muss. Diese Rechenzeitersparnis kommt mit einer weiteren Einschränkung: Wegen einer ähnlichen Überlegung wie bei der Gauß-Pyramide muss der Wert σ_0 so gewählt werden, dass $\sigma_0 \sqrt{f - 1}/f > 1$ gewährleistet ist.

Boxes

Noch schneller lassen sich die Faltungen berechnen, falls der Kernel nur aus wenigen konstanten Abschnitten besteht, wie es bei einem Box-Filter der

Fall ist.

Ein Algorithmus zur Berechnung der Faltung einer Zeitreihe h mit der Box-Funktion

$$b(x,a) := \begin{cases} 1 & -a \leq x \leq a \\ 0 & \text{sonst} \end{cases}$$

könnte wie folgt aussehen:

1. Berechne die Präfixsumme s in $O(n)$:

$$\begin{aligned} s_0 &= h(0) \\ s_i &= s_{i-1} + h(i) \quad 1 \leq i < n \end{aligned}$$

2. Berechne die Faltung $r := h * b$

$$r_i = \sum_{j=i-a}^{i+a} h(i) = s_{i+a} - s_{i-a-1} \qquad a+1 \leq i < a-n$$

was unabhängig von a in $O(n)$ berechnet werden kann! Die Randeffekte spielen dabei keine große Rolle - die Feature-Punkte am Rand können vernachlässigt werden.

Der LoG-Filter wurde von uns bis auf einen konstanten Vorfaktor mit der Boxfunktion (Vergleiche Abschnitt 10.3.4)

$$DD(x, \Delta(\sigma)) := \begin{cases} 0 & x < -3 \cdot \Delta \\ \frac{1}{\Delta} & -3 \cdot \Delta \leq x \leq -\Delta \\ -\frac{2}{\Delta} & -\Delta < x < \Delta \\ \frac{1}{\Delta} & \Delta \leq x \leq 3 \cdot \Delta \\ 0 & 3 \cdot \Delta < x \end{cases}$$

mit $\Delta = round(\frac{9}{10}\sigma)$ approximiert.

Offensichtlich kann DD als eine Linearkombination zweier Boxfunktionen $b(x,a)$ dargestellt werden und damit kann die Faltung mit Hilfe des obigen Algorithmus in $O(n)$ berechnet werden.

Betrachtet man die LoG-, DoG- und Boxes-Filter in Zeit- und Frequenz-Domainen (Abbildung 5.5), so fällt auf, dass der Boxes-Filter keine gute Low-Pass-Filter-Eigenschaften aufweist, was auf Grund seiner scharfen Kanten auch zu erwarten wäre.

Deswegen sorgen wir in unserer Implementierung gesondert für eine Glättung, indem wir die Zeitreihe h vor der Bildung der Präfixsumme s mit der Gaußfunktion $G(\cdot, \sigma_0)$ falten und damit die hohen Frequenzen unterdrücken. Ohne diesen Schritt gäbe es viele lokale Extrema, die weder robust noch charakteristisch und für uns nicht von Interesse sind. Die gesamte Vorgehensweise wird in Abbildung 5.6 zusammengefasst.

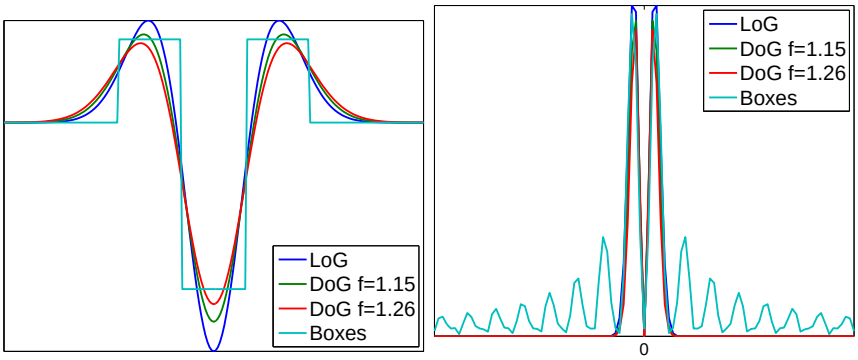

Abbildung 5.5: In der Zeit-Domain (Bild links) kann man keinen großen Unterschied zwischen LoG- und DoG-Filter feststellen. Wobei der DoG-Filter mit der kleineren Schrittweite f eine bessere Approximation des LoG-Filters darstellt. Die Probleme mit der Näherung durch Boxfunktionen wird erst in der Frequenz-Domain (rechts) offensichtlich: Die Low-Pass-Filter-Eigenschaften des LoG gehen weitgehend verloren.

5.2 Suche nach den lokalen Extrema

Im vorherigen Abschnitt wurde beschrieben, wie der diskretisierte Detektor, den wir allgemein als $\mathcal{D}(x, j, o)$ bezeichnen, unabhängig davon, ob es sich um eine Wavelet-Transformation, einen DoG-Filter oder Faltungen mit Boxfunktionen handelt, in Form einer Pyramide berechnet wird. Nun nutzen wir diese Darstellung um lokale Extrema zu finden.

In (x^*, j^*, o^*) befindet sich ein lokales Maximum, falls

$$\mathcal{D}(x^*, j^*, o^*) > \mathcal{D}(x, j, o^*)$$

für die Umgebung $(x, j) \in [x^* - 1,\, x^* + 1] \times [j^* - 1,\, j^* + 1] \setminus \{(x^*, j^*)\}$ gilt.

Analog für ein lokales Minimum: In (x^*, j^*, o^*) befindet sich ein lokales Minimum, falls

$$\mathcal{D}(x^*, j^*, o^*) < \mathcal{D}(x, j, o^*)$$

für die gleiche Umgebung wie beim Maximum erfüllt ist.

Mit diesen Definitionen wird auch die Notwendigkeit der Levels $j = -1$ und $j = k$ in jeder Oktave klar: Ohne sie könnte man nicht alle Skalierungen untersuchen, da man an den Schnittstellen der Oktaven die Information nicht in der passenden Auflösung hätte.

Es ist jedoch wünschenswert, die instablen Extrema schon bei der Suche auszusortieren. Dazu werden zwei Strategien benutzt: die Geräuschschwelle und die Nachbarschaftserweiterung.

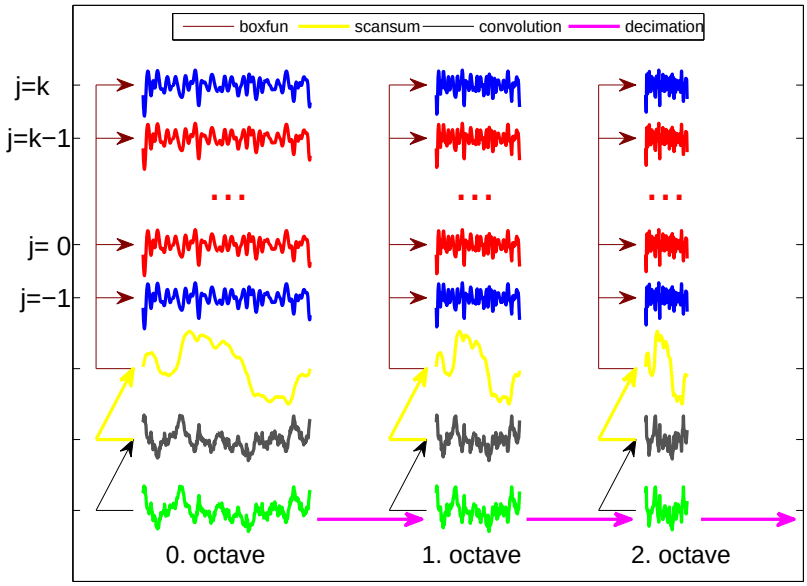

Abbildung 5.6: Unsere Vorgehensweise bei der Berechnung der Boxes-Pyramide: Als erstes wird die Originalzeitreihe (grün) passend geglättet (schwarze Pfeile). Anschließend wird die Präfixsumme (gelbe Pfeile) gebildet. Die Berechnung der Faltung mit den Boxfunktionen (maroonfarbene Pfeile) für $j = -1, \ldots, k$ liefert die Näherungen an die Faltungen mit den LoG-Filtern. Die Auflösung der originalen Zeitreihe wird anschließend halbiert (magentafarbene Pfeile) und der Vorgang wird in der nächsten Oktave wiederholt.

Geräuschschwelle

Kleine, instabile Extrema lassen sich nicht in der leicht gestörten Zeitreihe wiederfinden, unter anderem weil sie auch auf Rundungsfehler bei den Berechnungen mit der Gleitkommaarithmetik basieren können (dies ist vor allem dann der Fall, falls man FFT für die Berechnung der Faltung benutzt). Solche durch das Rauschen bedingte Extrema werden aussortiert (bzw. bleiben an der Geräuschschwelle "hängen"), indem wir verlangen, dass das Extremum um mindestens ϵ besser sein soll, als seine Nachbarschaft. Dabei wird ϵ relativ zum Wert von $\mathcal{D}(x^*, j^*, o^*)$, d.h. $\epsilon = \varepsilon \cdot \mathcal{D}(x^*, j^*, o^*)$ gewählt.

Der Test für die lokalen Extrema ändert sich zu

$$\mathcal{D}(x^*, j^*, o^*) \cdot (1 - \varepsilon) > \mathcal{D}(x, j, o^*) \text{ bzw.}$$
$$\mathcal{D}(x^*, j^*, o^*) \cdot (1 + \varepsilon) < \mathcal{D}(x, j, o^*)$$

für $(x, j) \in [x^* - 1,\, x^* + 1] \times [j^* - 1,\, j^* + 1] \setminus \{(x^*, j^*)\}$.

Die Wahl $\varepsilon = 10^{-4}$ erwies sich in unseren Experimenten als zweckmäßig. Aber auch ein $\varepsilon \in [10^{-5},\, 10^{-3}]$ lieferte vergleichbar gute Ergebnisse.

Nachbarschaftserweiterung

Eine weitere Möglichkeit nur die stabilsten Extrema auszuwählen besteht darin, die Nachbarschaft zu erweitern, in der dieser Wert extrem sein muss. Bis jetzt wurde nur die sogenannte 3-Nachbarschaft (d.h. nur die unmittelbaren Nachbarn) angeschaut, es ist aber möglich (zumindest was die zeitliche Dimension x angeht) die allgemeine $(2d+1)$-Nachbarschaft zu betrachten: Dabei werden die d nächsten Nachbarn auf jeder Seite berücksichtigt.

Für die $(2d+1)$-Nachbarschaft lauten die Bedingungen unter der Berücksichtigung der Geräuschschwelle

$$\mathcal{D}(x^*, j^*, o^*) \cdot (1-\varepsilon) > \mathcal{D}(x, j, o^*) \text{ bzw.}$$
$$\mathcal{D}(x^*, j^*, o^*) \cdot (1+\varepsilon) < \mathcal{D}(x, j, o^*)$$

für $(x, j) \in [x^* - d,\, x^* + d] \times [j^* - 1,\, j^* + 1] \setminus \{(x^*, j^*)\}$.

Während die einfache naive Überprüfung eine $O(d)$ worst-case-Laufzeit pro Element in der Zeitreihe aufweist, existieren Algorithmen, deren worst-case-Verhalten mit $O(1)$ abgeschätzt werden kann (siehe z.B. *Dynamic Block Algorithm* in Neubeck und Gool [53]).

Letztendlich verwenden wir die naive Implementierung, denn dafür sprechen folgende Argumente:

1. Die Laufzeitabschätzung $O(d)$ pro Pixel ist zu pessimistisch. Ohne Abbruch bzw. im worst-case wären $6 * d + 2$ Vergleiche pro Element notwendig, doch es ist nicht der Fall für unsere Anwendungen: Es ist viel wahrscheinlicher, dass die Vergleiche schon nach den ersten 1-2 Versuchen abgebrochen werden können, wonach es klar ist, dass es sich beim untersuchten Element nicht um ein Extremum handelt. Z.B. falls man eine stochastisch unabhängige Verteilung der zugrunde liegenden Werte voraussetzt, werden in $\frac{2}{3}$ aller Fälle nicht mehr als zwei Vergleiche durchgeführt (für eine Begründung siehe die Laufzeitanalyse in Neubeck und Gool [53]).

2. Die Verwendung der Geräuschschwelle erschwert den Einsatz des Dynamic Block Algorithmus. Es ist nicht offensichtlich, wie die Geräuschschwelle berücksichtigt werden kann ohne die Laufzeit des Algorithmus zu beeinträchtigen.

3. Die bei der Suche nach den Extrema verbrachte Rechenzeit beträgt nicht mehr als 10% der gesamten Rechenzeit zur Erstellung der Fingerprints, eine Beschleunigung würde kaum die Gesamtlaufzeit beeinflussen.

4. Der Dynamic Block Algorithmus war in unseren Experimenten für die vorhandenen Daten ungefähr genau so schnell wie die naive Vorgehensweise.

5.2.1 Bestimmung der Subpixel genauen Lage der Extrema

Da die zeitliche Auflösung mit jeder Oktave halbiert wird, gewinnt die Subpixel genaue Bestimmung der Lage der Extrema an Bedeutung.

Subpixel genaue x-Koordinate

Um die genaue x-Koordinate der Extrema zu bestimmen, wird eine Parabel $y = a_x x^2 + b_x x + c$ durch die Punkte

$$(x^* - 1, \mathcal{D}(x^* - 1, j^*, o^*)),\ (x^*, \mathcal{D}(x^*, j^*, o^*)) \text{ und}$$
$$(x^* + 1, \mathcal{D}(x^* + 1, j^*, o^*))$$

interpoliert. Die genaue x-Koordinate ergibt sich als die Lage des Extremums der Parabel

$$x_{ext} = -\frac{b_x}{2a_x}.$$

Oder im ursprünglichen, nicht durch die Auflösungshalbierung betroffenen Koordinatensystem:

$$x_{ext,original} = -\frac{b_x}{2a_x} \cdot 2^{o^*}.$$

Subpixel genaue σ-Koordinate

Eine ähnliche Vorgehensweise wählen wir, um die σ-Koordinate zu bestimmen: Die Parabel $y = a_\sigma \sigma^2 + b_\sigma \sigma + c_\sigma$ wird durch die Stützstellen

$$\left(\sqrt[k]{2}^{j^*-1}, \mathcal{D}(x_{ext}, j^* - 1, o^*)\right),\ \left(\sqrt[k]{2}^{j^*}, \mathcal{D}(x_{exp}, j^*, o^*)\right),$$
$$\left(\sqrt[k]{2}^{j^*+1}, \mathcal{D}(x_opt, j^* + 1, o^*)\right)$$

festgelegt, wobei die Werte $\mathcal{D}(x_{opt}, \cdot, \cdot)$ mit Hilfe der linearen Interpolation (siehe Abschnitt 10.2.1 im Anhang) bestimmt werden.

Dass die Approximation mit der Parabel auch für die σ-Koordinate möglich ist, wurde von uns schon in 3.8.1 (Abbildung 3.14) illustriert. Der so gefundene Wert $\sigma_{ext} = -\frac{b_\sigma}{2a_\sigma}$ muss anschließend in das originale Koordinatensystem mit voller Auflösung via

$$\sigma_{ext,original} = -\sigma_{ext} \cdot 2^{o^*}$$

überführt werden.

Gemeinsame Bestimmung von (x_{ext}, σ_{ext})

Es ist auch denkbar die beiden Koordinaten gleichzeitig zu bestimmen, indem man die Funktion

$$p(x, \sigma) = a_{xx}x^2 + a_{\sigma\sigma}\sigma^2 + a_{x\sigma}x\sigma + b_x x + b_\sigma \sigma + c$$

in der Umgebung von $\left(x^*, \sqrt[k]{2}^{j^*}\right)$ fittet. Nimmt man nur die unmittelbare Nachbarschaft, so sind es 9 Gleichungen mit 6 Unbekannten.

Diese Vorgehensweise brachte jedoch neben einer gesteigerten Laufzeit keinen Gewinn bei der Genauigkeit, sogar umgekehrt - die Performanz des Frameworks verschlechterte sich leicht. Deswegen wurde nur das schnellere zweistufige Verfahren angewandt.

5.3 Abtasten der Umgebung

Die Umgebung des Feature-Punktes, aus welcher später die Deskriptoren berechnet werden, soll skalierungsinvariant sein. Wie schon in Abschnitt 3.8 gesehen, passiert dies am besten, indem man die Umgebung in der Abhängigkeit der j^* abtastet. Mit einer vorgegebenen Schrittweite τ ergibt sich dann für den $(2 \cdot d + 1)$-dimensionalen Umgebungsvektor U aus der Scale-Space-Funktion F:

$$U_k := F(x_{ext} + k \cdot \sigma_{ext} \cdot \tau, j^*, o^*) \qquad -d \leq k \leq d$$

Bei der Wahl von τ sollte man das Abtasttheorem (Anhang 10.3.5) berücksichtigen:

$$f_{abstast} > 2f_{max}$$

Dies ist auch einer der Gründe, warum man zum Abtasten die mit dem passenden Gaußkernel gefaltete Zeitreihe und nicht die originale Zeitreihe verwendet: Die Eigenschaften des Gaußkernels als Tiefpassfilter sorgen dafür, dass die Bedingung aus diesem Theorem erfüllt wird.

Zur Berechnung von $F(x_{ext} + d \cdot \sigma_{ext} \cdot \tau, j^*, o^*)$ wird die lineare Interpolation verwendet, da die x-Koordinate $x_{ext} + d \cdot \sigma_{ext} \cdot \tau$ im Allgemeinen nicht ganzzahlig ist.

5.4 Berechnung des Deskriptors

Die Berechnung der unterschiedlichsten Deskriptoren, solche wie SIFT, PCA und BRIEF, aus der Umgebung U wird ausführlich im Abschnitt 6.1 besprochen, hier soll nur die generische Vorgehensweise erläutert werden.

Es wäre möglich, die skalierungsinvariante Umgebung U direkt zum Vergleich zu benutzen, doch die Umrechnung in einen Deskriptor $D(U)$ bringt einige Vorteile mit sich:

- Eine Reduktion der Dimension (SIFT, PCA) bzw. des Speicherbedarfs (BRIEF).
- Normalisierung und die damit verbundene Invarianz bezüglich einer additiven Konstante und einer Skalierung der Werte.
- Erhöhte Robustheit gegenüber Störungen und Messfehlern.

Handhabung der Orientierung

Es ist offensichtlich, welcher Zusammenhang zwischen der Umgebung U mit der Umgebung in der gespiegelten Zeitreihe U^s besteht:

$$U_i^s = U_{-i} \quad -d \leq i \leq d$$

Es stellt sich nun die Frage, wie man die Deskriptoren vergleichen kann, falls die Koordinatentransformation eine Spiegelung beinhalten soll. Uns stehen vier Möglichkeiten zur Auswahl:

- Der Deskriptor ist invariant unter Spiegelungen.
- Festlegung einer lokalen Orientierung.
- Es werden je ein Deskriptor aus U und U^s berechnet, im Speicher gehalten und nach Bedarf verwendet (wie z.B. PCA- und BRIEF-Deskriptoren).
- Der Deskriptor aus $D(U^s)$ lässt sich schnell aus dem Deskriptor $D(U)$ berechnen, es reicht also, nur eine Version im Speicher zu behalten (wie z.B. CC- und SIFT-Deskriptoren).

Die spiegelungsinvarianten Deskriptoren bringen den großen Vorteil mit sich, dass man die beiden Orientierungen der Zeitreihen zueinander mit einer Nearest-Neighbor-Anfrage gleichzeitig abhandeln kann. Der Nachteil besteht darin, dass man bei einer gefundenen Zuordnung $\Big((x^i, \sigma_x^i) \mapsto (y^i, \sigma_y^i), \varsigma\Big)$ nicht mehr aus ς (immer $= 1$) auf die Ausrichtung/Spiegelung der Koordinatensysteme beider zu vergleichenden Zeitreihen zurückschließen kann.

Die Festlegung der lokalen Orientierung ist ein naheliegender Weg, Deskriptoren, die nicht spiegelungsinvariant sind, in solche zu verwandeln. Wie jedoch im Abschnitt 6.4 gezeigt wird, muss man Einbußen bei der Performanz in Kauf nehmen - der Preis für mögliche Fehler bei der Festlegung der lokalen Orientierung.

Für SIFT- und CC-Deskriptoren lässt sich $D(U)$ in $D(U^s)$ durch eine Permutation der Einträge überführen (für mehr Details siehe 6.4). Dies macht es möglich, nur den Deskriptor $D(U)$ abzuspeichern und $D(U^s)$ *on the fly* zu berechnen.

5.5 Nearest-Neighbor-Anfrage

Die Funktionsweise unterschiedlicher Datenstrukturen werden im Abschnitt 6.7 ausführlich besprochen. Hier möchten wir auf die von uns letztendlich verwendete naive lineare Indexstruktur eingehen, die uns die Möglichkeit gibt, zum einen mehrere Vergleiche aufgrund der σ-Koordinate effizient auszuschließen und zum anderen den Vorgang frühzeitig abzubrechen.

Dazu liegt der Fingerprint als zwei nach der σ-Koordinate sortierte Arrays von Features vor. In einem Array befinden sich diejenigen Features, die als ein lokales Maximum detektiert wurden, im zweiten - diejenigen, die als ein Minimum detektiert wurden. Diese Aufteilung bringt einen Speedup-Faktor von 2, weil es nur sinnvoll ist, Minima mit Minima und Maxima mit Maxima zu vergleichen.

Oft kann die Skalierung der uns interessierenden Koordinatentransformationen eingeschränkt werden: So können wir z.B. in den von uns untersuchten Daten davon ausgehen, dass die Bedingung $\frac{2}{3} \leq |\alpha| \leq \frac{3}{2}$ von den Koordinatentransformationen $T_{(\alpha,\beta)}$ erfüllt wird. Dieses Vorwissen kann ausgenutzt werden, um die Anzahl der notwendigen Deskriptoren-Vergleiche zu reduzieren: Bei der Suche nach der richtigen Zuordnung eines Feature-Punktes mit σ-Koordinate $\hat{\sigma}$ betrachten wir nur diejenigen Features, deren σ-Koordinaten die Bedingung

$$\frac{2}{3}\hat{\sigma} \leq \sigma \leq \frac{3}{2}\hat{\sigma}$$

erfüllen. Diese in Frage kommenden Feature-Punkte stehen hintereinander im Feature-Array, weil dieses ja nach der σ-Koordinate sortiert wurde. Die Grenzen des Blocks können mit binärer Suche in logarithmischer Zeit gefunden werden.

Beim sequentiellen Abarbeiten der Features kann die Information über den bisher besten Treffer ausgenutzt werden, um den Vergleich der Deskriptoren früher abzubrechen. Z.B. falls schon nach der Berücksichtigung der ersten Dimension der Abstand der Deskriptoren größer ist als der Gesamtabstand bei dem bisher besten Treffer, so interessieren wir uns nicht wirklich für den genauen Abstand, sondern brechen die Berechnung einfach ab. Im Abschnitt 6.7.5 wurde diese Strategie als ausschlaggebend für die überlegene Performanz der linearen Datenstruktur ausgemacht.

5.6 Aussortieren der falschen Zuordnungen

Die Suche nach den nächsten Nachbarn anhand der Deskriptoren liefert nicht immer die richtigen Zuordnungen. Es ist sogar öfter der Fall, dass die Mehrheit der gefundenen Zuordnungen falsch ist und damit ähnelt die Suche nach den richtigen Zuordnungen, der Suche nach einer Nadel im sprichwörtlichen Heuhaufen.

Wir benutzen einige Heuristiken, um schnell die falschen Zuordnungen auszuschließen:

1. Falls der Abstand zwischen den Deskriptoren einer Zuordnung einen Schwellwert δ überschreitet, so werden die Zuordnungen direkt als falsch eingestuft. Dieser Schwellwert kann direkt als Abbruchkriterium bei der Suche mit dem linearen Index eingesetzt werden, um die Berechnungen zu beschleunigen.

2. Wir werten eine Zuordnung zwischen einem Deskriptor c aus einem Fingerprint C und einem Deskriptor p aus einem Fingerprint P als richtig, falls

$$\begin{aligned} &abstand(c,p) < abstand(c,d) \quad \forall d \in P \setminus \{p\} \quad \text{und} \\ &abstand(c,p) < abstand(d,p) \quad \forall d \in C \setminus \{c\} \end{aligned} \tag{5.2}$$

   gelten. Anders ausgedruckt: Nur wenn die beiden Deskriptoren sich gegenseitig für die beste Zuordnung halten, kann es sich um eine richtige Zuordnung handeln.

   Diese Vorgehensweise hat noch die angenehme "Nebenwirkung", dass der Vergleich des Fingerprints C mit dem Fingerprint P das gleiche Ergebnis liefert, wie der Vergleich des Fingerprints P mit dem Fingerprint C.

Nun widmen wir uns fortgeschritteneren Heuristiken und starten mit der Vorstellung des RANSAC-Verfahrens und der Hough-Transformation.

5.6.1 RANSAC und Hough-Transformation

Wir betrachten hier das RANSAC-Verfahren und die Hough-Transformation für T_{1d}. Für eine allgemeinere Behandlung siehe Fischler und Bolles [27] und Duda und Hart [25].

Unser Szenario (Abbildung 5.7) kann leicht als Suche nach einer Geraden in einer Ebene dargestellt werden:

- Gegeben sind n Punkte (x_i, y_i) mit $i = 1, \cdots, n$ auf einer Ebene, welche die gefunden Zuordnungen z_i zwischen den Features darstellen

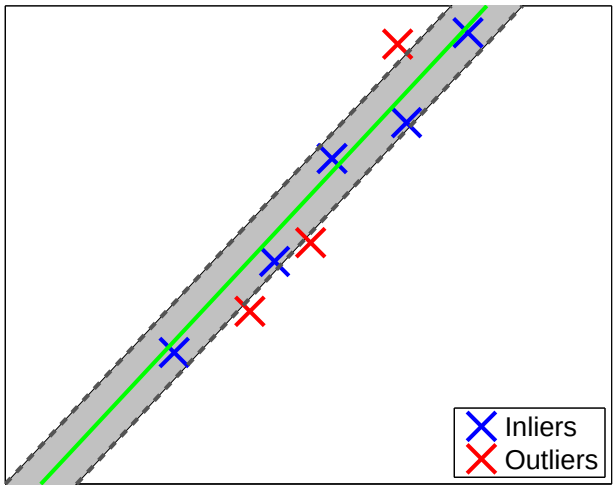

Abbildung 5.7: Die grüne Gerade und ihre graue Umgebung, die dem Abstand Δ zur Geraden entspricht, unterteilen die Punkte in die blauen Inliers*, die innerhalb der grauen Fläche liegen, und in die roten* Outliers*, die sich zu weit weg von der grünen Geraden befinden.*

(zumindest deren x-Koordinaten). Durch die Koordinatentransformation $T_{(\alpha\,,\beta)}$ werden in dieser Ebene Geraden $y = \alpha x + \beta$ indiziert. Dies ermöglicht uns, den Abstand zwischen einer Zuordnung z_i (repräsentiert durch den Punkt (x_i, y_i)) und einer Koordinatentransformation $T_{(\alpha\,,\beta)}$ (repräsentiert durch die Gerade $y = \alpha x + \beta$) zu definieren:

$$d(z_i, T_{(\alpha\,,\beta)}) := d((x_i, y_i), T_{(\alpha\,,\beta)}) := \min_{x \in \mathbb{R}} \left\| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x \\ \alpha x + \beta \end{pmatrix} \right\|_2 .$$

Offensichtlich lässt sich der Abstand via

$$d(z_i, T_{(\alpha\,,\beta)}) = |y_i - \alpha \cdot x_i - \beta| \, \frac{1}{\sqrt{1 + \alpha^2}}.$$

ausrechnen.

- Des Weiteren ist ein Fehlerschwellwert Δ vorgegeben und füllt das Zeichen $\approx$ in der Gleichung

$$y \approx T_{(\alpha\,,\beta)}(x)$$

mit Bedeutung dank folgender Definition:

$$y \approx T_{(\alpha\,,\beta)}(x) \Leftrightarrow d((x, y), T_{(\alpha\,,\beta)}) \leq \Delta. \tag{5.3}$$

Das bedeutet wir lassen kleinere Abweichungen von der Transformationsvorschrift bis hin zur Größe Δ zu. Alle größeren Abweichungen werden als Ausreißer interpretiert und nicht berücksichtigt.

- Als *Inliers* einer Geraden $T_{(\alpha\,,\beta)}$ werden all diejenigen Zuordnungen (x_i, y_i) bezeichnet, die nicht weiter als der vorgegebene Schwellwert Δ von der Geraden entfernt sind:

$$In\left(T_{(\alpha\,,\beta)}\right) := \left\{z_i \,|\, y \approx T_{(\alpha\,,\beta)}\right\}$$

- Gesucht ist: Eine Koordinatentransformation $T_{(\alpha^*\,,\beta^*)}$, die durch die meisten Inliers $In\left(T_{(\alpha^*\,,\beta^*)}\right)$ untermauert wird. Natürlich wird diese Koordinatentransformation in den meisten Fällen nicht eindeutig sein, all diese Transformationen werden aber (so ist die Hoffnung) ähnlich aussehen.

RANSAC

Als erstes stellen wir die deterministische Version von RANSAC vor (was zugegebenermaßen ein Widerspruch zum Namen des Verfahrens *RANdom SAmple Consensus* darstellt).

Für einen vorgegebenen Schwellwert Δ führe folgende Schritte aus:

1. Bestimme für alle Paare $\{i, j\}$ die durch die Punkte (x_i, y_i) und (x_j, y_j) festgelegte Gerade $y = \alpha_{ij}x + \beta_{ij}$.

2. Bestimme dasjenige Paar $\{i, j\}$, das für diejenige Gerade verantwortlich ist, die die meisten Inliers besitzt.

Diese Version weist kubische Laufzeit auf, denn, wenn mit n die Anzahl der Punkte auf der Ebene bezeichnet wird, so

1. müssen $\binom{n}{2}$ Paare bzw. Geraden untersucht werden,

2. nimmt die Suche der Inliers einer Geraden $O(n)$ Zeit in Anspruch.

An dieser Stelle sollte hinzugefügt werden, dass dieser Algorithmus nicht unbedingt das beschriebene Problem löst, denn es werden nur diejenigen Geraden betrachtet, die durch ein Punktepaar gehen. Diese Geraden stellen nicht unbedingt die bestmögliche Wahl dar (Abbildung 5.8). Diese Bedenken werden jedoch oft vernachlässigt - in der Praxis reichen die Genauigkeiten der gelieferten Ergebnisse aus und man gibt sich mit einer Näherung zufrieden.

Eine randomisierte Wahl der Punktepaare $\{i, j\}$ stellt eine Möglichkeit dar, die Laufzeit des Algorithmus zu reduzieren. Dabei werden nicht alle möglichen Paare ausprobiert, sondern nur eine Untermenge $\mathcal{T}$, die zufällig ausgewählt wird. Es stellt sich die Frage, wie groß die ausgewählte Menge $\mathcal{T}$

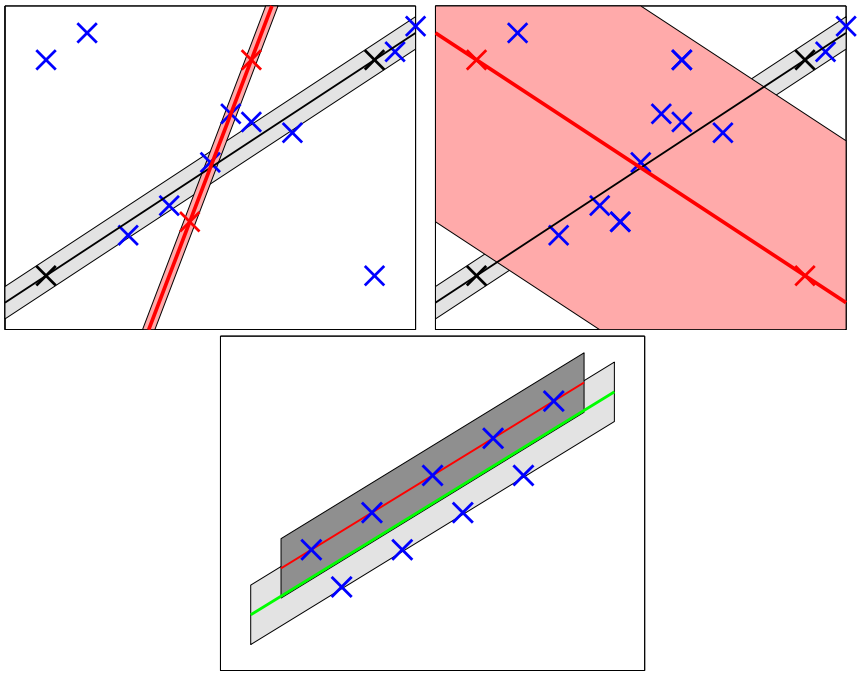

Abbildung 5.8: Links oben: Der Parameter Δ wurde zu klein gewählt, deswegen wird nicht die richtige Lösung (die schwarze Gerade), sondern eine falsche (rot) zurückgeliefert. Rechts oben: Der Parameter Δ wurde zu groß gewählt, dadurch wird wieder eine falsche (rote) Lösung zurückgeliefert. Unten: RANSAC liefert die rote Gerade, während die Grüne richtig gewesen wäre, denn sie wäre von allen 7 Punkten untermauert.

sein muss, um mit hoher Wahrscheinlichkeit die richtige Transformation zu finden.

Für unsere weiteren Überlegungen machen wir folgende Annahmen:

1. Um eine richtige Transformation zu finden, reicht es, ein Paar von richtigen Zuordnungen zu finden bzw. zu untersuchen.

2. Der Anteil der richtigen Zuordnungen beträgt ρ. Damit beträgt die Wahrscheinlichkeit, dass man ein Paar der richtigen Zuordnungen auswählt, ρ^2.

3. Einfachheitshalber suchen wir ein Paar aus, unabhängig davon, ob es schon gewählt bzw. untersucht wurde. Falls $|\mathcal{T}| << n^2$, so bringt diese vereinfachte Vorgehensweise keine Nachteile, weil die Wahrscheinlichkeit, ein Paar der Zuordnungen doppelt zu betrachten, recht klein ist und der Aufwand die schon untersuchten Paare zu merken sich nicht lohnen würde.

4. Es sollte gewährleistet werden, dass die Wahrscheinlichkeit, dass man nicht die richtige Transformation findet, weniger als $\alpha = 10^{-9}$ beträgt.

Offensichtlich beträgt die Wahrscheinlichkeit, dass man die richtige Transformation nicht findet

$$P_f = (1 - \rho^2)^{|\mathcal{T}|} \overset{!}{\leq} \alpha$$

und damit

$$|\mathcal{T}| \geq \frac{\log(\alpha)}{\log(1 - \rho^2)}.$$

Sollten wir annehmen, dass ρ mindestens 10% beträgt, so reichen uns die Untersuchungen von 2100 zufällig gewählten Zuordnungenpaaren, um die Vorgaben zu erfüllen.

Damit bleibt die Laufzeit bei $O(n)$, obwohl die Konstante recht groß ist.

Es bleibt noch anzumerken, dass die Wahl des Parameters Δ mit Bedacht erfolgen sollte (Abbildung 5.8): Wählt man Δ zu groß, so werden viele nicht aussortierte falsche Zuordnungen bzw. Ausreißer für falsche Ergebnisse sorgen, bei einem zu kleinen Wert kann es vorkommen, dass zwischen den falschen und der "richtigen" Geraden kein Unterschied festgestellt werden kann.

Die Hough-Transformation

Jedes Paar der Zuordnungen definiert eine Gerade $y = \alpha \cdot x + \beta$, die durch diese Punkte verläuft und damit auch einen Punkt (α, β) im Raum der Parameter. Intuitiv ist folgende Schlussfolgerung:

Die größte Anhäufung der Punkte (α, β) im Parameterraum entspricht der gesuchten Koordinatentransformation.

Um diese Anhäufung zu finden, unterteilen wir den Parameterraum in Zellen, auf welche die in den Parameterraum transformierten Zuordnungspaare $\{z_i, z_j\}$ aufgrund der Koordinaten $(\alpha_{ij}, \beta_{ij})$ verteilt werden. Anschließend wird diejenige Zelle gewählt, die die meisten Paare beinhaltet. Siehe Abbildung 5.9 für ein Beispiel.

Die Laufzeit dieses Algorithmus beträgt $O(n^2)$, da man alle möglichen Paare der Zuordnungen anschaut.

Es gibt zwei Nachteile dieser naiven Vorgehensweise im Vergleich zum langsameren RANSAC-Algorithmus:

1. Die Wahl der Größe der Zellen ist nicht so intuitiv, wie die Wahl des Parameters Δ beim RANSAC. Die zu feine Auflösung kann dazu führen, dass in einer Zelle nicht mehr als nur ein Punkt landet und man keine eindeutigen stabilen Ergebnisse bekommt. Es ist auch nicht klar,

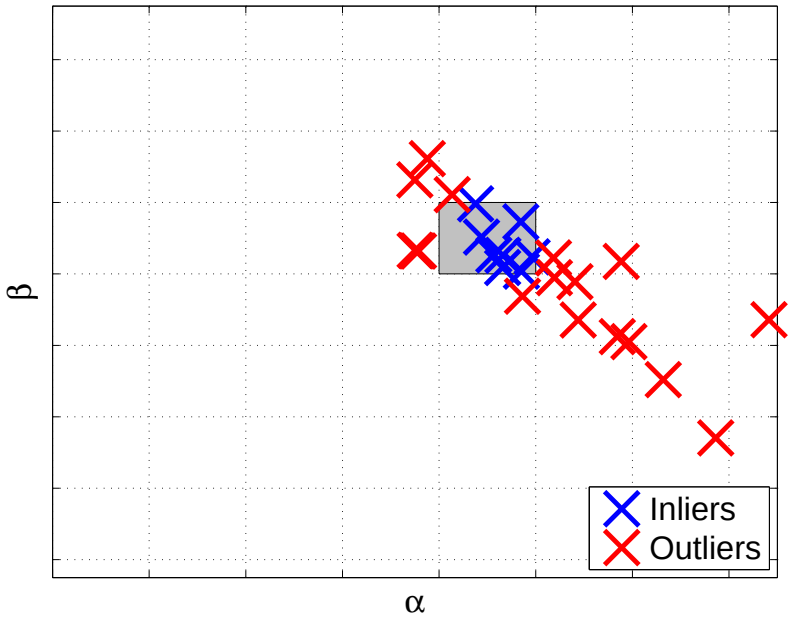

Abbildung 5.9: 28 *mögliche Paare der Zuordnungen* $\{i, j\}$ *ergeben Punkte* $(\alpha_{ij}, \beta_{ij})$ *im Parameterraum.* 14 *blau von insgesamt* 28 *Punkten befinden sich in der grau markierten Zelle, deren Mitte somit die gefundene Koordinatentransformation darstellt. Die roten Punkte* (α, β) *sind zerstreut auf die unterschiedlichen Zellen und basieren auf Zuordnungen, von denen mindestens eine falsch ist.*

dass eine kartesische Aufteilung des Raumes die besten Ergebnisse bringt.

2. Liegt der wahre Wert nahe an der Grenze einer Zelle, so werden die "wirklichen" Inliers auf mehrere Nachbarzellen verteilt. Dies erhöht die Wahrscheinlichkeit, dass eine falsche Zelle und damit eine falsche Transformation als die "beste" gewählt wird.

Im nächsten Abschnitt besprechen wir Möglichkeiten, diese Probleme zu umgehen.

5.6.2 1D-Hough-Transformation in $O(n)$

Die gefundenen Zuordnungen können in zwei Mengen unterteilt werden $\varsigma = 1$ und $\varsigma = -1$:

$$\mathcal{Z}_\varsigma := \{z_i \in \mathcal{Z} \,|\, \varsigma_i = \varsigma\},$$
$$\mathcal{Z} = \mathcal{Z}_{-1} \dot\cup \mathcal{Z}_1$$

Im weiteren Verlauf des Algorithmus werden wir $\mathcal{Z}_{-1}$ und $\mathcal{Z}_1$ einzeln betrachten.

Bestimmung von $|\alpha|$

Durch die Zuordnung der Feature-Punkte zueinander werden neben den zeitlichen Koordinaten x und y auch die Skalierungskoordinaten σ_x und σ_y einander zugeordnet. Aus der Formel (3.2) - Skalierungsinvarianz der Scale-Space-Funktion - wird ersichtlich: Falls $y \approx T_{(\alpha,\beta)}(x)$, so gilt auch $\sigma_y \approx |\alpha| \cdot \sigma_x$. Diese Erkenntnis legt es nahe, eine eindimensionale Hough-Transformation zu benutzen, um den Faktor $|\alpha|$ zu bestimmen.

Dafür werden $\log \frac{\sigma_y^i}{\sigma_x^i}$ in die Zellen c_m der Breite $b := \log f = \frac{1}{k} \log 2$ eingeteilt (mit f bzw. k wie sie bei der Berechnung der Pyramide (Abschnitt 5.1) gewählt wurden):

$$c_m = [b \cdot m, b \cdot (m+1)) \quad -k \cdot o \leq m < k \cdot o,$$

wobei mit o die Anzahl der Oktaven der Pyramide bezeichnet wird.. An dieser Stelle nutzen wir unser Wissen aus, dass der Faktor zwischen den gefundenen Features nicht 2^o übersteigen kann, da nur dieser Unterschied von der Pyramide abgedeckt werden kann.

Die Zellen können also in einem Array abgespeichert werden. Würde sich die Lage der Zellen, die von Interesse sind, nicht abschätzen lassen, so könnte man an der Stelle eine Hashtabelle benutzen.

Nach der Einteilung liegt uns ein Vektor C vor, der die Anzahl der Elemente in jeder Zelle beinhaltet:

$$C_m := \left| \left\{ i \mid \log \frac{\sigma_y^i}{\sigma_x^i} \in c_m \right\} \right|.$$

Anschließend suchen wir dasjenige m^*, so dass

$$C_{m^*} + C_{m^*+1} \geq C_m + C_{m+1} \quad \forall k \cdot o \leq m < k \cdot o - 1.$$

Der Hauptvorteil dieser Vorgehensweise besteht darin, dass das schon angesprochene Problem mit der Häufung der Punkte an der Grenze einer Zelle umgangen wird: Denn man betrachtet im Prinzip zwei Gitter mit der Breite der Zellen $2 \cdot b$, die um b relativ zu einander verschoben sind (siehe Abbildung 5.10).

Bestimmung von β

Nachdem $|\alpha|$ bestimmt wurde, kennen wir alle Parameter bis auf β:

$$\beta = y_i - \varsigma \cdot |\alpha| x_i,$$

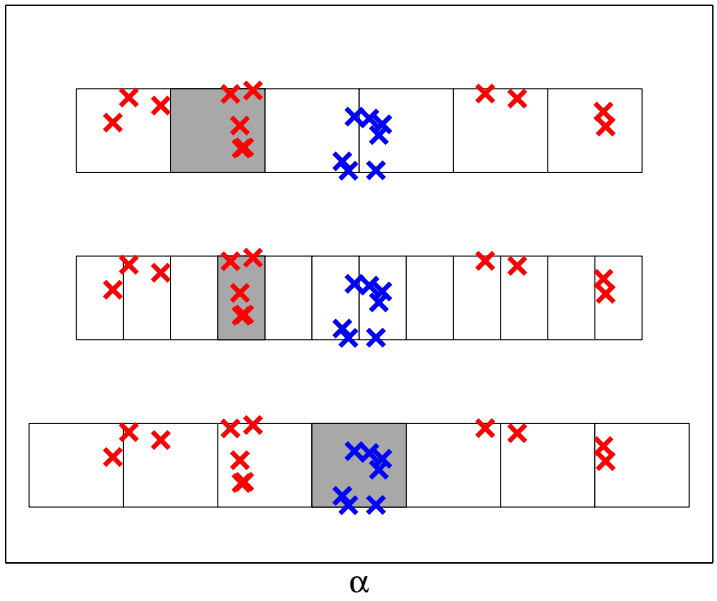

Abbildung 5.10: In der Mitte befindet sich die größte Häufung der Punkte (blau). Beim unglücklich gewählten Gitter in der oberen Zeile befindet sich diese Häufung an der Grenze zwischen zwei Zellen, so dass die Zelle mit den meisten Punkten (grau) die zweite Zelle von links ist, die zufälligerweise auch viele rote Treffer verzeichnet - das möchte man jedoch vermeiden! Halbiert man jede Zelle und kombiniert die Nachbarn neu, so entsteht das untere Gitter und alle blauen Punkte befinden sich in einer Zelle (diesmal grau, weil sie nun die meisten Treffer verzeichnet)! Man überzeugt sich schnell: Entweder das obige oder das untere um die halbe Breite verschobene Gitter passen gut. Die beiden Gitter können wir durch zwei unterschiedliche Kombinationen der halbierten Zellen erzeugen.

dabei werden natürlich nur diejenigen Zuordnungen berücksichtigt, die den richtigen Wert von $|\alpha|$ untermauern.

Doch nun können wir ähnlich wie bei der Bestimmung von $|\alpha|$ die eindimensionale Hough-Transformation benutzen, um β zu bestimmen.

Es hat sich herausgestellt, dass die besten Ergebnisse erreicht werden, falls die Breite der Hough-Zelle abhängig von den Längen der Zeitreihen und der Steigung α gewählt wird.

Dazu werden die Koordinaten x_i und y_i auf das Intervall [0, 1] skaliert:

$$x_i' := \frac{x_i}{x_{max}}$$
$$y_i' := \frac{y_i}{y_{max}}$$

mit x_{max} - die Länge der ersten Zeitreihe und y_{max}- die Länge der zweiten Zeitreihe. Dadurch ändern sich die α und β Parameter der Geraden im neuen

Koordinatensystem zu:

$$\alpha' := \alpha \cdot \frac{x_{max}}{y_{max}},$$
$$\beta' := \frac{\beta}{y_{max}}.$$

Wie schon erwähnt, soll die Breite der Zellen ausreichend groß sein, sonst sind die Ergebnisse der Hough-Transformation zu instabil. Für große $|\alpha|$-Werte führen schon kleine Abweichungen von der Geraden zu großen Abweichungen des β-Parameters. Deswegen wird die Breite der Zelle bei der Hough-Transformation auch von der Steigung α abhängig gewählt (siehe Abbildung 5.11)

$$b' := b_s \cdot \sqrt{1 + \alpha'^2} \tag{5.4}$$

mit b_s - die Breite der Zelle für den Fall $\alpha' = 0$. Bei unseren Experimenten hat sich der Wert $\beta_s \approx 0.01$ als zweckmäßig herausgestellt.

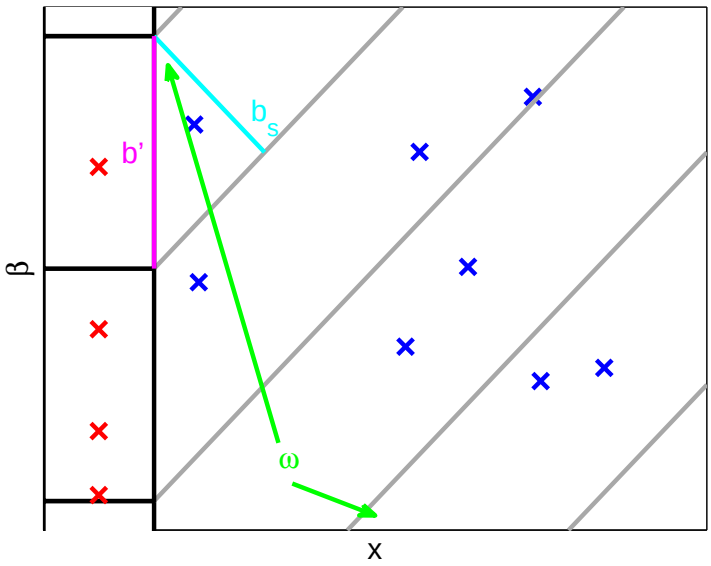

Abbildung 5.11: Die Zuordnungen (blaue Punkte) werden parallel zur Geraden $y = \alpha \cdot x$ auf die y-Achse projiziert und im Raum der Parameter in die Zellen aufgeteilt (rote Punkte). Die Breite der Zelle im Parameterraum ist $b' = \frac{b_s}{\cos\omega}$. Und wegen $\tan\omega = \alpha'$ und der Identität $\frac{1}{\cos^2\omega} = 1 + \tan^2\omega$ ergibt sich die Vorschrift (5.4).

Als letztes stellt sich die Frage: Welche Zellen im Parameterraum können belegt werden bzw. welche Werte kann β' annehmen? Offensichtlich werden die extremen Werte erreicht, falls sich die Zuordnung (x'_e, y'_e) in einer der Ecken

$$Ecken = \{(0,0),\ (0,1),\ (1,1),\ (1,0)\}$$

befindet. Dies führt zu den folgenden möglichen $\beta' = y'_e - \alpha \cdot x'_e$ Parametern:

$$\beta'(Ecken) = \{0,\, 1,\, 1 - \alpha',\, -\alpha'\}.$$

Damit müssen die Zellen im Parameterraum nur den Bereich

$$[\min \beta'(Ecken),\, \max \beta'(Ecken)]$$

abdecken.

5.7 Bestimmung des Alignments

Unsere Experimente haben gezeigt, dass die Bestimmung von α aufgrund der Statistik über $\frac{\sigma_x}{\sigma_y}$ nicht die ausreichende Genauigkeit aufweist. Deswegen nutzen wir die bei der Bestimmung von $|\alpha|$ und β gefundenen und als richtig eingestuften Zuordnungen, um die Koordinatentransformation genauer zu berechnen.

Dafür werden folgende Vorgehensweisen untersucht:

1. Anwendung des (deterministischen) RANSAC-Verfahrens, um immer noch möglicherweise vorhandene Ausreißer zu eliminieren. Anschließend wird Methode 2 oder 3 angewandt.
2. Fitten einer Ausgleichgeraden, welche die Fehler sowohl auf der x- als auch auf der y-Achse berücksichtigt.
3. Fitten einer Ausgleichgeraden, die nur die Fehler auf der y-Achse berücksichtigt bzw. die Werte auf der x-Achse als fehlerlos ansieht.

Wir konnten keine nennenswerte Verbesserung bei der Benutzung der rechenintensiveren Methode 2 gegenüber der Methode 3 feststellen, deswegen wird auch nur die einfachere Variante der Berechnung der Ausgleichgeraden benutzt.

Die Einschaltung des RANASAC-Verfahrens führte zu einer spürbaren Verbesserung der Ergebnisse für die Fälle, in denen die Anzahl der richtigen Zuordnungen klein war. Ein möglicher Ausreißer hatte hier viel dramatischere Auswirkungen als bei vielen vorhandenen Zuordnungen. Deswegen werden die Zuordnungen zusätzlich mit Hilfe von RANSAC von möglichen Ausreißern “gereinigt”, falls weniger als 50 Zuordnungen nach den beiden durchgeführten Hough-Transformationen übrig geblieben sind.

Einfaches Fitten einer Ausgleichgeraden

Bei dem von uns angewandten Verfahren wird davon ausgegangen, dass die x-Koordinaten genau sind bzw. die Fehler auf der x-Achse vernachlässigbar

sind. Es wird also

$$T_{(\alpha^*,\,\beta^*)} := \underset{T \in T_{1d}}{\operatorname{argmin}} \sum_i |y_i - T(x_i)|^2 \tag{5.5}$$

gesucht. Die Transformation $T(x)$ kann als ein Skalarprodukt aufgefasst werden

$$T(x) = (x,\ 1) \cdot \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

und somit lässt sich die Gleichung (5.5) zu

$$T_{(\alpha^*,\,\beta^*)} = \underset{T \in T_{1d}}{\operatorname{argmin}} \left\| y - A(x) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \right\|_2$$

mit $y = (y_1, \ldots, y_n)^t$, $x = (x_1, \ldots, x_n)$ und

$$A(x) := \begin{pmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}$$

umschreiben. Dies ist ein bekanntes lineares Ausgleichproblem (z.B. Hanke-Bourgeois [34]) mit der Lösung:

$$\begin{pmatrix} \alpha^* \\ \beta^* \end{pmatrix} = \left(A^t A\right)^{-1} A^t y,$$

Dabei gehen wir davon aus, dass die Matrix $A^t A$ vollen Rang hat, was immer dann der Fall ist, wenn es mindestens zwei Punkte i und j mit $x_i \neq x_j$ gibt.

5.8 Bestimmung der Anzahl der Inliers

Die Anzahl der Inliers z_i, d.h. der Zuordnungen, welche die gefundene Transformation $T_{(\alpha,\beta)}$ untermauern bzw. für die

$$y_i \approx T_{(\alpha\,,\,\beta)}(x_i),$$

gilt, wird als ein Maß dafür benutzt, wie wahrscheinlich es ist, dass die beiden Zeitreihen zusammenpassen.

Es gibt zwei Möglichkeiten die Relation $\approx$ zu definieren:

1. absolut
$$y_i \approx T_{(\alpha\,,\,\beta)}(x_i) \Leftrightarrow d((x_i, y_i)(T_{(\alpha,\beta)})) \leq \Delta$$

2. relativ zu den Längen der Zeitreihen:

$$y_i \approx T_{(\alpha\,,\beta)}(x_i) \Leftrightarrow d((x'_i, y'_i)(T_{(\alpha',\beta')})) \leq \Delta'$$

Dazwischen besteht folgender Zusammenhang:

$$\begin{aligned} d((x'_i, y'_i)(T_{(\alpha',\beta')})) &= |y'_i - \alpha' \cdot x'_i - \beta'| \, \frac{1}{\sqrt{1+\alpha'^2}} \\ &= \frac{|y_i - \alpha x_i - \beta|}{y_{max}} \cdot \frac{y_{max}}{\sqrt{y_{max}^2 + \alpha \cdot x_{max}^2}} \\ &= \frac{|y_i - \alpha x_i - \beta|}{\sqrt{1+\alpha^2}} \cdot \frac{\sqrt{1+\alpha^2}}{\sqrt{y_{max}^2 + \alpha \cdot x_{max}^2}} \\ &= d((x_i, y_i)(T_{(\alpha,\beta)})) \cdot \frac{\sqrt{1+\alpha^2}}{\sqrt{y_{max}^2 + \alpha \cdot x_{max}^2}} \end{aligned}$$

Der Schwellwert Δ legt fest, welche Abweichung von der Transformationsvorschrift aufgrund der vielen möglichen Fehlerquellen noch akzeptiert werden. Dieser Wert hängt von der Datenqualität ab. In unseren Experimenten hat sich gezeigt, dass die relative Definition mit $\Delta' = 0.005$ gut und zuverlässig für alle von uns betrachteten Testdatensätze funktioniert.

Kapitel 6

Analyse der Bausteine

Bis jetzt wurde die Beschreibung unseres Frameworks allgemein gehalten. In diesem Kapitel soll sich dies nun ändern: Die drei großen Bausteine des Algorithmus

- Deskriptor,
- Feature-Detektor,
- Fingerprint-Datenstruktur

werden systematisch mit dem Ziel untersucht, die beste und effizienteste Möglichkeit für den jeweiligen Baustein zu bestimmen.

6.1 Eindimensionale Deskriptoren

In diesem Abschnitt untersuchen wir diejenigen eindimensionalen Deskriptoren, die durch verwandte zweidimensionale Deskriptoren inspiriert wurden. Sie alle werden aus der Umgebung eines Feature-Punktes mit Koordinaten (x_0, σ_0) berechnet. Die Umgebung liegt als ein $2 \cdot n + 1$-dimensionaler Vektor $U \in \mathbb{R}^{2 \cdot n+1}$ vor. Dabei wird U aus der Scale-Space-Funktion $F(x, \sigma)$ für ein bestimmtes σ_0 wie folgt berechnet:

$$U_i := F(x_0 + (i - n) \cdot \tau(\sigma_0), \sigma_0) \quad i = 0, \ldots, 2 \cdot n,$$

die Länge des Schrittes τ variiert mit σ_0, womit die Skalierungsinvarianz der Umgebung gewährleistet wird (siehe auch Abschnitt 3.8).

Es ist auch möglich, die Umgebung nicht aus $F(x, \sigma)$ sondern aus $F_x(x, \sigma)$ zu berechnen. Es würde sich dabei um einen Deskriptor aus der ersten Ableitung handeln. Ein Vorteil solcher Vorgehensweise wäre, dass solche De-

skriptoren nach einer Normalisierung invariant gegenüber den Trends in der zugrunde liegenden Zeitreihen wären.

In dem Fall bezeichnen wir den $2 \cdot n$-dimensionalen Vektor mit D:

$$D_i := F_x(x_0 + (i - n + 0.5) \cdot \tau(\sigma_0), \sigma_0) \quad i = 0, \ldots, 2 \cdot n - 1.$$

Zuerst sollen diese Deskriptoren vorstellt werden. Ihre Performanz wird in 6.3 untersucht und bewertet.

6.1.1 Cross - Correlation basierte Deskriptoren

Bei der Analyse von Zeitreihen ist die Cross-Correlation zweier gleichlanger Messreihen ein erprobtes Werkzeug und wurde von uns schon im Abschnitt 3.1.2 vorgestellt:

$$\rho(U^1, U^2) := \frac{1}{N} \sum_i \frac{(U_i^1 - \mu_1) \cdot (U_i^2 - \mu_2)}{\sigma_1 \cdot \sigma_2},$$

wobei σ_i - die Standardabweichung, μ_i - der Mittelwert und N- die Dimension von U^i .

Um aus einer Menge der Messreihen $\mathcal{U} := \{U^j\}$, die Messreihe zu wählen, die zu der Messreihe U^* am besten passt, maximiert man die Cross-Correlation $\rho(U^j, U^*)$.

Benutzt man die mittelwertbereinigten und normierten Vektoren

$$\hat{U}^j := \frac{U^j - \mu_j}{\sqrt{N}\sigma_j} \quad j = 1, 2,$$

so lässt sich die Cross-Correlation als ihr Skalarprodukt auffassen:

$$\rho(U^1, U^2) = \langle \hat{U}^1, \hat{U}^2 \rangle.$$

Im weiteren Verlauf konzentrieren wir uns auf die normierten Vektoren $\hat{U}^j$. Es ist offensichtlich, dass

$$\langle \hat{U}, \hat{U} \rangle = \frac{1}{N} \sum_i \frac{(U_i - \mu_i)^2}{\sigma^2} = \frac{\sigma^2}{\sigma^2} = 1$$

gilt.

Die Maximierung der Cross-Correlation für die normierten Vektoren ist äquivalent zur Minimierung des $\mathcal{L}_2$-Abstands:

$$\begin{aligned} & \left\|\hat{U} - \hat{U}^*\right\|_2 \leq \left\|\hat{U} - \hat{U}'\right\|_2 \\ \Leftrightarrow \;& 2 - 2\langle \hat{U}, \hat{U}^* \rangle \leq 2 - 2\langle \hat{U}, \hat{U}' \rangle \\ \Leftrightarrow \;& \langle \hat{U}, \hat{U}^* \rangle \geq \langle \hat{U}, \hat{U}' \rangle. \end{aligned}$$

Es ist auch möglich, zu einer vorgegebenen Schranke für den $\mathcal{L}_2$-Abstand eine Schranke für die Cross-Correlation vorzugeben:

$$\begin{aligned} & \left\|\hat{U} - \hat{U}^*\right\|_2 \leq \delta \\ \Leftrightarrow \;& \langle \hat{U}, \hat{U}^* \rangle \geq 1 - \frac{\delta^2}{2}. \end{aligned}$$

Bei der Benutzung des Skalarproduktes spart man einige Subtraktionen. Die Benutzung des $\mathcal{L}_2$-Abstands hat jedoch den Vorteil, dass man den Fehlern am Rande des Vektors ein kleineres Gewicht (mit Hilfe einer Gewichtsfunktion) beimessen kann, als den Fehlern in der Mitte - dort, wo sich der Feature-Punkt auch befindet.

Orientierung

Die Festlegung der lokalen Orientierung kann wie folgt erfolgen:

$$\begin{aligned} g_l &:= \sum_{i=0}^{n-1} |\hat{U}_i| \\ g_r &:= \sum_{i=n+1}^{2 \cdot n} |\hat{U}_i| \end{aligned}$$

Falls $g_l < g_r$, so sollte die Orientierung gespiegelt werden.

Die Spiegelungsoperation S der Deskriptoren ist hier eine einfache Operation:

$$S(\hat{U})_i = \hat{U}_{2 \cdot n - i} \quad i = 0, \ldots, 2 \cdot n$$

6.1.2 Deskriptoren basierend auf der Cross-Correlation der Ableitung

Analog zu U kann man auch die Ableitungen der Umgebung D benutzen. Dann ergeben sich nach der Normierung die Deskriptoren $\hat{D}$.

Die Festlegung der Orientierung kann ähnlich zum obigen Fall ablaufen:

$$g_l := \sum_{i=0}^{n-1} |\hat{D}_i|$$
$$g_r := \sum_{i=n}^{2\cdot n-1} |\hat{D}_i|$$

und falls $g_l < g_r$, so sollte die Orientierung gespiegelt werden. Die Spiegelungsoperation S muss jedoch leicht abgeändert werden:

$$S(\hat{D})_i = -\hat{D}_{2\cdot n-1-i} \quad i = 0, \ldots, 2 \cdot n - 1.$$

Dies ist notwendig, weil die Änderung der Orientierung zum Vorzeichenwechsel der ersten Ableitung führt.

6.1.3 SIFT-Deskriptoren

Unsere Version von Scale-Invariant-Feature-Transform (SIFT) Deskriptoren orientiert sich an Lowe [48].

Im ersten Schritt werden aus dem $2 \cdot n + 1$-dimensionalen Umgebungsvektor U die Ableitungen berechnet:

$$R_i = (U_{i+1} - U_i) \cdot w_i \quad i = 0, \ldots, 2 \cdot n - 1$$

Es ist auch möglich den Vektor R direkt aus der Ableitung der Scale-Space-Funktion $F_x(x, \sigma)$ abzutasten. Die Experimente haben jedoch gezeigt, dass die Approximation mit $U_{i+1} - U_i$ fast identische Ergebnisse liefert und keine Verschlechterung der Güte der Algorithmen nach sich zieht. Deswegen wird auf die Berechnung von $F_x(x, \sigma)$ verzichtet, um Laufzeit zu sparen.

$w_i := e^{-(i-n-0.5)^2/2\sigma_D^2}$ sind Gewichte mit passend gewählten σ_D, die dafür sorgen, dass je weiter die Vektorelemente von dem Feature-Punkt entfernt sind, desto kleinere Auswirkung haben die möglichen Fehler - die Daten am Rand eines Deskriptors sind weniger vertrauenswürdig als in der Mitte!

Der Vektor der Ableitungen wird in m Abschnitte von jeweils k Elementen unterteilt (das ist möglich, falls $n = m \cdot k$ gewählt wurde):

$$I_i := \{i \cdot k, i \cdot k + 1, \ldots, i \cdot (k + 1) - 1\} \quad i = 0, \ldots, m - 1.$$

Um die Dimensionalität zu reduzieren wird für jeden Abschnitt ein Histogramm via

$$d_i^{\pm} := \sum_{j \in I_i,\ \pm R_j > 0} |R_j|$$

erstellt. Damit ergibt sich ein $2 \cdot m$-dimensionaler Vektor

$$d := \left(d_0^-, \ldots, d_{m-1}^-, d_0^+, \ldots, d_{m-1}^+\right),$$

der nach der Normierung

$$\hat{d} := \frac{d}{\|d\|_2}$$

als ein Deskriptor verwendet werden kann.

Orientierung

Die Festlegung der lokalen Orientierung erfolgt wie folgt:

$$g_l := \sum_{i=0}^{m-1} d_i^-$$
$$g_r := \sum_{i=0}^{m-1} d_i^+$$

Falls $g_l < g_r$, sollte die Orientierung gespiegelt werden.

Die Spiegelungsoperation S der Deskriptoren ist hier eine einfache Operation:

$$S(d)_i^+ := d_i^- \quad i = 0, \ldots, m-1$$
$$S(d)_i^- := d_i^+ \quad i = 0, \ldots, m-1$$

6.1.4 SURF ähnliche Deskriptoren

In den letzten Jahren wurden die SURF-Deskriptoren (Bay u. a. [5]) zum state-of-the-art in der zweidimensionalen Bildverarbeitung. Im eindimensionalen Fall würden die Deskriptoren ähnlich zu SIFT berechnet.

Dabei wird die Umgebung R nicht durch das Abtasten von $f(\cdot) * G_x(\cdot, \sigma)$, sondern durch das Abtasten von $f(\cdot) * H(\cdot, \sigma)$ gewonnen.

Mit dem Haar-Wavelet

$$H(x, \sigma) := \begin{cases} 0 & x < -\Delta(\sigma) \\ 1 & -\Delta(\sigma) \leq x < 0 \\ 0 & x = 0 \\ -1 & 0 < x \leq \Delta(\sigma) \\ 0 & \Delta < x \end{cases}$$

Da H als eine Approximation von $G(x, \sigma)$ mit Rechtecken angesehen werden kann (vergleiche auch den Abschnitt 10.3.3 im Anhang), kann R auch aus

$f(\cdot) * G_x(\cdot, \sigma)$ berechnet werden. Dann würde der einzige Unterschied in der Berechnung der Vektorkomponenten ${}^{SURF}d^{\pm}$ bestehen:

$$
\begin{aligned}
{}^{SURF}d_i^+ &:= \sum_{j \in I_i} R_j, \\
{}^{SURF}d_i^- &:= \sum_{j \in I_i} |R_j|.
\end{aligned}
$$

Dies führt zum folgenden Zusammenhang mit den Einträgen des SIFT-Deskriptors:

$$
\begin{aligned}
{}^{SURF}d_i^+ &= d_i^+ - d_i^- \\
{}^{SURF}d_i^- &= d_i^+ + d_i^- \\
\left\| {}^{SURF}d \right\|_2 &= \sqrt{2}\, \|d\|_2 \\
\begin{pmatrix} {}^{SURF}\hat{d}_i^+ \\ {}^{SURF}\hat{d}_i^- \end{pmatrix} &= \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \hat{d}_i^+ \\ \hat{d}_i^- \end{pmatrix} \\
&= R \begin{pmatrix} \hat{d}_i^+ \\ \hat{d}_i^- \end{pmatrix}.
\end{aligned}
$$

Offensichtlich handelt es sich bei einem SURF-Deskriptor um einen (mit der Rotationsmatrix R) rotierten SIFT-Deskriptor. Da die Rotation eine starre Transformation ist und somit die Abstände invariant lässt, sind die SURF-Deskriptoren im eindimensionalen Raum äquivalent zu den SIFT-Deskriptoren. Die SURF-Deskriptoren werden deswegen nicht mehr weiter untersucht.

6.1.5 PCA-Descriptoren

Es bietet sich an, *Principal Component Analysis* (PCA) (Jolliffe [39]) zu verwenden, um die Dimensionalität der Deskriptoren zu reduzieren, ohne ihre Unterscheidungskraft (wesentlich) zu mindern. Es sind sogar Arbeiten bekannt, die Fälle belegen, bei welchen die PCA-komprimierten Deskriptoren besser als die Originale abschneiden (Ke und Sukthankar [40]). Dies kann allerdings nicht für die allgemeinen Anwendungen zutreffen.

Es ist möglich die PCA-Reduktion sowohl für die Cross-Correlation-Deskriptoren als auch für die SIFT-Deskriptoren durchzuführen.

Orientierung

Die Festlegung der lokalen Orientierung wird durch die zugrunde liegende Deskriptoren übernommen. Leider ist es nicht mehr möglich, einen PCA-Deskriptor zu spiegeln. Eine Lösung besteht darin, die PCA-Deskriptoren für

beide Richtungen im Speicher zu halten. Dabei geht der Vorteil eines PCA-Deskriptors - der kleinere Speicherbedarf - zumindest zum Teil verloren.

6.1.6 Moment-Invarianten

Die Momente

$$M_p^m := \sum_i i^p \hat{U}_i^m \quad m = 1, \dots m_{max},\ p = 0, \dots, p_{max}$$

definieren einen $m_{max} \cdot (p_{max} + 1)$-dimensionalen Deskriptor, dessen Vorteil die Spiegelungsinvarianz wäre.

Die Nachteile überwiegen jedoch diesen Vorteil: Im Normalfall werden die äußeren Bereiche der Umgebung U stärker gewichtet, obwohl sie gewöhnlich anfälliger für Störungen sind. Außerdem muss man sich Gedanken über die Gewichtung der einzelnen Komponenten machen.

Nach den anfänglichen experimentellen Misserfolgen wurde dieser Ansatz nicht weiter verfolgt, da er uns nicht besonders erfolgversprechend erschien.

6.1.7 Quantisierte Deskriptoren

Für jede Komponente gilt $-1 \leq d_i \leq 1$, weil die oben beschriebenen Deskriptoren d normiert sind. Diese Tatsache kann ausgenutzt werden, um den Deskriptor zu quantisieren und mit weniger Bits pro Komponente abzuspeichern. Für das vorgegebene Maximum M lässt sich der quantisierte Deskriptor wie folgt berechnen:

$$Q(d_i) := \texttt{round}(d_i \cdot M) \quad \forall i$$

Möchte man also eine Komponente mit nur 4 Bits kodieren, so sollte man M am besten 3.49 wählen, um so den Wertebereich $[-3, 3]$ abdecken zu können. Damit wäre die Größe des Deskriptors auf ein Achtel im Vergleich zum normalen Deskriptor aus Floats reduziert.

Unsere Experimente haben gezeigt, dass diese Reduktion keine großen Verluste der Güte der Algorithmen mit sich bringt (vergleiche Abschnitt 6.3.4).

Man kann beim Umstieg auf die Ganzzahlarithmetik jedoch nicht auf eine deutliche Reduktion der Laufzeit hoffen, da die aktuellen Prozessoren über FPUs verfügen und im Stande sind, die Nacheinanderberechnung von Floatoperationen effizient in einer Pipeline durchzuführen.

6.1.8 BRIEF-Deskriptoren

Die *Binary Robust Independent Elementary Features* (BRIEF) Deskriptoren (Calonder u. a. [14]) treiben die Quantisierung der Deskriptoren noch

weiter - dabei wird Information in binären Strings abgespeichert. Im Prinzip kann BRIEF sowohl für die Umgebung U als auch für jede beliebige Art von Deskriptor angewandt werden. Die *VerBRIEFung* eines n-dimensionalen Vektors U zu einem binären String der Länge m kann wie folgt stattfinden:

- Wähle m Paare $p_k := (i_k, j_k) \in \{0, \dots, n-1\}^2 \quad k = 0, \dots m-1$.
- Erstelle nun m Einträge des verBRIEFten Deskriptors

$$BRIEF_k := \begin{cases} 1 & U_{i_k} > U_{j_k} \\ 0 & \text{sonst} \end{cases} \quad k = 0, ..., m-1$$

  Für ein Beispiel siehe die Abbildung 6.1.

- Als Abstandsmaß kann der Hamming-Abstand benutzt werden, welcher effizient mit Hilfe der Bit-Operationen XOR und $BitCount$ berechnet werden kann:

$$HD(BRIEF^1, BRIEF^2) = BitCount(BRIEF^1 \hat{} BRIEF^2)$$

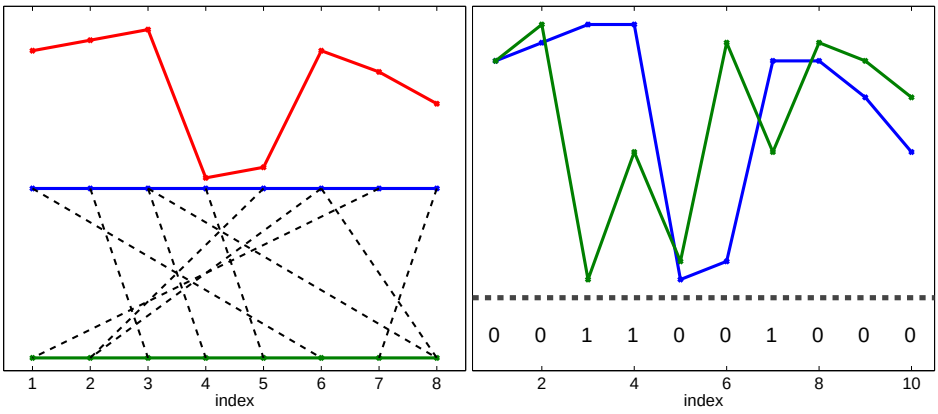

Abbildung 6.1: Links: Ein 8-dimensionaler Vektor V (rot) und die Vorschirft zum "VerBRIEFen": Es gibt 10 Paare $p_k = (i_k, j_k)$, repräsentiert durch eine gestrichelte Linie. Sie lauten (in lexikographischer Ordnung): $(1,6)$, $(2,3)$, $(3,4)$, $(3,8)$ usw. Rechts: Anwendung der Vorschrift auf den Vektor V. Die Ungleichung $V_{i_k} > V_{j_k}$ ist erfüllt, wenn die blau Linie über der grünen liegt. Der Vektor V wird damit als der binäre String 0011001000 dargestellt.

Die Wahl der Paare p_k ist entscheidend für die Qualität der entstandenen Deskriptoren.

Eine ähnliche Idee steckt auch hinter BRISC (Leutenegger u. a. [45]) und FREAK (Ortiz [55]) Deskriptoren. Der Unterschied besteht zum einen in einem festen Muster mit dem die Komponenten von U verglichen werden, zum anderen wird die Umgebung U auf folgende Art berechnet

$$U_i := F(x_0 + (i - n - 1) \cdot \tau(\sigma_0), \sigma_0 \cdot f^{|i-n|}) \quad f > 1$$

und damit die weiter von dem Feature-Punkt entfernten Komponenten der Umgebung stärker geglättet werden.

Im Unterschied zum zweidimensionalen Fall enthalten die eindimensionalen Umgebungen viel weniger Information. Unsere Experimente haben gezeigt, dass die eindimensionalen FREAK- und BRISC-Deskriptoren keine genügende Unterscheidungskraft besitzen und werden deswegen von uns nicht mehr weiter beleuchtet.

Orientierung

Bei "verBRIEFten" Deskriptoren sieht es mit der Orientierung ähnlich wie bei den PCA-Deskriptoren aus: Die Festlegung der lokalen Orientierung wird durch die zugrunde liegende Deskriptoren übernommen. Es nicht mehr möglich einen "verBRIEFten" Deskriptor zu spiegeln. Man kann jedoch auch hier die Versionen des Deskriptors für beide Orientierungen im Speicher halten.

6.2 Vergleichskriterien

Nun wollen wir die Performanz der oben beschriebenen Deskriptoren untereinander vergleichen. Es stellt sich die Frage, nach welchen Kriterien man die Güte der Deskriptoren vergleichen kann.

Zur Auswahl stehen uns unter anderem folgende Möglichkeiten:

- Eine naheliegende Möglichkeit, die Performanz der Features zu beurteilen, besteht darin, dass man die Genauigkeit der Suche (die Anzahl der Top-1-Treffer) auf einer großen Datenbank untersucht. Obwohl dies ein ultimativer Test ist, gibt es einige Nachteile: Zum einen ist diese Art des Testens zum Teil sehr zeitaufwendig, zum anderen ist sie oft nicht aussagekräftig genug - wenn die Zeitreihen in der Datenbank von guter Qualität sind und man eine beinahe hundertprozentige Genauigkeit erreichen kann, lässt sich der Unterschied zwischen den Deskriptoren nicht mehr feststellen.

- Der Benutzung von *recall* und $1 - precision$ zur Auswertung der Performanz ist eine Standardmethode und wurde unter anderem in Mikolajczyk und Schmid [51] verwendet.

- Der Vergleich der Verteilung der Abstände zwischen passenden und nicht passenden Deskriptoren.

- Manchmal sind die mittleren Werte ungenügend. So könnte die Anzahl der Zeitreihen-Paare, für die ein Deskriptor besser funktioniert als ein anderer, ein besserer Indikator der Performanz sein. Es ist auch

möglich, kritische, schlecht passende Paare stärker zu gewichten. Eine mögliche Darstellung zum Vergleich der Ergebnissen stellt der so genannte “Scatter Plot” dar.

Auf der Suche nach dem besten Deskriptor werden wir alle diese Kriterien unter die Lupe nehmen, doch zuerst machen wir uns Gedanken zu den Testszenarien.

6.2.1 Testumgebung

Es liegen uns 5 unterschiedliche Dicken- bzw. Breitenprofilesammlungen vor, die von unterschiedlichen Anlagen stammen:

Name	Anzahl der passenden Paare	Anlagenart	Profilart
`A`	1379	Alu	Dicke
`B`	1764	Stahl	Dicke
`C`	2166	Stahl	Breite
`D`	2166	Stahl	Dicke
`E`	63	Alu	Dicke

Bis auf die Sammlung `E` beinhalten die Testdatensätze eine relativ große Anzahl von passenden Paaren, so dass die gefundenen Ergebnisse über eine gewisse statistische Signifikanz verfügen. Alle passenden Paare wurden verifiziert - es kann also von der Richtigkeit der Zeitreihenzuordnungen ausgegangen werden.

Um zu gewährleisten, dass wirklich die Performanz der Deskriptoren verglichen wird, werden bei jedem Durchlauf immer dieselben Feature-Punkte untersucht. Diese Vorgehensweise enthält jedoch eine gewisse Gefahr, da die gewählten Feature-Punkte durch Zufall eine gewisse Art von Deskriptoren bevorzugen könnten und dadurch ein Bias entstehen könnte.

Es ist auch notwendig, für jeden Testfall (`A` bis `E`) und jede Art von Deskriptoren die bestmöglichen Parameter/Einstellungen zu ermitteln. Dabei ist ebenfalls nicht gänzlich klar, was der Begriff “die besten Einstellungen” bedeutet, denn je nachdem, welches Kriterium zur Bewertung gewählt wird, könnte es sich um unterschiedliche “beste” Parameter handeln.

Wir wählen die besten Einstellungen, indem wir die Zielfunktion

$$z(parameter\ set) = \#\ \text{Top}\ 1 + q \cdot recall.$$

maximieren, die einen Kompromiss darstellt.

Die Optimierung wird auf je 100 (bis auf den Fall `E`) zufällig gewählten Paaren durchgeführt, um eine annehmbare Laufzeit zu erreichen.

Mit der geeigneten Wahl der Gewichtung werden die Anzahl der richtig ermittelten Anfragen und die mittlere *recall*-Rate ungefähr gleich gewichtet:

- Für 100 Bänder variiert erfahrungsgemäß die Anzahl der richtig beantworteten Anfragen auf einem Intervall der Größe ≈ 2.
- *recall* variiert auf einem Intervall der Größe ≈ 0.1.
- $q = 50$ scheint eine vernünftige Wahl zu sein.

6.2.2 Kriterien

Nachdem die besten Einstellungen festgelegt wurden, wenden wir uns den oben erwähnten Kriterien zu.

Richtige Anzahl der Top-1-Treffer

Von jedem zusammengehörigen Zeitreihenpaar wird eine Zeitreihe (die wir als Vaterband/Vaterzeitreihe bezeichnen) in einer Datenbank abgespeichert. Für jedes Paar wird die zweite Zeitreihe des Paars (oft als *Kind* bezeichnet), die nicht in die Datenbank aufgenommen wurde, als eine Anfrage an die Datenbank gestellt. Anschließend wird die Zahl der richtig zurückgelieferten Vaterbänder gezählt. Da die Performanz des Algorithmus letztendlich durch die Anzahl der richtigen Top-1-Treffer bewertet ist, ist es der ultimative Test, dessen Ergebnisse leicht zu interpretieren sind.

Der große Nachteil neben der langen Laufzeit ist, dass, falls die Deskriptoren hinreichend gut sind und alle Anfragen richtig beantwortet werden, ist es nicht mehr möglich die Performanz dieser Deskriptoren zu unterscheiden.

recall und $1 - precision$

Um *recall* und $1 - precision$ zu definieren, betrachten wir zwei Mengen von Feature-Punkten D_p (z.B. der Fingerprint des Vaterbandes) und D_c (z.B. der Fingerprint des Kindes). Außerdem setzten wir voraus, dass wir $D_p^r \subset D_p$ und $D_c^r \subset D_c$ zusammen mit der bijektiven Funktion $r : D_p^r \to D_c^r$ kennen, wodurch die richtigen Zuordnungen zwischen den Features definiert werden. Natürlich kann es Deskriptoren geben, die nicht gleichzeitig in beiden Mengen vorkommen: $D_p \setminus D_p^r$ und $D_c \setminus D_c^r$.

Mit Hilfe der Deskriptoren wird nun versucht diese Zuordnungsvorschrift r zu schätzen. Dabei entsteht die (nicht unbedingt injektive oder surjektive) Funktion $h : D_p \supseteq D_p^h \to D_c^h$. Nun lässt sich die Anzahl der richtig bestimmen Zuordnungen definieren:

$$cm := \left| \{ x \in D_p^h \cap D_p^r \mid r(x) = h(x) \} \right| \tag{6.1}$$

und damit wird *recall* als

$$recall := \frac{cm}{|D_p^r|}$$

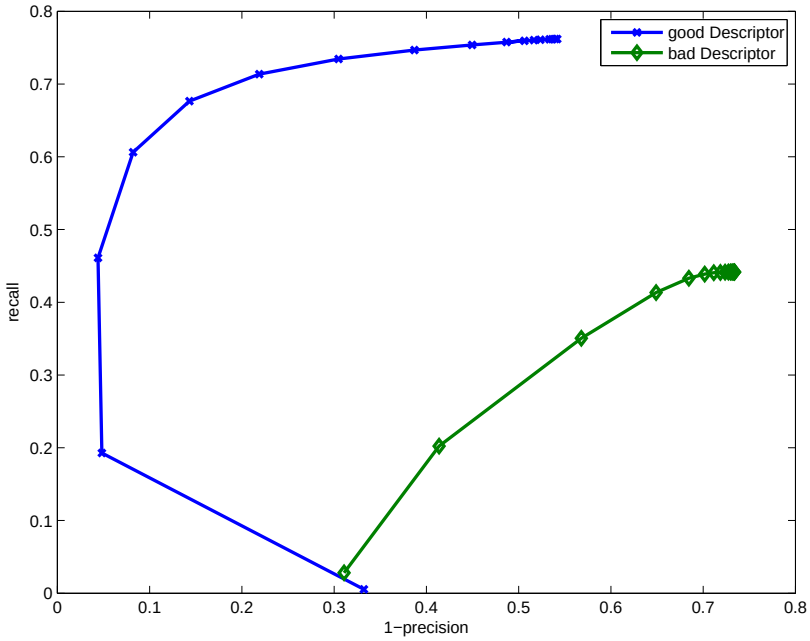

Abbildung 6.2: Ein beispielhafter recall vs $1-precision$ *Graph. Der Verlauf liegt für den besseren Deskriptor oberhalb vom Verlauf für den schlechteren Deskriptor. Zum einen kann mit Hilfe des guten Deskriptors mehr als* $70\,\%$ *der richtigen Zuordnungen gefunden werden (für den schlechten Detektor beträgt dieser Wert nur* $40\,\%$*). Zum anderen ist der Anteil der falsch bestimmten Zuordnungen für einen festen recall beim guten Deskriptor immer kleiner als beim schlechten Deskriptor: Um* $40\,\%$ *recall zu erreichen, muss beim schlechten Deskriptor in Kauf genommen werden, dass* $70\,\%$ *der Zuordnungen falsch sind. Beim guten Deskriptor beläuft sich dieser Fehler für* $recall = 0.4$ *nur auf rund* $10\,\%$*.*

definiert. *recall* ist damit der Anteil der richtigen Zuordnungen, die gefunden werden. $1 - precision$ wird dagegen definiert als:

$$1 - precision := \frac{|D_p^h| - cm}{|D_p^h|} =: \frac{fm}{|D_p^h|}.$$

$1 - precision$ ist damit der Anteil der gefundenen Zuordnungen, die falsch sind.

Um h zu bestimmen, wird eine recht naive Strategie angewandt: Für jeden Feature-Punkt $x \in D_p$ wird mit Hilfe der Deskriptoren der nächste Nachbar in der Menge D_c bestimmt. Anschließend werden nur diejenigen Zuordnungen aufgenommen, deren Abstand kleiner als ein vorgegebener Schwellwert

δ ist:

$$D_p^h := \{x | \min_{y \in D_c} abstand(x, y) < \delta\}$$
$$h(x) := \operatorname*{argmin}_{y \in D_c} abstand(x, y) \quad \forall x \in D_p^h$$

Ein Problem bleibt jedoch ungelöst: Die "richtige" Funktion r ist normalerweise unbekannt! Um die oben genannte Werte zu berechnen, reichen jedoch folgende Abschätzungen:

- $|D_p^r|$ wird konservativ durch $\min\{D_p, D_c\}$ abgeschätzt.
- Für *cm* - Anzahl der richtigen Zuordnungen - nehmen wir die Anzahl der Zuordnungen, welche die richtige Koordinatentransformation untermauern.

In einer perfekten Welt, wären $recall = 1$ und $1 - precision = 0$ für jede Wahl von δ. In realen Anwendungen steigen sowohl *recall* (was erstrebenswert ist) als auch $1 - precision$ (nicht erstrebenswert) mit wachsendem δ und so muss man bei der Wahl von δ diese Effekte erwägen.

Der *recall* vs. $1 - precision$ Graph (für ein Beispiel siehe Abbildung 6.2) eines besseren Deskriptors liegt oberhalb des Graphen eines schlechteren Deskriptors, denn dies bedeutet, dass bei gleicher *precision* der *recall* höher ist. Schwieriger ist die Beurteilung, falls die beiden Graphen sich kreuzen: Abhängig vom Schwellwert δ sind unterschiedliche Deskriptoren vorzuziehen.

Verteilung der Abstände

Eine weitere Methode das Potenzial der Deskriptoren zu visualisieren, ist die Darstellung der Verteilungen der Abstände von richtig bzw. falsch zugeordneten Deskriptoren. Sind diese Verteilungen fast identisch, so kann man auch schlecht zwischen einer richtigen und einer falschen Zuordnung nur aufgrund des Abstandes unterscheiden.

Obwohl diese Form der Illustration für die Güte der Deskriptoren von einigen Autoren benutzt wurde (z.B. Calonder u. a. [14], Lowe [48], Rublee u. a. [57]), scheint diese Möglichkeit etwas stiefmütterlich behandelt zu sein. Deswegen und weil man viel Information aus der Verteilung der Abstände ableiten kann, möchten wir uns damit genauer auseinandersetzen.

Typische Verteilungen, wie sie oft in der Literatur gezeigt werden, lassen sich in der Abbildung 6.3 finden.

Wie wir im weiteren Verlauf sehen werden, kann man aus diesen Verteilungen viele Erkenntnisse gewinnen. Zum einen werden wir den Verlauf des

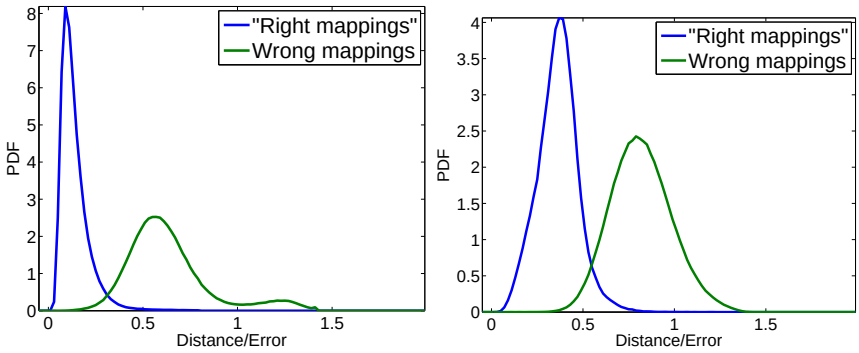

Abbildung 6.3: Das linke Diagramm zeigt die Verteilungen für gute Daten, während das rechte Diagramm - für etwas schlechtere Daten steht. Man sieht, dass die Lücke zwischen richtigen Zuordnungen (blau) und falschen Zuordnungen (grün) für die guten Daten größer ist.

recall vs. $1 - precision$ Graphen berechnen können, zum anderen werden wir im Stande sein, die Mächtigkeit der Mengen D_p^r und D_c^r genauer abzuschätzen, was vor allem in Abschnitt 6.5 (Eindimensionale Detektoren) zum Tragen kommt.

Diese Erkenntnisse liegen jedoch nicht auf der Hand und wir müssen etwas tiefer in die Problematik eintauchen.

Als erstes beschäftigen wir uns mit dem Abstand der falschen Zuordnungen. Es ist ziemlich einfach, diese Verteilung aus den Daten zu bekommen: Man nimmt zwei zufällig gewählte Deskriptoren von zwei Bändern, die nicht zu einander passen. Diese Information ist jedoch nicht unbedingt das, was wir brauchen: Der Abstand der richtigen Zuordnung muss sich nicht gegen eine beliebige falsche Zuordnung durchsetzen, sondern gegen den kleinsten Abstand einer Menge von falschen Zuordnungen.

Wir sollten uns daher vor allem für die Verteilung der besten falschen Zuordnungen interessieren, die beim Vergleichen gefunden wurden. Betrachtet man diese Verteilungen, so wird es offensichtlich, dass die Situation (Abbildung 6.4) nicht so gutartig ist, wie es die Abbildung 6.3 noch andeutete.

Während man die Verteilungsdichten für die falschen Zuordnungen bzw. die besten falschen Zuordnungen ziemlich einfach berechnen kann, indem man unpassende Bänder nimmt, ist es nicht mehr so einfach mit der Verteilung der richtigen Zuordnungen. Zum einen sind die richtigen Zuordnungen unbekannt zum anderen ist es undenkbar, eine hinreichend große Menge der richtigen Zuordnungen von einem Menschen zuweisen zu lassen.

Ähnlich wie beim Schätzen des *recalls* werden diejenigen Zuordnungen als richtig gewertet, welche die richtige Koordinatentransformation untermauern. Diese Verteilung wurde in den Abbildungen 6.3 und 6.4 als *"right mappings"* bezeichnet.

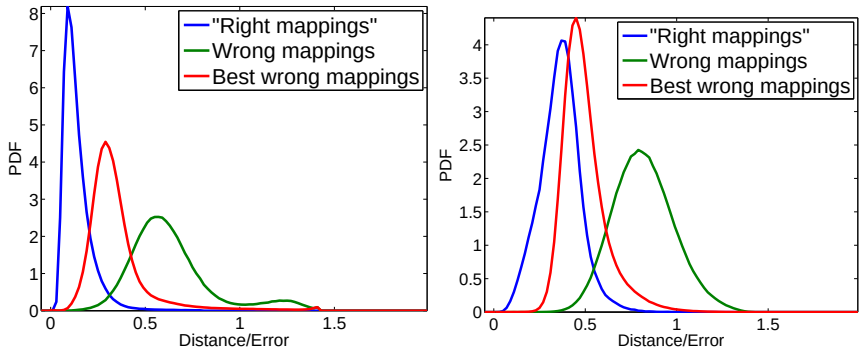

Abbildung 6.4: Der Unterschied zwischen den guten Datensätzen links und den schlechten rechts ist offensichtlicher, falls man sich die Verteilung der besten falschen Zuordnungen (rot) anschaut.

Man muss sich jedoch im Klaren sein, dass man bei dieser Vorgehensweise nicht die Verteilungsdichte der Abstände für richtige Zuordnung bekommt, sondern die bedingte Verteilung, unter der Bedingung, dass der Abstand zwischen den richtig zugeordneten Deskriptoren kleiner ist als der Abstand für die beste falsche Zuordnung.

Je größer die Menge der Deskriptoren, gegen die sich der richtige Deskriptor durchsetzen muss, desto ungünstiger ist die Situation. Deswegen erwarten wir für Bänder mit größerer Anzahl der Features kleinere *recall*-Raten.

Im Testsatz `A` befinden sich Bänder mit unterschiedlicher Anzahl von Features, und so sind wir im Stande die obige Behauptung zu überprüfen: In der Abbildung 6.5 sinkt die *recall*-Rate mit steigender Anzahl von Features.

Nun fragen wir uns, wie die richtige Verteilung der Abstände von richtig zugeordneten Deskriptoren aussieht und ob wir diese aus den experimentell ermittelten Verteilungen berechnen können.

Wir betrachten folgende Zufallsvariablen:

- Δ_r, auch als "Abstand einer richtigen Zuordnung" bezeichnet.
- Δ_w, auch als "der kleinste Abstand aller falschen Zuordnung" bezeichnet.
- $R = \begin{cases} 1 & \Delta_r < \Delta_w \\ 0 & \text{sonst} \end{cases}$, welche wir auch als *recall*-Indikator bezeichnen werden. Der Indikator ist 1, falls der Deskriptor bei der Suche richtig zugeordnet wurde und 0 andererfalls.

Des Weiteren nehmen wir an, dass die Zufallsvariablen Δ_r und Δ_w stochastisch unabhängig sind. Damit ergibt sich für die gemeinsame Verteilung bzw. Verteilungsdichte:

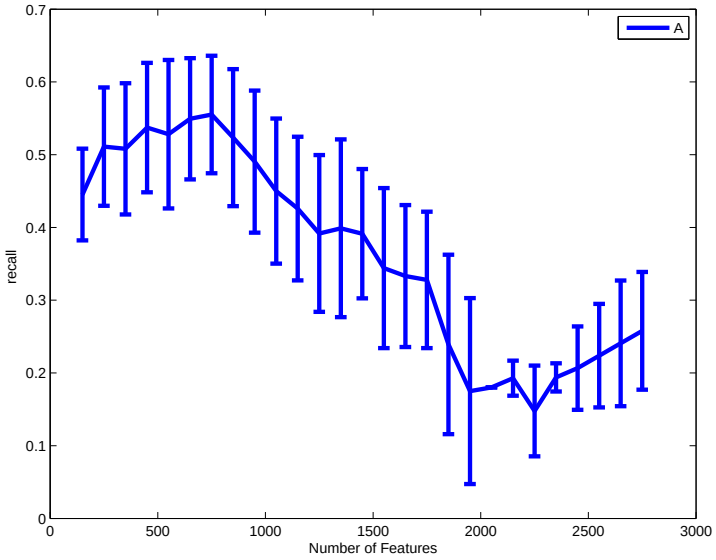

Abbildung 6.5: Der Verlauf spricht für die Behauptung "Sinkende *recall*-Rate mit steigender Anzahl von Features".

$$F(\Delta_r, \Delta_w) = F_r(\Delta_r) \cdot F_w(\Delta_w),$$
$$f(\Delta_r, \Delta_w) = f_r(\Delta_r) \cdot f_w(\Delta_w)$$

Die blaue Kurve in den Abbildungen 6.3 und 6.4, die als *"right mappings"* bezeichnet wird, ist dann nichts anders als $f_{rr}(\Delta_r) := f_r(\Delta_r | R = 1)$ - die bedingte Verteilungsdichte für Δ_r unter der Bedingung, dass $\Delta_r < \Delta_w$ bzw. dass der richtige Abstand kleiner ist, als der kleinste Abstand aller falschen Zuordnungen.

Es ist denkbar, eine Zuordnung zweier Deskriptoren nicht in Betracht zu ziehen, falls der Abstand zwischen ihnen größer als ein vorgegebener aber frei wählbarer Schwellwert δ ist. Die Wahrscheinlichkeit dafür, dass bei diesem Szenario eine richtige Zuordnung als solche erkannt wird beträgt:

$$\begin{aligned} Recall(\delta) &:= P(\{R = 1\} \wedge \Delta_r < \delta) \\ &= \int_{x<y, x<\delta} f(x, y) dx\, dy \\ &= \int_{-\infty}^{\delta} f_r(x) \int_{x}^{\infty} f_w(y) dy\, dx \\ &= \int_{-\infty}^{\delta} f_r(x) \cdot (1 - F_w(x)) dx \end{aligned} \tag{6.2}$$

und insbesondere

$$Recall(\infty) = P(\{R = 1\}) = 1 - \int_{\mathbb{R}} f_r(x) F_w(x) dx.$$

Die Bezeichnung $Recall(\delta)$ ist nicht zufällig gewählt und wie wir später sehen werden, hängt diese mit dem Erwartungswert für die *recall*-Rate zusammen.

Die Verteilungsdichten $f_w(x)$ und $f_{rr}(x)$ können aus den Daten geschätzt werden - die Ergebnisse sind z.B. in der Abbildung 6.3 illustriert. Der Zusammenhang zwischen f_w, f_{rr} und f_r ermöglicht es uns, die Verteilungsdichte f_r zu berechnen:

$$\begin{aligned} F_{rr}(x) \cdot P(R = 1) &= F_{rr}(x) \cdot Recall(\infty) \\ &= \int_{-\infty}^{x} f_r(y)(1 - F_w(y))dy \overset{\frac{d}{dx}}{\Longrightarrow} \\ f_{rr}(x) \cdot Recall(\infty) &= f_r(x)(1 - F_w(x)) \Longrightarrow \\ f_r(x) &= \begin{cases} \frac{f_{rr}(x) \cdot Recall(\infty)}{1 - F_w(x)} & x \in \{F_w(x) < 1\} \\ 0 & x \in \{F_w(x) = 1\} \end{cases} \end{aligned}$$

$Recall(\infty)$ ist nur eine Konstante und kann mit der Normierungsbedingung $\|f_r(x)\|_1 = 1$ bestimmt werden:

$$\begin{aligned} 1 = \int_{\mathbb{R}} f_r dx &= Recall(\infty) \cdot \int_{\{F_w(x)<1\}} \frac{f_{rr}(x) \cdot}{1 - F_w(x)} dx \\ Recall(\infty) &= \left(\int_{\{F_w(x)<1\}} \frac{f_{rr}(x) \cdot}{1 - F_w(x)} dx \right)^{-1}. \end{aligned}$$

Nun kann die Formel (6.2) so umgeschrieben werden, dass sie nur von den experimentell ermittelbaren Verteilungen F_{rr} und F_w abhängt

$$Recall(\delta) = \int_{-\infty}^{\delta} f_{rr}(x) dx \cdot Recall(\infty). \tag{6.3}$$

All diese Überlegungen angewandt auf die obigen Daten ergeben Verteilungsdichten $f_r(d)$, die in der Abbildung 6.6 dargestellt sind.

Im nächsten Schritt beantworten wir die Frage, wie groß der Anteil von D_p^r in D_p ist. Zur Erinnerung: Nicht alle gesuchten Feature-Punkte im ersten Fingerprint haben einen passenden Feature-Punkt im zweiten Fingerprint. Dies kann z.B. dadurch bedingt werden, dass bei der Detektion von Feature-Punkten einige wegen Rauschen/Störungen nicht wiedergefunden werden.

Wir benutzen folgende Bezeichnungen:

- $n_r := |D_p^r|$ - Anzahl der Features, die in den beiden Fingerprints vorkommen.

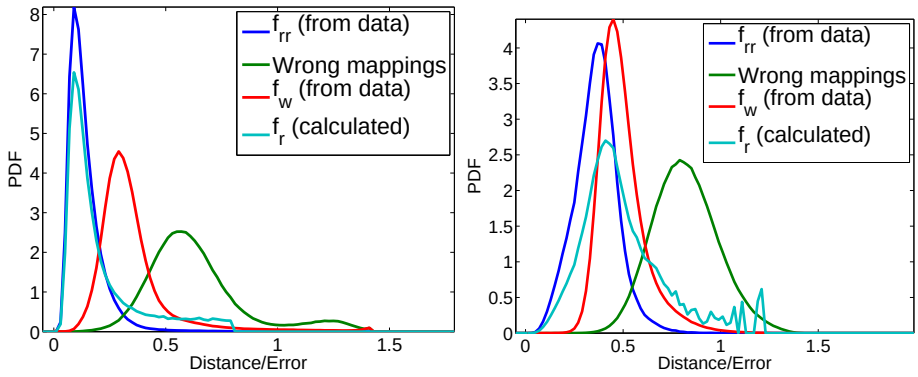

Abbildung 6.6: Unsere Berechnungen liefern plausible Ergebnisse: Die wirkliche Verteilung (f_r, cyan) hat höhere Anteile für größere Abstände als die experimentell ermittelbare Verteilungsdichte f_{rr}. Es ist auch klar, dass die recall-Rate für das rechte Datensatz deutlich kleiner sein wird, da die meisten richtigen Zuordnungen große Abstände haben und mit hoher Wahrscheinlichkeit schlechter als die besten falschen Zuordnungen sind. Der Grund fürs "Zappeln" im rechten Bild ist die kleine Anzahl der Ereignisse im diesem Bereich, was dazu führt, dass kleine Fehler zu großen Abweichungen führen.

- $n_w := |D_p \setminus D_p^r|$ - Anzahl der Features die nur im ersten Fingerprint vorhanden sind.
- $\alpha_r := \frac{n_r}{n_r+n_w}$ -Anteil der Features, die in beiden Fingerprinrs vorkommen.

Man sollte beachten, dass n_r, n_w und α_r Zufallsvariablen sind.

Kennen wir $E(\alpha_r)$, so sind wir im Stande die Feature-Detektoren zu bewerten: Denn je größer $E(\alpha)$, desto besser ein Feature-Detektor.

Nun bestimmen wir den Erwartungswert für den *recall* zum vorgegebenen Schwellwert δ. Die Wahrscheinlichkeit, dass die Zuordnung für ein Element $e \in D_p^r$ richtig bestimmt wird, ist

$$P(e \text{ richtig zugeordnet}) = P(\Delta_r < \Delta_w \wedge \Delta_r < \delta) = Recall(\delta).$$

Nimmt man nun an, dass die Ereignisse $\{e \text{ richtig zugeordnet}\}\ \forall e \in D_p^r$ stochastisch unabhängig sind, so ist cm (wie in der Gleichung (6.1) definiert) für ein festes n_r binomialverteilt:

$$\begin{aligned} cm &\stackrel{d}{=} B(Recall(\delta), n_r) \\ P(cm) &= \binom{n_r}{cm} Recall(\delta)^{cm}(1 - Recall(\delta))^{n_r - cm} \quad 0 \leq cm \leq n_r \\ E(cm) &= Recall(\delta) \cdot n_r \end{aligned}$$

mit dem bedingten Erwartungswert

$$E(recall(\delta)|n_r) = \frac{E(cm)}{n_r} = \frac{Recall(\delta)n_r}{n_r} = Recall(\delta)$$

und dem Zusammenhang zwischen $E(recall(\delta))$ und $Recall(\delta)$

$$\begin{aligned} E(recall(\delta)) &= \sum_{n_r} P(n_r) \cdot E(recall(\delta)|n_r) \\ &= \sum_{n_r} P(n_r) \cdot Recall(\delta) = Recall(\delta). \end{aligned}$$

Komplizierter ist es mit *precision*: Zusätzlich muss bestimmt werden, wie viele Deskriptorenzuordnungen (richtige und falsche) überhaupt gefunden wurden. Diese Anzahl bezeichen wir mit n_f.

Im Allgemeinen gilt:

$$precision = \frac{cm}{n_f}$$

Ist der Schwellwert $\delta = \infty$ gewählt, so ist die Frage nach n_f leicht zu beantworten: $n_f = n_r + n_w$. Für $\delta \leq \infty$ können wir nur von $n_f \leq n_r + n_w$ ausgehen.

Um n_f zu berechnen, betrachten wir zuerst die Situation für feste Werte n_r und n_w, dabei gehen wir wieder davon aus, dass die Zuordnungen der Feature-Punkte wieder (stochastisch) unabhängig stattfinden.

1. Fall: Der Deskriptor kommt aus der Menge D_p^r.

- Mit Wahrscheinlichkeit $Recall(\delta)$ wird die richtige Zuordnung gefunden.
- Mit Wahrscheinlichkeit

$$\begin{aligned} P(\{R = 0\} \wedge \Delta_r < \delta) &= \int_{x>y, y<\delta} f(x,y)dx\,dy \\ &= \int_{-\infty}^{\delta} f_w(y) \int_y^{\infty} f_r(x)dx\,dy \\ &= \int_{-\infty}^{\delta} f_w(y) \cdot (1 - F_r(y))dy =: p_1(\delta) \end{aligned}$$

  wird eine falsche Zuordnung gefunden.

Mit Annahme der stochastischen Unabhängigkeit ergibt sich eine multibinomiale Verteilung für die Anzahl der richtigen Zuordnungen (cm) und der falschen Zuordnungen der 1. Art (w_1):

$$\begin{aligned} P(cm, w_1|n_r) = &\binom{n_r}{cm}\binom{n_r - cm}{w_1} Recall(\delta)^{cm} \cdot \\ &\cdot p_1(\delta)^{w_1}(1 - Recall(\delta) - p_1(\delta))^{n_r - cm - w_1}, \qquad cm + w_1 \leq n_r. \end{aligned}$$

und damit

$$E(cm|n_r) = Recall(\delta) \cdot n_r,$$
$$E(w_1|n_r) = p_1(\delta) \cdot n_r.$$

2. Fall: Der Deskriptor kommt aus der Menge $D_p \setminus D_p^r$.

Dann wird mit Wahrscheinlichkeit $F_w(\delta) =: p_2(\delta)$ eine falsche Zuordnung gefunden. Auch hier ergibt sich eine Binomialverteilung für die Anzahl der falschen Zuordnungen der 2. Art (w_2):

$$P(w_2|n_w) = \binom{n_w}{w_2} p_2(\delta)^{w_2}(1-p_2(\delta))^{n_w-w_2}, \quad w_2 \leq n_w,$$
$$E(w_2|n_w) = p_2(\delta) \cdot n_2.$$

Zusammen ergibt sich

$$\begin{aligned} E(precision(\delta)|n_r, n_w) &= E\left(\frac{cm}{n_f}\middle| n_r, n_w\right) \\ &= E\left(\frac{cm}{cm + w_1 + w_2}\middle| n_r, n_w\right) \end{aligned} \tag{6.4}$$

Für $\delta = \infty$ ist $n_f = r + w_1 + w_2 = n_r + n_w$ konstant was zu

$$\begin{aligned} E(precision(\infty)|n_r, n_w) &= \frac{E(r|n_r, n_w)}{n_r + n_w} = \frac{E(recall(\infty)|n_r)}{n_r + n_w} \\ &= \frac{Recall(\infty)n_r}{n_r + n_w} = Recall(\infty) \cdot \alpha_r \end{aligned}$$

und

$$\begin{aligned} E(precision(\infty)) &= \sum_{n_r} P(n_r)E(precision(\infty)|n_r) \\ &= \sum_{n_r} P(n_r)Recall(\infty) \cdot \alpha_r \\ &= Recall(\infty) \cdot \sum_{n_r} P(n_r)\alpha_r \\ &= Recall(\infty) \cdot E(\alpha_r) \end{aligned} \tag{6.5}$$

führt.

Die letzte Gleichung (6.5) ist deswegen interessant, weil man die beiden Werte $E(precision(\infty))$ und $Recall(\infty)$ experimentell bestimmen kann. Dies ermöglicht die Berechnung vom Erwartungswert des Koeffizienten α_r, mit dessen Hilfe die Feature-Detektoren bewertet werden!

Im nächsten (und letzten) Schritt soll nun der Verlauf des *recall* vs $1 -$ *precision* Graphen berechnet werden.

Es ist möglich die Gleichung (6.4) numerisch auszuwerten, wir möchten jedoch die Näherung

$$E\left(\frac{X_1}{X_2}\right) \approx \frac{E(X_1)}{E(X_2)} \tag{6.6}$$

anwenden. X_1 und X_2 sind dabei Zufallsvariablen mit $\frac{\sqrt{Var(X_i)}}{E(X_i)} << 1$ mit $i = 1, 2$. Daher könne diese fast wie Konstanten behandelt werden können.

Es ist bekannt, dass für eine binomialverteilte Zufallsvariable $X \stackrel{d}{=} B(p, n)$ das obige Verhältnis mit wachsenden n gegen 0 geht:

$$\frac{\sqrt{Var(X)}}{E(X)} = \frac{\sqrt{p(1-p)n}}{pn} \stackrel{n\to\infty}{\longrightarrow} 0$$

Damit ist die Näherung für große n_r und n_w gut anwendbar und ergibt nach der Vereinfachung der Gleichung (6.4):

$$\begin{aligned} E(precision(\delta)|n_r,\, n_w) &\approx \frac{E(cm|n_r\, n_w)}{E(cm + w_1 + w_2|n_r, n_w)} \\ &= \frac{Recall(\delta) \cdot n_r}{(Recall(\delta) + p_1(\delta)) \cdot n_r + p_2(\delta) n_w} \\ &= \frac{Recall(\delta) \cdot \alpha_r}{(Recall(\delta) + p_1(\delta)) \cdot \alpha_r + p_2(\delta) \cdot (1 - \alpha_r)} \end{aligned}$$

Nehmen wir an, dass α_r kleine Varianz aufweist, so ist es wieder möglich die Näherung (6.6) anzuwenden:

$$E(precision(\delta)) \approx \frac{Recall(\delta) \cdot E(\alpha_r)}{(Recall(\delta) + p_1(\delta)) \cdot E(\alpha_r) + p_2(\delta) \cdot (1 - E(\alpha_r))}. \tag{6.7}$$

Die Gleichung (6.7) gilt nur für große n_r, n_w und fast konstante α_r.

Grenzen der Anwendbarkeit

Im Prinzip können unsere Überlegungen anhand der Daten überprüft werden: Zum einen bestimmen wir den *recall* vs $1 -$ *precision* Graphen auf dem "experimentellen" Weg durch das Variieren des Schwellwertes δ, zum anderen durch die Berechnungen aus den Verteilungen der Abstände. Es lohnt sich jedoch, nach dem Gültigkeitbereich unserer Überlegungen zu fragen.

Die wichtigste Voraussetzung für die oben hergeleiteten Formeln ist die stochastische Unabhängigkeit zwischen den Zufallsvariablen Δ_r und Δ_w. Diese Voraussetzung ist jedoch nicht selbstverständlich.

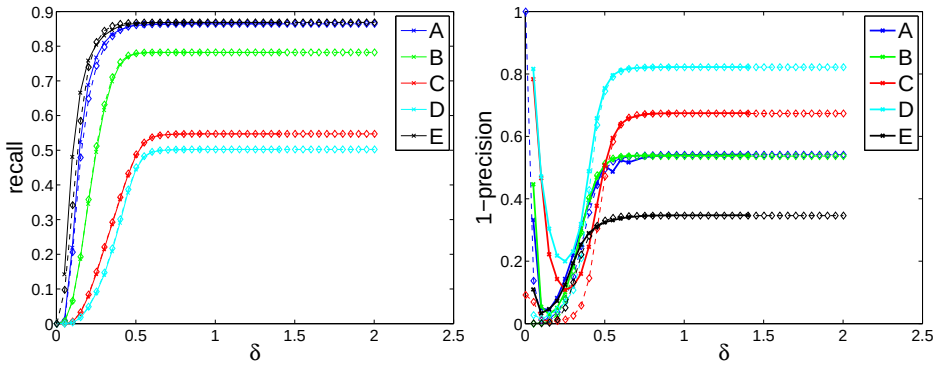

Abbildung 6.7: Wirkliche Verläufe (x-markierte Linien) und aus den Verteilungen berechnete Verläufe (Linien markiert mit ◇) stimmen gut für recall (linkes Bild) überein. Für $1 - precision$ *sieht man im rechten Bild vor allem für die kleinen Schwellwerte* δ *große Abweichungen. Dies gilt für alle* 5 *Testdatensätze.*

Betrachten wir die experimentellen Ergebnisse (Abbildung 6.7), so fällt auf: Die Ergebnisse für *recall* passen sehr gut zu den theoretischen Vorhersagen, während sie für *precision* zwar für kleine Schwellwerte δ einige Abweichungen aufweisen, asymptotisch jedoch gut übereinstimmen. Die Gründe für dieses Verhalten sind nicht ganz klar: Die Vermutung, dass die Näherung (6.6) für diesen Fehler verantwortlich sein könnte, wurde in den Simulationen nicht bestätigt. Möglicherweise liegt dieses Verhalten daran, dass die Verteilungen der Abstände F_r und F_w leicht korreliert sind.

Wegen der Fehler beim $1 - precision$ Verlauf sind die berechneten *recall* vs. $1 - precision$ Graphen nur zur Darstellung qualitativer Verläufe geeignet, wie man in der Abbildung 6.8 sehen kann.

Scatter-Plots

Vergleicht man nur zwei Deskriptoren, so ist es möglich ihre Performanz mit Hilfe von sogenannten *Scatter-Plots* zu visualisieren (siehe Abbildung 6.9).

Es ist möglich, diese Idee etwas formaler aufzufassen: Unsere Nullhypothese lautet *"Beide Deskriptoren haben gleich gute Performanz"*. Diese Annahme bedeutet, dass für jedes Bandpaar die Wahrscheinlichkeit, dass der Deskriptor vom ersten Typ besser abschneidet, als der Deskriptor vom zweiten Typ, genau so groß, wie dass der zweite Typ besser abschneidet.

Messen wir die Performanz als Anzahl der richtig bestimmten Zuordnungen durch die Deskriptoren der ersten Art (cm_1) und der zweiten Art (cm_2), so lässt sich die Hypothese wie folgt ausdrücken:

$$P(cm_1 > cm_2 | cm_1 \neq cm_2) = 0.5 = P(cm_2 > cm_1 | cm_1 \neq cm_2).$$

Haben wir n unabhängige Bandpaare, für welche die Bedingung $cm_1 \neq cm_2$

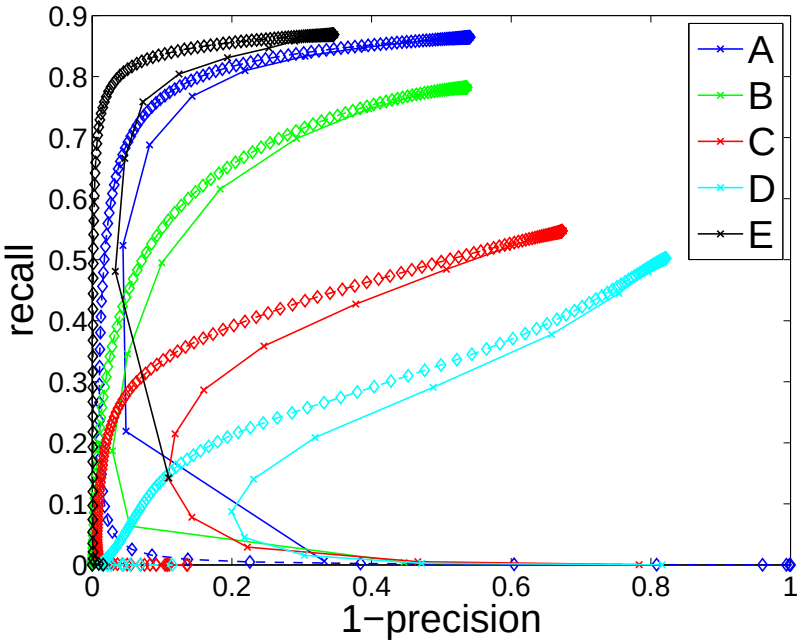

Abbildung 6.8: Wirkliche Verläufe (x-markierte Linien) und aus den Verteilungen berechnete Verläufe (Linien markiert mit ◇) des recall vs $1 - precision$ Graphen für alle 5 Testdatensätze.

gilt, so ist die Anzahl der Bänder b_1, für die der erste Deskriptor bessere Ergebnisse liefert, binomialverteilt:

$$P(b_1 = i) = \binom{n}{i} \left(\frac{1}{2}\right)^n$$

Für eine empirisch ermittelte Anzahl b_1^e (o.B.d.A. $b_1^e \leq \frac{n}{2}$, sonst betrachte $n - b_1$) wird die Nullhypothese auf dem Signifikanzniveau α abgelehnt, falls

$$\sum_{i=0}^{b_1^e} P(b_1 = i) + \sum_{i=n-b_1^e}^{n} P(b_1 = i) < \alpha$$

erfüllt ist.

6.3 Vergleich der Deskriptoren

Zuerst vergleichen wir Cross-Correlation (CC), Cross-Correlation der Ableitungen (CCD) und SIFT-Deskriptoren untereinander. Anschließend widmen wir uns PCA- und BRIEF-Deskriptoren.

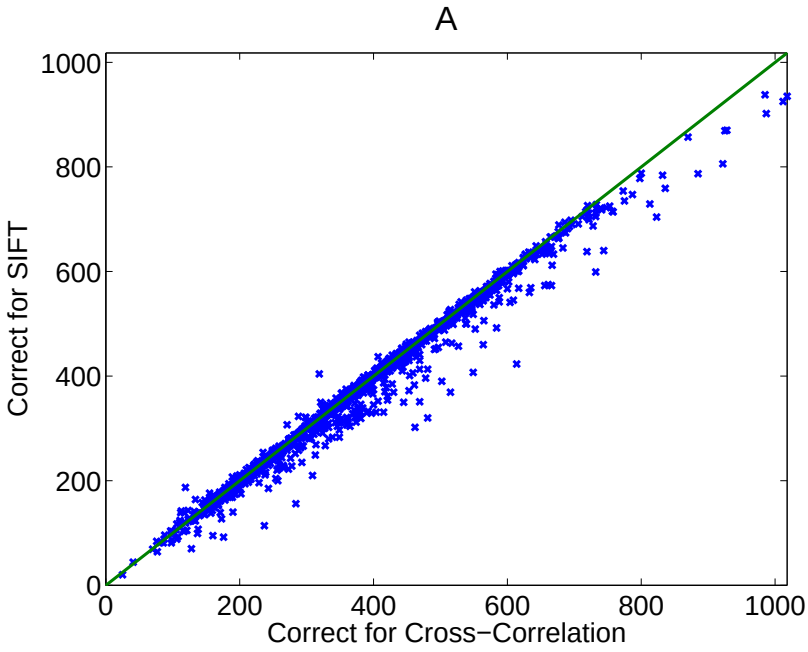

Abbildung 6.9: Ein typisches Scatter-Plot, das SIFT- und Cross-Correlation-Deskriptoren (CC-Deskriptoren) vergleicht. Jeder Punkt stellt Ergebnisse für ein Bandpaar dar: Die x-Koordinate ist das Ergebnis bei der Benutzung der CC-Deskriptoren, die y-Koordinate - bei SIFT-Deskriptoren. Die grüne Winkelhalbierende stellt die Grenze da: Für Punkte oberhalb dieser Linie haben die SIFT-Deskriptoren einen besseren Wert erreicht, für die Punkte unterhalb haben die CC-Deskriptoren bessere Performanz.

6.3.1 CC-, CCD- und SIFT-Deskriptoren

Beste Einstellung für Deskriptoren

Die durchgeführten Optimierungen lieferten folgende optimale Deskriptorengrößen:

Datensatz	CC	CCD	SIFT
`A`	33	32	24
`B`	17	16	12
`C`	33	48	24
`D`	33	40	24
`E`	33	48	16
`Overall`	33	32	20

Die unter `Overall` ausgewiesenen Einstellung lieferten die besten Ergebnisse kombiniert über alle Datensätze.

Anzahl der richtigen Top-1-Treffer

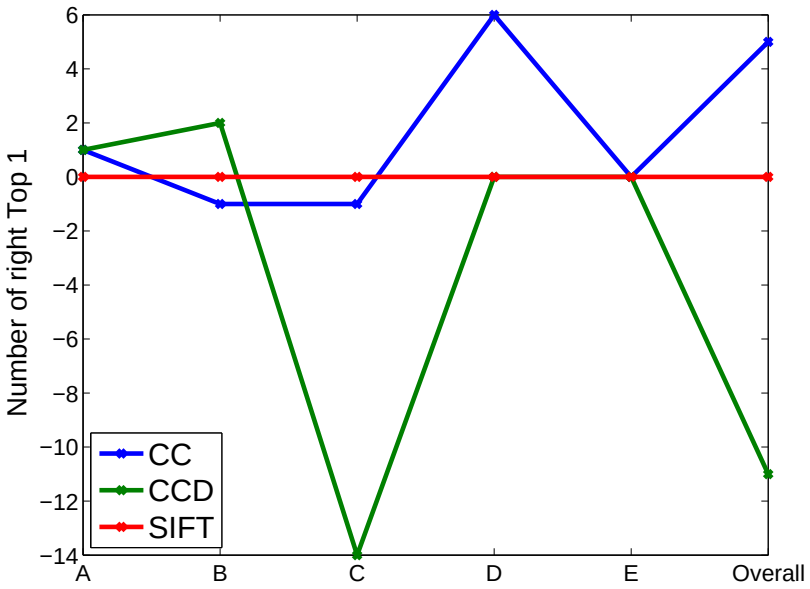

Abbildung 6.10: Anzahl der richtigen Top-1-Treffer. Die Werte beziehen sich auf die Anzahl der Top-1-Treffer des SIFT-Deskriptors.

Beim ersten und wichtigsten Kriterium liegt der Cross-Correlation-Deskriptor etwas vorne (Abbildung 6.10), was vor allem am guten Abschneiden beim Datensatz `D` liegt. Cross-Correlation der Ableitung liegt wegen der schlechten Performanz beim Datensatz `C` an der dritten Stelle.

Alles in einem ist es nicht einfach, einen eindeutigen Sieger festzulegen.

Recall vs. 1-precision

Betrachtet man die Verläufe der *recall* vs. $1-$*precision* Graphen (Abbildung 6.11), so stellt man Folgendes fest:

- Für die Datensätze `A` und `E` sind alle Deskriptoren ungefähr gleich gut.
- Beim Datensatz `D` ist der Cross-Correlation-Deskriptor eindeutig die beste Wahl.
- SIFT-Deskriptor ist besonders gut für die Datensätze `B` und `C` geeignet.

Scatter-Plot

In diesem Abschnitt sollen nur zwei Arten von Deskriptoren miteinander verglichen werden: SIFT- und Cross-Correlation-Deskriptoren. Man konnte

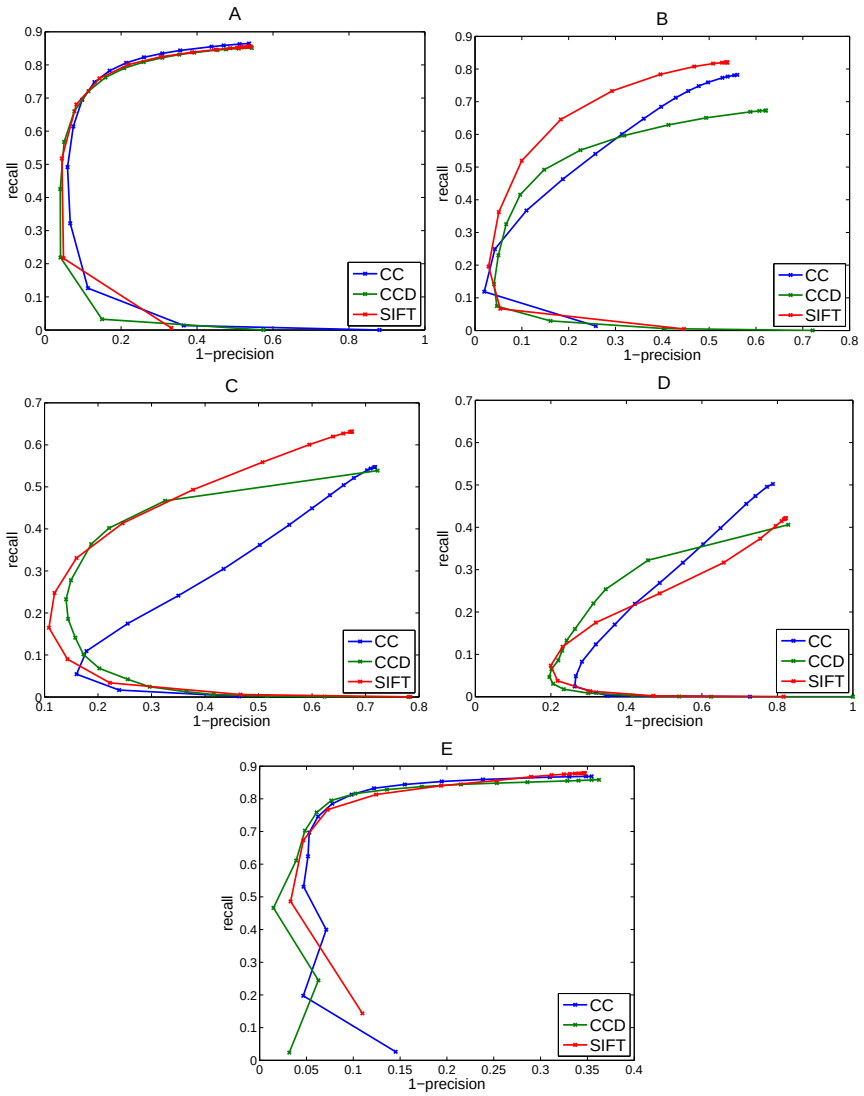

Abbildung 6.11: Vorteile für SIFT-Deskriptoren bei Datensätzen `B` *und* `C`*, während Cross-Correlation-Deskriptoren eindeutig die bessere Wahl für den Datensatz* `D` *darstellten. Das Zeichen für die bessere Performanz ist, dass die entsprechende Kurve oberhalb anderer Verläufe liegt: z.B. für den Datensatz* `C` *kann mit Hilfe von SIFT-Deskriptoren ein recall von* 0.6 *erreicht werden, für die CC-Deskriptoren ist ein recall* ≈ 0.5 *möglich.*

sich anhand der anderen Kriterien schon davon überzeugen, dass diese performanter als die Cross-Correlation-Deskriptoren auf Basis der Ableitung sind.

Die in der Abbildung 6.12 als Scatter-Plots visualisierten Ergebnisse sind nochmal in folgender Tabelle zusammengefasst:

Datensatz	CC besser	SIFT besser	NH abgelehnt ($\alpha = 10^{-6}$)
A	800	502	Ja
B	253	1457	Ja
C	334	1770	Ja
D	1841	266	Ja
E	15	47	Ja

Es ist bemerkenswert, dass für alle Datensätze die Nullhypothese *"Beide Deskriptoren haben gleiche Performanz"* sogar auf dem Signifikanzniveau $\alpha = 10^{-6}$ abgelehnt wurde. Damit konnte man für jeden Datensatz einen eindeutigen Sieger feststellen.

6.3.2 PCA-Deskriptoren

Nun möchten wir untersuchen, in wie weit es möglich ist, dass man mit Hilfe von PCA die Dimensionalität der Deskriptoren vermindert, ohne ihre Performanz entscheidend zu verschlechtern. Eine Reduktion der Dimensionen hätte mehrere Vorteile: schnellere Vergleiche, kleinerer Speicherbedarf, größere Effizienz der im Abschnitt 6.7 untersuchten Datenstrukturen zum Suchen im mehrdimensionalen Raum.

Die schon in Abschnitt 5.4 erwähnte Problematik mit der Orientierung wird dadurch umgangen, dass im Speicher gleichzeitig die beiden Versionen der Deskriptoren für unterschiedliche Orientierungen gehalten werden. Dies ist zwar gegenläufig zu dem Bestreben, möglichst wenig Speicher zu benutzen, die Geschwindigkeitsvorteile bleiben jedoch erhalten.

Die Ergebnisse in der Abbildung 6.13 legen Folgendes nahe: Die PCA-Reduktion kommt vor allem für CC-Deskriptoren in Frage, denn schon mit Projektion auf die ersten 14 Dimensionen werden im Schnitt 99% des $recalls$ der vollständigen Deskriptoren erreicht. Teilweise haben PCA-Deskriptoren bessere Ergebnisse als die zugrunde liegenden Deskriptoren - dies kann dadurch erklärt werden, dass die Störungen vor allem bei den verworfenen Dimensionen eine Rolle spielen. Nimmt man diese verworfenen Dimensionen nach und nach hinzu, so nimmt der positive Effekt auch wieder ab.

6.3.3 BRIEF-Deskriptoren

Die Performanz der BRIEF-Deskriptoren hängt von der Wahl der Paare p_k ab. Alle von uns ausgedachten, regelmäßigen Muster bzw. Vorschriften p_k waren den zufälligen Paaren p_k deutlich unterlegen. Ähnliche Erfahrungen haben auch die Erfinder der BRIEF-Deskriptoren machen müssen (Calonder u. a. [14]).

Um einen ersten Eindruck zu bekommen, werden zufällig 40 Muster $\{p_k | k = 1, \ldots, size\}$ mit $size = 32$, 64, 128 oder 256 gewählt und deren Performanz

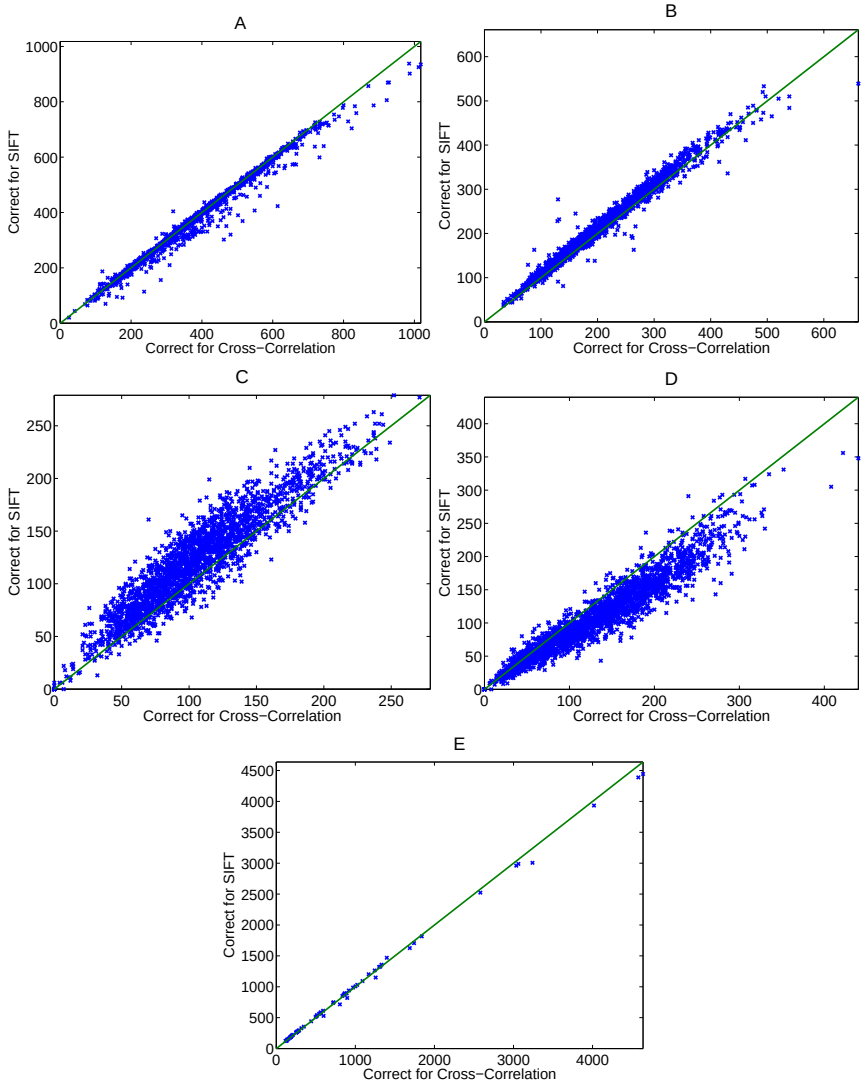

Abbildung 6.12: Vorteile für SIFT-Deskriptoren bei Datensätzen `B`, `C` und E, für Cross-Correlation-Deskriptoren - bei Datensätzen `A` und `D`.

mit der Performanz der zugrunde liegenden “unverBRIEFten” Deskriptoren verglichen. Ähnlich wie bei PCA-reduzierten Varianten werden die ”verBRIEFten“ Deskriptoren für beide Orientierungen gleichzeitig im Speicher gehalten.

Ein typisches Ergebnis ist in der Abbildung 6.14 dargestellt:

Für alle Datensätze und Deskriptorenarten gilt: die “verBRIEFten” Versionen kommen nicht ganz an die Performanz der originalen Deskriptoren heran. Erst die 256 bit BRIEF-Deskriptoren würden in Betracht kommen, doch wie wir sehen werden, gibt es bessere Techniken den Speicherbedarf zu

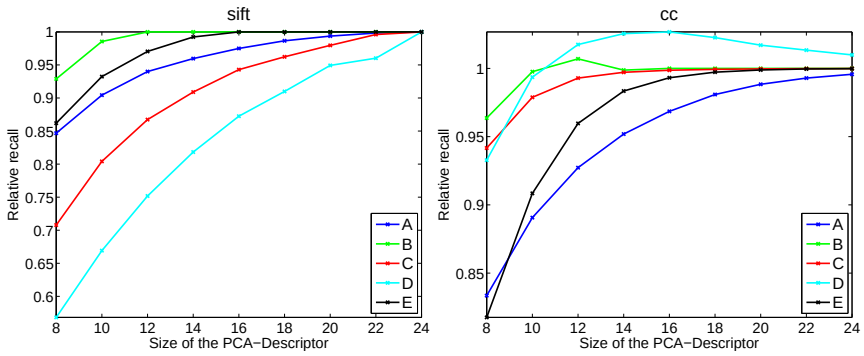

Abbildung 6.13: Mittlerer recall der PCA-Deskriptoren in Abhängigkeit von der Dimensionalität relativ zum mittleren recall vollständiger SIFT-Deskriptoren (linke Seite) bzw. CC-Deskriptoren (rechte Seite).

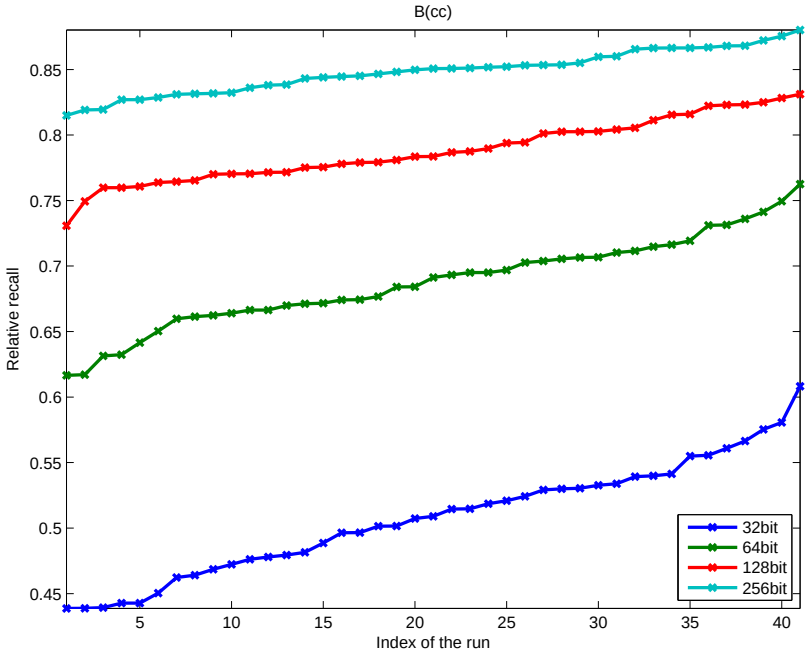

Abbildung 6.14: Mittlerer recall der "verBRIEFten"-Deskriptoren zum mittleren recall vollständiger Deskriptoren für 40 *zufällig gewählten Konfigurationen* p_k *mit* $k = 1, \ldots,$ *size für size* $= 32$ *(blau), size* $= 64$ *(grün), size* $= 128$ *(rot) und size* $= 256$ *(cyan).*

reduzieren.

Wir möchten die BRIEF-Deskriptoren nicht länger untersuchen, weil sie sich bei allen unseren Versuchen den andern Arten der Deskriptoren als unterlegen erwiesen haben.

6.3.4 Quantisierte Deskriptoren

Bei der Betrachtung der Resultate mit quantisierten Deskriptoren (Abbildung 6.15) stellt man fest, dass die Float- bzw. Double-Genauigkeit gar nicht notwendig ist, schon ab 6 Bit pro Wert unterscheiden sich die Ergebnisse kaum von den Originalen.

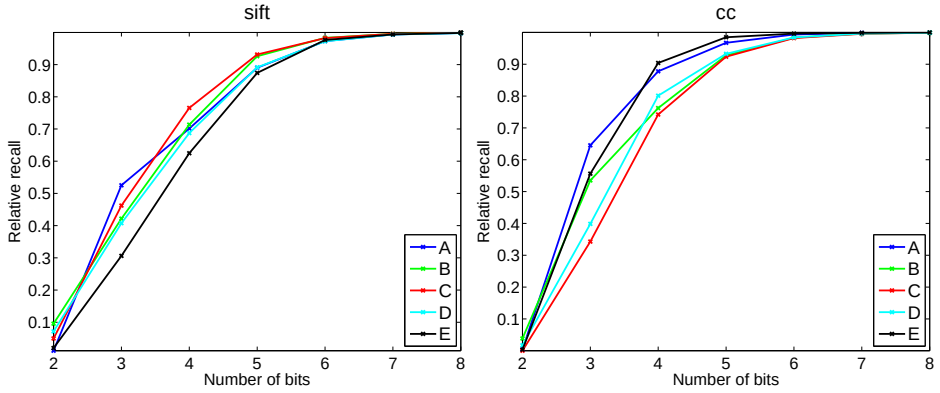

Abbildung 6.15: Mittlerer recall der quantisierten Deskriptoren in Abhängigkeit von Bit pro Wert relativ zum mittleren recall vollständiger - 32 bit pro Wert - SIFT-Deskriptoren (linke Seite) bzw. CC-Deskriptoren (rechte Seite).

6.3.5 Fazit

Bei den Datensätzen aus unseren Anwendungsszenarien haben sich CC- und SIFT-Deskriptoren als die bessere Wahl herausgestellt. Die CC-Deskriptoren sind besonders im Zusammenhang mit PCA-Reduktion interessant, da man mit nur 14 Werten praktisch gleiche Performanz wie mit den vollständigen Deskriptoren erreichen kann. Nachteil dieser Vorgehensweise ist, dass man die PCA-Matrix aus den Daten "erlernen" muss. Möchte man eine Lösung, die ohne irgendwelche Anpassungen gute Ergebnisse liefert, so sind die SIFT-Deskriptoren eine bessere Wahl.

Um den Speicherbedarf zu reduzieren, lohnt es sich auf jeden Fall, die Deskriptoren zu quantisieren, denn schon bei 6 bit pro Wert (statt 32 im Falle eines Floats) stellt man so gut wie keinen Unterschied mehr fest.

6.4 Lokale Orientierung

Da es im eindimensionalen Fall nur zwei Orientierungen gibt: x und $-x$, ist es kein Problem, beide auszuprobieren. Die Festlegung der lokalen Orientierung der Deskriptoren würde dazu führen, dass man statt zwei nur einen Vergleich

braucht. Diesen speed-up bezahlt man jedoch mit den sinkenden *recall*-Raten, da man auch bei der Festlegung der lokalen Orientierung, bedingt durch Ausreißer bzw. andere Störungen, Fehler machen kann.

In diesem Abschnitt untersuchen wir unterschiedliche Strategien zur Festlegung der Orientierung.

Für eine gespiegelte Funktion $f'(x) = f(-x) =: [Sf](x)$ wird auch die Scale-Space-Funktion an der y-Achse gespiegelt sein:

$$F'(x,\, \sigma) = F(-x, \sigma)$$

Damit ändert sich der $2 \cdot n + 1$-dimensionale Umgebungsvektor U zu SU via:

$$SU_i = U_{2n+1-i}$$

Dies führt zu folgenden Spiegelungsfunktionen für unterschiedliche Deskriptoren

CC Siehe Abschnitt 6.1.1:

$$S(d)_i = \hat{U}_{2 \cdot n - i} \quad \text{für} \quad i = 0, \ldots, 2 \cdot n$$

SIFT Siehe Abschnitt 6.1.3:

$$\begin{aligned} S(d)_i^+ &:= d_i^- \quad \text{für} \quad i = 0, \ldots, m-1 \\ S(d)_i^- &:= d_i^+ \quad \text{für} \quad i = 0, \ldots, m-1 \end{aligned}$$

CCD Siehe Abschnitt 6.1.2:

$$S(\hat{D})_i = -\hat{D}_{2 \cdot n - 1 - i} \quad \text{für} \quad i = 0, \ldots, 2 \cdot n - 1 \tag{6.8}$$

Die Orientierungsfunktion $o(d) \in \{1, -1\}$ sollte folgende Eindeutigkeitseigenschaft aufweisen:

$$o(d) \neq o(S(d)) f.s.$$

$f.s.$ - *"fast sicher"* bedeutet, dass die Wahrscheinlichkeit für das Vorkommnis $o(d) = o(S(d))$ 0 beträgt. Dabei wird mit d ein generischer Deskriptor bezeichnet, welcher ein CC-, CCD- oder SIFT-Deskriptor sein könnte.

Die Wahl einer perfekten Orientierungsfunktion, die bei der Festlegung der lokalen Orientierung keinen Fehler macht, würde sich zweifach auszahlen: Zum einen durch schnellere Suche, weil man nicht jede Orientierung einzeln untersuchen muss, zum anderen durch die Möglichkeit von PCA- bzw. BRIEF-Deskriptoren zu profitieren, ohne sie für jede Orientierung gesondert abspeichern zu müssen.

Im Folgenden werden Orientierungsfunktionen untersucht:

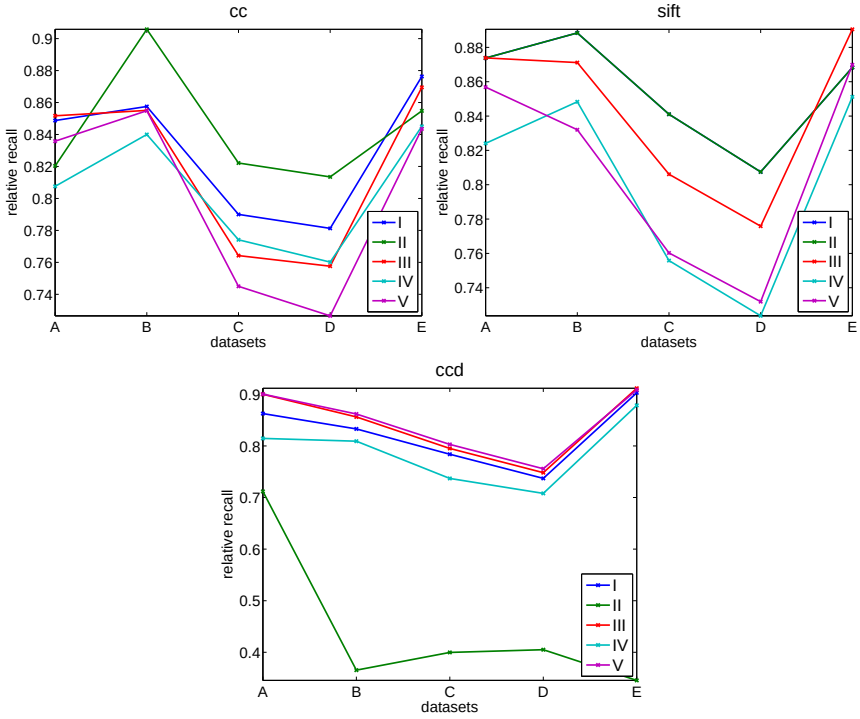

Abbildung 6.16: Es werden die relativen recall-Raten in Abhängigkeit von Datensätzen für unterschiedliche Orientierungsfunktionen dargestellt. Die Variante II scheint die besten Ergebnisse für CC- und SIFT-Deskriptoren zu erreichen. Für den - wie oben schon festgestellt - weniger interessanten Fall, den CCD-Deskriptor, schneidet die Variante V am besten ab.

I $o(d) = \begin{cases} 1 & \sum_{0 \le i < n} |d_i| < \sum_{n+1 \le i \le 2n} |d_i| \\ -1 & sonst \end{cases}$

Es wird also verlangt, dass die rechte Seite des Deskriptors "schwerer" ist, als die linke.

II $o(d) = \begin{cases} 1 & \sum_{0 \le i < n} d_i < \sum_{n+1 \le i \le 2n} d_i \\ -1 & sonst \end{cases}$

Im Unterschied zur ersten Variante, werden vorzeichenbehaftete Werte genommen.

III $o(d) = \begin{cases} 1 & \sum_{0 \le i < n} w_i \cdot |d_i| < \sum_{n+1 \le i \le 2n} w_i \cdot |d_i| \\ -1 & sonst \end{cases}$, wobei Gewichte w_i so gewählt werden, dass die Werte in der Mitte stärker gewichtet werden.

Die Variante **I** wird insofern verbessert, dass die fehleranfälligen Werte am Rande weniger stark berücksichtigt werden.

IV $o(d) = \begin{cases} 1 & \max_{0 \le i < n} |d_i| < \max_{n+1 \le i \le 2n} |d_i| \\ -1 & sonst \end{cases}$

Das "schwerste" Element gibt den Ausschlag für die Orientierung - es muss sich auf der rechten Seite befinden.

V $o(d) = \begin{cases} 1 & \max_{0 \le i < n} w_i \cdot |d_i| < \max_{n+1 \le i \le 2n} w_i \cdot |d_i| \\ -1 & sonst \end{cases}$, wobei Gewichte w_i so gewählt werden, dass die Werte in der Mitte stärker gewichtet werden.

Wie **IV** mit dem Unterschied, dass die fehleranfälligen Werte am Rande weniger stark berücksichtigt werden.

Die Gleichheit $o(d) = o(S(d))$ gilt nur, wenn die beiden Seiten der Ungleichung identisch sind. Setzt man eine kontinuierliche Verteilung der Deskriptoren voraus, so trifft dies nur für eine Nullmenge zu.

Natürlich sind auch andere Möglichkeiten denkbar, wir beschränken unsere Untersuchungen jedoch auf diese Varianten.

Für die SIFT-Deskriptoren gibt es keinen Unterschied zwischen der Variante I und II, weil alle Einträge nicht negativ sind. Für die CCD-Deskriptoren besitzt die Variante II nicht die Eindeutigkeitseigentschaft, weil beim Spiegeln die Einträge mit -1 multipliziert werden (vgl. Gleichung (6.8)).

Bei der Betrachtung der Resultate (Abbildung 6.16) stellt man fest, dass insgesamt für ca. 20% der Fälle die lokale Orientierung falsch festgelegt wird. Besonders anfällig sind die Datensätze mit schlechter Datenqualität `C` und `D`.

Fazit: Von der Festlegung der lokalen Orientierung sollte bei schlechter Datenqualität Abstand genommen werden!

6.4.1 Zielscheibenfehler

Wie in Batista u. a. [4] zu recht hingewiesen wurde, ist es einfach einen Zielscheibenfehler zu begehen: Die Schlussfolgerung zu treffen, dass die Variante II besser bei den Datensätzen `B`, `C` und `D` abschneidet, nachdem man diese (unter Umständen zufällige) Ergebnisse in der Abbildung 6.16 gesehen hat. Dabei verfährt man ähnlich einem *Texas sharpshooter* (der Zielscheibenfehler ist in der englischen Literatur als *Texas sharpshooter fallacy* bekannt), der zuerst schießt und erst später das Ziel festlegt - nämlich dorthin, wo er getroffen hat.

Die schon vorgestellte Vorgehensweise beim Scatter-Plot kann benutzt werden, um die Hypothese zu widerlegen, dass es sich um einen Zufall handelt. Hier soll jedoch der sogenannte *Texas Sharpshooter Plot* vorgestellt werden, welcher schon in Batista u. a. [4] benutzt wurde.

Die Idee dieser Darstellung besteht darin, dass man anhand von wenigen Probebändern vorhersagen kann, wie sich die beiden Verfahren auf einem großen Datensatz verhalten werden. Dazu definieren wir das folgende Maß für zwei Varianten $A1$ und $A2$:

$$gain = \frac{E(recall_{A1})}{E(recall_{A2})}$$

Nun können wir den vorhergesagten Gewinn $gain_{part}$ aufgrund der Probebänder mit dem tatsächlich erzielten Gewinn $gain_{all}$ vergleichen, der für alle vorhandenen Bänder bestimmt wurde. Um auch hier die Wahrscheinlichkeit für ein zufälliges gutes Ergebnis zu reduzieren, werden die Probebänder (ca. 10% der gesamten Anzahl) mehrfach zufällig gewählt, während die restlichen Bänder zu $gain_{all}$ beitragen. Es ergeben sich folgende Bereiche:

RS: $gain_{part} < 1$ und $gain_{all} < 1$. Es wurde richtig vorhergesagt, dass $A1$ schlechter als $A2$ abschneidet.

RB: $gain_{part} > 1$ und $gain_{all} > 1$. Es wurde richtig vorhergesagt, dass $A1$ besser als $A2$ abschneidet.

FS: $gain_{part} > 1$ und $gain_{all} < 1$. Es wurde vorhergesagt, dass $A1$ besser als $A2$ abschneidet, aber beim großen Datensatz hat $A2$ besser abgeschnitten.

FB: $gain_{part} > 1$ und $gain_{all} < 1$. Es wurde vorhergesagt, dass $A1$ schlechter als $A2$ abschneidet, aber beim großen Datensatz hat $A1$ besser abgeschnitten.

Sollten die Ergebnisse nicht zufällig sein (also liegt kein Zielscheibenfehler vor), so würden die Punkte $(gain_{all},\ gain_{part})$ auf der Geraden $y = x$ liegen. Betrachtet man die Ergebnisse der Orientierungsfunktionen II und III für SIFT-Deskriptoren (Abbildung 6.17), so stellt man fest, dass kein Zielscheibenfehler vorliegt: für Testsätze `B`, `C` und `D` lassen sich kleine aber eindeutige Vorteile für Orientierungsfunktion **II** feststellen. Für die Datensätze `A` und `E` sind die Ergebnisse fast identisch - alle Punkte liegen nahe an $(1, 1)$. Für die anderen Vergleiche ergeben sich ähnliche Bilder, welche hier nicht angeführt werden.

6.5 Eindimensionale Detektoren

In diesem Abschnitt untersuchen wir die Performanz der Feature-Punkte-Detektoren. Als das Maß für ihre Performanz wird der im Abschnitt 6.2.2 (Verteilung der Abstände) eingeführte Faktor α_r verwendet, der den Anteil

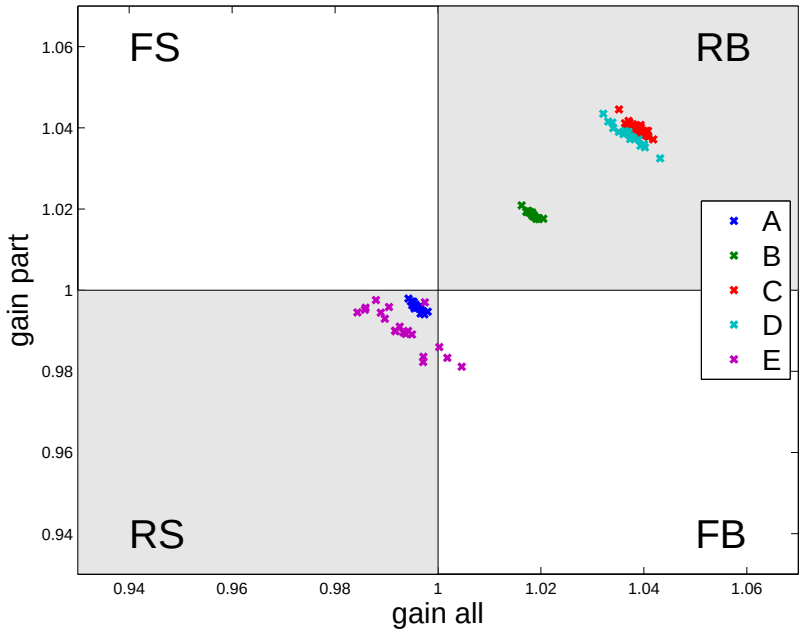

Abbildung 6.17: Das sogenannte Texas sharpshooter plot *zeigt, dass die Aussage "die Orientierungsfunktion II für SIFT-Deskriptoren performt besser als die Orientierungsfunktion III" nicht auf dem Zielscheibenfehler basiert, weil die Punkte* $(gain_{all},\ gain_{part})$ *sich gut durch die Gleichung* $y = x$ *beschreiben lassen.*

der wiedergefundenen Feature-Punkte beschreibt. Idealerweise sollte $\alpha_r = 1$ sein - ein Wert der in den realen Anwendungen unerreichbar bleibt.

Es wurde folgende Vorgehensweise hergeleitet, um $E(\alpha_r)$ zu bestimmen:

- Bestimme experimentell die Verteilungen F_w und F_{rr}. Berechne daraus $E(recall(\infty))$ mit der Formel (6.3).
- Bestimme experimentell $E(precision(\infty))$.
- Benutze

$$E(\alpha_r) = \frac{E(recall(\infty))}{E(precision(\infty))}$$

α_r kann nicht allein für sich betrachtet werden. Dies kann leicht mit dem folgenden Gedankenexperiment ad absurdum geführt werden:

Nehme jedes Element der Zeitreihe als Feature-Punkt, was zum idealen Wert $\alpha_r = 1$ führen würde. Dies ist nicht der bestmögliche Detektor, deswegen sollte der Faktor α_r in Abhängigkeit von der Anzahl der Feature-Punkte n, d.h. $\alpha_r(n)$ untersucht werden (dabei gehen wir davon aus, dass die Anzahl der Features in beiden Messungen ungefähr gleich ist).

Die mittlere Anzahl der Feature-Punkte n kann dadurch beeinflusst werden, dass man nur einen Anteil der lokalen Maxima und Minima betrachtet (siehe Abschnitt 5.2).

Wir untersuchen folgende Detektoren:

- LoG-Detektor (auch als *Mexican Hat* oder *Ricker-Wavelet* bekannt),
- DoG-Detektor,
- Box-Detektor (Annäherung von LoG durch Boxen),
- Cauchy-Detektor (Wavelet basiert auf $\phi = \frac{1}{\pi(1+x^2)}$),
- Wavelet basierend auf $\phi = x^2 \cdot G(x, \sigma)$, das wir als X2 oder x^2Gauß-Detektor bezeichnen.

Da natürlich diese Vorgehensweise fehlerbehaftet ist, bietet sich an, sie für unterschiedliche Deskriptoren durchzuführen, sodass man den Wert α_r genauer abschätzen kann.

6.5.1 Ergebnisse mit realen Daten

Wir benutzen die oben beschriebene Vorgehensweise für SIFT-, CC- und CCD-Deskriptoren um die Menge $A := \{\alpha_r^{SIFT}, \alpha_r^{CC}, \alpha_r^{CCD}, \}$ zu berechnen. Diese Menge wird in den Abbildungen 6.18-6.19 wie folgt dargestellt:

- $\alpha_r = \frac{1}{|A|} \sum_{a \in A} a$,
- Die angezeigten Fehler sind $\min A$ und $\max A$.

Die Anzahl der Features wird dadurch variiert, dass man die Nachbarschaftsbeziehung ändert: Bei der Ermittlung der Extrema werden nicht nur die direkten Nachbarn betrachtet, sondern alle Punkte mit dem zeitlichen Abstand $\leq \delta_N$. Bei den vorliegenden Experimenten wurde $\delta_N = 1$, 2, 4, 6, 8, 10 und 15 gewählt (mehr Details in Abschnitt 5.2).

Man kann Folgendes feststellen:

- *Boxes*- und x^2Gauß-Detektoren sorgen für viele lokale Extrema, was nicht verwunderlich ist, weil sie sich als keine so gute (z.B. im Vergleich zum DoG-Detektor) Tiefpassfilter erweisen.
- Der Cauchy-Detektor produziert die instabilsten Feature-Punkte und sollte daher nicht mehr von Interesse sein.
- Der DoG-Detektor ist eine gute Näherung an den LoG-Detektor und liefert sogar teilweise stabilere Features.
- Boxes- und x^2Gauß-Detektoren liefern die stabilsten Features.

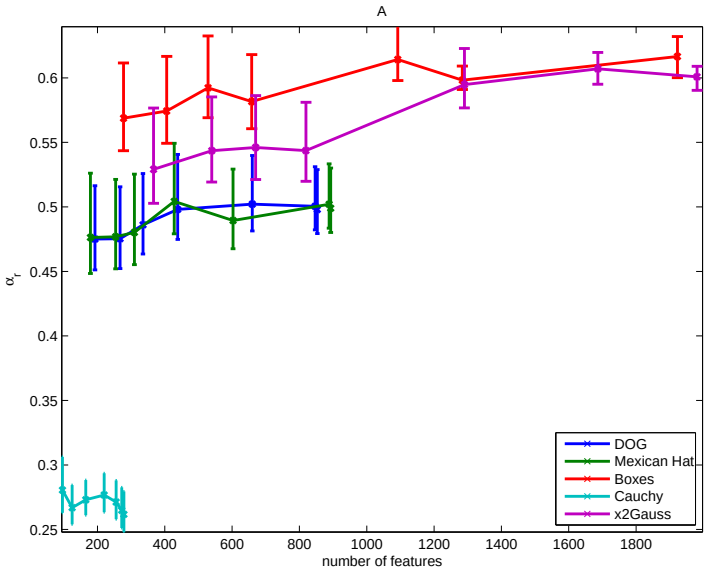

Abbildung 6.18: Der Koeffizient α_r für unterschiedliche Detektoren in Abhängigkeit von der Anzahl der gewählten Features für den Testdatensatz `A`.

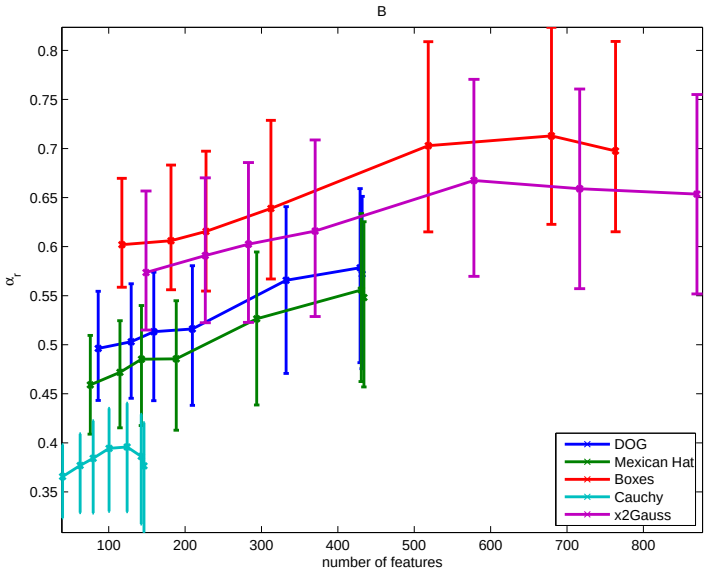

Abbildung 6.19: Der Koeffizient α_r für unterschiedliche Detektoren in Abhängigkeit von der Anzahl der gewählten Features für den Testdatensatz `B`.

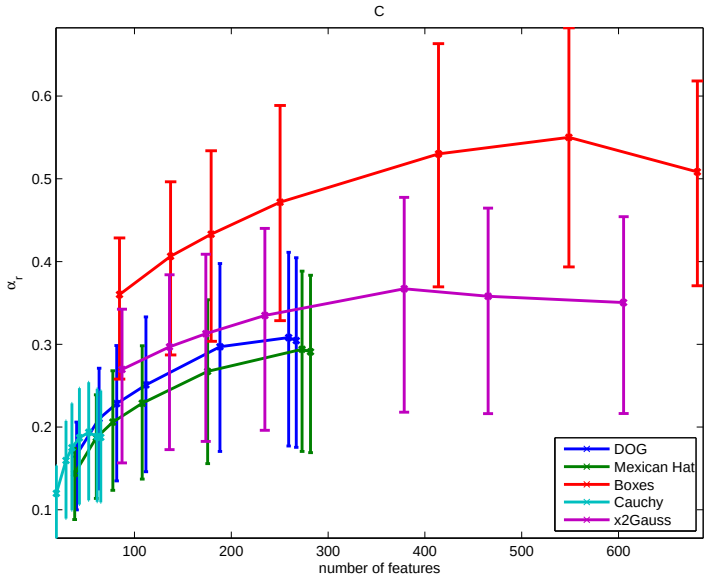

Abbildung 6.20: Der Koeffizient α_r für unterschiedliche Detektoren in Abhängigkeit von der Anzahl der gewählten Features, angewandt auf den Testdatensatz C.

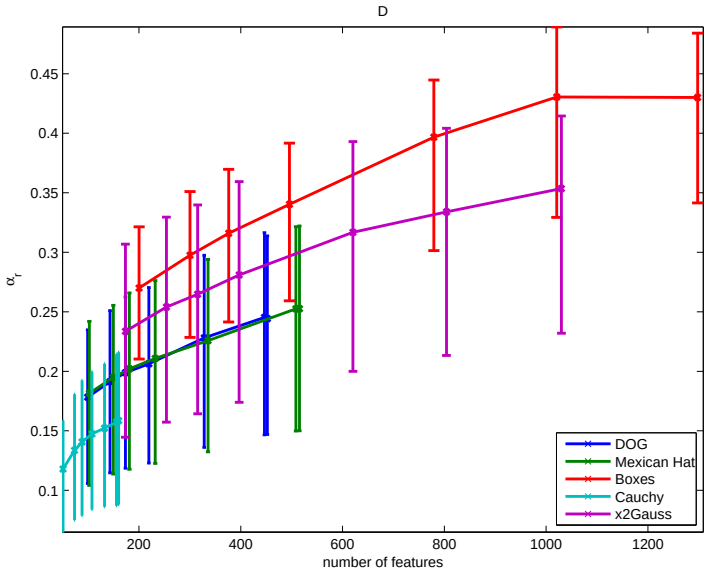

Abbildung 6.21: Der Koeffizient α_r für unterschiedliche Detektoren in Abhängigkeit von der Anzahl der gewählten Features, angewandt auf den Testdatensatz D.

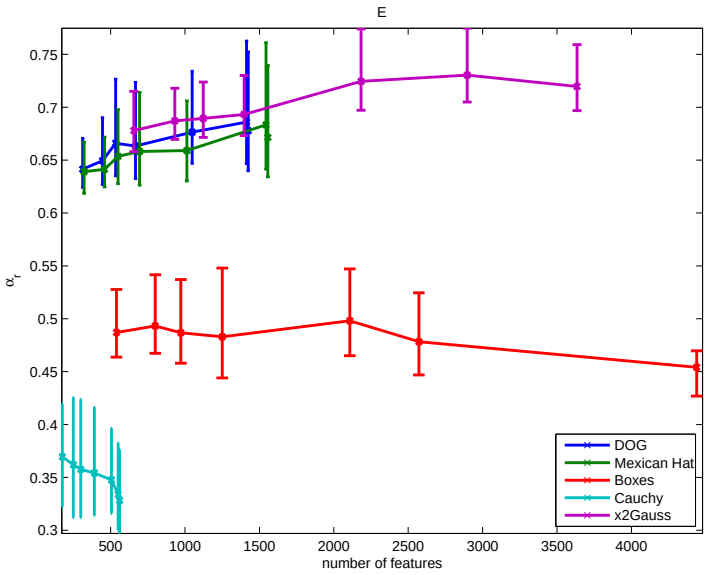

Abbildung 6.22: Der Koeffizient α_r für unterschiedliche Detektoren in Abhängigkeit von der Anzahl der gewählten Features, angewandt auf den Testdatensatz E.

6.6 Skalierungsinvarianz

Die Datensätze, mit deren Hilfe wir bis jetzt die Performanz des Frameworks untersuchten, haben eine Reihe von Fehlern vorzuweisen, angefangen mit "einfachem" Gaußrauschen, über Trends bis hin zu Messstörungen und Messausfällen. Die vorhandenen Skalierungen der x-Achse wichen aber nur um wenige Prozente von der Identität 1.00 ab. In diesem Abschnitt wollen wir das Verhalten des Frameworks für größere Skalierungen untersuchen. Dafür skalieren wir die Kinderbänder mit einem Skalierungsfaktor $scale \in [1, 2]$ und erzeugen damit "halbsynthetische" Datensätze.

Als erstes untersuchen wir die Anzahl von richtig zugeordneten Feature-Punkten in Abhängigkeit von der Skalierung *scale*. Im weiteren Verlauf möchten wir uns der Übersichtlichkeit halber auf den Datensatz `A` beschränken, der das typische Verhaltensmuster zeigt (Abbildung 6.23).

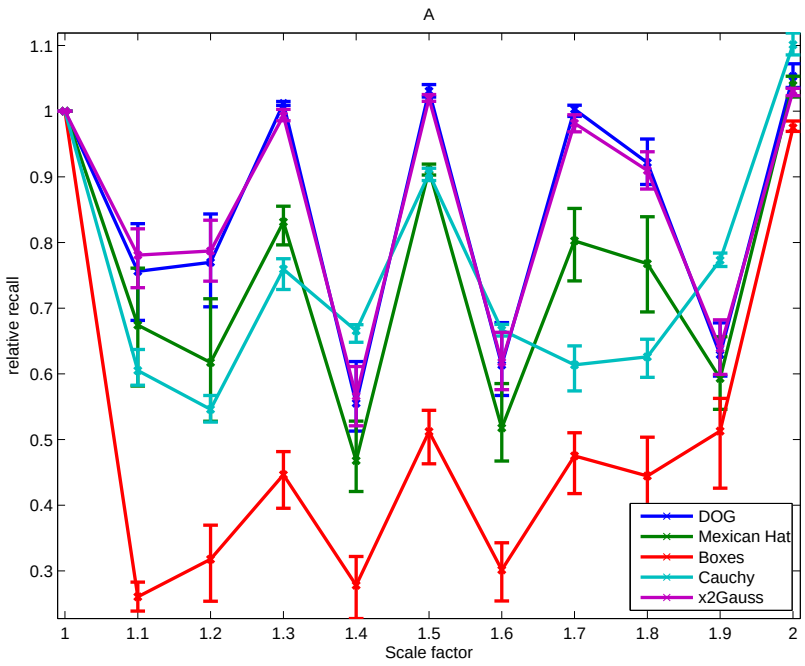

Abbildung 6.23: Es wird das Verhältnis der recall-Rate zur recall-Rate bei scale = 1.00 dargestellt. Es gibt einige Einbrüche des recalls für bestimmte Skalierungen.

Für die leicht schlechtere Performanz bei bestimmten Skalierungen kommen folgende Ursachen in Frage:

A: Der Detektor ist nicht im Stande, die gleichen Features für skalierte Versionen zu detektieren.

B: Für die Berechnung der Umgebung wird nicht $F(x, \sigma_{extremum})$ sondern

$F(x, \sigma')$ mit $\sigma' \approx \sigma_{extremum}$ verwendet. Dies führt zu Fehlern, die zu den schlechteren Ergebnissen führen.

C: Die σ-Koordinate des Extremums wurde falsch berechnet und dadurch auch die Umgebung, die zur Berechnung des Deskriptoren verwendet wird.

D: Lässt sich auf die Fehler bei der Berechnung der skalierten Versionen zurückführen.

Die Ursache **D** kann als Hauptproblem jedoch ausgeschlossen werden, da man damit nicht erklären kann, warum die Performanz für bestimmte Skalierungen schlecht ist.

Zuerst widmen wir uns der Hypothese **A** und untersuchen nebenbei welcher Detektor am besten mit den größeren Skalierungen klar kommt.

Um ungefähr gleiche Anzahl der Features für alle Detektoren zu haben und damit gerechte Bedingungen zu schaffen, wird $\delta_N = 1$ für LoG-, DoG- und Cauchy-Detektoren gewählt, für Boxes- und x^2Gauß-Detektoren wird δ_N auf 6 gesetzt.

Die Abbildungen 6.24 - 6.27 lassen folgende Schlüsse zu:

- Während der Boxes-Detektor für große Skalierungen nicht geeignet zu sein scheint, kann der x^2Gauß-Detektor seinen Vorsprung gegenüber den DoG- und LoG-Detektoren über alle Skalierungen erhalten.
- Die schlechte Performanz der Detektoren ist verantwortlich für die Einbrüche in der *recall*-Rate für den Testsatz `A` (vergleiche Abbildung 6.23).

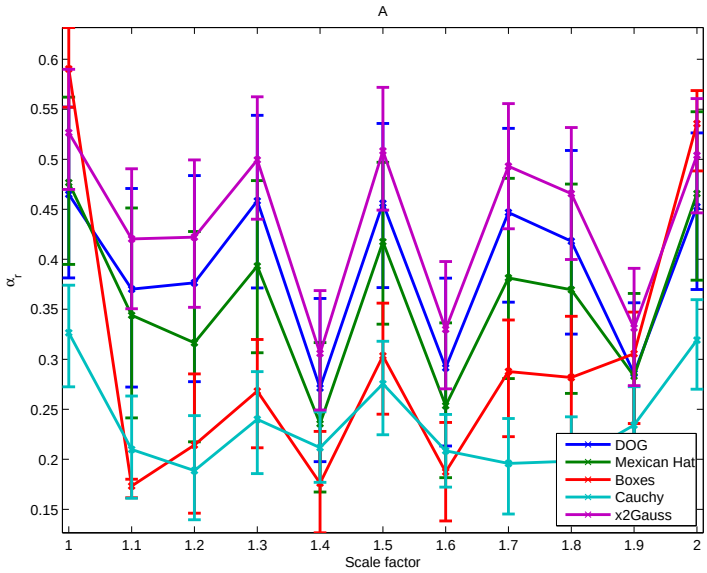

Abbildung 6.24: Der Koeffizient α_r für unterschiedliche Detektoren in Abhängigkeit vom Skalierungsfaktor, dargestellt für den Testdatensatz `A`.

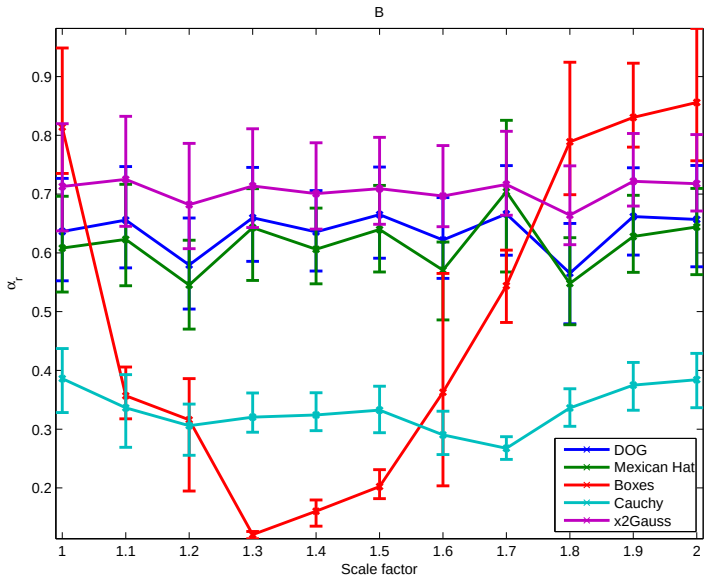

Abbildung 6.25: Der Koeffizient α_r für unterschiedliche Detektoren in Abhängigkeit vom Skalierungsfaktor, dargestellt für den Testdatensatz `B`.

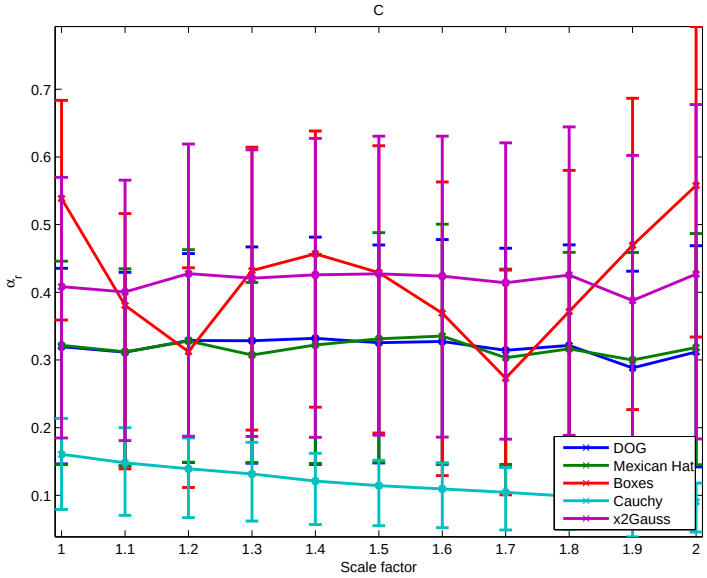

Abbildung 6.26: Der Koeffizient α_r für unterschiedliche Detektoren in Abhängigkeit vom Skalierungsfaktor, dargestellt für den Testdatensatz `C`.

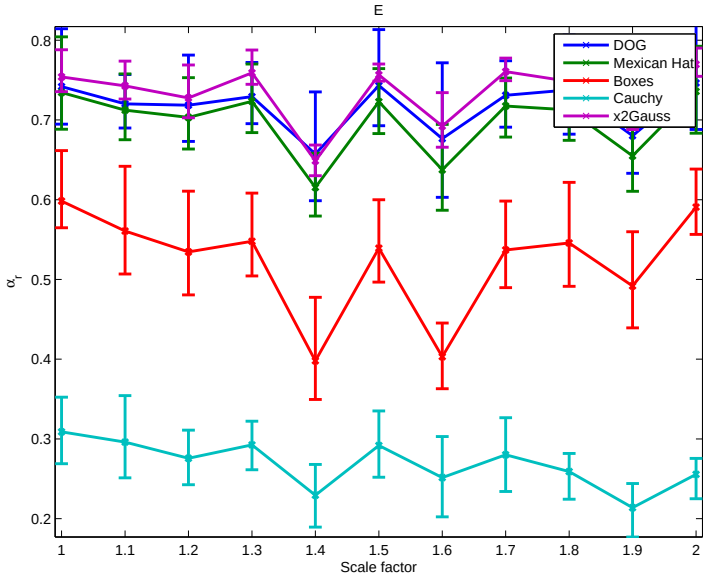

Abbildung 6.27: Der Koeffizient α_r für unterschiedliche Detektoren in Abhängigkeit vom Skalierungsfaktor, dargestellt für den Testdatensatz `E`.

Um die Ursache **B** auszuschließen, untersuchen wir die Verteilungen der Abstände der Deskriptoren f_r und f_w, wie sie schon in Abschnitt 6.2.2 (Verteilung der Abstände) definiert wurden, für unterschiedliche Skalierungen $scale = 1.00$, 1.40 und 1.90. Die Abbildung 6.28 zeigt, dass die vorhandenen Unterschiede nicht groß genug sind, um die Einbrüche der Performanz zu erklären. Deswegen kann die Hypothese **B** verworfen werden.

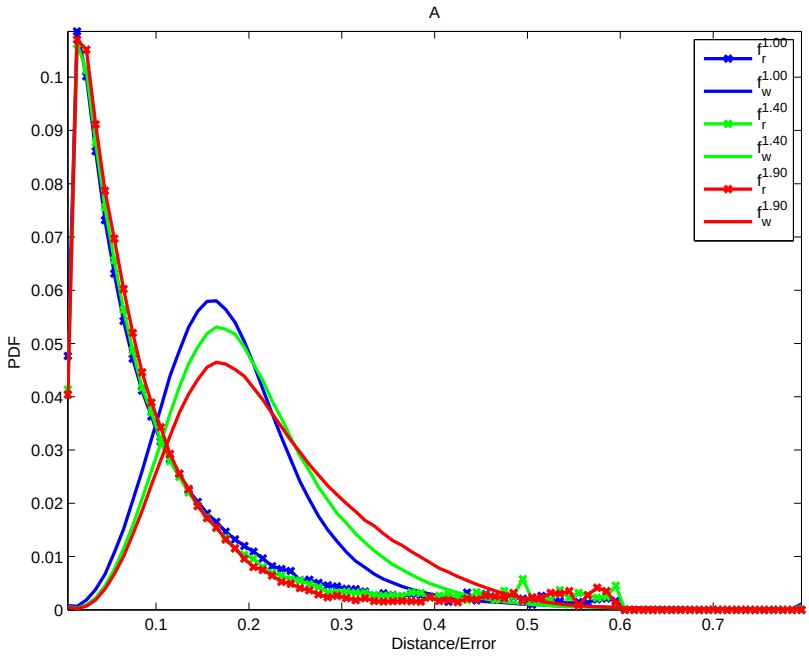

Abbildung 6.28: Verteilungsdichten f_r und f_w für Skalierungen 1.00, 1.40 *und* 1.90 *für den Testdatensatz* `A`.

Um festzustellen, welchen Einfluss die Ursache **C** hat, wollen wir die Verteilungen von $k := \frac{\sigma_C}{\sigma_P}$ für die richtig zugeordneten Feature-Punkte (x_C, σ_C) und (x_P, σ_P) betrachten. Im Idealfall würde es sich nach der Bereinigung mit der Steigung der gefundenen Koordinatentransformation α um eine Konstante $k = 1$ handeln, was selbstverständlich bei unseren Daten nicht der Fall sein wird. Unterschiedliche Verteilungen für k^{scale} für verschiedene Skalierungsfaktoren *scale* würden darauf deuten, dass die Ursache in der fehlerhaften Berechnung der σ-Koordinate der Feature-Punkte liegt.

Die gefundenen Verteilungen für Skalierungen 1.00, 1.40 und 1.90 (Abbildung 6.29) sprechen dafür, dass die Ursache **C** keine große Rolle spielen kann.

All diese Überlegungen legen nahe, dass die Ursache `A` für das Verhalten des *recalls* in der Abbildung 6.23 verantwortlich ist. Man kann dem zu einem gewissen Grad entgegenwirken, indem man die Anzahl der berechneten Levels (k) pro Oktave erhöht.

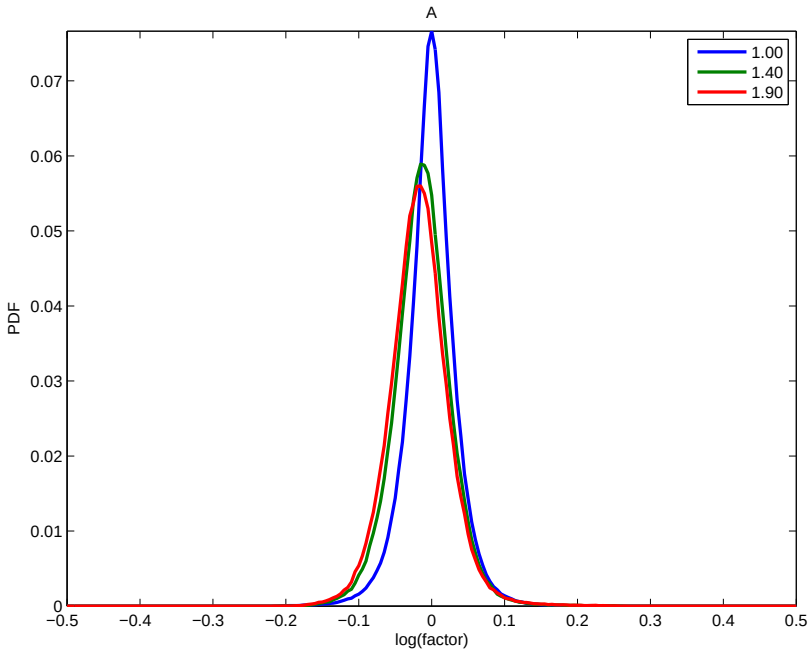

Abbildung 6.29: Die Fehler der σ-Koordinaten scheinen für keine der gewählten Skalierungen dramatisch zu sein.

6.7 Nearest-Neighbor-Anfragen

Die meiste Rechenzeit beim Abgleich zweier Fingerprints wird für die Suche des nächsten Nachbarn für die Deskriptoren verbraucht. Obwohl es für die niederdimensionalen Fälle durchaus effiziente Algorithmen gibt (z.B. kd-Bäume (Friedman u. a. [28])) ist es bekannt, dass diese Algorithmen für hochdimensionale Daten nicht effizienter als die einfache naive Suche sind. Dieser Umstand wird in der Literatur oft als *"the curse of dimensionality"* bezeichnet.

Diese Erkenntnis führte zur Entwicklung von approximativen Nearest-Neighbor-Anfragen (wir werden dies auch als *NN-Anfrage* bzw. *NN-Problem* bezeichnen): Diese Anfragen sind zwar zum Teil deutlich schneller, jedoch garantieren sie nicht, immer den nächsten Nachbarn zurückzuliefern.

Es stellt sich für uns die Frage: Sollten wir die Features/Deskriptoren eines Fingerprints in einer Datenstruktur organisieren, die eine schnelle Suche nach dem bestpassenden Deskriptor ermöglicht? Es wäre unproblematisch, wenn z.B. nur für 95% der Anfragen der nächste Nachbar und damit der bestpassende Deskriptor gefunden wird: Der nächste Nachbar führt eh nicht immer zur richtigen Zuordnung! Diese Verschlechterung würde kaum ins Gewicht fallen, während die Vorgehensweise viel Rechenzeit sparen könnte.

Wir betrachten folgende Strategien:

- Naive Strategie, bei der während der Suche nach dem nächsten Nachbarn ein Vergleich mit allen Deskriptoren durchgeführt wird. Dabei ist es möglich, falls die Skalierung zwischen den beiden Zeitreihen bekannt ist, die Anzahl der Vergleiche zu reduzieren, indem man einige Feature-Punkte anhand ihrer σ-Koordinate aussortiert. Diese Methode liefert immer den richtigen Nachbarn und dient uns als Goldstandard.
- Randomisierte Linked Cells (RLC).
- R*-Bäume (Beckmann u. a. [6]).
- Local Sensitive Hashtables (Datar u. a. [22], Gionis u. a. [31]).
- Randomisierte k-d-Bäume (Muja und Lowe [52]).

6.7.1 Local Sensitive Hashtables

Während die Locality-Sensitive Hash (LSH) Funktionen zuerst mit Erfolg für Hamming-Abstände entwickelt wurden (Gionis u. a. [31]), folgend wir in unseren Ausführungen Datar u. a. [22]. Die Autoren benutzen p-stabile Distributionen um Locality-Sensitive Hashfunktionen zu erzeugen.

Wir betrachten eine Familie von Hashfunktionen $\mathcal{H} := \left\{h : \mathbb{R}^d \to \mathbb{Z}^k\right\}$ mit einer Wahrscheinlichkeit P_H, die angibt, wie diese gewählt werden.

Für zwei Vektoren $u, v \in \mathbb{R}^d$ (beliebig aber fest) stellt sich die Frage nach den Werten folgender bedingter Wahrscheinlichkeiten:

$$P_H\left(h(u) = h(v) | \left\|u - v\right\|_p \leq \varepsilon\right) =: p_\varepsilon$$
$$P_H\left(h(u) = h(v) | \left\|u - v\right\|_p \geq \delta\right) =: p_\delta$$

Von Interesse sind solche Familien von Hashfunktionen, für die gilt:

$$\varepsilon < \delta \Rightarrow p_\varepsilon > p_\delta.$$

D.h. je großer der Abstand zwischen den Vektoren, desto unwahrscheinlicher soll es sein, dass sie den gleichen Hashwert zugewiesen bekommen.

In Datar u. a. [22] wurde vorgeschlagen,

$$\hat{h}_{a,b}(v) := \frac{a \cdot v + b}{w}$$
$$h_{a,b}(v) := \left\lfloor \hat{h}_{a,b}(v) \right\rfloor$$

für ein festes $w \in \mathbb{R}_{>0}$, $a \in \mathbb{R}^d$ und $b \in [0, w)$ als Hashfunktionen ($k = 1$) zu benutzen. Während die Komponenten von a $a_i \overset{d}{=} F_p \ \forall i$ i.i.d. gewählt werden mit F_p eine p-stabile Verteilung, wird b aus der uniformen Verteilung auf $[0, w)$ gezogen. Damit ist auch implizit die Wahrscheinlichkeitsverteilung auf $\mathcal{H}$ gegeben.

Die Verteilung F_p für $p > 0$ heißt p-stabil, falls für $a_1, \ldots, a_d \overset{d}{=} F_p$ i.i.d. und $\forall v \in \mathbb{R}^d$ gilt:

$$\sum_{i=1}^{d} v_i a_i \overset{d}{=} \|v\|_p \, a_1$$

Wir konzentrieren uns auf $p = 2$. Man sieht leicht ein, dass die Normalverteilung eine 2-stabile Verteilung ist: Seien $a_i \overset{d}{=} N(0, 1)$, dann gilt wegen der Symmetrie der Normalverteilung:

$$v_i a_i \overset{d}{=} |v_i| a_i \overset{d}{=} N(0, |v_i|).$$

Das Skalarprodukt ist dann verteilt wie die Faltung von d Gaußfunktionen (für die Eigenschaften der Gaußfuntionen in Verbindung mit Faltung siehe Anhang 10.3.3):

$$\begin{aligned} v \cdot a &\overset{d}{=} N(0, |v_1|) * \cdots * N(0, |v_d|) \overset{d}{=} N\left(0, \sqrt{\sum |v_i|^2}\right) \\ &\overset{d}{=} \|v\|_2 \, a_1 \end{aligned} \tag{6.9}$$

Nun untersuchen wir die Frage, wie groß die Wahrscheinlichkeit ist, dass beide Vektoren u, v den gleichen Hashwert besitzen, vorausgesetzt $\|u - v\|_2 = c$. Zuerst betrachten wir den Fall, dass $|(v - u) \cdot a| = \tau$ mit $0 \leq \tau \leq w$. Es ist klar, dass für $\tau > w$ die Hashwerte der beiden Vektoren niemals gleich sein werden.

Des Weiteren, weil b zufällig und uniform gewählt wurde und damit auch die Grenzen der Bins gleichverteilt sind, kommt es nur auf den Abstand τ und nicht auf die Werte von $v \cdot a$ und $u \cdot a$ an. Also beträgt die bedingte Wahrscheinlichkeit:

$$P(u, v \text{ im gleichen Bin} || (v - u) \cdot a| = \tau) = \begin{cases} 1 - \frac{\tau}{w} & 0 \leq \tau \leq w \\ 0 & \text{sonst} \end{cases}$$

Da $a_i \overset{d}{=} N(0, 1)$ gewählt wurde, gilt nach Formel (6.9)

$$\tau \overset{d}{=} N(0, \|v - u\|_2) = N(0, c)$$

und damit beträgt die Gesamtwahrscheinlichkeit im gleichen Bin zu landen:

$$\begin{aligned} p_c(w) :&= P(u, v \text{ im gleichen Bin } |c = \|u - v\|_2) \\ &= \int_{\mathbb{R}} P(u, v \text{ im gleichen Bin } ||(v-u)\cdot a| = \tau) G(\tau, c)\, d\tau \\ &= \int_{-w}^{w} G(\tau, c) \cdot \left(1 - \frac{|\tau|}{w}\right) d\tau \end{aligned}$$

Einfachheitshalber nehmen wir an, dass die richtig zugeordneten Deskriptoren den Abstand ε und die falsch zugeordnete Deskriptoren den Abstand δ aufweisen.

So kann man $p_\varepsilon(w)$ - die Wahrscheinlichkeit, dass die passenden Vektoren in einem Bin landen - erhöhen, indem man den Parameter w vergrößert. Dafür bezahlt man jedoch auch mit der größeren Wahrscheinlichkeit für falsche Treffer $p_\delta(w)$.

Welche Möglichkeit hat man noch? Es ist möglich, bei festem w nicht nur eine, sondern mehrere Hash-Tabellen zu benutzen. Die Ergebnisse jeder einzelnen Abfrage werden zu einer gesamten Antwort vereinigt. Wir gehen davon aus, dass die Hashfunktionen der unterschiedlichen Tabellen stochastisch unabhängige Hashwerte berechnen. Damit beträgt die Wahrscheinlichkeit, dass die Abfrage das richtige Ergebnis liefert, d.h. dass der nächste Nachbar in der Gesamtantwort der Anfrage vorhanden ist:

$$recall(n, w) = 1 - (1 - p_\varepsilon(w))^n,$$

mit n - Anzahl der Tabellen. Dafür wird ein nicht passender Vektor mit Wahrscheinlichkeit

$$checked(n, w) = 1 - (1 - p_\delta(w))^n$$

in der Gesamtantwort enthalten sein. Zu vorgegebener *recall* versuchen wir nun n und w so zu wählen, dass $checked(n, w)$ möglichst klein ausfällt. Numerische Berechnungen liefern dann unter der Annahme $\varepsilon = 0.1$ und $\delta = 0.6$ Ergebnisse, die in Abbildung 6.30 zusammengefasst wurden.

Man sieht, dass sogar bei der günstigsten Wahl von $w \approx 0.20$ man immer noch ca. 35% der Features vergleichen muss, und so scheint die LSH-Methode keinen großen Gewinn zu bringen.

Es werden oft auch mehrdimensionale Hashfunktionen

$$h_{a,b} : \mathbb{R}^d \to \mathbb{Z}^k$$

$$h_{a,b}(v) = \begin{pmatrix} \left\lfloor \frac{a\cdot v + b_1}{w} \right\rfloor \\ \vdots \\ \left\lfloor \frac{a\cdot v + b_k}{w} \right\rfloor \end{pmatrix}$$

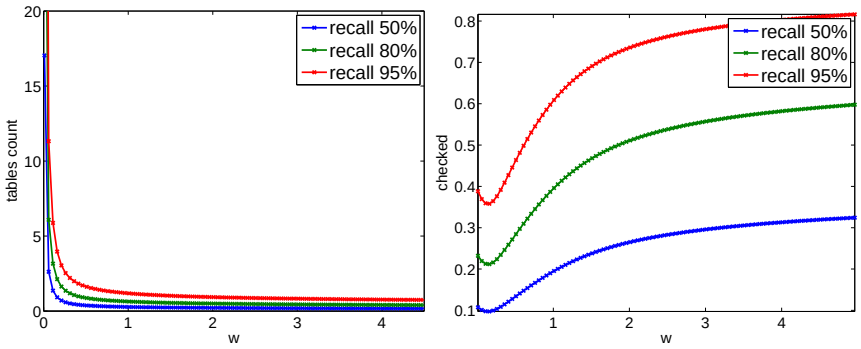

Abbildung 6.30: Aus den vorgegebenen w und $recall(n)$ lässt sich n - die Anzahl der notwendigen Tabellen (linkes Bild)und anschließend auch checked (rechtes Bild) berechnen. Vor allem ist $recall = 95\%$ (rot) von Interesse. Die beste Wahl für w ist ≈ 0.2, dann müssen nur ca. 35% der Deskriptoren verglichen werden.

benutzt. Dabei wird $b \in [0, w)^k$ zufällig nach uniformer Verteilung gewählt.

Man kann auch für diese Version der Hashfunktionen analoge Überlegungen anstellen, es ändert sich dabei nur die bedingte Wahrscheinlichkeit (für ein festes k):

$$P(u, v \text{ im gleichen Bin } ||(v-u)\cdot a| = \tau) = \begin{cases} (1-\frac{\tau}{w})^k & 0 \le \tau \le w \\ 0 & \text{sonst} \end{cases}.$$

Dies führt zu ähnlichen Resultat wie in der Abbildung 6.30, auf die hier nicht eingegangen wird.

6.7.2 Randomisierte Linked Cells

Linked Cells (siehe Bentley u. a. [7])Randomisierte linked cells werden mit Erfolg bei der Simulation von Mehrkörpersystemen in 3d benutzt: Dabei wird der Raum in mehrere Quader (Zellen) der Breite δ unterteilt. Bei der Suche betrachtet man nur die Elemente in der eigenen Zelle so wie in den Nachbarzellen. Die Anzahl der Nachbarzellen beträgt jedoch $3^d - 1$ und dieses Verfahren kommt für unsere Deskriptorengrößen nicht in Frage.

Wir möchten nun das Konzept der *randomisierten Linked Cells* vorstellen: Die Anfrage wird nur in der eigenen Zelle durchgeführt, deren Koordinaten $I(v) \in \mathbb{Z}^d$ für $v \in \mathbb{R}^d$ wie folgt komponentenweise berechnet werden:

$$I_b^i(v) := \left\lfloor \frac{v_i + b_i}{\delta} \right\rfloor \quad \forall i = 1, \ldots, k.$$

wobei b ähnlich, wie bei LSH, zufällig nach uniformer Verteilung aus $[0, \delta)^d$ gewählt wird. Damit wird eine Hashfunktionen-Familie $\mathcal{H} = \{h : \mathbb{R}^d \to \mathbb{Z}^d \,|\, I_b(v)\}$ definiert.

Es werden mehrere Hashtabellen benutzt, wobei die Gesamtantwort aus der Vereinigung von Antworten der einzelnen Tabellen besteht.

Im Prinzip kann man ähnliche Berechnungen wie im oben vorgestellten LSH-Fall durchführen. Wir möchten uns jedoch nur auf die praktischen Ergebnisse beschränken, welche in 6.7.5 vorgestellt werden.

6.7.3 R*-Baum

Die von Guttman (Guttman [33]) vorgestellten R-Bäume stellen eine Möglichkeit dar, für einen vorgegebenen Quader $Q_q := I_1 \times \cdots \times I_d \subset \mathbb{R}^d$ und die Menge der Vektoren $V \subset \mathbb{R}^d$ den Schnitt $V \cap Q_q$ zu berechnen. Dabei wird die Menge V bei der Vorverarbeitung in einer Baumstruktur angeordnet.

Offensichtlich kann diese Struktur für eine approximative NN-Suche verwendet werden, indem man den Anfragevektor $v \in \mathbb{R}^d$ in den Anfragequader

$$Q_q = [v_1 - \frac{\delta}{2}, v_1 + \frac{\delta}{2}] \times \cdots \times [v_d - \frac{\delta}{2}, v_d + \frac{\delta}{2}]$$

umwandelt.

Die zurückgelieferte Menge $Q_q \cap V$ muss anschließend linear untersucht werden.

Natürlich ist nicht gewährleistet, dass man den nächsten Nachbarn findet, falls der Abstand größer als $\frac{\delta}{2}$ beträgt. Ist der Abstand größer als $\sqrt{d}\frac{\delta}{2}$, so wird der nächste Nachbarn sicherlich nicht gefunden.

In dieser Hinsicht verhält sich der R-Baum ähnlich wie die Linked Cells, hat aber den Vorteil, dass der Abfragequader bei jeder Abfrage flexibel gewählt werden kann. Bei Linked Cells dagegen wird der Abfragequader zum Zeitpunkt der Vorverarbeitung gewählt.

In unserer Implementierung folgen wir der Implementierung von R*-Bäumen beschrieben in Beckmann u. a. [6]. Ein R*-Baum unterscheidet sich lediglich durch die Vorverarbeitung-Routine von der originalen R-Baum-Struktur, weist aber überlegene Performanz auf, wie es in Beckmann u. a. [6] gezeigt wurde.

Funktionsweise

Beim R*-Baum bzw. R-Baum handelt es sich um eine B-Baum-ähnliche Datenstruktur (Mehr über B-Bäume findet man in Cormen u. a. [20]):

- Auf der untersten Ebene befinden sich die Blätter, die die Vektoren $v \in V$ beinhalten.
- Ein Vaterknoten kann m bis M Kinder haben, wobei $2 \leq m \leq \frac{M}{2}$ erfüllt sein muss. Die einzige Ausnahme kann die Wurzel darstellen.

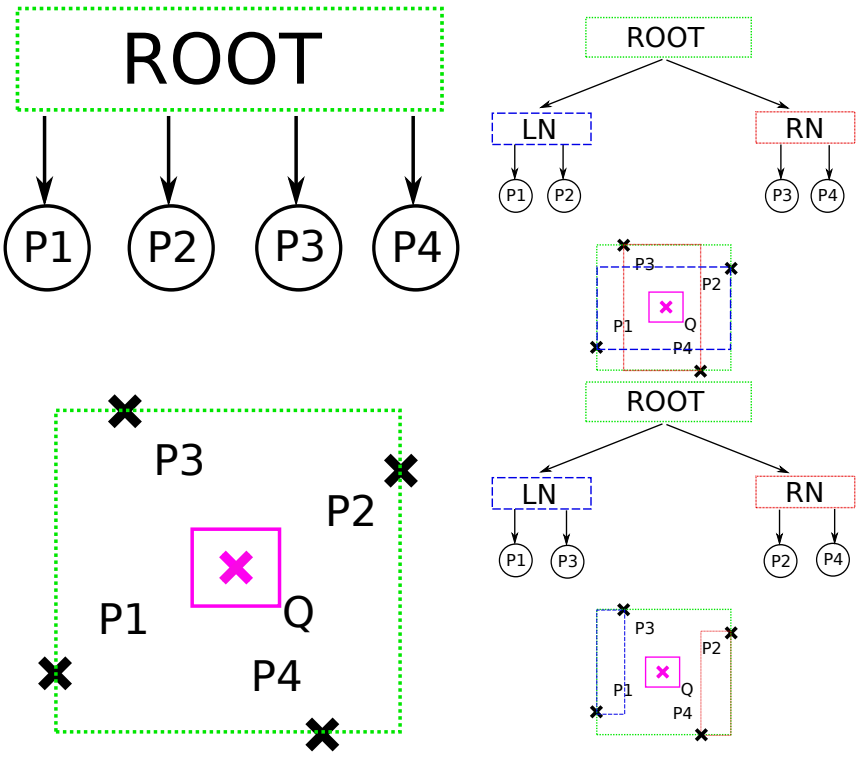

Abbildung 6.31: Es gibt mehrere Möglichkeiten einen Knoten zu spalten: Die obige ist nicht so gut, wie man anhand der Anfrage Q sehen kann, denn nicht nur ROOT*, sondern auch die Kinderknoten* LN *und* RN *müssen besucht werden. Die Möglichkeit unten ist besser - bei der Abfrage muss in keinem der Kinderknoten gesucht werden.*

- Jeder Knoten gibt den möglichst engen achsenparallelen Quader Q an, der alle seine Kinder umschließt.
- Bei der Abfrage mit Q_q werden nur die Knoten mit $Q \cap Q_q \neq \emptyset$ weiter untersucht.
- Einfügen eines Elements $v \in V$ wird analog zu den B-Bäumen durchgeführt: Ein neues Blatt wird erstellt, überfüllte Knoten werden gesplittet, was sich bis zur Wurzel propagieren kann.

Der wichtige Punkt ist, wie man die Elemente beim Spalten eines Knotens aufteilt bzw. zu welchen Knoten ein neues Element hinzugefügt wird.

Die zweite Frage ist recht einfach zu beantworten: Soll ein neues Element hinzugefügt werden, so wird rekursiv, mit der Wurzel beginnend, derjenige Knoten gewählt, für den das Hinzufügen des Elementes die kleinste Änderung des Volumens des Quaders Q bewirkt.

Wie entscheidend der richtige Spaltung des Knotens ist, wird in der Abbildung 6.31 illustriert. Für die genaue Vorgehensweise und weitere Details verweisen wir auf Beckmann u. a. [6].

6.7.4 Randomisierte kd-Bäume

Die kd-Bäume zum Lösen des NN-Problems wurden in Friedman u. a. [28] vorgestellt. Diese Datenstruktur gehört zu den Standardwerkzeugen für die NN-Suche in niederdimensionalen Räumen. Die Performanz verschlechtert sich jedoch rapide mit wachsender Anzahl der Dimensionen: Im hochdimensionalen Raum müssen im Normalfall viele Zweige untersucht werden, um sicher zu stellen, dass man wirklich den nächsten Nachbarn gefunden hat.

Silpa-Anan und Hartley [59] haben vorgeschlagen, mehrere randomisierte kd-Bäume zu erzeugen und bei der Suche nur die viel versprechenden Zweige zu untersuchen, deren Anzahl beschränkt werden kann.

Worin besteht nun der Unterschied zwischen einem "normalen" und einem randomisierten kd-Baum? Bei einem "normalen" kd-Baum werden die Punkte immer anhand der Dimension gespalten, welche die größte Varianz aufweist. Beim randomisierten kd-Baum wird diese Dimension zufällig gewählt, wobei allerdings nur die Dimensionen mit größerer Varianz in Frage kommen.

In unserer Implementierung folgen wir dem Abschnitt über die randomisierten kd-Bäume in Muja und Lowe [52].

6.7.5 Auswertung der Datenstrukturen

Nun soll die Performanz der oben dargestellten Datenstrukturen angewandt auf unsere Testdatensätze betrachtet werden.

Zusätzlicher Speicherbedarf

Wir fassen die Erkenntnisse aus den vorherigen Betrachtungen zusammen:

Datenstruktur	Linear	LSH	RLC	kd-Baum	R*-Baum
Speicherbedarf	-	$O(n \cdot N)$	$O(n \cdot N)$	$O(N)$	$O(d \cdot N)$

mit N- Anzahl der Elemente in der Datenstruktur, n - Anzahl der Tabellen bei LSH bzw. Random Cell und d - die Dimensionenanzahl der abgespeicherten Elemente.

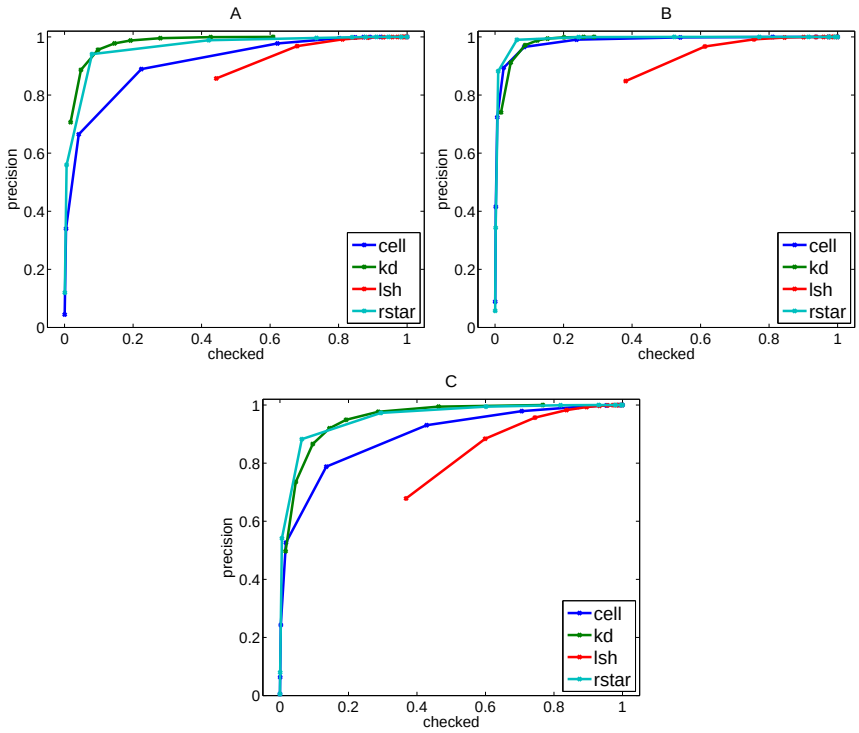

Abbildung 6.32: Die randomisierten kd-Bäume und der R-Baum zeigen das beste asymptotische Verhalten: Für den Datensatz B ist es notwendig nur ca. 10% der Elemente im Schnitt anzuschauen, um bei der Suche nach dem nächsten Nachbarn in 99% der Fälle das richtige Ergebnis zu erhalten. Damit wäre ein 10-facher speed-up möglich. Wie im theoretischen Teil schon hergeleitet, kann man von LSH nicht wesentlich profitieren.*

Asymptotisches Verhalten

In diesem Abschnitt möchten wir den Overhead, der bei der Benutzung dieser Datenstrukturen auftritt, vernachlässigen und uns nur auf die folgende Frage konzentrieren: Welchen Anteil der Elemente in der Datenstruktur *checked* muss man sich anschauen, wenn man bei der Suche nach dem nächsten Nachbarn einen bestimmten *recall* Wert erreichen möchte?

Bei der linearen Suche ist die Lage einfach: Es kann davon ausgegangen werden, dass $checked = recall$. Andere Datenstrukturen machen nur dann Sinn, wenn die Bedingung $recall > checked$ erfüllt ist. Das Trade-Off zwischen *checked* und *recall* lässt sich durch das Variieren mehrerer Parameter beeinflussen.

Der Übersichtlichkeit halber konzentrieren wir uns auf die Datensätze `A`, `B`, `C` und die SIFT-Deskriptoren:

- SIFT-Deskriptoren für den Datensatz `A` bestehen aus 24 Einträgen und

die Daten weisen ein gute Qualität auf.

- SIFT-Deskriptoren für den Datensatz `B` bestehen aus nur 12 Einträgen und die Daten weisen eine ähnlich gute Qualität wie im Datensatz `A` auf.
- SIFT-Deskriptoren für den Datensatz `C` bestehen aus 24 Einträgen und die Daten sind von schlechter Qualität.

Die Ergebnisse unserer Experimente werden in der Abbildung 6.32 dargestellt: Die randomisierten kd-Bäume und der R*-Baum sind anderen Ansätzen überlegen.

Laufzeiten

Nun vergleichen wir die wirklichen Laufzeiten, die beim Vergleich der Deskriptoren verbracht werden. Dafür wählen wir Einstellungen für jede Datenstruktur so, dass die *recall*-Rate von mindestens 95% erreicht wird.

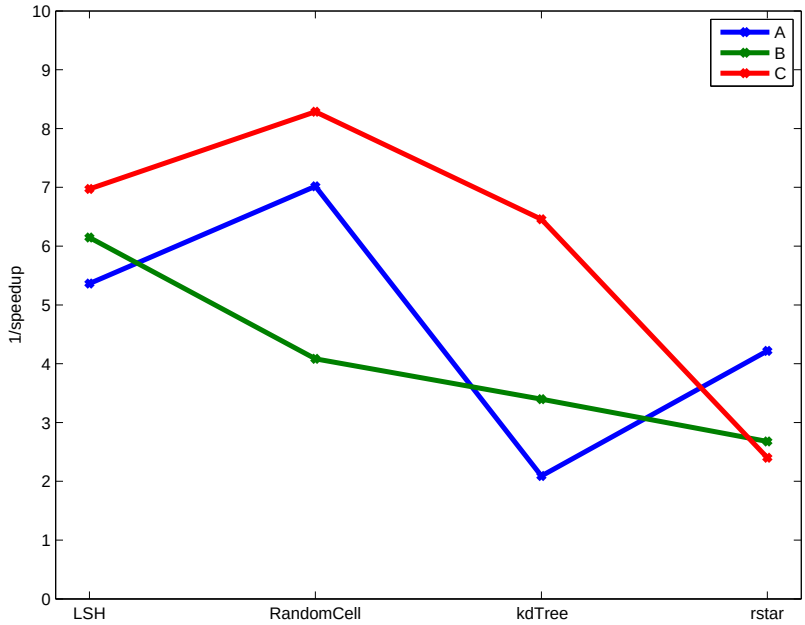

Abbildung 6.33: Es wird gezeigt um welchen Faktor die Datenstrukturen (x-Achse) langsamer sind, als die naive lineare Suche. Sowohl für den Datensatz `A` (blau) als auch `B` (grün) als auch `C` (rot) ist die lineare Suche die bestmögliche Strategie.

Obwohl man anhand der Ergebnisse (Abbildung 6.33) nicht genau sagen kann, ob nun die kd-Bäume oder die R*-Bäume die bessere Wahl darstellen, ist es klar, dass alle betrachteten Datenstrukturen mindestens um Faktor 2 langsamer sind als die naive lineare Suche.

Early Exit als dominante Strategie

Warum schneidet die naive lineare Suche so gut ab? Es ist natürlich so, dass man nicht gezwungen ist, die ganzen d-dimensionalen Vektoren zu vergleichen, sondern kann schon früher die Berechnungen abbrechen, falls der Fehler schon einen früher gefundenen kleineren Abstand übersteigt. Schaut man sich die Verteilung der notwendigen Anzahl der Vergleiche (Abbildung 6.34) an, so stellt man fest, dass im Schnitt nur ca. $d/3$ Vergleiche notwendig sind.

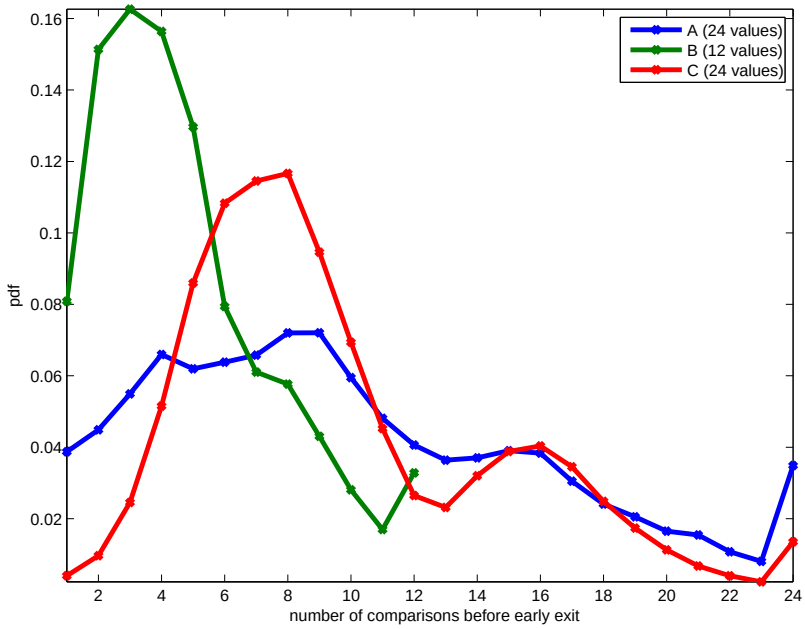

Abbildung 6.34: Die Verteilungsdichte für die Anzahl der notwendigen Vergleiche. Im Schnitt braucht man beim Datensatz `A` *und* `C` *ca. 10 Vergleiche für $d = 24$, für den Datensatz* `B` *mit der Deskriptorgröße $d = 12$ braucht man im Schnitt weniger als 5 Vergleiche.*

Man könnte diesen Effekt noch verstärken, indem man die Reihenfolge der Differenzbildung so abändert, dass zuerst die betragsmäßig größten Elemente eines Desktiptors berücksichtigt werden. Es wäre jedoch zu speicheraufwendig für jeden Deskriptor noch die Permutation für diese Sortierung abzuspeichern.

6.7.6 Fazit

Wie so oft bei der Suche in mehreren Dimensionen ist die lineare Suche die beste Strategie für unsere Datensätze. Es kann sein, dass sich dieser Sachverhalt ändert, wenn man Fingerprints mit größerer Anzahl von Fea-

tures zulässt. Doch man sollte jedoch umgekehrt bestrebt sein, die Anzahl der Features in einem Fingerprint zu reduzieren, wie wir es im folgenden Abschnitt sehen werden.

6.8 Reduktion der Fingerprints

Bis jetzt hatten wir alle gefundenen Features in den Fingerprint aufgenommen. Dies führt dazu, dass die so entstandenen Fingerprints ungefähr den Speicherbedarf der originalen Zeitreihen haben.

Natürlich wäre es möglich, mit Hilfe der Quantisierung (Abschnitt 6.3.4) den Speicherbedarf zu reduzieren. In diesem Kapitel möchten wir jedoch die Möglichkeiten betrachten, die Anzahl der Features im Fingerprint zu verringern: Es ist oft nicht notwendig, alle 1000 Features zu betrachten - auch $200 - 300$ Features pro Fingerprint wären ausreichend.

Man sollte bestrebt sein, solche Features zu wählen, die besonders stabil sind. Mit besonders stabil ist gemeint, dass nicht nur die Feature-Punkte vom Detektor zuverlässig gefunden werden, sondern, dass auch die Deskriptoren der Feature-Punkte aussagekräftig genug sind, um diese eindeutig zuordnen zu können.

Natürlich müssen die stabilsten Features nicht unbedingt die beste Wahl sein: Haben z.B. alle Messungen im Datensatz einen markanten, gleichen, sehr stabilen Feature-Punkt in ihrem Verlauf, so profitieren wir nicht davon, diesen zu behalten: Man kann diese Fingerprints nicht anhand dieser Eigenschaft unterscheiden. Im Prinzip braucht man ungewöhnliche, stabile Features. Wir beschränken uns in dieser Arbeit nur auf die einfacheren Heuristiken.

Im Prinzip stehen uns folgende Heuristiken zur Verfügung:

- Naive Methode: Man wählt zufällig k Features von den vorliegenden n. Diese Vorgehensweise führt dazu, dass der *recall* nur noch $\left(\frac{k}{n}\right)$ des ursprünglichen Wertes beträgt (die Wahrscheinlichkeit, dass beide Features einer richtigen Zuordnung in der Auswahl landen, beträgt $\frac{k}{n} \cdot \frac{k}{n}$). Deswegen scheint diese Vorgehensweise nicht besonders erfolgversprechend zu sein.

- Statische Methoden: Man wählt nur die größten Extrema, die man mit Hilfe von Detektoren findet, in der Hoffnung, dass diese besonders stabil sein müssten. Solche Methoden (Nachbarschaftserweiterung und Geräuschschwelle) wurden in Abschnitt 5.2 vorgestellt. Ein Nachteil besteht darin, dass man die endgültige Anzahl der Features im Fingerprint nicht einfach festlegen kann.

- Dynamische Methoden: Betrachtet man zwei Messungen m und $\hat{m}$ der gleichen Observablen, wobei $\hat{m}$ eine verzerrte Version der Messung m darstellt:

$$\hat{m} = V(m),$$

wobei V aus einer Familie der Verzerrungen $V \in \mathcal{V}$ zufällig gewählt wird, so kann man sich fragen, welche Features besonders stabil für die Verzerrungsfamilie $\mathcal{V}$ sind. Dies kann man erlernen, indem man für die Messung m unterschiedliche verzerrte Versionen $m_i = V_i(m)$ erstellt, diese mit dem Original m abgleicht und auf diese Weise feststellen kann, welche Features besonders stabil sind, d.h. bei den Abgleichvorgängen besonders oft richtig zugeordnet wurden. Die k stabilsten können so gewählt und weiter benutzt werden. Wir betrachten folgende Verzerrungsfamilien:

I *Gaußstörung:* Die Werte werden mit einem Gaußfehler behaftet:

$$\hat{m_i}(x) = m(x) + \xi,$$

wobei $\xi \stackrel{d}{=} N(0, \sigma)$ mit σ passend gewählt: Ist σ zu klein, so werden alle Features als stabil eingestuft, ist σ zu groß, so werden keine stabilen Features gefunden.

II *Selbstaddition:* Schaut man sich die Datensätze an, so stellt man schnell fest, dass die Verzerrungen nicht ein einfaches Gaußrauschen sein kann: vielmehr sehen sie wie die Messungen selbst aus: Randomwalks mit einer kleineren Amplitude. Das legt nahe

$$\hat{m_i}(x) = m(x) + a_i \cdot m(x + \Delta_i)$$

zu wählen, mit a_i und Δ_i zufällig. Auch hier gilt, dass die a_i passend gewählt werden müssen, sonst lassen sich die stabileren Features nicht von den weniger stabileren unterscheiden.

III *Erlernen von $\mathcal{V}$:* Uns stehen normalerweise zwei Messungen eines Coils zur Verfügung, das Vaterband p und das Kindband c, aus dem Alignment zwischen p und c. Mit Hilfe von DTW (Abschnitt 3.3) können die Verzerrungen V_i gelernt werden, die für diese Messanlagen typisch sind. Anschließend können sie angewandt werden, um $\hat{m} = V_i(m)$ zu erzeugen.

6.8.1 Auswertung

Wir betrachten zwei Szenarien:

1. Sowohl der Fingerprint des Vaterbandes als auch der des Kindbandes werden reduziert.

2. Nur der Fingerprint des Vaterbandes wird reduziert. Der Fingerprint des Kinderbandes wird ohne (zum Teil aufwändige) Reduktion benutzt. Dieses Szenario ist dann denkbar, wenn nur die Fingerprints der Vaterbänder in einer Datenbank abgespeichert werden und die Suche über nicht allzu viele Bänder erfolgt, so dass die für die Reduktion notwendige Rechenzeit bei der Suche nicht zurückgewonnen werden kann. Da die Ergebnisse bei dieser Vorgehensweise etwas besser sind (wie wir sehen werden), kann sie ein Kompromiss zwischen rechenzeitintensiver Suche mit nicht reduzierten Fingerprints, die jedoch sehr genau ist, und der schnellen aber etwas weniger genauen Suche mit reduzierten Fingerprints darstellen.

Statische Methode

Als erstes variieren wir die Größe der Nachbarschaft, in der ein lokales Extremum extrem sein muss. Die experimentellen Ergebnisse (Abbildung 6.35) zeigen, dass man damit die Anzahl der Features im Fingerprint reduzieren kann, ohne die *recall*-Rate zu verschlechtern.

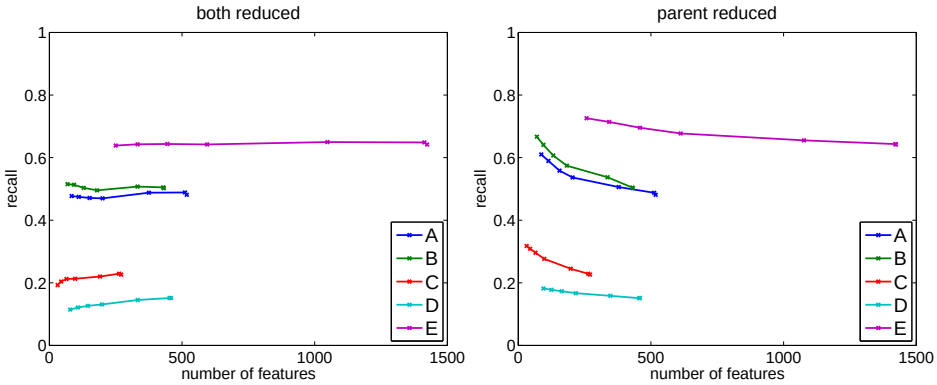

Abbildung 6.35: Links sehen wir das Verhalten der recall-Rate, wenn beide Fingerprints reduziert wurden. Man stellt fest, dass die recall-Rate konstant bleibt auch wenn die Anzahl der Features deutlich reduziert wird. Eine bessere Entwicklung stellt der zweite Fall dar, bei dem nur der Vaterfingerprint reduziert wird: Die recall-Rate steigt mit Verringerung der Anzahl von Features im Fingerprint.

Gaußrauschen

Im ersten Schritt wurde untersucht, welches Level des Rauschens die besten Ergebnisse liefert. Anhand von 200 zufällig gewählten Bändern wurden für jeden Testdatensatz die besten Einstellungen ermittelt. Es stellte sich heraus, dass die Ergebnisse am besten waren, falls der Rauschpegel an die

Standardabweichung der Zeitreihe gekoppelt war und ca. 15% davon betrug. Diese Einstellung wurde anschließend für alle Testdatensätze benutzt.

Des Weiteren, im Unterschied zur statischen Methode, bei der man die Anzahl der Features nur indirekt beeinflussen kann, lässt sich nun die genaue Anzahl der Features im Fingerprint festlegen. Es stehen zwei Möglichkeiten zur Verfügung:

- Die Anzahl der Features im endgültigen Fingerprint ist fest und unabhängig von der anfänglichen Anzahl der detektierten Features. Man könnte z.B. 200 beste Features für den reduzierten Fingerprint auswählen.

- Ein fester Komprimierungsfaktor wird angegeben. Man könnte z.B. nur die besten 20% aller detektierten Features in den endgültigen Fingerprint aufnehmen.

Beide Lösungen haben ihre Vor- und Nachteile und schnitten in unseren Experimenten ähnlich ab. Im Weiteren konzentrieren wir uns auf die Variante, bei der der Komprimierungsfaktor fest angegeben wird.

Für die Tests, deren Ergebnisse in Abbildung 6.36 vorgestellt wurden, wurden die stabilen Features anhand von 20 verzerrten Versionen einer Zeitreihe festgestellt.

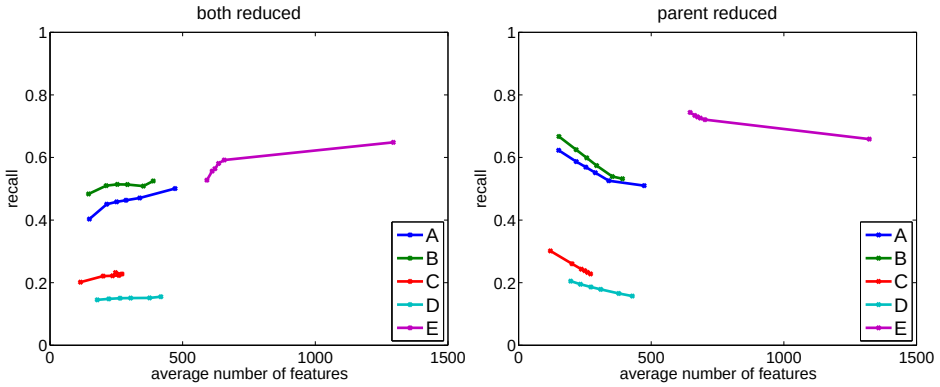

Abbildung 6.36: Das Verhalten der recall-Rate in Abhängigkeit von der Anzahl von Features, falls beide Fingerprints (links) oder nur einer (rechts) mit Hilfe vom Gaußrauschen reduziert wurden. Die Reduktion der beiden Fingerprints führt zu schlechteren recall-Raten. Die Schwächen der Strategie sind besonders bei den extrem langen Zeitreihen im Datensatz `E` offensichtlich: Man kann nicht gut genug zwischen guten und schlechten Features unterscheiden, so kommt es dazu, dass die mittlere Anzahl der Features im Fingerprint kaum reduziert wird. Nach diesen Ergebnissen kommt nur die Reduktion des Vaterfingerprints in Frage.

Selbstaddition

Auch hier musste zuerst die Verteilung des Parameters a_i bestimmt werden. Wir stellten in den Experimenten mit 200 zufällig gewählten Zeitreihen aus jedem Testdatensatz fest, dass der Parameterwert $a_i = 0.3$ die besten Ergebnisse für jeden der fünf Datensätze liefert.

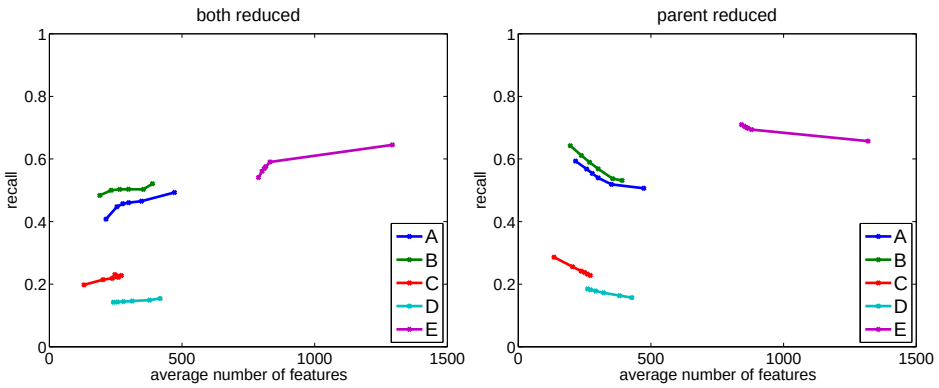

Abbildung 6.37: Das Verhalten der recall-Rate in Abhängigkeit von der Anzahl von Features falls beide Fingerprints (links) oder nur einer (rechts) mit Hilfe von der Selbstaddition reduziert wurden. Die Reduktion der beiden Fingerprints führt zu schlechteren recall-Raten. Auch hier bereitet der Datensatz `E` *besondere Schwierigkeiten: Die Anzahl der Features in Fingerprints lassen sich nicht entscheidend reduzieren. Nach diesen Ergebnissen kommt nur die Reduktion des Vaterfingerprints in Frage.*

Auch diese Heuristik (für die Ergebnisse siehe Abbildung 6.37) schneidet schlechter als die unflexible und weniger rechenintensive statische Methode ab.

Erlernte stabile Features

Mit Hilfe von DTW (Abschnitt 3.3) erlernen wir für jeden Testdatensatz die Verzerrungsfamilie anhand von 100 zufällig gewählten Zeitreihen. Diese Verzerrungen (lokale Änderungen der Skalierung und additive Fehler) werden dann auf die originale Zeitreihe angewandt.

Auch hier wurden die stabilsten Features nach einer Analyse der Ergebnisse für 20 verzerrten Versionen gewählt. Die Ergebnisse (Abbildung 6.38) legen nahe, dass diese Vorgehensweise, zumindest so lange man die DTW zugrunde legt, nicht besonders erfolgversprechend sind.

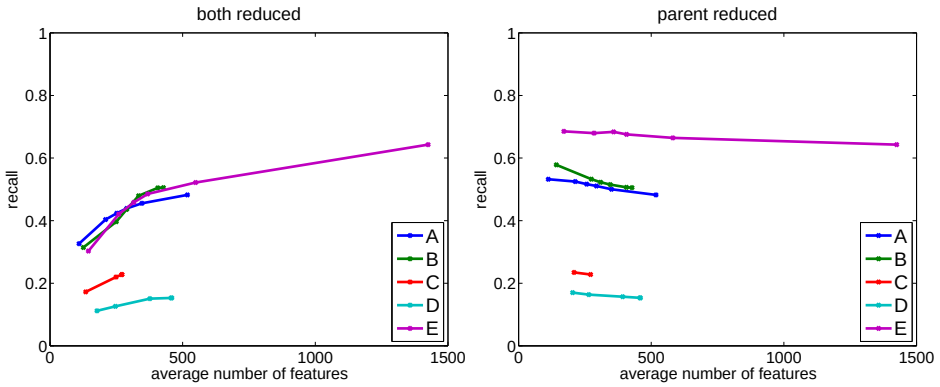

Abbildung 6.38: Das Verhalten der recall-Rate in Abhängigkeit von der Anzahl der Features falls beide Fingerprints (links) oder nur einer (rechts) mit Hilfe von der gelernten Verzerrungen reduziert wurden.

6.8.2 Fazit

Angesicht der experimentellen Ergebnisse kann man für die Anwendung nur die statische Reduktion mit Hilfe der Nachbarschaftserweiterung empfehlen. Hinzu kommt noch, dass es keine zusätzlichen Berechnungen dafür notwendig sind.

Die mit dem zusätzlichen Rechenaufwand verbundenen Berechnungen der stabilsten Features mit Hilfe des Gaußstörung oder der Selbstaddition lohnen sich nicht - diese Vorgehensweisen schneiden nicht besser ab, als die einfache Nachbarschaftserweiterung.

DTW scheint nicht die richtige Methode zu sein, um die kleinen lokalen Störungen der Koordinatentransformationen festzustellen bzw. richtig zu beschreiben.

Kapitel 7

Experimentelle Ergebnisse

In diesem Kapitel werden die mit Hilfe unseres Frameworks erreichten Ergebnisse präsentiert und mit den Resultaten der schon bekannten Vorgehensweisen verglichen. Bei dem Vergleich mit anderen Algorithmen geht es vor allem um die bereits vorgestellte Problemstellung `A` (Problem `A`, Abschnitt 2.2.1), da der Problemstellung `B` (Problem `B`, Abschnitt 2.2.2) bisher in der Literatur kaum Beachtung geschenkt wurde.

Wir untersuchen folgende Aspekte:

- Vergleich mit den etablierten Algorithmen
- Genauigkeit
- Effizienz bzgl. der Laufzeit und des Speicherbedarfs
- Robustheit gegenüber großen Skalierungen der x-Achse
- Datenbanksuche mit Teilabschnitten

7.1 Vergleich mit den vorherigen Algorithmen

Folgende zwei Versionen des Algorithmus werden dabei ins "Rennen" geschickt:

DOG Wir benutzen den DoG-Filter als Detektor, dazu die SIFT-Deskriptoren mit den Einstellungen wie diese in Abschnitt 6.3 bestimmt wurden. Des Weiteren werden die falschen Zuordnungen anhand der Formel (5.2) aussortiert. Um die Berechnungen zu beschleunigen, wird auch unser Vorwissen ausgenutzt, dass nur Skalierungen der x-Achse $\alpha \in [0.9, 1.1]$ vorkommen können, indem man nur diejenigen Feature-Punkte vergleicht, deren σ-Koordinaten sich nicht um mehr als den Faktor 1.3 unterscheiden (für mehr Informationen siehe Abschnitt 5.5).

X2 Wie **DOG**, mit dem Unterschied, dass man das auf $\phi = x^2 \cdot G(x, \sigma)$ basierende Wavelet als Detektor benutzt. Wie in Abschnitt 6.5 gezeigt wurde, ist dieser Detektor dem DoG-Filter überlegen, was jedoch auf Kosten der fast doppelt so langen Vorverarbeitungsphase geschieht.

Die Wahl fiel auf die SIFT-Deskriptoren, weil diese, wie im Abschnitt 6.1 festgestellt wurde, bei einer kleineren Deskriptorgröße im Durchschnitt eine leicht bessere Leistung erzielen. Auf unsere Versuche mit den CC-Deskriptoren, die vergleichbare (obwohl leicht schlechtere) Ergebnisse ergaben, soll nicht näher eingegangen werden.

Verglichen werden diese Varianten mit:

1. *Alignieren der Cross-Correlation* (Abschnitt 3.1.2). Zur Beschleunigung wird die FFT benutzt. Die Skalierung α wird dadurch berücksichtigt, dass die Anfrage mit

   $$\alpha \in \{0.9, 0.94, 0.96, 0.98, 1.00, 1.02, 1.04, 1.06, 1.10\}$$

   gestreckt wird, d.h. es werden neun unterschiedliche Anfragen untersucht, entsprechend den unterschiedlichen Werten, die die Skalierung α annehmen kann.

2. *Weiterentwickelter Shotgun* (Abschnitt 3.5). Es werden die Schnipsellängen L

   $$L \in \{64, 128, 256, 512\}$$

   untersucht und jeweils die beste Länge für den jeweiligen Datensatz gewählt.

3. *BDTW* (Abschnitt 3.3) berechnet mit Hilfe von UCR suite (Rakthanmanon u. a. [56]).

4. Die Darstellung der Zeitreihen als SAX-Zeichenreihen (Abschnitt 3.4), um diese dann mit Hilfe des Smith-Waterman-Algorithmus (Smith und Waterman [60]) zu alignieren. Die Laufzeit des Smith-Waterman-Algorithmus ist jedoch quadratisch, so dass man diese Vorgehensweise nur für kleinere Datenmengen einsetzen kann.

Als Kriterium für die Genauigkeit der Verfahren verwenden wir die schon im Abschnitt 6.2.2 vorgestellte "richtige Anzahl der Top-1-Treffer": Dabei wird eine Hälfte der Zeitreihen (auch als Vaterbänder/-zeitreihen bezeichnet) in einer Datenbank abgespeichert. Die dazu passenden Zeitreihen (auch als Kindbänder/-zeitreihen bezeichnet) werden für die Anfragen benutzt. Der Anteil der so richtig bestimmten Vater-Kindzeitreihenpaare wird von uns als *Genauigkeit* bezeichnet.

Experimentell ermittelte Genauigkeit und die Konfidenzintervalle

Bevor wir uns den experimentellen Ergebnissen zuwenden, sind einige Worte zur Signifikanz der Ergebnisse notwendig.

Es ist natürlich denkbar, dass man bei der Auswahl seiner Daten "Glück hat" und nur "gute" Zeitreihen erwischt und damit die experimentellen Ergebnisse zu gut ausfallen. Es stellt sich also die Frage, inwieweit man den Ergebnissen vertrauen kann bzw. wie groß der Fehler ist.

Eine Möglichkeit besteht darin, unterschiedliche Teilmengen aus dem gesamten Datenbestand zu wählen und über mehrere solche Stichproben eine Statistik zu erstellen, anhand welcher man den Fehler bzw. die Konfidenzintervalle für die erzielten Erfolgsraten angeben kann.

Es ist jedoch nicht notwendig, den "experimentellen" Weg zu gehen. In unserem Modell nehmen wir an, dass es zwei Sorten von Zeitreihen gibt:

- "gute" Zeitreihen, die in der Datenbank mit Hilfe unseres Algorithmus richtig gefunden werden
- "schlechte" Zeitreihen, die von unserem Algorithmus nicht gefunden werden.

Einfachheitshalber nehmen wir an, dass die Eigenschaft "gut" unabhängig davon ist, welche Zeitreihen sich in der Datenbank befinden.

Der Anteil der "guten" Zeitreihen p gibt den Anteil der richtig beantworteten Anfragen an die Datenbank und damit die Genauigkeit der zu untersuchenden Vorgehensweise an. Angenommen, wir betrachten nun n Zeitreihen und stellen fest, dass r davon in der Datenbank wiedergefunden wurden. Aufgrund der gemachten Annahmen handelt es sich bei der Anzahl der richtig gefundenen Zeitreihen r um eine binomialverteilte Variable

$$r \stackrel{d}{=} B(p, n).$$

Damit können wir nicht nur den "guten" Anteil p schätzen, sondern auch einen Fehler der Abschätzung angeben:

$$\hat{p} := \frac{r}{n},$$
$$\Delta\hat{p} := \sqrt{\frac{\hat{p} \cdot (1 - \hat{p})}{n - 1}}.$$

Während $\Delta\hat{p}$ dem Konfidenzintervall von ca. 68% entspricht, stimmt der Fehler offensichtlich nicht für $p = 1$ oder $p = 0$. Diesen Fall haben wir jedoch schon in der Formel (3.1), Abschnitt 3.5.1, abgehandelt:

$$\Delta\hat{p} \approx -\frac{\log \alpha}{m}$$

mit $\alpha = 0.68$ im weiteren Verlauf.

Die obigen Ergebnisse lassen sich auch intuitiv deuten: Die aufgrund von 1000 Zeitreihen berechneten Werte sind viel vertrauenswürdiger als die Ergebnisse aufgrund von 10 Zeitreihen - es ist sehr unwahrscheinlich, dass man 1000 mal Glück hatte.

7.1.1 Vergleich der Genauigkeit

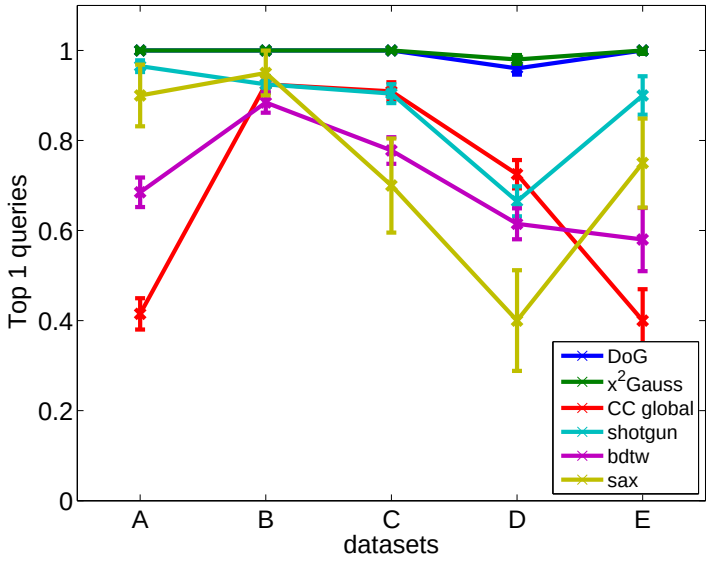

Abbildung 7.1: Für unsere Datenauswahl aus den Testdatensätzen `A`*,* `B`*,* `C` *und* `D` *haben DOG und X2 beide* 100*%-tige Genauigkeit. Bei dem Datensatz* `E` *performt die X2-Variante etwas besser als die DOG-Variante. Für alle Testdatensätze sind DOG und X2-Varianten deutlich besser als die Konkurrenz. Der größere Fehler bei der SAX-Kurve ist dadurch bedingt, dass sie nur aufgrund von* 20 *Zeitreihen bestimmt wurde.*

Nun soll die Performanz unseres Algorithmus mit der Performanz anderer Ansätze verglichen werden. Um einen ersten Eindruck zu bekommen, wählen wir zufällig je 200 Bänder aus den Datensätzen `A`, `B`, `C` und `D` aus und 50 aus dem Datensatz `E` und bestimmen experimentell die Anzahl der richtigen Antworten für das Problem `A` (Abschnitt 2.2.1). Für die Bewertung der SAX-inspirierten Vorgehensweise wurden jedoch nur 20 Zeitreihen pro Testdatensatz betrachtet, was durch die äußerst lange Laufzeit des Smith-Waterman-Algorithmus bedingt ist.

Wie man der Abbildung 7.1 entnimmt, ist der Vorteil der sowohl DOG- als auch X2-Varianten gegenüber den anderen Algorithmen offensichtlich. Keiner der "renomierten" Algorithmen zeigt eine Performanz, die ihren Einsatz rechtfertigen würden. Am besten schneidet noch das Shotgun-Verfahren

ab, doch auch Shotgun erreicht nicht mehr als 90% Genauigkeit, während die DOG/X2-Varianten auf fast 100% der Anfragen eine richtige Antwort liefern.

7.1.2 Vergleich der Laufzeiten

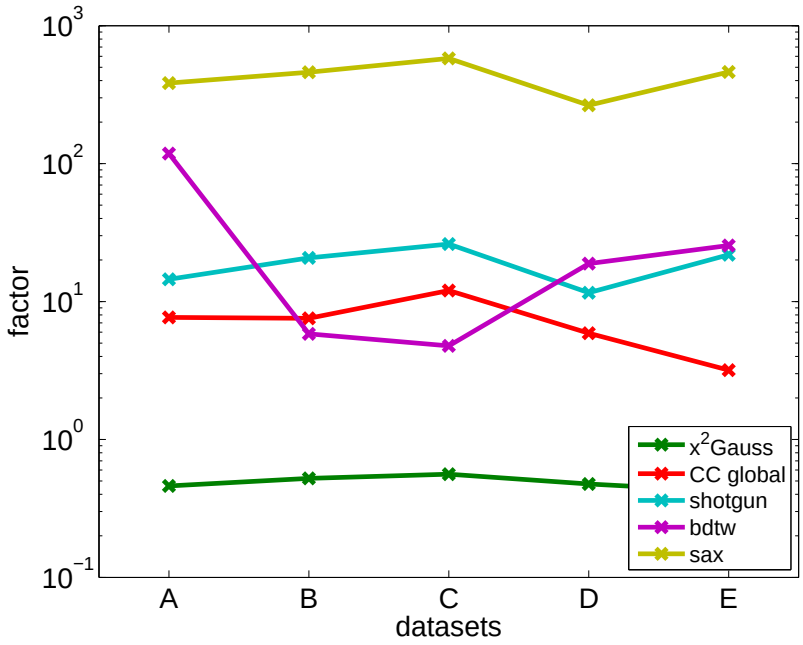

Abbildung 7.2: Vergleichszeiten der Algorithmen normiert auf die Laufzeit der DOG-Variante. Die X2-Variante ist etwas schneller, weil bei den gewählten Einstellungen etwas weniger Features produziert werden. BDTW (bzw. UCR suite) performt am schlechtesten beim Datensatz `A`*. Ansonsten sind BDTW, CC-Alignment und Shotgun ungefähr gleich schnell aber deutlich langsamer als die DOG- oder X2-Varianten.*

Vergleicht man die asymptotischen Laufzeiten der obigen Verfahren in Abhängigkeit von der Länge der Datenbank N_D (damit bezeichnen wir die Summe der Längen aller Zeitreihen in der Datenbank) und der Länge der Abfrage N_Q, so ergibt sich Folgendes:

Algorithmus	Vorverarbeitung	Anfrage
DOG/X2	$O(N_D)$	$O(N_D \cdot N_Q)$
CC-Alignment	$O(N_D \log N_D)$	$O(N_D \log N_D)$
Shotgun	$O(N_D \log N_D)$	$O(N_Q \cdot N_D \log N_D)$
BDTW	$O(N_D)$	$O(N_D \cdot N_Q)$
SAX	$O(N_D)$	$O(N_D \cdot N_Q)$

Die asymptotischen Laufzeiten werden durch folgende Operationen dominiert:

DOG/X2: Vergleich der Deskriptoren. Unsere Untersuchungen haben gezeigt, dass die Anzahl der Deskriptoren im Fingerprint proportional zur Länge der Zeitreihe ist. Dies erscheint logisch und stimmt zumindest für alle von uns untersuchten Datensätze. Da die Laufzeit der Suche quadratisch in der Anzahl der Deskriptoren ist, ist sie auch quadratisch in der Länge der Zeitreihen. Der Vorfaktor ist allerdings bedingt durch die von uns eingesetzten Heuristiken recht klein.

CC-Alignment: Berechnung der FFT. Asymptotisch ist dies das schnellste Verfahren.

Shotgun: Die Suche nach dem besten Alignment eines Schnipsels wird FFT-beschleunigt, was sich jedoch erst ab der Schnipsellänge 128 wirklich lohnt.

BDTW: Berechnung des DTW-Abstands. Auch Heuristiken, wie die in der *UCR suite* eingesetzten, ändern nicht viel an dem quadratischen Rechenaufwand. Wir nutzten PAA 8, damit die Berechnungen im zeitlich absehbaren Rahmen durchgeführt werden können.

SAX: Die quadratische Laufzeit des Smith-Waterman-Algorithmus.

Wie die Abbildung 7.2 illustriert, variieren die Laufzeiten der Anfrage in einer Datenbank bestehend aus 200 Bändern von ca. 1-2 Sekunden (DOG/X2) bis ca. eine Stunde (SAX). Während das SAX-Verfahren am langsamsten ist und nicht in Frage für größere Datenbanken kommt, sind Shotgun, BDTW und CC-Alignment ca. um den Faktor 10 langsamer.

Damit würde die Auswertung der Verfahren für die kompletten vorhandenen Testdatensätze mehrere Monate Rechenzeit in Anspruch nehmen. Darauf wird verzichtet, wir konzentrieren uns auf die Auswertung der von uns vorgestellten Verfahrensvarianten DOG und X2.

Entzauberung der UCR suite

In Rakthanmanon u. a. [56] wurde die sogenannte *UCR suite* vorgestellt. Dieses Framework ermöglicht eine schnelle Suche der Subsequenzen wobei DTW als das Abstandsmaß benutzt wird. Dieses Verfahren gilt als *state-of-the-art*. Die Hauptidee des Algorithmus besteht darin, dass man den Vergleich zweier Zeitreihen schon früh abbrechen kann und man nur für einen kleinen Teil der Vergleiche die wirkliche DTW-Distanz berechnen muss. Für viele auf der UCR-Homepage (`http://www.cs.ucr.edu/~eamonn/UCRsuite.html`)

angeführte Beispiele wurde DTW nicht einmal für 1% der Vergleiche notwendig.

Diese beeindruckende Leistung lässt sich nicht auf unsere Testdatensätze übertragen, was sich in der langen Laufzeit niederschlägt (vergleiche Abbildung 7.2). Folgende Tabelle fasst den mittleren (prozentualen) Anteil der notwendigen DTW-Vergleiche für unsere Testdatensätze zusammen, falls eine Skalierung (α) von 5% bzw. 10% zugelassen wird (dabei betrachteten wir die Daten nach der PAA-Stufe 8):

Datensatz	A	B	C	D	E
DTW-Anteil (in %, $\alpha = 5\%$)	76	48	16	48	30
DTW-Anteil (in %, $\alpha = 10\%$)	74	17	8	31	27

Man sieht, dass vor allem die Zeitreihen des Testdatensatzes A der *UCR suite* große Probleme bereiten. Aber auch für die anderen Testdatensätze muss man für deutlich mehr als in 1% der Fälle die DTW-Berechnung bemühen, so dass kein Speedup-Faktor größer als 10 im Vergleich zum naiven BDTW-Verfahren erzielt werden kann.

Dies führt dazu, dass wir die Untersuchungen nur mit den reduzierten Daten (PAA 8) in akzeptabler Zeit durchführen konnten. Damit ist der Vergleich zwar nicht mehr gänzlich fair, weil die anderen Verfahren die volle Auflösung benutzen können - ohne Reduktion der Dimension würden die Berechnungen jedoch Monate in Anspruch nehmen!

7.1.3 Fazit

Schon aufgrund ihrer Laufzeit wären die von uns entwickelten Verfahren den bisher bekannten Algorithmen vorzuziehen. Die einzige Begründung für die längere Laufzeit der anderen Algorithmen wäre eine höhere Genauigkeit, doch diese ist nicht gegeben. Sogar umgekehrt: Die bisher bekannten Verfahren vermögen es nicht, eine ausreichende Genauigkeit zu erreichen, wobei das Shotgun-Verfahren wohl den weit abgeschlagenen zweiten Platz in der Genauigkeitswertung beanspruchen kann.

Für den von uns entwickelten Algorithmus spricht außerdem Folgendes:

- Es ist nicht notwendig, die Daten zu bereinigen bzw. zu normalisieren. Die Normalisierung kann bei Dateien mit Ausreißern zu großen Abweichungen führen.
- Es können alle Skalierungsintervalle behandelt werden. Die vom Algorithmus behandelbare Skalierung hängt nur von der Anzahl der Oktaven o bei der Berechnung der Fingerprints (siehe Abschnitt 5.1) ab, ohne dabei drastisch höhere Laufzeiten in Kauf nehmen zu müssen (wie dies z.B. bei der BDTW der Fall ist).

- Robustheit gegenüber Messausfällen: Es reicht schon ein "richtiger" Teilbereich, um einen erfolgreichen Vergleich durchführen zu können. Es ist natürlich möglich, z.B. mit Hilfe von CC-Alignment, BDTW oder SAX lediglich mit einem Teilbereich zu suchen. Der große Unterschied besteht darin, dass man bei diesen Vorgehensweisen vorher wissen muss, welcher Teilbereich die richtigen Werte beinhaltet und welcher durch den Messausfall zustande kommt. Unser Algorithmus setzt dieses Wissen nicht voraus.

7.2 Genauigkeit für reale Daten

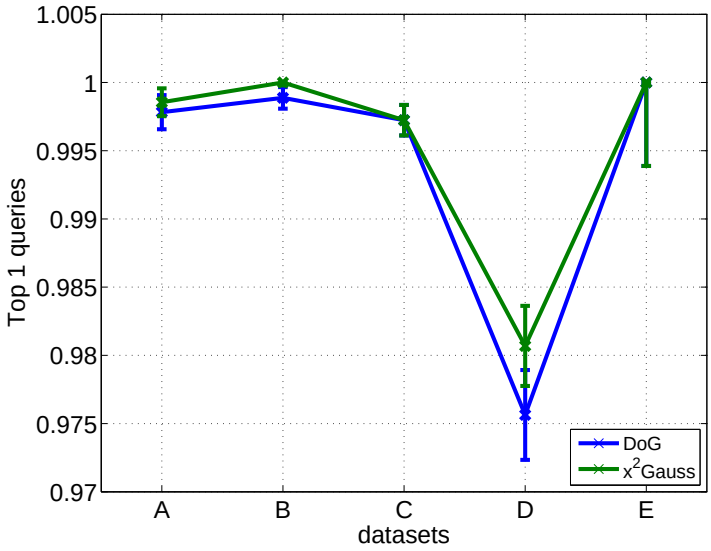

Abbildung 7.3: Die Genauigkeit der Anfragen beträgt sowohl für die DOG als auch für die X2-Variante fast 100 *%. Die X2-Variante scheint die Nase vorn zu haben, doch auch bei dem besonders deutlich gewonnenen Fall* `D` *überschneiden sich noch die Fehlerintervalle, so dass der Vorteil der X2-Variante nicht über alle Zweifel erhaben ist.*

In diesem Abschnitt möchten wir nun die Performanz der beiden Algorithmus-Varianten auf den gesamten Testdatensätzen untersuchen. Auf einen Vergleich mit den anderen Ansätzen wird an dieser Stelle verzichtet, denn zum einen haben die kleineren Tests schon eindrucksvoll die Überlegenheit unserer Vorgehensweise bestätigt, zum anderen würde die dafür notwendige Rechenzeit von mehreren Monaten den minimalen Wissensgewinn nicht rechtfertigen.

Zur Erinnerung: Die Genauigkeit eines Verfahrens ist durch den Anteil der richtigen Top-1-Antworten gegeben.

Die Ergebnisse wurden in der Abbildung 7.3 zusammen getragen. Bis auf die Datensätze `C` und `D` wurden Genauigkeiten von ca. 99.8%-99.9% erreicht! Im Abschnitt 7.7 wird eine Möglichkeit besprochen, wie man auch für die Datensätze `C` und `D` die Genauigkeit erhöhen könnte.

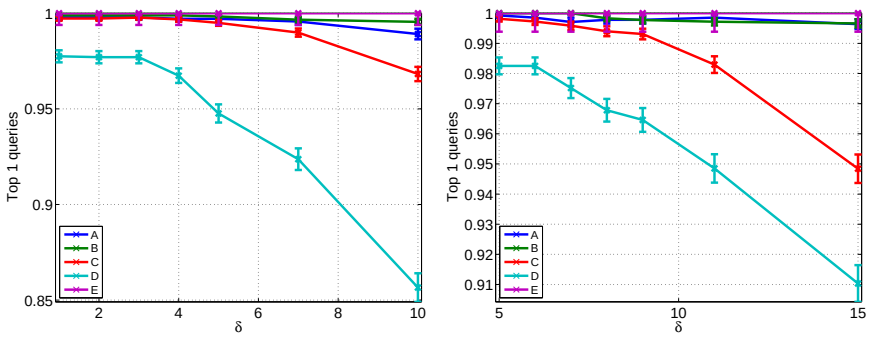

Abbildung 7.4: Links befindet sich die Entwicklung der Genauigkeit in Abhängigkeit von der Nachbarschaftsgröße δ für die DOG-Variante, rechts - für die X2-Variante für alle fünf vorhandenen Datensätze. Man stellt fest, dass für die Daten mit guter Qualität (`A`, `B`, `E`) die Reduktion der Fingerprints kaum zu einer Reduktion der Genauigkeit führt.

7.2.1 Genauigkeit der reduzierten Fingerprints

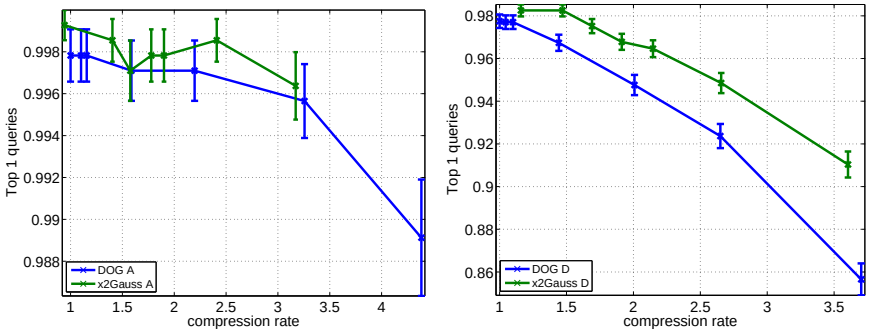

Abbildung 7.5: Es wird beispielhaft für die Datensätze `A` (links) und `D` (rechts) die Abhängigkeit der Genauigkeit von der Kompressionsrate der Fingerprints gezeigt. Die Kompressionsrate wird relativ zur Größe der Fingerprints der DOG-Variante mit Parameter $\delta = 1$ angegeben. Sowohl für das linke als auch für das rechte Diagramm verläuft die X2-Variante (grün) über der DOG-Variante (blau), d.h. bei gleicher Fingerprintgröße schneidet die X2-Variante besser ab.

Nun betrachten wir die Performanz des Frameworks nachdem die Anzahl der Features mit Hilfe der im Abschnitt 6.8 besprochenen statischen Methode reduziert wurde. Dabei wird bei der Suche der lokalen Extrema die

Nachbarschaft δ erweitert: Je größer die Nachbarschaft, desto weniger lokale Extrema existieren und umso weniger Features befinden sich in einem Fingerprint, was zu einer schnelleren Suche führt.

Die Abbildung 7.4 zeigt, dass die reduzierten Fingerprints in Sachen Genauigkeit den originalen Fingerprints kaum in etwas nachstehen, vorausgesetzt die Datenqualität ist gut genug (was vor allem für die Datensätze `A`, `B` und `E` gilt). Die nächste Tabelle fasst die möglichen Einstellungen für den Parameter δ zusammen, die sich aus der Abbildung 7.4 ergeben. Dabei wird in Klammern das Verhältnis zwischen der Genauigkeit der reduzierten Fingerprints und der Genauigkeit der originalen Fingerprints angegeben:

Datensatz	A	B	C	D	E
X2-Variante	9 (0.999)	9 (0.998)	7 (0.998)	6 (1.000)	15 (1.000)
DOG-Variante	5 (0.999)	5 (0.999)	4 (0.999)	3 (0.999)	10 (1.000)

Welche Auswirkungen die Reduktion der Fingerprints auf die Laufzeit hat, wird von uns im Abschnitt 7.3.1 genauer untersucht.

Ein Vergleich der Performanz der DOG/X2-Varianten ist nicht so einfach: Für verschieden Varianten/Einstellungen ergeben sich unterschiedliche Fingerprintgrößen. Betrachten wir jedoch die Performanz in Abhängigkeit von der Fingerprintgröße, indem wir den Parameter δ variieren, so wird ein kleiner aber stabiler Vorsprung der $X2$-Variante erkennbar (Abbildung 7.5).

7.2.2 Fazit

Beide Varianten des Algorithmus weisen eine hohe Genauigkeit auf: Beide kommen für die meisten von uns untersuchten Datensätze auf eine Genauigkeit von 99.9%, und machen damit bei 2000 Zeitreihen weniger Fehler als die "renomierten" Algorithmen auf einer kleinen Datenbank von 200 Datensätzen. Dies ist ein weiterer Grund, diese Algorithmen nicht auf einer großen Datenbank testen zu müssen.

Obwohl beide Versionen für einen Einsatz geeignet sind, liegt die X2-Variante leicht vorne, was vor allem bei den Daten mit schlechterer Qualität - `C` und `D` - deutlich wird.

7.3 Geschwindigkeit für reale Daten

Auch wenn wir schon gezeigt haben, dass unser Algorithmus ungefähr um den Faktor 10 schneller als die "Konkurrenz" ist, sollen nun die absoluten Werte genannt werden, damit man die Dauer der Anfragen in größeren Datenbanken besser abschätzen kann. Für unsere Tests in diesem Abschnitt

verwenden wir einen handelsüblichen Desktop-PC mit Intel-Core 2 Duo CPU 3.00 GHz, 3.8 GiB Arbeitsspeicher und 32bit Ubuntu Linux. Performanzen, die mit Hilfe einer stärkeren Hardware erreicht werden können, werden von uns im Abschnitt 7.3.2 besprochen.

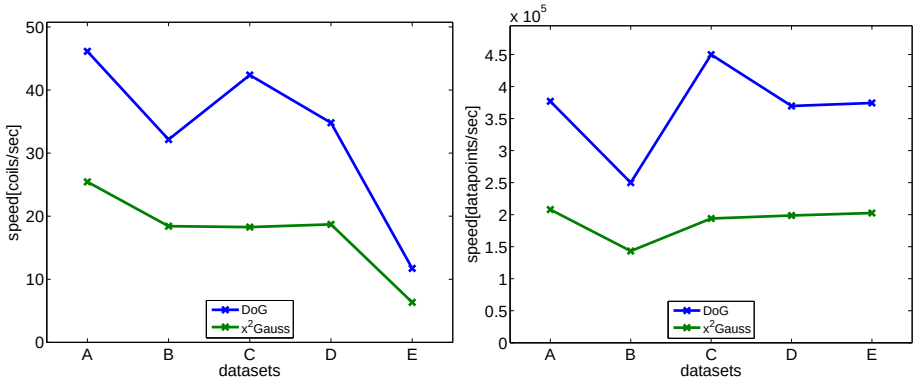

Abbildung 7.6: Das linke Bild zeigt die Anzahl der vorverarbeiteten Zeitreihen pro Sekunde. Die Geschwindigkeit der DOG-Variante (blau) ist hierbei fast doppelt so hoch, was daran liegt, dass die Berechnung der Faltung mit dem DOG-Filter nur die Hälfte der Zeit in Anspruch nimmt (Abschnitt 5.1). Man stellt fest, dass die Anzahl der vorverarbeiteten Zeitreihen für den Datensatz `E` stark abnimmt. Dies liegt daran, dass diese Zeitreihen außergewöhnlich lang sind. Ein fairerer Vergleich ergibt sich, wenn man die Anzahl der vorverarbeiteten Datenpunkte pro Sekunde untersucht. Sie entspricht ca. $2 \cdot 10^5$ für die X2-Variante und ist fast doppelt so groß für die DOG-Variante. Der Einbruch der Geschwindigkeit für den Datensatz `B` lässt sich durch eine besonders große Feature-Punkte-Dichte erklären, der Sprung beim Datensatz `C` ist durch eine besonders kleine Feature-Punkte-Dichte zu erklären.

Wir benutzen folgende Einheiten, um die Geschwindigkeit zu charakterisieren:

Zeitreihe/s: Anzahl der pro Sekunde vorverarbeiteten Zeitreihen (v_z)

Datenpunkte/s: Anzahl der Datenpunkte, die pro Sekunde vorverarbeitet werden (v_d)

Zeitreihenvergleiche/s: Anzahl der pro Sekunde durchgeführten Zeitreihenvergleiche (V_z)

Feature-Vergleiche/s: Anzahl der pro Sekunde durchgeführten Feature-Vergleiche (V_f)

Datenpunkte^2/s: Produkt der Längen der Zeitreihen, welche pro Zeiteinheit verglichen werden (V_d)

Es werden alle Größen gebraucht: Auch wenn wir uns vor allem für die Geschwindigkeit interessieren, mit welcher die Zeitreihen miteinander verglichen werden, ist dieser Wert nicht aussagekräftig genug bzw. unfair - die

längeren Zeitreihen werden sicherlich mehr Zeit in Anspruch nehmen als die kürzeren. Die längere Zeitreihe bedeutet aber nicht zwangsläufig mehr Feature-Punkte, deren Anzahl letztendlich für die Länge der Rechnung ausschlaggebend sind.

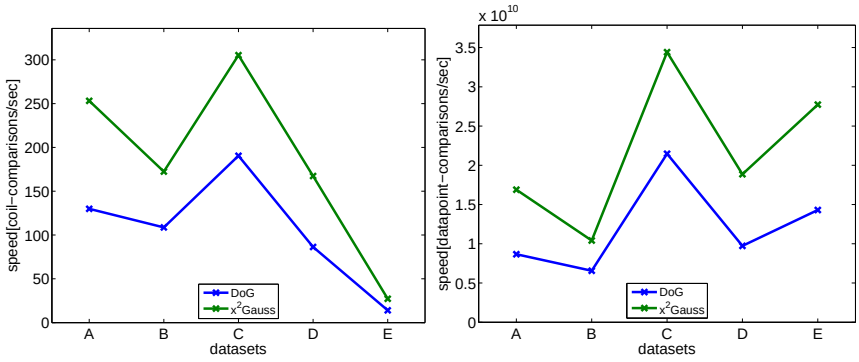

Abbildung 7.7: Das erste Diagramm zeigt die Anzahl der verglichenen Zeitreihen pro Sekunde. Die X2-Variante ist dabei schneller, was darauf zurückzuführen ist, dass die erzeugten Fingerprints weniger Features beinhalten. Auch für die Vergleichsgeschwindigkeit stellt man fest, dass die langen Zeitreihen einen negativen Einfluss darauf haben (Datensatz `E`*). Der im rechten Diagramm dargestellte Kennwert (Datenpunkte^2/s) lässt sich für den Vergleich mit den anderen Algorithmen benutzen: So bedeutet* $1.5 \cdot 10^{10}$ *Datenpunkte^2/s für den Datensatz* `A`*, dass die Anfrage der Länge* 10^3 *in einer Datenbank der Länge* $1.5 \cdot 10^7$ *in ca. einer Sekunde beantwortet wird.*

Obwohl die Vorverarbeitungszeit (Abbildung 7.6) nicht so relevant ist, wenn die Fingerprints nach der Vermessung einer Zeitreihe berechnet und in einer Datenbank abgespeichert werden, kann dieser Prozess zu einem Bottleneck werden, falls die Fingerprints *on the fly* berechnet werden müssen. Wie erwartet, ist die Erzeugung der DOG-Fingerprints fast doppelt so schnell wie die Erzeugung der X2-Fingerprints.

Betrachtet man die Geschwindigkeit beim Vergleich der Zeitreihen (V_z) in der Abbildung 7.7, so stellt man einen kleinen Vorsprung der X2-Variante fest: So ist die Zeitreihenvergleichsgeschwindigkeit V_z fast um den Faktor 2 höher als bei der DOG-Variante und erreicht z.B. für den Datensatz `A` ca. 250 Vergleiche pro Sekunde.

Vergleicht man diese Geschwindigkeit mit der Vorverarbeitungsgeschwindigkeit v_z, welche für den Datensatz `A` nur 25 Zeitreihen pro Sekunde betrug (Abbildung 7.6), so stellt man fest, dass, falls die Fingerprints bei der Suche *on the fly* berechnet werden müssen, die DOG-Variante vorzuziehen ist, für welche fast 50 Zeitreihen pro Sekunde in ihre entsprechenden Fingerprints umgewandelt werden können.

Der Normalfall sollte jedoch eine Datenbank sein, welche die vorberechneten Fingerprints enthält: Verglichen mit der Zeit, die zur Vermessung der

Zeitreihe notwendig ist, kann man die zur Umwandlung in einen Fingerprint notwendige Zeit vernachlässigen. Dank der im Abschnitt 6.3.4 vorgestellten Quantisierung ist auch der zur Speicherung des Fingerprints notwendige Speicher nur ein Bruchteil dessen, was zur Speicherung der originalen Zeitreihe notwendig ist (mehr dazu im Abschnitt 7.4).

7.3.1 Vergleichsgeschwindigkeit für reduzierte Fingerprints

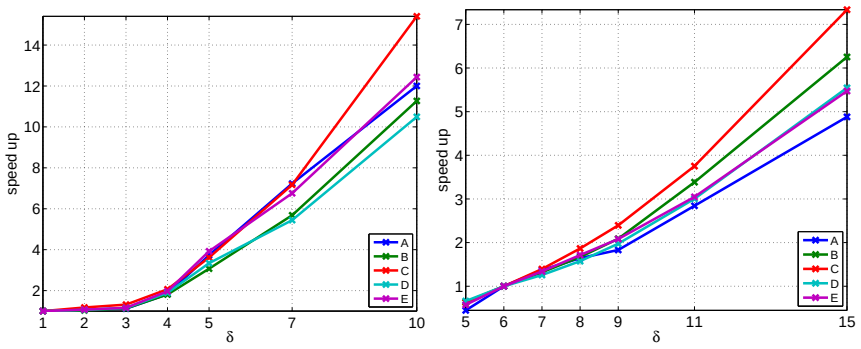

Abbildung 7.8: Es werden die Speedup-Faktoren für reduzierte Fingerprints in Abhängigkeit vom Parameter δ für die DOG-Variante (links, speed-up relativ zum $\delta = 1$) und X2-Variante (rechts, speed-up relativ zum $\delta = 6$). Erst ab $\delta \leq 4$ wird die Anzahl der Features im Fingerprint merkbar reduziert, sodass der Speedup für kleinere δ's vernachlässigt werden kann.

Für die im Abschnitt 6.8 vorgestellte statische Reduktion der Feature-Anzahl in einem Fingerprint wurde schon im Abschnitt 7.2.1 eine mit den originalen Fingerprints vergleichbare Genauigkeit attestiert. Nun möchten wir der Frage nachgehen, wie sich dabei die Vergleichsgeschwindigkeit der Zeitreihen verhält.

An dieser Stelle soll vermerkt werden, dass sich die Vorverarbeitungszeit bei der statischen Reduktion nicht von der Vorverarbeitungszeit zur Berechnung der originalen Fingerprints unterscheidet.

Aus der Abbildung 7.8 ergeben sich folgende Speedup-Faktoren für die im Abschnitt vorgeschlagenen δ-Einstellungen (vermerkt in Klammern):

Datensatz	A	B	C	D	E
X2-Variante	1.8 (9)	2.1 (9)	1.3 (7)	1 (6)	5.5 (15)
DOG-Variante	3.9 (5)	3.1 (5)	2.0 (4)	1.1 (3)	10.5 (10)

Von der Möglichkeit der Reduktion profitieren die Daten mit guter Qualität (A und B), vor allem wenn sie extrem lang sind (Datensatz E).

7.3.2 Parallelisierung

Müssen große Datenbanken abgefragt werden, so ist es denkbar, die Datenbank in mehrere Teile auf unterschiedliche Maschinen zu verteilen und damit auf diese einfache Art und Weise den Algorithmus zu parallelisieren. In diesem Abschnitt beschäftigen wir uns mit der Möglichkeit Anfragen an kleinere Datenbanken für Multiprozessoren zu parallelisieren.

Unser Ansatz sieht wie folgt aus: Der Abgleich zweier Fingerprints könnte von einem Thread übernommen werden. Auf diese einfache Weise kann die Suche auf so viele Threads, wie es Zeitreihen in der Datenbank gibt, verteilt werden. Dies ist jedoch nicht besonders effizient, denn der mit der Erzeugung der Threads verbundene Aufwand würde einen großen negativen Einfluss auf die Gesamtlaufzeit haben.

In unserer Implementierung benutzen wir dazu eine mit Hilfe von OpenMP[1] parallelisierte `for`-Schleife und verteilen damit die Fingerprintvergleiche zufällig auf die unterschiedlichen Kerne.

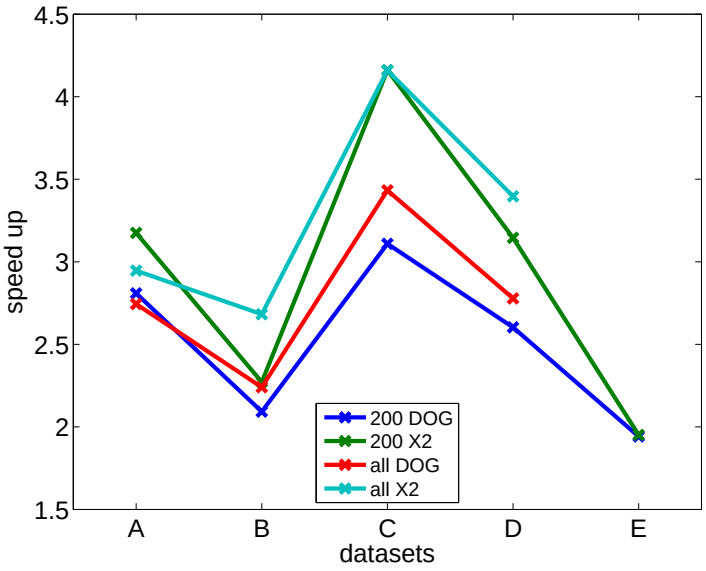

Abbildung 7.9: Erzielte Speedup-Faktoren für die Anfragen an die Datenbank mit 200 *(blau, grün) und mit ca.* 2000 *(rot, cyan) Einträgen. Der Speedup für die größere Datenbank fällt etwas höher aus und bewegt sich im Schnitt um den Faktor* 3*.*

Wir vergleichen die Laufzeiten für die Single-Core Anwendungen mit der parallelisierten Version für zwei Szenarien:

- Die Datenbank ist klein (200 Zeitreihen bzw. 63 für den Datensatz `E`)
- Die gesamte Datenbank (ca. 1500 - 2000 Zeitreihen)

[1] `http://openmp.org`

Um die Skalierbarkeit zu untersuchen benutzen wir einen Rechner mit einer *Intel Core i7 CPU 970 3.20 GHz* mit 6 Kernen und 64bit Ubuntu Linux.

Die Ergebnisse werden in der Abbildung 7.9 vorgestellt. Insgesamt lässt sich ein Speedup von ca. 3 erzielen. Dies ist zwar etwas vom denkbaren Speedup von 6 entfernt, trotzdem sind damit ca. 1000 Zeitreihenvergleiche pro Sekunde möglich und damit rückt die Suche in riesigen Datenbanken mit mehreren tausenden Einträgen in greifbare Nähe.

7.3.3 Fazit

Sowohl die DOG- als auch die X2-Varianten unseres Algorithmus ermöglichen es, mehrere hundert Zeitreihenvergleiche pro Sekunde sogar auf einer gewöhnlichen Hardware durchzuführen. Dies würde in etwa einer Anfrage der Länge 10^3 an eine Datenbank der Länge $\approx 10^7$ entsprechen.

Müssen die Fingerprints jedoch erst aus den originalen Datensätzen noch erzeugt werden, so stellt sich heraus, dass diese Vorverarbeitung zu einem *bottle neck* wird: Die Fingerprints werden um den Faktor 10 langsamer erzeugt, als sie verglichen werden! Dies ist ein Szenario, bei dem die DOG-Variante ihre Stärke ausspielen kann: Die Erzeugung der Fingerpints dauert in dem Fall nur halb so lange!

Geht man davon aus, dass die Fingerprints vorberechnet in einer Datenbank abgespeichert sind, so ist die X2-Variante wegen ihrer leicht höheren Genauigkeit bei gleicher Vergleichsgeschwindigkeit vorzuziehen.

7.4 Komprimierungsraten

Neben der kurzen Laufzeit ist natürlich auch der kleine Speicherbedarf des Fingerprints von Bedeutung: Kleinere Fingerprints bedeuten, dass weniger Daten über das Netzwerk von der Datenbank übertragen werden müssen und dass mehr Fingerprints im Arbeitsspeicher gecacht werden können. Dies alles trägt zu einer schnelleren Suche bei.

Neben der Reduktion der Anzahl der Features im Fingerprint wurde von uns im Abschnitt 6.3.4 die Quantisierung der Deskriptoren betrachtet. Dabei kamen wir zum Schluss, dass 8 Bit ausreichend sind: Die Quantisierung auf 8 Bit kann erfolgen, ohne die Ergebnisse negativ zu beeinflussen.

Die Abbildung 7.10 zeigt die Kompressionsraten für die von uns untersuchten Datensätze. Dabei wurde die 8-Bit-Quantisierung genutzt und eine Kompressionsrate von ca. 5 erreicht. Es wurde davon ausgegangen, dass die originale Zeitreihe mit 32 Bit pro Datenpunkt angegeben wurde.

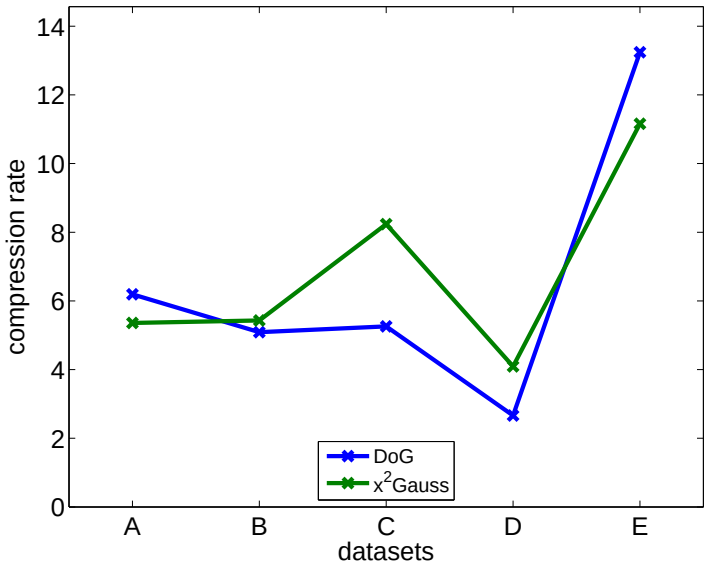

Abbildung 7.10: Das Verhältnis zwischen der Größe der originalen Zeitreihen (32 Bit/Datenpunkt) und der Größe der erzeugten Fingerprints nach der statischen Reduktion mit Parametern, wie sie im Abschnitt 7.2.1 bestimmt wurden, und 8-Bit-Quantisierung.

7.5 Genauigkeit für skalierte Daten

Wie schon angesprochen wurde, handelt es sich bei unseren Testdatensätzen um Skalierungen, die 10% nicht übersteigen. Wir möchten jedoch auch das Verhalten des Algorithmus für größere Skalierungen untersuchen.

In diesem Abschnitt soll das folgende Experiment betrachtet werden: Jede Zeitreihe in der Datenbank wird mit einem zufällig gewählten Skalierungsfaktor $\alpha \in [1, 4]$ (uniform verteilt) gestreckt. Anschließend wird die Suche in der Datenbank ganz normal durchgeführt.

Wie schon bei der Betrachtung der Detektoren (Abschnitt 6.6) festgestellt, ist mit einer leichten Verschlechterung der Feature-Punkte-Wiedererkennung zu rechnen. Inwieweit hat dies einen Einfluss auf die Top-1-Quote bei den Datenbank-Anfragen? Die Abbildung 7.11 beantwortet diese Frage:

- Für die guten Datensätze `A`, `B` und `E` stellt man so gut wie keinen Unterschied fest:

rel. Genauigkeit	A	B	E
DOG	1.000	1.001	1.000
X2	0.999	1.000	1.000

- Bei den "schlechteren" Datensätzen `C` und `D` macht sich jedoch diese

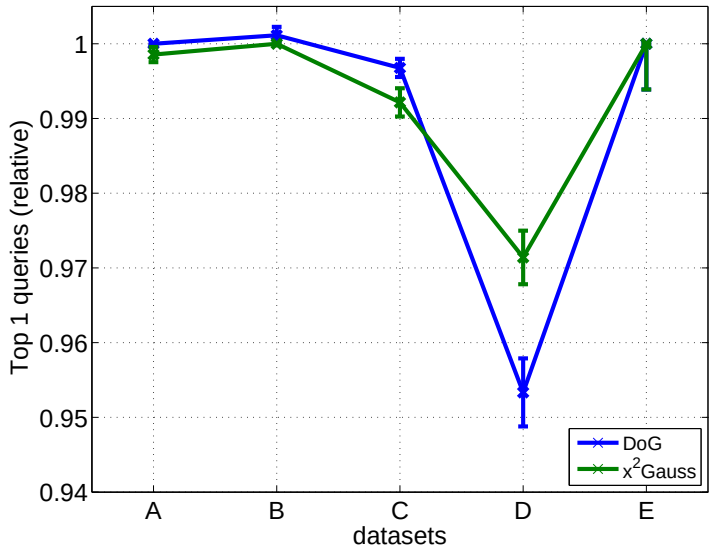

Abbildung 7.11: Die Genauigkeit für große Skalierungsunterschiede. Die Genauigkeit für die zufällig skalierten Zeitreihen wurde mit der Genauigkeit für die unskalierte Zeitreihen normiert, damit man die Unterschiede besser sehen kann. Dies führt dazu, dass der Wert für den Datensatz `B` für die DOG-Variante über 1 liegt - bei den skalierten Anfragen war die Genauigkeit sogar höher als bei den unskalierten!

verschlechterte Wiedererkennung der Feature-Punkte bemerkbar, auch wenn die Erkennungsraten immer noch oberhalb von 95% bleiben.

rel. Genauigkeit	`C`	`D`
DOG	0.997	0.953
X2	0.992	0.971

Unsere Experimente lassen den Schluss zu, dass sowohl die DOG- als auch die X2-Variante robust gegenüber einer größeren Skalierung sind.

7.6 Suche mit Teilabschnitten

Es gibt mindestens zwei gute Gründe, die Suche mit Teilabschnitten der Zeitreihen zu untersuchen:

- Wegen einer Störung im Betriebsablauf wurde nur ein Teil der Messung durchgeführt. Reicht diese vorhandene Information aus?
- Man möchte einen Notstopp durchführen können: Schon nach wenigen vermessenen Metern will man wissen, ob die Zeitreihen zu einer Observablen gehören und falls dies nicht der Fall ist, den Prozess abbrechen.

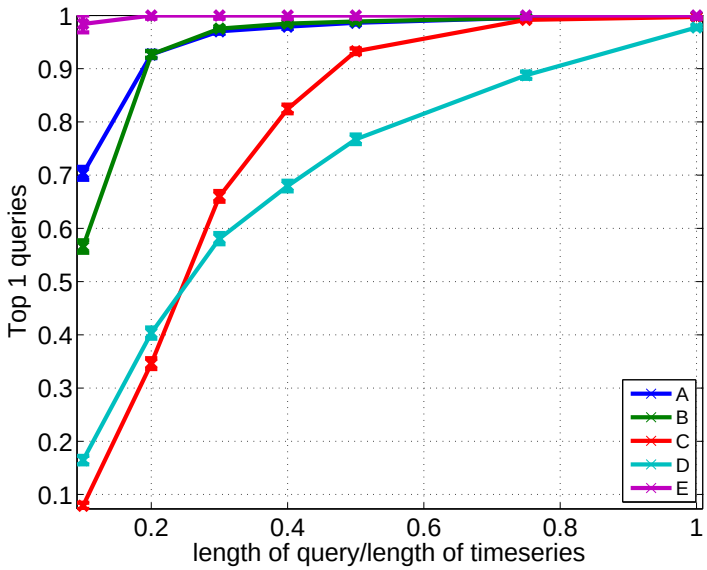

Abbildung 7.12: Anteil der richtigen Top-1-Anfragen in Abhängigkeit von der relativen Länge des Teilabschnittes (hier exemplarisch die DOG-Variante). Für die Datensätze mit schlechter Datenqualität (`C` und `D`) bricht die Genauigkeit massiv ein.

Ob die Suche mit Teilabschnitten erfolgreich sein wird, hängt zum größten Teil von der Datenqualität ab. So ist es nicht verwunderlich, dass (siehe Abbildung 7.12) für die Datensätze `C` und `D` die Genauigkeit deutlich einbricht, sobald nicht mit Hilfe der ganzen Zeitreihe gesucht wird. Besser sieht es für die Datensätze `A`, `B` und `D` aus.

Einer der Hauptgründe für das relativ schlechte Abschneiden besteht darin, dass man oft das Pech hat, dass entweder der gewählte Teilabschnitt oder das dazu passende Gegenstück durch einen Messausfall unbrauchbar gemacht wurde. Des Weiteren ist das Ergebnis besonders schlecht für kurze Zeitreihen - die kurzen Teilabschnitte reichen nicht mehr für eine zuverlässige Zuordnung aus.

Deswegen ist die Frage interessanter, wie lang ein Teilabschnitt (absolut) sein muss, damit eine erfolgreiche Suche möglich ist. Diese Fragestellung ist näher an der Notstopp-Aufgabe: Die Länge der Zeitreihe ist a-priori nicht bekannt, deswegen ist es wichtiger, das Verhalten für die absoluten Längen von Teilabschnitten zu kennen.

Diese Frage untersuchen wir anhand des Testdatensatzes `B`, welcher *bereinigt* wurde: D.h. es wurde sicher gestellt, dass die Datenbank für jeden Teilabschnitt, mit dem gesucht wird, auch ein passendes Gegenstück beinhaltet.

In unseren Experimenten (Abbildung 7.13) stellten wir fest, dass 2000 Datenpunkte (was in etwa 200 Metern entspricht) ausreichend sind: Es wird

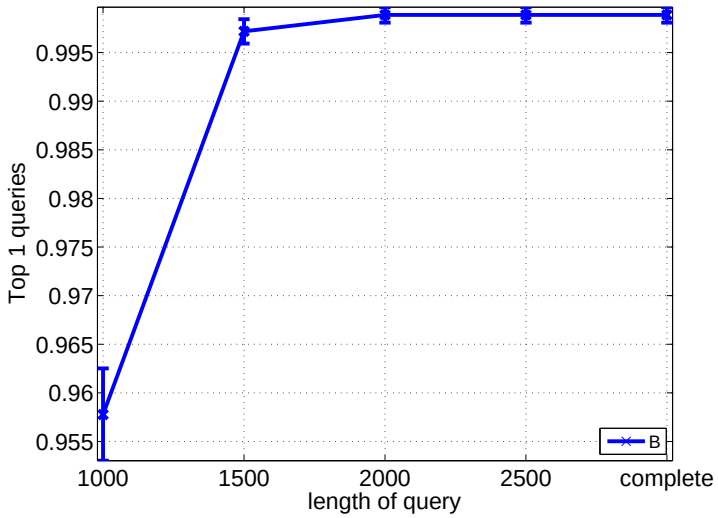

Abbildung 7.13: Anteil der richtigen Top-1-Anfragen in Abhängigkeit von der Länge des Teilabschnittes (exemplarisch die DOG-Variante) für den bereinigten Datensatz `B`*. Ab der Länge der Abfrage* 2000 *wird die gleiche Genauigkeit erreicht, wie bei den Abfragen mit den kompletten Zeitreihen, welche im Schnitt auf die Länge von ca.* 16000 *kommen.*

die gleiche Genauigkeit erreicht wie bei den Anfragen mit der kompletten Zeitreihe. Bei einer mittleren Zeitreihenlänge von ca. 16000 würde dies eine achtfache Beschleunigung mit sich bringen.

7.7 Mehrspurige Fingerprints

Ist die Qualität der Messungen nicht ausreichend, wie dies z.B. im Falle der Testdatensätze `C` und insbesondere `D` der Fall ist, so kann man die Ergebnisse verbessern, indem man die Informationen kombiniert.

Bis jetzt wurde die Tatsache noch nicht erwähnt, dass die Zeitreihen im Testdatensatz `C` die Breitenprofile und die Zeitreihen im Testdatensatz `D` die Dickenprofile der gleichen Metallbänder sind.

Wir können die Fingerprints zu einem mehrspurigen Fingerprint zusammenfassen. Zum Beispiel im obigen Fall wäre das Breitenprofil (Datensatz `C`) die erste Spur und das Dickenprofil (Datensatz `B`) die zweite Spur.

1. Der Abgleich der Deskriptoren passiert für alle Spuren unabhängig voneinander, genauso wie bei den “einfachen” Fingerprints.

2. Die so gefundenen Zuordnungen $\mathcal{Z}^s$ für die einzelnen Spuren werden zu einer gemeinsamen Menge der Zuordnungen

$$\mathcal{Z} := \cup_{s \in Spuren} \mathcal{Z}^s$$

zusammengefasst.

3. Die gemeinsame Menge der Zuordnungen $\mathcal{Z}$ wird benutzt, um die Koordinatentransformation zwischen den Messungen zu bestimmen und die Anzahl der Inlier zu finden (siehe Abschnitt 5.6).

Die Benutzung des mehrspurigen Fingerprints hat auch seine Kosten: Die Laufzeit wächst linear mit der Anzahl der analysierten bzw. benutzten Spuren. So ist es nicht sinnvoll, eine "gute" Spur durch eine Spur mit schlechter Qualität der Messung zu erweitern: Die Genauigkeit würde davon nicht profitieren, während die Laufzeit sich verdoppeln würde.

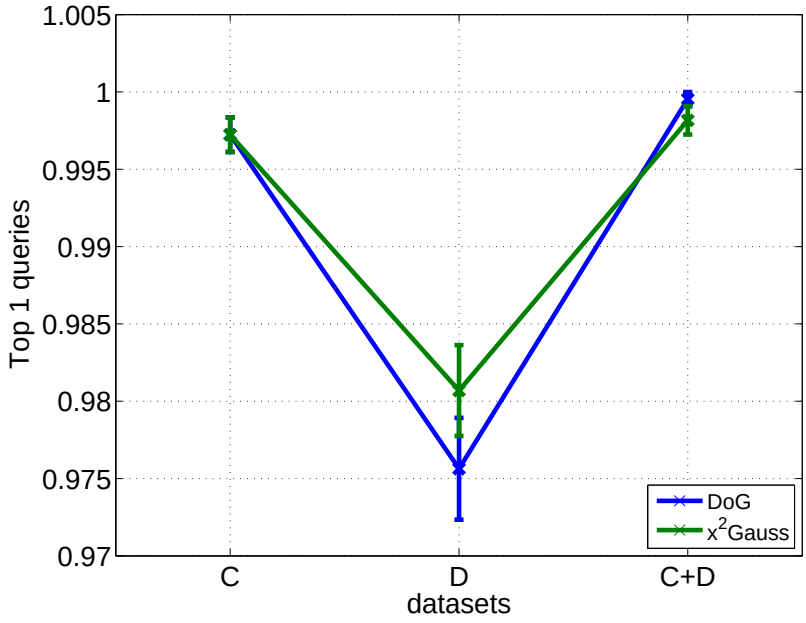

Abbildung 7.14: Die Genauigkeit der DOGs-Variante nimmt von 99.72% allein für `C` auf 99.95% für `C` und `D` kombiniert zu. Die Anzahl der falsch gefundenen Zeitreihen hat sich um den Faktor 6 reduziert! Der noch größere Anstieg für `D` ist nicht so beeindruckend: Es wurde ja auch mit einer weitaus besseren Spur kombiniert.

Aus der Abbildung 7.14 sind die Verbesserungen dieser Methode ersichtlich: Die Genauigkeit wächst von 99.72% für die "normalen" Fingerprints auf 99.95% für mehrspurige Fingerprints!

Für die Verwendung der mehrspurigen Fingerprints sprechen folgende Gründe:

- Die Wahrscheinlichkeit, dass ein Messausfall bei allen Spuren gleichzeitig passiert, ist umso geringer, je mehr Spuren verwendet werden. Somit ist ein System, das mehrere Spuren verwendet, deutlich stabiler.

- Man kann die Bänder sogar dann erkennen, wenn die Erkennung anhand einzelner Spuren nicht möglich wäre.

Ist die Genauigkeit der einzelnen Spuren nicht ausreichend, so bieten die mehrspurigen Fingerprints eine Möglichkeit die Performanz des Frameworks deutlich zu steigern.

7.8 Genauigkeit für das Problem B

Beim zuvor betrachteten Problem A wurde nur nach der bestpassenden Zeitreihe gesucht bzw. man ging implizit davon aus, dass sich zu jeder Anfrage eine passende Zeitreihe in der Datenbank befindet.

Für das Problem B muss die Frage beantwortet werden, ob es sich bei zwei vorgelegten Zeitreihen um die Messung der gleichen Observablen handelt.

Hätte man eine große Datenbank zur Hand, so wäre folgende Strategie denkbar, um das Problem B zu lösen:

- Füge eine der vorliegenden Zeitreihen zu der Datenbank hinzu.
- Führe die Anfrage mit der zweiten Zeitreihe durch. Falls die erste eingefügte Zeitreihe die Top-1-Antwort der Anfrage ist, so gelten die Zeitreihen als passend, sonst als unpassend.

Diese Vorgehensweise führt dazu, dass bei einer Datenbank der Größe N in $\frac{1}{N}$ der Fälle eine falsche positive Antwort gegeben wird. Wenn es keine passende Zeitreihe gibt, muss irgendeine Zeitreihe die am besten passende sein und genau für diese dann die Anfrage fälschlicherweise mit “Ja, sie passen zusammen” beantwortet wird. Es ist möglich, diesen Fehler zu reduzieren, indem man zwischen den Top-1-Antwort und der zweitbesten Antwort einen Mindestunterschied verlangt.

Obwohl wir in der Praxis ohne langwierige Suche in einer Datenbank auskommen möchten, sollte uns diese Methode als Goldstandard dienen und ist deswegen von Interesse.

Für die in der nachfolgenden Tabelle vorgestellten Ergebnisse wurde ein Unterschied von mindestens 10 Inliers (auch andere Werte sind denkbar und führen zu vergleichbaren Ergebnissen) zwischen den beiden Top-Antworten verlangt. Die Testdatensätze C und D wurden zu mehrspurigen Fingerprints zusammengefasst (vergleiche Abschnitt 7.7), weil nur so eine befriedigende Genauigkeit erzielt werden konnte. Die Anzahl der falsch positiven Antworten wird pro eine Million und die Anzahl der falsch negativen wird pro Tausend angegeben - der Testdatensatz E wurde wegen zu kleiner Anzahl der Zeitreihen ausgelassen:

Datensatz	A	B	C und D zusammen
falsch positiv (aus 1 Mio.)	12	10	16
falsch negativ (‰)	1	5	3
Datenbankgröße	1379	1764	2166

Im Abschnitt 3.5.1 wurde schon ein Ansatz vorgestellt, um die Frage nach der Zusammengehörigkeit der beiden Zeitreihen zu beantworten, ohne irgendwelche a-priori Informationen zu besitzen.

Dem gegenüber stehen Methoden, die den Unterschied zwischen den zusammengehörigen und unterschiedlichen Zeitreihen aufgrund einer Lernmenge erkennen.

Als nächstes sollen stochastische bzw. statistische Modelle betrachtet werden, welche eine Unterscheidung zwischen den beiden Zeitreihen ermöglichen.

7.8.1 Stochastische Sicht auf RANSAC

Zur Erinnerung: Wir stellen uns die Frage, wie wahrscheinlich es ist, dass für n zufällige gleichverteilte Punkte $(x_i, y_i) \stackrel{d}{=} U([0,1] \times [0,1])$ mit $i = 1, \cdot n$, der RANSAC-Algorithmus (Abschnitt 5.6.1) k *Inliers* findet. Die Anzahl der Inliers $k(n, \delta)$ hängt von n und dem Parameter δ im RANSAC-Verfahren ab. Eine analytische Lösung dieses Problems scheint kompliziert zu sein, deswegen möchten wir stochastische Simulationen benutzen.

Als erstes untersuchen wir die Verteilung der Inlieranzahl in Abhängigkeit von der Anzahl n bzw. vom Parameter δ. Es ist natürlich nicht möglich für jede denkbare Kombination von n und δ eine Simulation durchlaufen zu lassen.

Eine gängige Vorgehensweise besteht darin, eine experimentell ermittelte Verteilung der Inliers $F_{n,\delta}(k)$ durch eine Normalverteilung $N(\mu, \sigma)$ zu approximieren. Man kann durch weitere Experimente das Verhalten der Parameter $\mu(n, \delta)$ und $\sigma(n, \delta)$ untersuchen und dadurch die Verteilung $F_{n,\delta}$ für beliebige n und δ interpolieren. Wie die Abbildung 7.15 jedoch zeigt, kann dieser Ansatz nicht genügend genaue Ergebnisse liefern, da die Approximation mit einer Gaußfunktion nicht ausreichend genau erscheint.

Da wir uns jedoch letztendlich für solche Regeln interessieren, wie "Falls mehr als k Inliers gefunden wurden, so kann die Hypothese, dass die Zeitreihen nichts miteinander zu tun haben, auf dem Signifikanz-Niveau α abgelehnt werden", ist es geschickter den Verlauf dieser Linien $k_{ablehnen}(\alpha, n, \delta)$ zu bestimmen. Die Verläufe von $k_{ablehnen}$ wurden anhand von 10^6 Simulationen berechnet und in der Abbildung 7.16 dargestellt.

Betrachtet man jedoch die Experimente mit realen Daten, so stellt man fest, dass die Ergebnisse deutlich von den Vorhersagen der Simulation abweichen.

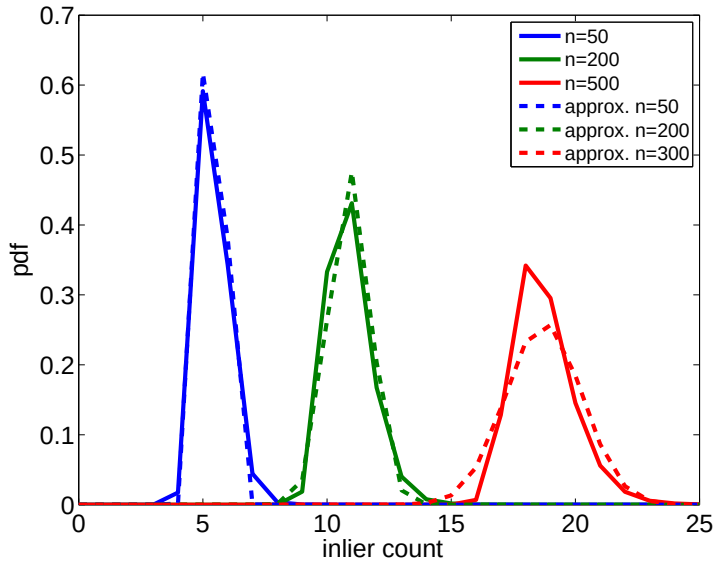

Abbildung 7.15: Die Annäherung der Verteilung der Anzahl der Inliers $F_{n,\delta}(k)$ mit Hilfe von der Normalverteilung für eine unterschiedliche Anzahl der Zuordnungen n bei festen $\delta = 0.05$. Vor allem qualitativ gibt es große Unterschiede (so z.B. fällt die Gaußfunktion zu schnell für $n = 50$ ab), so dass dieser Ansatz nicht anwendbar ist.

In der folgenden Tabelle wird die experimentell ermittelte Wahrscheinlichkeit dargestellt, unsere Nullhypothese ("die Zeitreihen passen nicht") in Abhängigkeit vom Signifikanzniveaus α fälschlicherweise abzulehnen:

Datensatz	$\alpha = 0.01$	$\alpha = 10^{-3}$	$\alpha = 10^{-4}$	$\alpha = 10^{-5}$
A	0.06	0.05	0.05	0.05
B	0.16	0.08	0.04	0.04
C	0.17	0.09	0.05	0.02
D	0.20	0.08	0.04	0.02
E	0.33	0.32	0.32	0.32
A vs. B	0.01	0.003	0.002	0.0007
B vs. C	0.02	0.006	0.003	0.002
C vs. D	0.17	0.08	0.04	0.02

Man stellt fest, dass die Vorhersagen unseres Modells nicht mit den experimentellen Ergebnissen übereinstimmen. So sollte man auf dem Signifikanzniveau $\alpha = 10^{-5}$ nur ca. 10^{-5} der Fälle fälschlicherweise als passend erkennen, in Experimenten wurde aber in mehr als 2% der Fälle dieser Fehler gemacht!

Besonders extrem ist dies beim Datensatz E: In einem Drittel der Fälle werden Inlieranzahlen erreicht, die sich durch Zufall in weniger als in einem Fall aus 100000 ergeben würden. Besser sieht es für den Vergleich der Zeitreihen

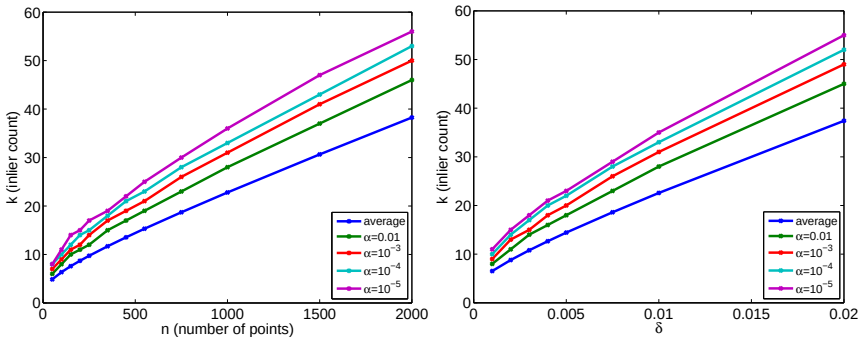

Abbildung 7.16: Betrachtet man den Verlauf der Signifikanz-Niveaus in der Abhängigkeit von n (links, $\delta = 0.005$ fest) und δ (rechts, $n = 500$ fest), so stellt man fest, dass es nichts Ungewöhnliches ist, falls mehr als 40 Inliers gefunden werden, wenn es viele Zuordnungen gibt bzw. der Parameter δ den Wert 0.01 übersteigt.

aus, die aus unterschiedlichen Datensätzen kommen, wie z.B. `A` vs. `B`: Zumindest für die Signifikanzniveaus $\alpha = 10^{-2}$ und $\alpha = 10^{-3}$ erreichen sie in etwa die vorhergesagten Raten. Dies ist wiederum nicht mehr der Fall für `C` vs. `D`, was jedoch auch nicht verwunderlich ist: Es sind die Breiten- und Dickenmessungen gleicher Metallbänder, sodass dort anscheinend Zusammenhänge bestehen.

Bei genauer Untersuchung stellt sich heraus, dass die Annahme "Nicht passende Zeitreihen führen zu uniformverteilten Zuordnungen" nicht stimmt. Dafür gibt es folgende Gründe:

- Nicht passende Zeitreihen (hier Dickenprofile) besitzen oft ähnliche Verläufe, vor allem wenn die zugrunde liegenden Observablen bei fast gleichen Bedingungen erstellt wurden, was unter anderem für zeitlich kurz nacheinander erzeugte Objekte oft zutrifft.

- Die Deskriptorenzuordnungen sind nicht unbedingt stochastisch unabhängig voneinander: Die Deskriptoren überlappen sich, stochastische Unabhängigkeit geht verloren, was zur Häufungen von Zuordnungen führt, welche bei stochastisch unabhängigen Deskriptorenzuordnungen äußerst selten wären.

Die Überlappung der Deskriptoren lässt sich jedoch nicht vermeiden, es bedarf also komplexere Modelle, um die Zugehörigkeit zweier Zeitreihen mit hoher Genauigkeit richtig vorherzusagen.

Auch wenn die hier besprochene Methode nicht die erhoffte Genauigkeit erreichen konnte, zeigten die Ergebnisse und die Modelle, dass es durchaus möglich ist, das Problem `B` zu bearbeiten.

7.8.2 Erlerntes Unterscheiden

Während die Ergebnisse des vorherigen Abschnittes keinerlei Wissen über die vorliegenden Daten erforderten, möchten wir als nächstes solche Methoden betrachten, für die man einige Parameter aus einem Lernsatz lernen bzw. schätzen muss.

Effektiver δ-Parameter

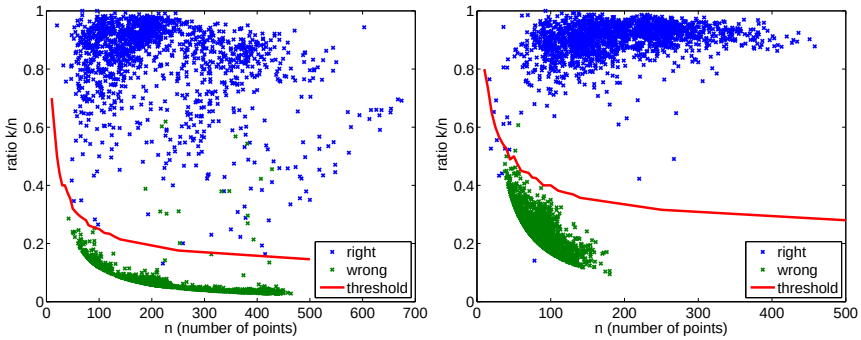

Abbildung 7.17: Eine mögliche Aufteilung in passende (blau) und nicht passende (grün) Zeitreihenpaare anhand des Verhältnises $\frac{|Inliers|}{|Zuordnungen|}$ *(y-Achse) für die Testdatensätze* `A`*(links) und* `B`*(rechts). Der Verlauf des Schwellenwerts entspricht dem Verlauf des Signifikanzniveaus* $\alpha = 10^{-6}$ *aus der RANSAC-Simulation mit* $\delta = 0.03$ *für den Testdatensatz* `A` *und* $\delta = 0.07$ *für den Testdatensatz* `B`*.*

Unser erster Versuch besteht darin, der Abhängigkeit zwischen den Feature-Zuordnungen insoweit Rechnung zu tragen, dass wir in unseren Simulationen den Parameter δ durch einen größeren Wert δ_{eff} ersetzen. Dies führt dazu, dass die Anzahl der falschen positiven Antworten (nicht passende Zeitreihenpaare werden als passend eingestuft) drastisch gesenkt wird, ohne die Anzahl der falschen negativen (passende Zeitreihenpaare werden als nicht passend eingestuft) wesentlich zu erhöhen.

Die Ergebnisse für die Testdatensätze `A` und `B` werden in der Abbildung 7.17 dargestellt. Die nachfolgende Tabelle fasst die erzielten Ergebnisse zusammen:

Datensatz	`A`	`B`	`C` und `D` zusammen
falsch positiv (aus 1 Mio.)	9	1	5
falsch negativ (‰)	1	4	3
gewähltes δ_{eff}	0.03	0.07	0.04

Das Erfreuliche ist, dass die Ergebnisse mindestens genau so gut sind, wie beim aufwendigen Verfahren mit der großen Datenbank. Der Nachteil liegt

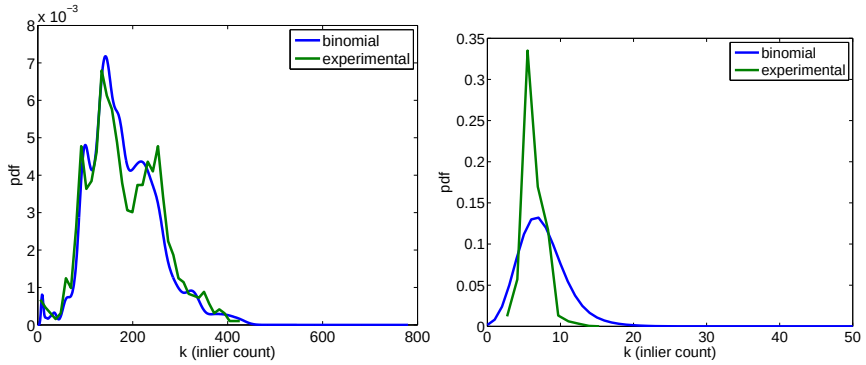

Abbildung 7.18: Theoretisch und experimentell ermittelte Verteilungen der Inlieranzahl k für den Testdatensatz (hier beispielhaft) B. Links - für die passenden Zeitreihenpaare ($p = 0.564$), rechts - für die nicht passenden Paare ($p = 0.027$). Vor allem rechts sieht man, dass die Annahme der binomialen Verteilung den Sachverhalt nicht richtig beschreibt, trotzdem kann dies als eine Näherung verwendet werden.

darin, dass der δ_{eff}-Parameter für jeden Testdatensatz gesondert bestimmt werden muss und dafür mehrere hunderte Zeitreihen notwendig sind.

Ein binomiales stochastisches Modell

Auch wenn die nicht passenden Zeitreihenpaare mehr Inliers aufweisen können als ihnen statistisch "zusteht", sind es immer noch weniger als bei den passenden Zeitreihenpaaren - sonst wäre es unmöglich, das Problem A zu lösen.

In der Praxis erreicht man bessere Ergebnisse, wenn man nicht eine Nullhypothese ablehnen/akzeptieren muss, sondern wenn man sich zwischen zwei Optionen (sprich zwei Verteilungen) entscheiden muss.

Ein denkbares Modell wurde in Brown und Lowe [11] vorgeschlagen. Dabei geht man davon aus, dass die Feature-Punkte unabhängig voneinander als Inliers erkannt werden. Die Wahrscheinlichkeit dafür beträgt:

$$P(z_i \; ist \; ein \; Inlier) = \begin{cases} p_r & \text{passende Zeitreihen} \\ p_w & \text{nicht passende Zeitreihen} \end{cases}$$

Die genauen Werte für p_r und p_w müssen gelernt bzw. aus den Experimenten geschätzt werden. Wie die Abbildung 7.18 zeigt, führt die Annahme der binomialen Verteilung der Inliers zu plausiblen Ergebnissen und kann durchaus als eine Näherung angenommen werden.

Bei n Feature-Punkten, die zuordenbar sind (d.h. in einem Überschneidungsgebiet beider Zeitreihen liegen)[2] und von uns mit dem Minimum aus der

[2]Beachte: Bei n handelt es sich um die Anzahl der Feature-Punkte und nicht um die

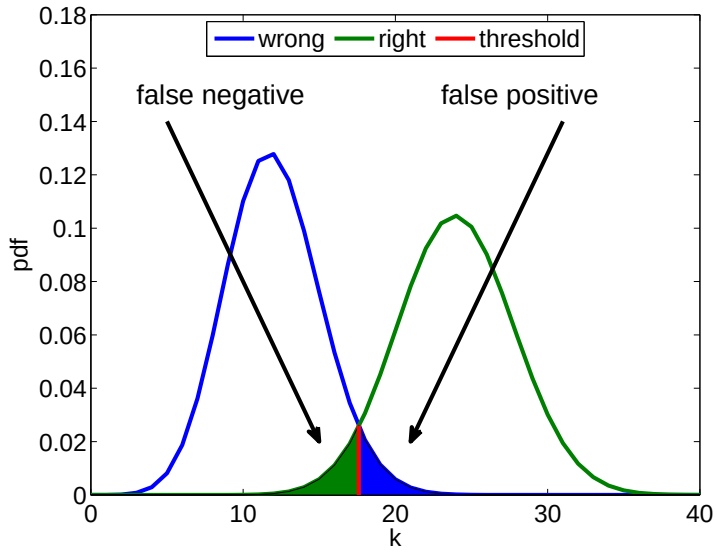

Abbildung 7.19: Die Wahl von c (im linken Bild $c = 1$ (rot) für $P_{a-priori}(r) = P_{a-priori}(w)$) für $\Lambda > c$ beeinflusst die beiden möglichen Fehlerarten: Vergrößert man c, so reduziert man die Anzahl der nicht passenden Zeitreihenpaaren, die man fälschlicherweise als passend *eingestuft hat (blaue Fläche), dafür vergrößert sich die Anzahl der passenden Zeitreihen, die man als* nicht passend *eingestuft hat (grüne Fläche). Mit der Wahl von c muss man sich für ein Trade-off entscheiden, dabei wird die Summe der beiden Fehler nie kleiner als die Schnittfläche beider Verteilungsdichten.*

Anzahl der Features in den beiden verglichenen Fingerprints abgeschätzt werden, und k Inliers beträgt die Wahrscheinlichkeit diesen Fall anzutreffen:

$$P(n,k) = P_{a-priori}(r) \cdot B(k;n,p_r) + P_{a-priori}(w) \cdot B(k;n,p_w),$$

mit den binominalen Verteilungen $B(\cdot;n,p_i)$ zu den Parametern n und p_i und den a-priori Wahrscheinlichkeiten $P_{a-priori}(i)$, dass das Zeitreihenpaar passend $(i = r)$ oder nicht passend $(i = w)$ ist. Betrachtet man nun den Likelihood-Quotienten

$$\Lambda := \frac{P_{a-priori}(r) \cdot B(k;n,p_r)}{P_{a-priori}(w) \cdot B(k;n,p_w)} = \frac{P_{a-priori}(r)}{P_{a-priori}(w)} \left(\frac{p_r}{p_w}\right)^k \left(\frac{1-p_r}{1-p_w}\right)^{n-k} \quad (7.1)$$

$$\Lambda_e := \left(\frac{p_r}{p_w}\right)^k \left(\frac{1-p_r}{1-p_w}\right)^{n-k}$$

und damit

$$\Lambda = \frac{P_{a-priori}(r)}{P_{a-priori}(w)} \Lambda_e,$$

Anzahl der wirklich gefundenen Zuordnungen, wie dies im vorherigen Abschnitt noch der Fall war.

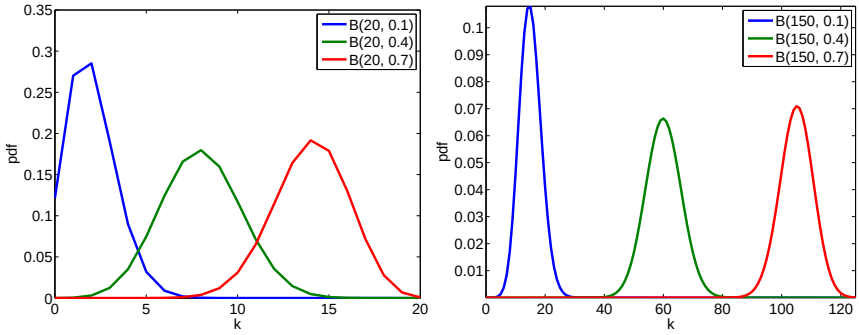

Abbildung 7.20: Es ist möglich, zwischen zwei Verteilungen zu unterscheiden, wenn die Schnittfläche beider gering ist. Dies wird durch einen großen Unterschied zwischen p_r und p_w und große n begünstigt. Linkes Bild: Die blaue Verteilung ($p_w = 0.1$) und rote Verteilung ($p_r = 0.7$) lassen sich gut unterscheiden, weil der Unterschied zwischen p_r und p_w groß ist. Rechtes Bild: Für große n lassen sich blau ($p_w = 0.1$) und grün ($p_r = 0.4$) bzw. grün ($p_w = 0.4$) und rot ($p_r = 0.7$) gut unterscheiden, was für kleine n noch nicht der Fall war.

so ist es wahrscheinlicher, dass es sich um ein passendes Zeitreihenpaar handelt, wenn $\Lambda > 1$ gilt. Man kann jedoch die Anzahl von "falsch positiven" bzw. "falsch negativen" beeinflussen, indem man $\Lambda > c$ mit $c \neq 1$ wählt (vergleiche Abbildung 7.19).

Klar ist auch (Abbildung 7.20):

- Je größer der Unterschied zwischen p_r und p_w, desto weniger Fehleinschätzungen wird man produzieren.

- Sind die Unterschiede zwischen p_r und p_w nicht so groß, so sind große n notwendig, um eine genaue Unterscheidung zu ermöglichen.

Es ist möglich, eine einfache Testfunktion zu konstruieren, die zu vorgegebenen k und n eines Vergleichsvorgangs zweier Fingerprints bestimmt, ob es sich dabei um passende oder nicht passende Zeitreihen handelt:

$$
\begin{aligned}
c < \Lambda \Leftrightarrow \\
\log c < \log \frac{P_{a-priori}(r)}{P_{a-priori}(w)} + k \cdot \log \frac{p_r}{p_w} + (n-k) \cdot \log \frac{1-p_r}{1-p_w} \Leftrightarrow \\
\log \frac{p_w \cdot (1-p_r)}{p_r \cdot (1-p_w)} k > \log \frac{P_{a-priori}(r)}{c \cdot P_{a-priori}(w)} + \log \frac{1-p_r}{1-p_w} n
\end{aligned}
$$

bzw.

$$
\begin{aligned}
& k > \alpha n + \beta, \\
& \alpha := \frac{\log(1-p_r) - \log(1-p_w)}{\log(p_w \cdot (1-p_r)) - \log(p_r \cdot (1-p_w))}, \\
& \beta := \frac{\log P_{a-priori}(r) - \log(c \cdot P_{a-priori}(w))}{\log(p_w \cdot (1-p_r)) - \log(p_r \cdot (1-p_w))}.
\end{aligned}
$$

Es soll vor allem $c = 1$ betrachtet werden, bei dem die Summe der beiden Fehler (falsch negativ und falsch positiv) minimal ist. Damit würde der Test für den in der Abbildung 7.18 illustrierten Testdatensatz B ($p_r = 0.564$, $p_w = 0.027$) und $P_{a-priori}(r) = P_{a-priori(w)}$ lauten:

$$
test(k,n) := \begin{cases} \text{passend} & k > 0.21n \\ \text{nicht passend} & k \leq 0.21n \end{cases}
$$

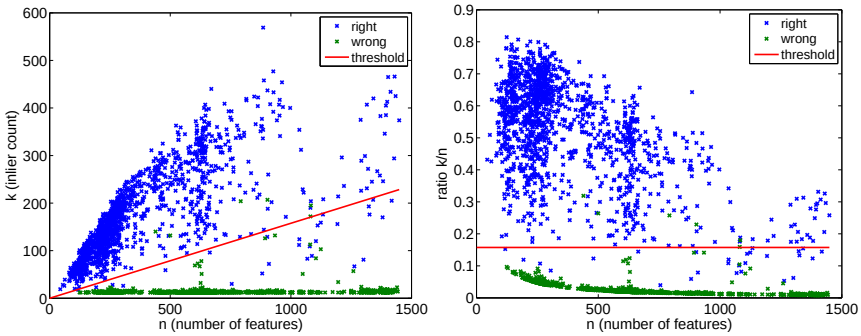

Abbildung 7.21: Der Verlauf der Testfunktion $test(k,n)$ für den Testdatensatz A. Dabei wurden passende (blau) und nur ca. 2000 nicht passende Zeitreihenpaare (grün), welche die meisten Inliers aufweisen, angezeigt. Vor allem in der rechten Abbildung, in der das Verhältnis $\frac{k}{n}$ betrachtet wird, fällt auf, dass eine einfache Gerade das Verhalten der Testfunktion nicht ausreichend genau beschreiben kann - die recall-Rate $\frac{k}{n}$ scheint von n abzuhängen.

Diese Vorgehensweise liefert gute Ergebnisse in Bezug auf die fälschlicherweise einander zugeordneten falschen Zeitreihenpaaren (falsch positiv) und etwas höhere Ungenauigkeit bei nicht gefundenen passenden Zeitreihenpaaren (falsch negativ).

Datensatz	A	B	C und D zusammen
falsch positiv (aus 1 Mio.)	3	1	11
falsch negativ (‰)	20	2	6

Betrachtet man die Abbildung 7.21, so stellt man fest, dass die in diesem Abschnitt vorgeschlagene Testfunktion vor allem für die großen n schlechte Ergebnisse liefert. Dafür sind folgende Gründe denkbar:

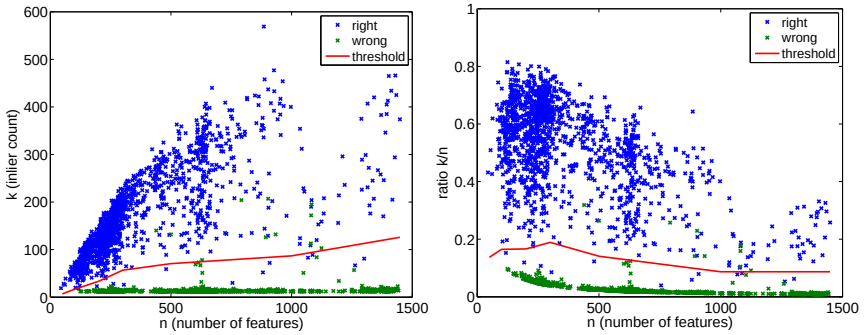

Abbildung 7.22: Stückweise Anpassung des Parameters α für die Testfunktion $test(k, n)$ liefert eine bessere Trennlinie im Vergleich zur Abbildung 7.21. Es ist jedoch ersichtlich, dass noch bessere Ergebnisse möglich sind, sollte die Trennlinie weiter unten verlaufen.

- Wie schon vermutet und diskutiert wurde, kann der Parameter p_r von der Anzahl der Features im Fingerprint abhängen (vergleiche Abschnitt 6.2.2 und insbesondere die Abbildung 6.5).

- Wie in der Abbildung 7.18 (rechts) gesehen, schätzt die binomiale Verteilung den Abfall der vorliegenden Verteilung der Inlier für die nicht passende Zeitreihenpaare zu konservativ ab - in Wirklichkeit passiert dieser viel schneller. D.h. die richtige Testfunktion sollte auch weniger steil für größere n verlaufen, womit die Anzahl der falschen negativen Antworten reduziert wäre, was vor allem auf der rechten Seite in der Abbildung 7.21 deutlich wird. Ein Verlauf der Testfunktion, der qualitativ ähnlich zum Verlauf der Signifikanzniveaus in der Abbildung 7.16 aussieht, wäre wünschenswert.

Trotz alldem ist dieses Modell gut anwendbar, vor allem wenn die Anzahl der Feature-Punkte in den Fingerprints nicht so stark variiert. Es liegt also auf der Hand, den Parameter α der Testfunktion in der Abhängigkeit von n zu wählen. Dafür müssen nur die Parameter $p_r(n)$ und $p_w(n)$ geschätzt werden.

Das Ergebnis einer solchen Vorgehensweise wurde in der Abbildung 7.22 dargestellt - die Anzahl der falschen negativen Antworten wurde dabei mehr als halbiert!

Die "Freihand-Heuristik"

Während $k < \alpha n$ eine gute Testfunktion für Fingerprints mit wenigen Features ist, wäre $k < const$ für größere n passender. Deswegen definieren und

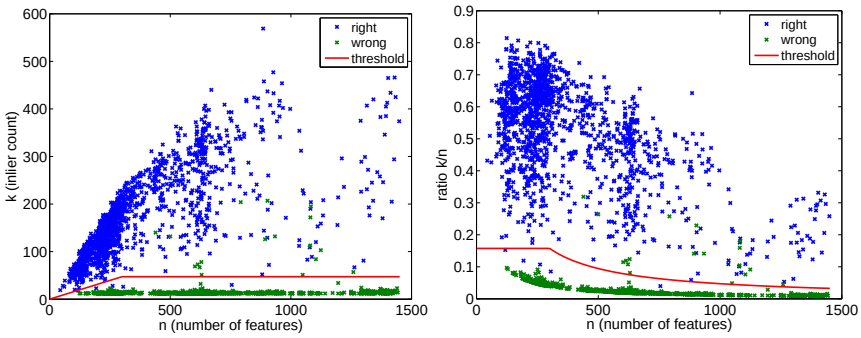

Abbildung 7.23: Die "Freihand Testfunktion" (rot) liefert eine kleinere Anzahl der falsch negativen Antworten (blau unter der roten Trennlinie), weil sie sich besser an den Verlauf der falschen Zeitreihenpaare (grün) anschmiegt.

betrachten wir die "Freihand-Testfunktion"

$$test_{N_0}(k,n) := \begin{cases} \text{passend} & (k > \alpha n \wedge n \leq N_0) \\ \text{passend} & (k > \alpha \cdot N_0 \wedge n > N_0) \\ \text{nicht passend} & \text{sonst.} \end{cases}$$

In der Tabelle unten zusammengefasste Ergebnisse werden für den Testdatensatz `A` in der Abbildung 7.23 präsentiert.

Datensatz	`A`	`B`	`C` und `D` zusammen
falsch positiv (aus 1 Mio.)	9	1	16
falsch negativ (‰)	4	2	5
gewähltes N_0	300	300	500

Die Freihand-Heuristik liefert also trotz ihrer Einfachheit mit unserem Standardverfahren (Suche in einer großen Datenbank) vergleichbare Ergebnisse.

7.8.3 Fazit

Das stochastische Modell, welches ohne a-priori Wissen auskommen sollte, weist für die Testdaten keine ausreichende Genauigkeit auf. Dies ist vor allem darauf zurückzuführen, dass die Annahme der stochastischen Unabhängigkeit nicht für die Zeitreihen von der gleichen Anlage gemacht werden kann.

Trotzdem ist es möglich, für jedes Zeitreihenpaar mit großer Genauigkeit zu sagen, ob diese zusammenpassend sind oder nicht. Die dafür notwendigen Parameter sind wenige und anhand von wenigen hundert Zeitreihen gut erlernbar.

Die besten Ergebnisse erzielen die "Freihand-Heuristik" und die etwas aufwendigere δ_{eff}-Heuristik. Ihre Ergebnisse stehen der Suche in einer Datenbank in nichts nach:

- mindestens 99.5% der passenden Zeitreihenpaare werden als solche erkannt.
- 99.999% der nicht passenden Zeitreihenpaare werden erkannt. Bedenkt man, dass beim normalen Produktionsprozess in 1000 Abfragen im Schnitt nur eine aus dieser Kategorie kommt. Da 5000 Bänder in etwa einer Woche des Betriebs entsprechen, so wird sich in ca. 400 Jahren nur ein falsches Zeitreihenpaar durchmogeln können!

7.9 Zusammenfassung

Der von uns vorgeschlagene Algorithmus erwies sich auf den vorliegenden Daten den bis dato etablierten Vorgehensweisen in puncto Genauigkeit und Schnelligkeit als überlegen. DTW, Shotgun und FFT-Alignment zeigten sich allesamt als zu langsam und nicht genügend robust gegenüber möglichen Messfehlern und Messausfällen (für Beispiele siehe Abschnitt 2.2.4), um in der Praxis eingesetzt zu werden.

Umso beeindruckender sind die Resultate unserer Vorgehensweise:

- Sowohl die X2- als auch DOG-Varianten beantworten 99.8% der Anfragen an eine große Datenbank richtig. Damit kann das Problem `A` (Abschnitt 2.2.1) als erfolgreich gelöst betrachtet werden.
- Es können mehrere hundert Zeitreihenvergleiche pro Sekunde auf einem Commodity-Singlecore-Prozessor durchgeführt werden. Die vorberechneten Fingerprints der Zeitreihen nehmen nur ca. $\frac{1}{5}$ des Speichers in Anspruch, der für die Originaldaten notwendig wäre. Damit kann das Problem `A` als *ausreichend effizient* gelöst betrachtet werden.
- Beide Varianten zeigten sich gegenüber großen Skalierungen der Zeitachse robust. Unser Algorithmus zeigt sich auch als äußerst robust gegenüber einer großen Anzahl möglicher Messfehler, deswegen ist keine Vorverarbeitung/Bereinigung bzw. Normalisierung der Zeitreihen notwendig.
- Die Suche mit kleinen Abschnitten ist erfolgreich, so lange die Datenqualität ausreichend ist. Auch die Suche mit einer Zeitreihe, von der in der Datenbank nur ein kleiner Teil vorhanden ist, ist möglich.
- Die auf dem $x^2 \cdot G(x, \sigma)$ induzierten Wavelet basierte X2-Variante ist der DOG-Variante leicht überlegen, so lange es nicht auf die Vorberechnungszeit ankommt.

- Unsere Vorgehensweise ermöglicht es, ohne große Probleme mehrere Spuren (z.B. Dicken- und Breitenprofile) bei der Suche auszunutzen, was vor allem für die Daten schlechter Qualität einen entscheidenden Vorteil bringen kann. An der Stelle sei vermerkt, dass eine gemeinsame Benutzung mehrerer Spuren für die etablierten Verfahren nicht ohne weiteres möglich ist: Die Fehler für verschiedene Dimensionen müssen unterschiedlich gewichtet werden. Diese Gewichtung muss von außen vorgegeben sein. Bei unserer Vorgehensweise entfallen diese Probleme vollkommen.

- Es ist durchaus denkbar, nur anhand von zwei vorhandenen Zeitreihen zu sagen, ob diese passend sind oder nicht. Die besten Ergebnisse erreicht man jedoch, wenn man einige wenige Parameter lernt: In mehr als 99.5% der Fälle werden die passenden Zeitreihen als solche erkannt, während für die nicht passenden Zeitreihenpaare nur in einer Abfrage von 10^5 ein Fehler gemacht wird. Damit kann das Problem `B` (Abschnitt 2.2.2) als ausreichend gut gelöst betrachtet werden.

Kapitel 8

Zusammenfassung und Ausblick

In dieser Arbeit untersuchten wir die Möglichkeit einer Verifizierung der Materialverfolgung in Walzwerken anhand von Breiten- bzw. Dickenprofilen. Diese Aufgabe lässt sich auf die Suche bis zu 10^5 Messpunkte langer Zeitreihen in einer mehrere tausend Einträge großen Datenbank reduzieren.

Die in der Zeitreihenanalyse etablierten Algorithmen, wie BDTW oder $\mathcal{L}^2$-Abstandsminimierung, waren dieser Aufgabe nicht gewachsen: Zum einen führen die langen Zeitreihen zu inakzeptablen Laufzeiten, zum anderen ist die Genauigkeit für die Zwecke der Materialverfolgung nicht ausreichend, was vor allem durch viele Messfehler in den Zeitreihen bedingt ist.

In unser Framework, das nicht nur schnell sondern auch besonders robust gegenüber einer großen Anzahl an Messstörungen (unter anderem Rauschen, Trends, Offsets und Ausreißer) ist, flossen Ideen aus der Musikindustrie (Shazam), Bio-Informatik (Shotgun) und Computer-Vision (SIFT) ein: Besonders interessante Zeitreihenabschnitte (*Features*) werden erst detektiert, dann in eine zum schnellen Vergleich geeignete Darstellung (*Deskriptor*) umgewandelt und letztendlich zu einem *Fingerprint* der Zeitreihe zusammengefasst. Die in der Vorverarbeitungsphase erstellten Fingerprints werden für eine schnelle Suche in der Datenbank verwendet.

Neben der Robustheit gegenüber den Messfehlern ist die Invarianz gegenüber Skalierung, Translation und Spiegelung der x- bzw. Zeitachse entscheidend.

Wir stellten fest, dass es sich bei den bis dato eher experimentell ermittelten Feature-Detektoren um eine kontinuierliche Wavelet-Transformation handelt. Damit kann das Framework auf ein mächtiges Analysewerkzeug zurückgreifen.

Der entstandene Algorithmus besticht durch seine Schnelligkeit und Genauigkeit:

- Es sind Vergleiche von einigen hundert typischen Dicken- bzw. Breitenprofilen pro Sekunde möglich ($\approx$ 10 mal schneller, als die uns bekannten bisherigen Verfahren).
- In mehr als 99.5% der Anfragen an die Datenbank mit Daten aus den realen Walzwerken wurde die richtige Antwort geliefert, während die bisherigen Verfahren nicht einmal 90% erreichen konnten.
- Eine Suche mit oder nach Teilstücken einer Zeitreihe ist möglich.

Diese Ergebnisse machten es der iba AG möglich, die Materialverfolgung in den Walzwerken mit Hilfe von Breiten- und Dickenprofilen zu prüfen und im Fehlerfall notwendige Korrekturhinweise zu liefern. Bis jetzt wurde der Algorithmus für Aluminium- und Stahlbänder mit Dicken- bzw. Breitenprofilen erfolgreich angewendet.

Nicht nur die Materialverfolgung, sondern auch ein aktives Eingreifen in den Fertigungsprozesses wurde erfolgreich ausprobiert: Dabei meldet das Framework einen schon nach wenigen hundert vermessenen Metern festgestellten Verwechsler, wonach die Operation abgebrochen wird, um die Anlage vor (weiteren) Schäden zu bewahren.

Es stellt sich die Frage, ob das Framework für weitere Materialien und Prozesse anwendbar ist. Eine Ausweitung auf weitere Zeitreihentypen wird sicherlich dazu führen, dass weitere Feature-Detektoren ihre Verwendung finden. Interessant ist, wie der "perfekte" Detektor mit den zu untersuchenden Zeitreihen zusammenhängt und vielleicht sogar aus diesen erzeugt werden kann.

Es könnte notwendig sein, die Geschwindigkeit der Vergleiche weiter zu erhöhen. Ein möglicher Weg wäre eine intelligente Reduktion der Anzahl von Features im Fingerprint. In dieser Arbeit wurden einige dafür geeignete Strategien untersucht, weitere Ansätze sollen genauer unter die Lupe genommen werden. Eine andere Möglichkeit die Geschwindigkeit der Vergleiche zu erhöhen liegt in der Parallelisierung. Dies wurde in dieser Arbeit nur stiefmütterlich behandelt und es konnte nicht das ganze Potenzial der modernen Hardware aufgedeckt werden.

Eine Optimierung des Vorberechnungsprozesses würde eine *on the fly* Berechnung der Fingerprints ermöglichen. Damit wäre eine effiziente Suche in nicht vorverarbeiteten Datenbeständen möglich.

Als abschließende Bemerkung sei noch auf den Anhang 9 hingewiesen, in dem zwei weitere Ideen zur Benutzung bzw. Weiterentwicklung des Frameworks kurz vorgestellt werden.

Anhang

Kapitel 9

Weitere Anwendungen

In diesem Kapitel sollen weitere Anwendungsmöglichkeiten/Einsatzmöglichkeiten des vorgestellten Frameworks, welche auf ähnlichen Ansätzen basieren, aufgezeigt werden. Dabei wird auf eine ausführliche Auswertung verzichtet: Vielmehr soll es sich ein *proof of principle* handeln.

9.1 Kopf-/Fußschrotterkennung

Eine der Anfangsmotivationen für die Diplomarbeit (Kemeter [41]) war die Schätzung des sogenannten Kopf/Fußschrottes: Die Anfangs- und Endabschnitte eines Metallbandes werden nach dem Walzen abgeschnitten, da diese keine ausreichende Qualität aufweisen. Man möchte feststellen, wie viel Schrott dabei entsteht, um z.B. die entstehenden Kosten besser einschätzen zu können.

Eine leichte Modifikation des Frameworks ermöglicht es, diese Aufgabenstellung erfolgreich zu erledigen:

Eigentlich interessieren wir uns im Framework nur für die Anzahl der Inlier. Doch die Koordinatentransformation des besten Alignments zwischen den Zeitreihen wird implizit mit ausgerechnet, denn sie wird benutzt, um die Inlier von Ausreißern zu unterscheiden. So kann die im Laufe der Berechnung gefundene Koordinatentransformation benutzt werden, um die Zeitreihen zu alignieren. Die Lage des Anfangs/Endes der zuletzt vermessenen Zeitreihe gibt die Lage des "Filetstücks" im originalen gewalzten Metallband an - die äußeren Stücke wurden als Kopf/Fußschrott abgeschnitten.

Auch wenn die absolute Genauigkeit der Koordinaten-Transformation nicht die höchste Bedeutung für unser Framework hat, ist ihre Qualität ausreichend für diese neue Aufgabe. Es gibt außerdem folgende Möglichkeiten sie zu erhöhen:

- Die Reduktion des Parameters δ des RANSAC-Algorithmus. Damit werden nur die äußerst zuverlässige Koordinatenzuordnungen berücksichtigt, was zu genaueren Ergebnissen führt.

- Es werden nur die Zuordnungen in unmittelbaren Nähe des Randes berücksichtigt, damit wird dem Umstand Rechnung getragen, dass die Annahme der linearen Koordinatentransformation zwischen den Zeitreihen am Rande nicht unbedingt stimmen muss.

In unseren Experimenten zeigte sich die erste Strategie als durchaus ausreichend. Die Abbildung 9.1 zeigt ein gefundenes Alignment, womit auch der Kopf- bzw. Fußschrott vermessen werden kann.

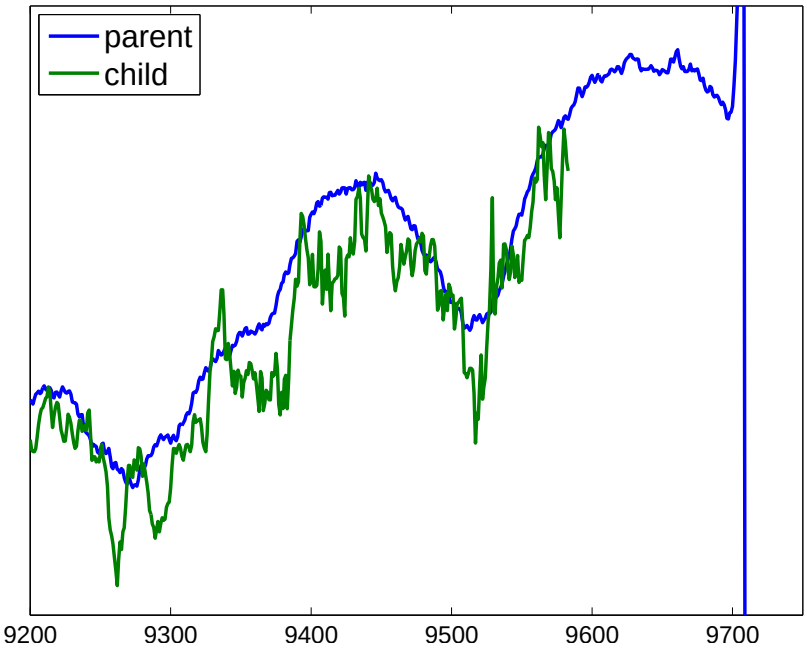

Abbildung 9.1: Ein Beispiel für den gefundenen Kopf/Fußschrottabschnitt für ein Metallband aus dem Testdatensatz `C`*. Die* parent*-Zeitreihe (blau) ist um ca.* 100 *Punkte länger, was ungefähr einer Länge von* 10 *Metern entspricht.*

9.2 Suche mit Röntgenaufnahmen

Die Verbreitung der Röntgengeräten in den Zahnarztpraxen führt dazu, dass für die meisten Personen mehrere dentale Röntgenaufnahmen existieren. Diese können gegebenfalls benutzt werden, um z.B. bei forensischen Untersuchungen das Opfer zu identifizieren. Dafür würde es reichen, einige Röntgenaufnahmen anzufertigen und mit einer Datenbank der vermissten

Personen abzugleichen. Ein Framework, welches solche Suche ermöglichen würde, könnte auch z.B. bei großen Naturkatastrophen, wie einem Tsunami oder einem Erdbeben, zum Einsatz kommen.

In den letzten Jahren wurden einige Bemühungen unternommen, um solch ein Framework zu entwickeln:

- In Chen [17] werden Zahn- und Füllungenkonturen für den Vergleich der Aufnahmen benutzt. Die oft schlechte Qualität der Aufnahmen verlangt jedoch eine Interaktion des Benutzers.
- Die pixelbasierte *Phase-Only Correlation* (POC) wird benutzt (Ito u. a. [38]), um den besten Treffer für eine Aufnahme in einer Datenbank zu ermitteln. Der große Vorteil - dies passiert völlig automatisch - überwiegt dabei den Nachteil, dass die POC nur unter Rotationen und Translationen invariant bleibt, während der Zusammenhang zwischen zwei aus unterschiedlichen Geometrien aufgenommenen Röntgenaufnahmen komplizierter ist.

Beide Ansätze wurden anhand einer kleinen Röntgenaufnahmen-Datenbank (33 für Chen [17] und 25 für Ito u. a. [38]) getestet.

Unser generisches Framework kann auch auf den zweidimensionalen Fall erweitert werden, und zwar wie folgt:

- Als Detektor wird ein zweidimensionaler DoG-Filter benutzt, wie z.B. in Lowe [48].
- Für die Features werden SIFT-Deskriptoren mit lokaler Orientierung benutzt.
- Als eine Vereinfachung nehmen wir an, dass alle gefundenen Features in einer Ebene liegen. Deswegen kann die Koordinatentransformation zwischen zwei Aufnahmen als eine Homographie bzw. projektive Transformation (Hartley und Zisserman [36]) aufgefasst werden. Dabei werden die homogenen Koordinaten der Features $x = [x_1 : x_2 : x_3]^t$ und $y = [y_1 : y_2 : y_3]^t$ mit Hilfe einer projektiven Matrix $H \in \mathbb{R}^{3\times 3}$ verknüpft:
$$y = Hx.$$
Es soll beachtet werden, dass die Matrix H nur 8 unabhängige Parameter hat: Wegen der Homogenität der Koordinaten, kann die Matrix mit einem beliebigen Faktor ($\neq 0$) multipliziert werden, ohne das Ergebnis zu verändern. Es reichen also 8 Gleichungen, die aus 4 Zuordnungen berechnet werden können. Dafür verwenden wir den *direct Linear Transformation (DLT) Algorithmus* (Hartley und Zisserman [36]).

- Mit dem RANSAC-Algorithmus können die Ausreißer der Zuordnungen aussortiert und die “beste” Koordinatentransformation gefunden werden.

- Die Anzahl der *Inlier* wird als Maß dafür benutzt, ob die zwei untersuchten Aufnahmen zu einander passen. Aus einer Datenbank wird diejenige Aufnahme als bester Treffer gewählt, welche die meisten *Inlier* aufweist.

9.2.1 Testumgebung

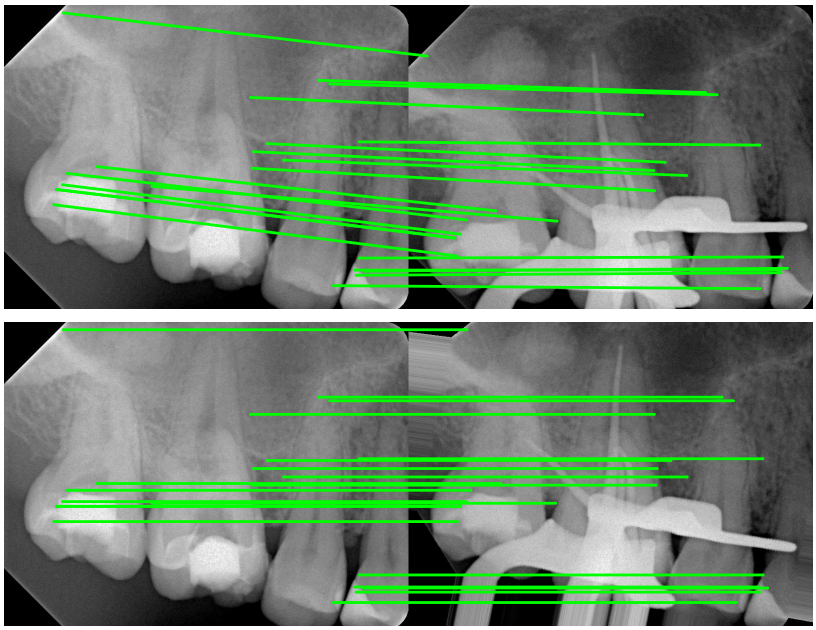

Abbildung 9.2: Oben: Alle gefundenen Zuordnungen, die durch die grünen Abschnitte verbunden sind. Unten: Alle Zuordnungen, die als richtig zugeordnet eingestuft wurden, nach der Anwendung der gefundenen Koordinatentransformation.

Für die Untersuchungen wurden uns 60 Röntgenaufnahmen von der Röntgenabteilung der Klinik für Zahn-, Mund- und Kieferkrankheiten der Universität Mainz zur Verfügung gestellt. Diese zu 15 anonymisierten Patienten gehörenden Daten wurden über mehrere Jahre mit unterschiedlichen Geräten erstellt. Über diese Zeitspanne gab es viele Veränderungen: neue Füllungen, Kunst- oder auch gezogene Zähne.

Um zu bestimmen, welche der Aufnahmen eines Patienten überhaupt einander zuordenbar sind, wurden die Aufnahmen drei menschlichen Experten vorgelegt, die sich diese anschauten und die zusammengehörigen Aufnahmen benannten. Ein Paar der Röntgenaufnahmen gilt als zuordenbar, falls sie zu

einem Patienten gehören und von mindesten einem der drei menschlichen Experten als zusammenpassend erkannt wurde.

Insgesamt wurden auf diese Art und Weise 109 zuordenbare Paare festgestellt.

Die Bewertung des Frameworks passiert wie folgt:

- Eine Röntgenaufnahme wird mit allen anderen vorhandenen Aufnahmen abgeglichen. Jeder Abgleich bekommt eine Bewertung.
- Ein zuordenbares Paar gilt als gefunden, wenn seine Bewertung besser ist, als die Bewertungen der Vergleiche mit nicht passenden Aufnahmen.
- Die Anzahl der richtig gefundenen zuordenbaren Paare gibt die Güte des Frameworks an.

Abbildung 9.3: Zwei Beispiele für zuordenbare Aufnahmenpaare, die vom Framework nicht gefunden wurden. Man sieht, dass die bestimmten Zuordnungen (grüne Verbindungen) falsch bestimmt wurden.

9.2.2 Ergebnisse

Bei unseren Tests wurden 51 der 109 zuordenbaren Paare vom unserem Framework gefunden. Einige Beispiele werden in den Abbildung 9.2 und 9.3 dargestellt.

Der Hauptvorteil der hier vorgeschlagenen Vorgehensweise besteht darin, dass sie gegenüber einer größeren Anzahl der Koordinatentransformationen als die bisherigen Ansätze invariant ist. Weitere Untersuchungen sollten sich auf die Entwicklung von Detektoren und Deskriptoren konzentrieren, die unter für die Röntgenaufnahmen üblichen Transformationen stabil bzw. invariant bleiben.

Kapitel 10

Mathematische Grundlagen

In diesem Kapitel besprechen wir einige mathematische Grundlagen, die als eine kurze Erinnerung dienen sollten. Das Kapitel kann ausgelassen werden und erst bei Bedarf durchgelesen werden.

10.1 Faltung

Zuerst beschäftigen wir uns mit Faltung.

Definition 1 *(Faltung $*$)*
Seien $f, g : \mathbb{R}^d \to \mathbb{R}$ Funktionen, dann heißt die Funktion

$$h(x) := (f * g)(x) := \int_{\mathbb{R}^d} f(x-y) \cdot g(y)\, dy$$

die Faltung *der Funktionen f und g. Dieses Konstrukt ist z.B. für $f, g \in \mathcal{L}^1\left(R^d\right)$ wohldefiniert, vergleiche Amann und Escher [2].*

Oft wird eine der gefalteten Funktionen als *Kernel* bezeichnet.
Die Faltung besitzt folgende Eigenschaften:

- Kommutativität: $f * g = g * f$,
- Assoziativität: $(f * g) * h = f * (g * h)$,
- Distributivität: $(f + g) * h = f * h + g * h$.

Dies kann leicht anhand der Definition eingesehen werden.
Sind f und g differenzierbar, so ist es auch $f * g$ und es gilt

$$\partial(f * g) = (\partial f) * g = f * (\partial g)$$

Anschließend noch das Faltungstheorem für die Fouriertransformation.

Definition 2 *(Fouriertransformation $\hat{\mathcal{F}}$)*
Sei $f : \mathbb{R}^d \to \mathbb{C}$ eine Funktion, dann heißt die Funktion

$$\hat{\mathcal{F}}(f)[w] := \frac{1}{\sqrt{2\pi}^d} \int_{\mathbb{R}^d} f(x) \cdot e^{-ixw}\, dx \quad w \in \mathbb{R}^d$$

die Fouriertransformierte von f. Auch hier ist die Fouriertransformierte für $f \in \mathcal{L}^1\left(R^d\right)$ wohldefiniert (Amann und Escher [2]).

Man kann sich leicht überzeugen, dass die inverse Fourier-Transformation

$$\hat{\mathcal{F}}^{-}(f)[w] := \frac{1}{\sqrt{2\pi}^d} \int_{\mathbb{R}^d} f(x) \cdot e^{ixw}\, dx \quad w \in \mathbb{R}^d$$

lautet.

Des Weiteren erwähnen wir das wichtige Faltungstheorem:

Theorem 1 *(Das Faltungstheorem)*
Für $f, g \in \mathcal{L}^1\left(R^d\right)$ gilt

$$\hat{\mathcal{F}}(f * g) = \sqrt{2\pi}^d \hat{\mathcal{F}}(f) \cdot \hat{\mathcal{F}}(g)$$

Für einen Beweis kann unter anderem Amann und Escher [2] konsultiert werden.

10.2 Diskrete Funktionen

Im Laufe dieser Arbeit werden vor allem die diskreten Funktionen betrachtet, die oft als Zeitreihen bezeichnet werden.

Wir machen eine Vereinfachung und betrachten in diesem Abschnitt periodische Funktionen auf einem endlichen Gitter Ω:

$$\Omega := [0, N-1] \cap \mathbb{N}$$

Die Menge dieser Funktionen bezeichnen wir mit

$$Abb_{\text{per}}(\Omega, \mathbb{R}) := \{f : \Omega \to \mathbb{R}\}$$

bzw.

$$Abb_{\text{per}}(\Omega, \mathbb{C}) := \{f : \Omega \to \mathbb{C}\}$$

für den komplexen Fall, der für die Fouriertransformation von Interesse ist.

Definition 3 *(Diskrete (periodische) Faltung $*$)*
Seien $f, g \in Abb_{per}(\Omega, \mathbb{R})$, dann heißt die Funktion

$$h(x) := (f * g)(x) := \sum_{i=0}^{N-1} f(x-i) \cdot g(i)$$

die (diskrete) Faltung von f und g.

Dabei wird mit $x - i$ die Äquivalenzklasse modulo N bezeichnet.

Auch die Fouriertransformation kann für die diskreten Funktionen definiert werden:

Definition 4 *(Diskrete Fouriertransformation $\hat{\mathcal{F}}$)*
Sei $f \in Abb_{per}(\Omega, \mathbb{C})$, dann heißt die Funktion

$$\hat{\mathcal{F}}(f)[w] := \sum_{j=0}^{N-1} f(j) \cdot e^{-2\pi i \frac{j \cdot w}{N}}$$

die (diskrete) Fourier-Transformierte von f.

Die inverse Fourier-Transformation lautet dann

$$\hat{\mathcal{F}}^{-}(f)[w] := \frac{1}{N} \sum_{j=0}^{N-1} f(j) \cdot e^{2\pi i \frac{j \cdot w}{N}}$$

Auch im diskreten Fall gilt das Faltungstheorem:

Theorem 2 *(Das diskrete Faltungstheorem)*
Für $f, g \in Abb_{per}(\Omega, \mathbb{C})$ gilt

$$f * g = \hat{\mathcal{F}}^{-}\left(\hat{\mathcal{F}}(f) \cdot \hat{\mathcal{F}}(g)\right)$$

Beweis:
Wir rechnen die Identität einfach nach und benutzen dabei die folgende Eigenschaft:

$$\sum_{j=0}^{N-1} e^{2\pi i \frac{j \cdot (k-k')}{N}} = N \delta_{k,k'}$$

$$\delta_{k,k'} = \begin{cases} 1 & k \equiv k' \mod N \\ 0 & \text{sonst} \end{cases}$$

Man sieht dies leicht ein:

1. Fall $k' = k + a \cdot N$ für $a \in \mathbb{Z}$:

$$\sum_{j=0}^{N-1} e^{2\pi i \frac{j \cdot aN}{N}} = \sum_{j=0}^{N-1} 1 = N$$

2. Fall $k' = k + m + a \cdot N$ für $m = 1, \ldots, N-1$ und $a \in \mathbb{Z}$ unter Benutzung der Eigenschaften der geometrischen Reihe:

$$\begin{aligned}\sum_{j=0}^{N-1} e^{2\pi i \frac{j \cdot m}{N}} &= \frac{1 - \left(e^{2\pi i \frac{m}{N}}\right)^N}{1 - e^{2\pi i \frac{m}{N}}} \\ &= \frac{1-1}{1 - e^{2\pi i \frac{m}{N}}} = 0\end{aligned}$$

Damit

$$\begin{aligned}&\hat{\mathcal{F}}^{-}\left(\hat{\mathcal{F}}(f) \cdot \hat{\mathcal{F}}(g)\right)(x) = \frac{1}{N} \sum_{j=0}^{N-1} \hat{\mathcal{F}}(f)(j) \cdot \hat{\mathcal{F}}(g)(j) \cdot e^{2\pi i \frac{j \cdot x}{N}} \\ &= \frac{1}{N} \sum_{j=0}^{N-1} \left(\sum_{k=0}^{N-1} f(k) \cdot e^{-2\pi i \frac{j \cdot k}{N}} \cdot \sum_{k'=0}^{N-1} g(k') \cdot e^{-2\pi i \frac{j \cdot k'}{N}} \right) \cdot e^{2\pi i \frac{j \cdot x}{N}} \\ &= \frac{1}{N} \sum_{k'=0}^{N-1} \sum_{k=0}^{N-1} f(k) g(k') \sum_{j=0} N - 1 e^{-2\pi i \frac{j \cdot (k + k' - x)}{N}} \\ &= \frac{1}{N} \sum_{k'=0}^{N-1} \sum_{k=0}^{N-1} f(k) g(k') \cdot N \cdot \delta_{x-k,k'} \\ &= \sum_{k=0}^{N-1} f(k) g(x-k) \\ &= (f * g)(x)\end{aligned}$$

□

Fast Fourier Transformation (FFT)

Die diskrete Fouriertransformation lässt sich als Matrix-Vektor-Produkt darstellen, denn sie ist ein Automorphismus auf $\mathbb{C}^N$. Die naive Berechnung würde dann $O(N^2)$ dauern. Die Laufzeit kann jedoch mit Hilfe der Fast Fouriertransformation auf $O(N \log N)$ reduziert werden, was zur großen Popularität der diskreten Fouriertransformation führte.

Der an der Funktionsweise der FFT und deren inversen Variante interessierte Leser wird auf Nussbaumer [54], Cooley und Tukey [19] und Frigo und Johnson [29] verwiesen.

Die Benutzung der FFT ermöglicht, das Falten zweier periodischen Funktionen in $O(N \log N)$: Dank dem Faltungstheorem kann die Faltung als zwei FFT (in $O(N \log N)$) mit der anschließenden punktweisen Multiplikation (in $O(N)$) und der inversen FFT (in $O(N \log N)$ realisiert werden.

10.2.1 Lineare Interpolation

Es besteht oft die Notwendigkeit die diskreten Funktionen zu interpolieren, z.B. wenn eine Koordinatentransformation angewendet wird. Obwohl es mehrere Möglichkeit existieren, setzen wir im Laufe der Arbeit verstärkt die lineare Interpolation ein, was vor allem durch die schnelle Berechnung dieser Art der Interpolation bedingt ist.

Eine $f : [0, N) \to \mathbb{C}$ können wir zu $\hat{f} \in Abb_{\text{per}}(\Omega, \mathbb{C})$ via

$$\begin{aligned} \hat{f}(x) &= (1 - (x - x_{fl}))f(x_{fl}) + (x - x_{fl})f(x_{fl} + 1) \\ x_{fl} &:= \lfloor x \rfloor = \max\{z \leq x \mid z \in \mathbb{Z}\} \end{aligned}$$

fortsetzen.

Lineare Interpolation und die $\mathcal{L}^2$-Norm

Verwendet man die lineare Interpolation in Verbindung mit der $\mathcal{L}^2$-Norm, so sollte man einiger Probleme beim Optimieren bewusst sein, welche anhand des folgenden Beispiel ersichtlich sind:

In unserem Szenario betrachten wir zwei Zeitreihen $f(x) = a \cdot x^2$ und $g(x) = f(x) + err(x)$, wobei von $err(x) \stackrel{d}{=} N(0, \sigma)$ ausgegangen wird. Versuchen wir nun also das beste Aligment zu finden, so erwarten wir, dass $T_{(1,0)}$ gefunden wird, vorausgesetzt die Zeitreihen sind lang genug. Betrachten wir jedoch den Verlauf der Zielfunktionen

$$\begin{aligned} z_\beta(\beta) &:= \|f - g(\cdot - \beta)\|_2 \quad \text{für } \beta \in \mathbb{R} \\ z_\alpha(\alpha) &:= \|f - g(\alpha \cdot)\|_2 \quad \text{für } \alpha \in \mathbb{R}_{>0} \end{aligned}$$

so stellen wir ein seltsames Verhalten (Abbildung 10.1) in der Nähe von $\beta = 0$ und $\alpha = 1$ fest: Die Zielfunktionen wachsen!

Um dieses Verhalten zu erklären, betrachten wir $\tilde{f} \equiv 0$ und das dazu passende $\tilde{g} = err(x)$. Dann gilt zum einen

$$E(z_\beta(0)) = E\left(\left\|\tilde{f} - \tilde{g}\right\|_2\right) = n\sigma,$$

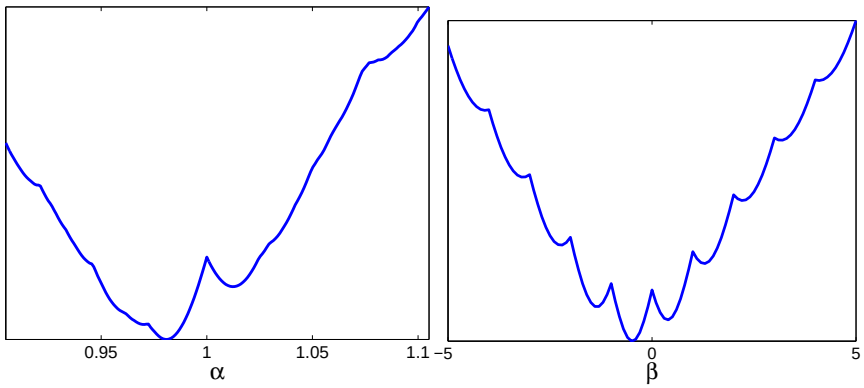

Abbildung 10.1: Unerwarteterweise ergeben die Verläufe der Zielfunktionen $z_\beta(\beta)$ (rechts) und $z_\alpha(\alpha)$ (links) ein Maximum für $\beta = 0$ und $\alpha = 1$, obwohl $T_{(1,0)}$ die richtige Transformation ist!

und zum anderen wegen $g(x) \stackrel{d}{=} N(0,\sigma)$ i.i.d.

$$\begin{aligned} E(z_\beta(0.5)) &= \sum_x E\left(\frac{g(x)+g(x+1)}{2}\right) \\ &= n\frac{\sigma}{\sqrt{2}}. \end{aligned}$$

Damit wird $\beta = 0.5$ mit hoher Wahrscheinlichkeit als beste Transformation gewählt. Ähnliche Überlegungen kann man auch für z_α anstellen.

Spielen diese Effekte eine größere Rolle (was immer dann der Fall sein wird, wenn f keine hohen Frequenzen hat), so sollte man den Einsatz der Nearest-Neighbor-Interpolation

$$\hat{f}(x) = f(round(x))$$

in Betracht ziehen. Eine andere Möglichkeit wäre, nur $\beta \in \mathbb{Z}$ und $\alpha \in \mathbb{Z} \cup \{z|\frac{1}{z} \in \mathbb{Z}\}$ zuzulassen, um dann mit Hilfe von Interpolation die subpixel genaue Bestimmung von α und β zu erreichen.

Es stellt sich allerdings die Frage, in wieweit eine subpixel genau Bestimmung möglich ist, wenn f keine hohen Frequenzen beinhaltet.

10.3 Gaußfunktion

Definition 5 *(Die Gaußfunktion)*

$$G^d(x,\sigma) := \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^d e^{-\frac{x^t x}{2\sigma^2}} \quad x \in \mathbb{R}^d$$

bzw. für $d = 1$:

$$G(x,\sigma) := G^1(x,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \quad x \in \mathbb{R}$$

Eigenschaften von $G(x,\sigma)$:

- Es handelt sich um eine symmetrische Funktion.
- $\int_{\mathbb{R}} |G(x,\sigma)|dx = 1$.

Die Ableitungen nach x der Gaußfunktion $G(x,\sigma)$ sehen wie folgt aus:

$$\begin{aligned}
\partial_x G(x,\sigma) &= \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \left(-\frac{x}{\sigma^2}\right) \\
\partial_{xx} G(x,\sigma) &= \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \left(\frac{x^2-\sigma^2}{\sigma^4}\right) \\
\partial_{xxx} G(x,\sigma) &= \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \left(\frac{3\sigma^2 x - x^3}{\sigma^6}\right)
\end{aligned}$$

und die Ableitung nach σ:

$$\partial_\sigma G(x,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \left(\frac{x^2-\sigma^2}{\sigma^3}\right)$$

Es gilt also

$$\partial_{xx} G(x,\sigma) = \frac{1}{\sigma} \partial_\sigma G(x,\sigma). \tag{10.1}$$

Die mehrdimensionale Variante gilt für beliebige Dimension d:

$$\begin{aligned}
G^d(x,\sigma) &= \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^d e^{-\frac{\sum_{i=1}^d x_i^2}{2\sigma^2}}, \\
\triangle_x G^d(x,\sigma) &= G^d(x,\sigma) \cdot \left(\sum_{i=1}^d \frac{x_i^2}{\sigma^4} - \frac{d}{\sigma^2}\right), \\
\partial_\sigma G^d(x,\sigma) &= G^d(x,\sigma) \cdot \left(\sum_{i=1}^d \frac{x_i^2}{\sigma^3} - \frac{d}{\sigma}\right),
\end{aligned}$$

und damit

$$\triangle_x G^d(x,\sigma) = \frac{1}{\sigma} \partial_\sigma G^d(x,\sigma).$$

Eigenschaften von $\partial_x G(x,\sigma)$:

- Es handelt sich um eine ungerade Funktion.

- $\int_{\mathbb{R}} |\partial_x G(x,\sigma)| dx = 2 \cdot \frac{1}{\sqrt{2\pi}\sigma}$.

Weitere Untersuchungen von $F_\sigma(x) := \partial_{xx} G(x,\sigma)$ ergeben Folgendes:

- $F_\sigma(x)$ ist gerade.
- $\int_{\mathbb{R}} F_\sigma(x)\, dx = 0$.
- Nullstellen sind $(\pm\sigma,\, 0)$.
- Minimum ist $\left(0,\, -\frac{1}{\sqrt{2\pi}\sigma^3}\right)$
- Maxima sind $\left(\pm\sqrt{3}\sigma,\, \frac{2}{\sqrt{2\pi}\sigma^3} e^{-\frac{3}{2}}\right)$
- $\int_{[\sigma,\,\infty)} F_\sigma(x)\, dx = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{1}{2}}$
- $\int_{[-\sigma,\,\sigma]} F_\sigma(x)\, dx = -\frac{2}{\sqrt{2\pi}\sigma^2} e^{-\frac{1}{2}}$
- $\int_{\mathbb{R}} |F_\sigma(x)|\ dx = \frac{4}{\sqrt{2\pi}\sigma^2} e^{-\frac{1}{2}}$.

10.3.1 Gaußfunktion und Skalierungen der x-Achse

In diesem kurzen Unterabschnitt wird der Leser auf eine interessante Eigenschaft der Gaußfunktion und deren Ableitungen aufmerksam gemacht (für $\alpha \neq 0$):

$$|\alpha| G(\alpha x, |\alpha|\sigma) = |\alpha| \frac{1}{\sqrt{2\pi}|\alpha|\sigma} e^{\frac{-\alpha^2 x^2}{2\alpha^2\sigma^2}} = G(x,\sigma)$$

oder als Folgerung daraus:

$$G(\alpha x, \sigma) = \frac{1}{|\alpha|} G(x, \frac{1}{|\alpha|}\sigma)$$

Ähnlich ist auch das Verhalten der ersten

$$\begin{aligned} |\alpha|^2 G_x(\alpha x, |\alpha|\sigma) &= G(x,\sigma) \quad \text{bzw.} \\ G_x(\alpha x, \sigma) &= \frac{1}{|\alpha|^2} G_x(x, \frac{1}{|\alpha|}\sigma) \end{aligned}$$

und der zweiten

$$|\alpha|^3 G_x(\alpha x, |\alpha|\sigma) = G(x,\sigma) \quad \text{bzw.}$$
$$G_x(|\alpha|x,\sigma) = \frac{1}{|\alpha|^3} G_x(x, \frac{1}{|\alpha|}\sigma)$$

Ableitungen.

10.3.2 Faltungen der Gaußfunktionen

Die Faltung zweier Gaußfunktionen ergibt wieder eine Gaußfunktion:

$$[G(\cdot,\sigma_1) * G(\cdot,\sigma_2)](x) = G(x,\sigma^*) \qquad \sigma^* := \sqrt{\sigma_1^2+\sigma_2^2} \Rightarrow$$
$$[G(\cdot,\sigma_1) * \cdots * G(\cdot,\sigma_n)](x) = G(x,\sigma^*) \qquad \sigma^* := \sqrt{\sum_{i=1}^{n}\sigma_i^2}$$

Dies gilt nach folgender Rechnung:

$$\begin{aligned}
[G(\cdot,\sigma_1) * G(\cdot,\sigma_2)](x) &= \int_{\mathbb{R}} \frac{1}{2\pi\sigma_1\sigma_2} e^{-\frac{\sigma_2^2(x-y)^2+\sigma_1^2y^2}{2\sigma_1^2\sigma_2^2}} dy \\
&= \int_{\mathbb{R}} \frac{1}{2\pi\sigma_1\sigma_2} e^{-\frac{(\sigma_1^2+\sigma_2^2)\left(y-\frac{\sigma_2^2 x}{\sigma_1^2+\sigma_2^2}\right)^2+\frac{\sigma_1^2\sigma_2^2x^2}{\sigma_1^2+\sigma_2^2}}{2\sigma_1^2\sigma_2^2}} dy \\
&= \int_{\mathbb{R}} \frac{1}{2\pi\sigma_1\sigma_2} e^{-\frac{(\sigma_1^2+\sigma_2^2)\left(y-\frac{\sigma_2^2 x}{\sigma_1^2+\sigma_2^2}\right)^2+\frac{\sigma_1^2\sigma_2^2x^2}{\sigma_1^2+\sigma_2^2}}{2\sigma_1^2\sigma_2^2}} dy \cdot e^{-\frac{x^2}{2(\sigma_1^2+\sigma_2^2)}} \\
&= \frac{1}{\sqrt{2\pi(\sigma_1^2+\sigma_2^2)}} e^{-\frac{x^2}{2(\sigma_1^2+\sigma_2^2)}}
\end{aligned}$$

Für die Faltung mit der Ableitung einer Gaußfunktion gilt dann:

$$\begin{aligned}
[G(\cdot,\sigma_1) * G_x(\cdot,\sigma_2)](x) &= \partial_x [G(\cdot,\sigma_1) * G(\cdot,\sigma_2)](x) \\
&= G_x(x,\sigma^*) \qquad \sigma^* := \sqrt{\sigma_1^2+\sigma_2^2}
\end{aligned}$$

10.3.3 Näherungen von Gaußfunktion und deren Ableitungen

Für die Faltung mit dem Gaußkernel wird oft die Gaußfunktion durch

$$\tilde{G}_a(x,\sigma) := G(x,\sigma) \cdot \mathbb{I}_{[-a,a]}$$

bzw. ähnlich für deren Ableitungen angenähert.

Welchen Fehler macht man dabei in Abhängigkeit vom gewählten a? Das heißt welchen Anteil hat

$$\int_{[-a,\,a]} |f(x)|\,dx$$

von

$$\int_{\mathbb{R}} |f(x)|\,dx$$

für $f = G,\ G_x,\ G_{xx}, \ldots$? Numerische Berechnungen liefern:

f	$a = \sigma$	$a = 2\sigma$	$a = 3\sigma$	$a = 4\sigma$
$G(x,\sigma)$	0.68%	0.95%	0.997%	0.9999%
$\partial_x G(x,\sigma)$	0.39%	0.86%	0.989%	0.9997%
$\partial_{xx} G(x,\sigma)$	0.50%	0.78%	0.972%	0.9989%

10.3.4 Näherung der Gaußfunktion durch ein Rechteck

Wird die Gaußfunktion mit $\hat{G}_a$ angenähert, ist es möglich, die Faltung in $O(a \cdot n)$ zu berechnen. Die Faltung kann in $O(n)$ bewerkstelligt werden, falls man die Gaußfunktion mit einem Rechteck

$$R(x,\Delta) := \begin{cases} 0 & x < -\Delta \\ \frac{1}{2\cdot\Delta} & -\Delta \le x \le \Delta \\ 0 & \Delta < x \end{cases},$$

annähert, was allerdings auf Kosten der Genauigkeit geht. Zu beachten: $\|R\|_1 = 1$.

Die Frage nach der besten Wahl von Δ in Abhängigkeit von σ beantworten wir mit Hilfe von numerischen Experimenten: Für eine Testfunktion $t(x) \stackrel{d}{=} U([0,1])$ für $\forall x$ werden Faltungen mit $G(x,\sigma)$ und $R(x,\Delta)$ und dasjenige Δ bestimmt, für welches der $\mathcal{L}^2$-Abstand zwischen $G{*}t$ und $R{*}t$ am geringsten ist (siehe Abbildung 10.2). Diese Tests lieferten $\Delta \approx \frac{3}{2}\sigma$ als den besten Zusammenhang.

Die erste Ableitung $\frac{G_x(x,\sigma)}{\|G_x\|_1}$ kann durch

$$D(x,\Delta) := \begin{cases} 0 & x < -\Delta \\ \frac{1}{2\Delta} & -\Delta \le x \le 0 \\ -\frac{1}{2\Delta} & 0 < x \le \Delta \\ 0 & \Delta < x \end{cases}$$

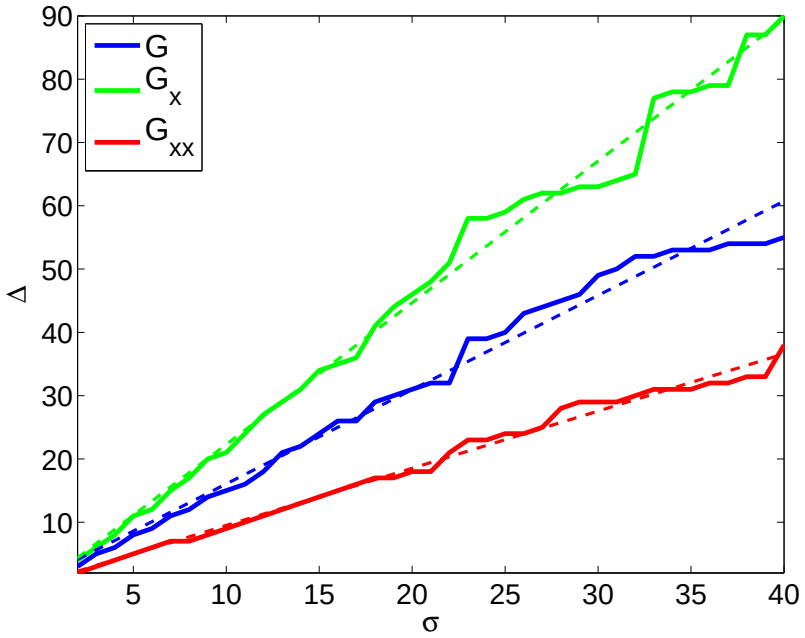

Abbildung 10.2: Die numerisch ermittelten Zusammenhänge zwischen σ und Δ bei Näherung der Gaußfunktion durch Rechtecke. Vor allem für kleine σ-Werte stimmen die durchgelegten Ausgleichgeraden mit den Ergebnissen der Simulationen gut überein.

angenähert werden. Die zweite Ableitung G_{xx} nähern wir durch

$$DD(x,\Delta) := \begin{cases} 0 & x < -3\cdot\Delta \\ \frac{A_{DD}}{\Delta} & -3\cdot\Delta \le x \le -\Delta \\ -\frac{2A_{DD}}{\Delta} & -\Delta < x < \Delta \\ \frac{A_{DD}}{\Delta} & \Delta \le x \le 3\cdot\Delta \\ 0 & 3\cdot\Delta < x \end{cases}$$

mit $A_{DD} = \frac{1}{8}\cdot\frac{4}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2}}$ an.

Numerische Experimente legen den Zusammenhang $\Delta \approx \frac{9}{10}\sigma$ für die zweite Ableitung nahe (Abbildung 10.2).

10.3.5 Gaußfunktion als Low-Pass-Filter

Eine Faltung mit der Gaußfunktion hat den Effekt der Anwendung eines Low-Pass-Filters. In diesem kleinen Abschnitt möchten wir die Low-Pass-Filter-Eigenschaften von G, G_x und G_{xx} untersuchen. Mit dem Wissen, welche Frequenzen vorhanden sind, kann man die Abtastfrequenz festlegen, wie

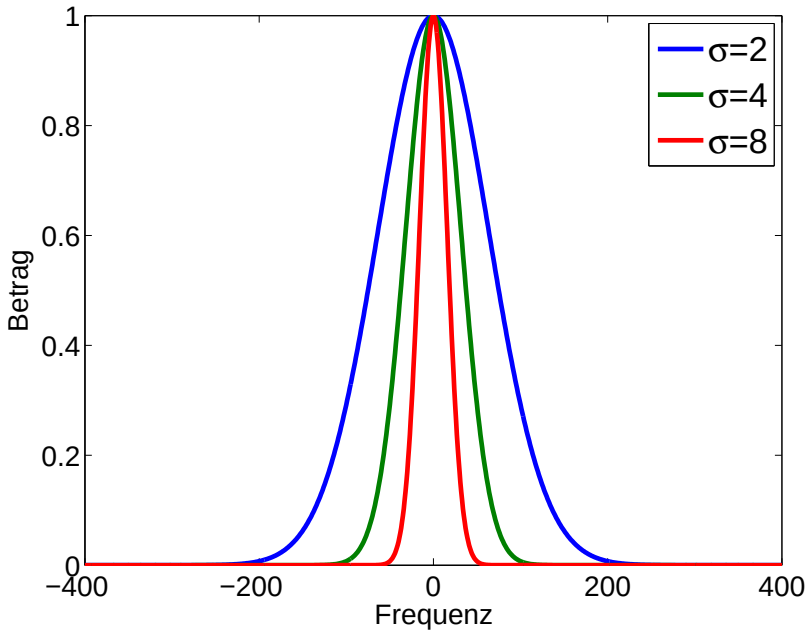

Abbildung 10.3: Die Fourierspektren von $G(x, 2)$, $G(x, 4)$ und $G(x, 8)$. Die Abstastrate beträgt in diesem Beispiel 800 *und damit können nach dem Abtasttheorem die Frequenzen bis zu* 400 *korrekt dargestellt werden. Nach der Faltung mit $G(x, 2)$ werden die Frequenzen größer als* 200 *mit einem Faktor kleiner 10^{-2} gedämpft und können praktisch vernachlässigt werden. Deswegen ist es ausreichend, in diesem Fall die Abstastrate zu halbieren -* 400 *Stützpunkte wären dann ausreichend.*

es vom Nyquist-Shannon-Abtasttheorem gefordert wird:

$$f_{abtast} > 2 \cdot f_{max}$$

d.h. die Abtastfrequenz sollte mindestens doppelt so groß sein wie die größte Frequenz im Spektrum.

Benutzt man die Eigenschaft der Fouriertransformation

$$f * g = \hat{\mathcal{F}}^{-} \left(\hat{\mathcal{F}}(f) \cdot \hat{\mathcal{F}}(g) \right)$$

so wird klar, dass es reicht, die Beträge der Fourierkoeffizienten einer Funktion zu betrachten, um zu wissen, welche Frequenzen durch die Faltung mit dieser Funktion gefiltert werden.

Als Beispiel betrachten wir zuerst die (diskrete) Fouriertransformierte von $G(x, 2)$ (Abbildung 10.3, blau) und kommen zum Schluss, dass eine Halbierung der Abtastfrequenz möglich wäre.

Die Abbildung 10.3 zeigt auch, dass nach der Faltung mit $G(x, 4)$ 200 Stützpunkte ausreichend wären , nach $G(x, 8)$ - sogar nur 100.

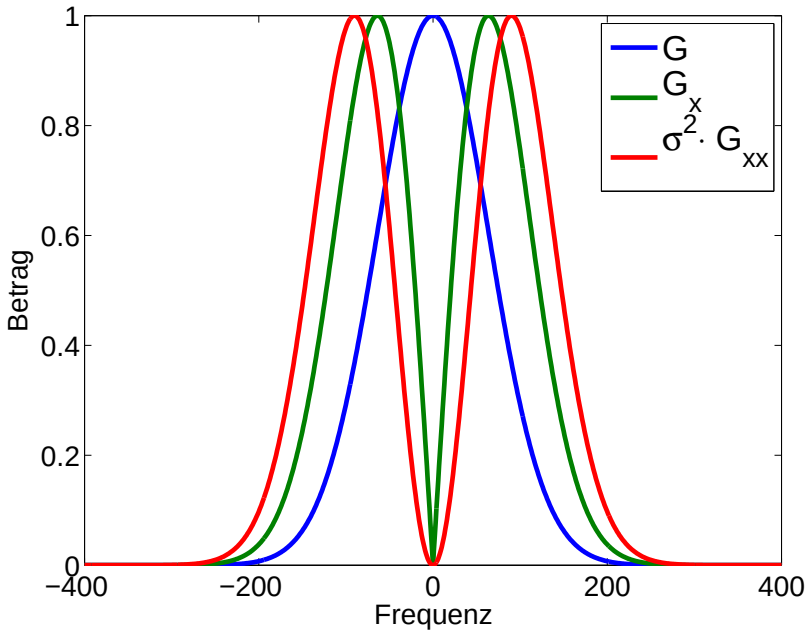

Abbildung 10.4: G_{xx} unterdrückt höhere Frequenzen weniger stark als G, dabei wurde das Maximum der Amplituden auf 1 *normiert.*

Je breiter die Gaußkurve, desto schmaler ist die Fouriertransformierte. So könnte man erwarten, dass G_{xx} die höheren Frequenzen stärker "unterdrückt" als G_x und G, doch dies ist nicht der Fall, wie man sich anhand von der Abbildung 10.4 überzeugen kann.

Das Fazit dieses Abschnitts lautet: Ist die Abtastung ausreichend für das Produkt der Faltung mit dem DoG-Filter bzw. mit $\sigma^2 G_{xx}(x, \sigma)$, so ist sie auch ausreichend für das Produkt der Faltung mit $G_x(x, \sigma)$ oder $G(x, \sigma)$.

10.4 Downsampling vs. Decimation

Nun möchten wir der Frage nachgehen, wie man richtig die Auflösung einer Zeitreihe verringert. Dabei interessieren wir uns vor allem für die Halbierung der Auflösung, wie es von uns z.B. für die Berechnung der Gauß-Pyramide (Abschnitt 5.1.1) verwendet wird.

Die einfachste Methode, die Auflösung zu reduzieren, wäre das sogenannte Downsampling: Im Falle einer Halbierung würde jedes zweite Element der Zeitreihe gelöscht. Diese Vorgehensweise ist vom Nyquist-Shannon-Abtasttheorem nur dann erlaubt, wenn in der Zeitreihe keine hohen Frequenzen (höher als $\frac{f_{max}}{2}$) vorhanden sind. Wird dies nicht beachtet, so können unerwünschte Muster auftreten, die in der Bildverarbeitung als Moiré-Effekt

bekannt sind (siehe Abbildung 10.5).

Das in der Bildverarbeitung als Decimation bekanntes Verfahren umgeht das Problem mit dem Moiré-Effekt, indem das Bild mit einem Low-Pass-Filter gefaltet wird, bevor es downgesampelt wird.

Wie wir schon gesehen haben, kann $G(x, 2)$ als Low-Pass-Filter verwendet werden, der die Frequenzen oberhalb von $\frac{f_{max}}{2}$ so gut wie auslöscht. Deswegen wird von uns bei der Halbierung der Auflösung nicht $f(x)$ sondern $[f * G(\cdot, 2)](x)$ mit halber Auflösung abgetastet.

Den Moiré-Effekt und den Unterschied zwischen Downsampling und Decimation kann man in der Abbildung 10.5 sehen.

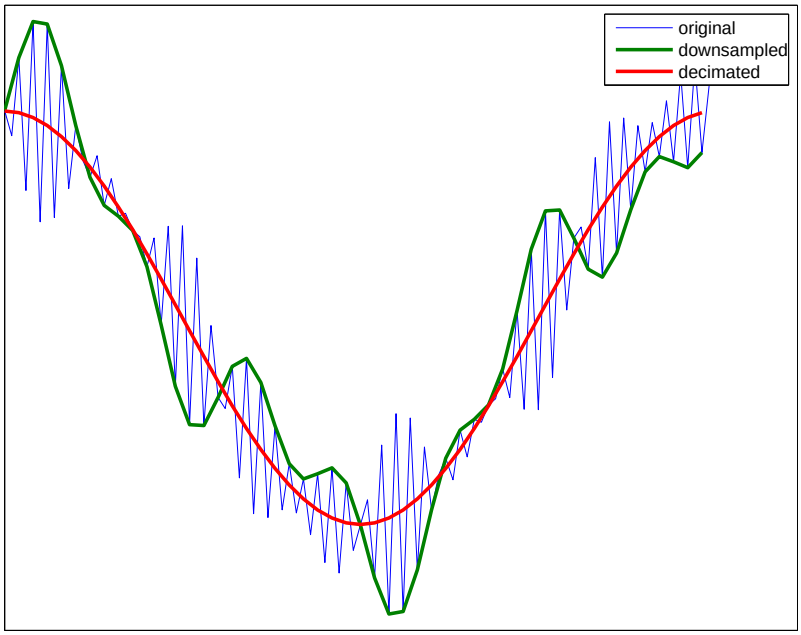

Abbildung 10.5: Die Entstehung des Moiré-Effekts beim Downsampling: Aliasing hoher Frequenzen. Die originale Funktion (blau) besteht aus zwei Frequenzen: einer niedrigen und einer hohen. Beim einfachen Downsampling wird die Bedingung des Abtasttheorems verletzt, so dass das Ergebnis statt nur einer niedrigen Frequenz noch eine weitere höhere Frequenz (grün) beinhaltet. Eine mögliche Lösung ist die Faltung der originalen Funktion mit einem Low-Pass-Filter mit dem anschließenden Downsampling. Dieses als Decimation bekanntes Verfahren führt zum roten Verlauf.

Anschließend sei noch auf das Problem der Vorgehensweise zu Halbierung der Auflösung hingewiesen, bei der zwei benachbarte Werte der originalen Funktion f gemittelt werden, d.h. die neu reduzierte Zeitreihe f^h wird durch die Vorschrift

$$f_i^h = \frac{f_{2i} + f_{2i+1}}{2} \quad i = 0, \ldots, \frac{n}{2}$$

gegebenen. Es kann also als Faltung mit dem Kernel

$$mean(x) = \begin{cases} \frac{1}{2} & 0 \leq x < 2 \\ 0 & \text{sonst} \end{cases}$$

mit dem anschließenden Downsampling aufgefasst werden.

Es gibt zwei Gründe, diese Vorgehensweise nicht zu benutzen (siehe auch Abbildung 10.6):

1. Der Kernel $mean(x)$ ist nicht symmetrisch, deswegen kommt es nach der Faltung auch zu einer (kleinen) translatorischen Verschiebung.

2. Die Low-Pass-Filter-Eigenschaften des $mean(x)$-Kernels sind denen von $G(x, 2)$ unterlegen.

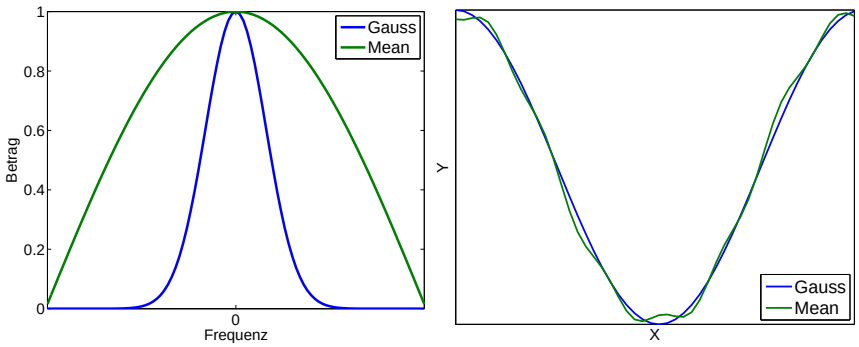

Abbildung 10.6: Das linke Bild zeigt das Spektrum der beiden Kernels: Offensichtlich besitzt $G(x, 2)$ die besseren Low-Pass-Filter-Eigenschaften. Im rechten Bild werden die Ergebnisse der Decimation mit Hilfe von Gauß- und $mean(x)$-Kerneln verglichen.

Literaturverzeichnis

[1] Agrawal, Rakesh ; Faloutsos, Christos ; Swami, Arun: Efficient Similarity Search In Sequence Databases, 1993

[2] Amann, Herbert ; Escher, Joachim: *Analysis III*. 1rd. Birkhaeuser-Verlag, Basel/Boston/Berlin, 2001

[3] Anhaus, Horst: *Verfahren und Vorrichtung zur Identifizierung eines Teilstücks eines Halbzeugs*. 2007

[4] Batista, Gustavo E. A. P. A. ; Wang, Xiaoyue ; Keogh, Eamonn J.: A Complexity-Invariant Distance Measure for Time Series. In: *SDM*, 2011, S. 699–710

[5] Bay, Herbert ; Ess, Andreas ; Tuytelaars, Tinne ; Van Gool, Luc: Speeded-Up Robust Features (SURF). In: *Comput. Vis. Image Underst.* 110 (2008), Juni, Nr. 3, S. 346–359

[6] Beckmann, Norbert ; Kriegel, Hans-Peter ; Schneider, Ralf ; Seeger, Bernhard: The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In: *SIGMOD Conference*, 1990, S. 322–331

[7] Bentley, Jon L. ; Stanat, Donald F. ; Jr., E.Hollins W.: The complexity of finding fixed-radius near neighbors. In: *Information Processing Letters* 6 (1977), Nr. 6, S. 209–212

[8] Berndt, Donald J. ; Clifford, James: Using Dynamic Time Warping to Find Patterns in Time Series. In: *KDD Workshop*, 1994, S. 359–370

[9] Besl, Paul J. ; McKay, Neil D.: A Method for Registration of 3-D Shapes. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 14 (1992), Februar, Nr. 2, S. 239–256

[10] Boyer, Robert S. ; Moore, J. S.: A fast string searching algorithm. In: *Commun. ACM* 20 (1977), Nr. 10, S. 762–772

[11] Brown, M. ; Lowe, D. G.: Recognising Panoramas. In: *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, IEEE Computer Society, 2003 (ICCV '03)

[12] Burt, Peter J.: Fast filter transform for image processing. In: *Computer graphics and image processing* (1981)

[13] Burt, Peter J. ; Edward ; Adelson, Edward H.: The Laplacian Pyramid as a Compact Image Code. In: *IEEE Transactions on Communications* 31 (1983), S. 532–540

[14] Calonder, Michael ; Lepetit, Vincent ; Fua, Pascal: *BRIEF: Binary robust independent elementary features.* 2010

[15] Canny, J: A Computational Approach to Edge Detection. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 8 (1986), Juni, Nr. 6, S. 679–698

[16] Chakrabarti, Kaushik ; Keogh, Eamonn ; Mehrotra, Sharad ; Pazzani, Michael: Locally adaptive dimensionality reduction for indexing large time series databases. In: *ACM Trans. Database Syst.* 27 (2002), Nr. 2, S. 188–228

[17] Chen, Hong: *Automatic forensic identification based on dental radiographs.* East Lansing, MI, USA, Dissertation, 2007. – AAI3264152

[18] Cole, Richard ; Hariharan, Ramesh ; Paterson, Mike ; Zwick, Uri: Tighter Lower Bounds on the Exact Complexity of String Matching. In: *SIAM J. Comput.* 24 (1995), Nr. 1, S. 30–45

[19] Cooley, James W. ; Tukey, John W.: An Algorithm for the Machine Calculation of Complex Fourier Series. In: *Mathematics of Computation* 19 (1965), Nr. 90, S. 297–301

[20] Cormen, Thomas ; Leiserson, Charles ; Stein, Ronald Rivest C.: *Introduction to Algorithms.* 2rd. MIT Press and McGraw-Hill, 2001

[21] Cover, Thomas M. ; Thomas, Joy A.: *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing).* Wiley-Interscience, 1991

[22] Datar, Mayur ; Immorlica, Nicole ; Indyk, Piotr ; Mirrokni, Vahab S.: Locality-sensitive hashing scheme based on p-stable distributions. In: *Proceedings of the twentieth annual symposium on Computational geometry.* New York, NY, USA : ACM, 2004 (SCG '04), S. 253–262

[23] Daubechies, Indrid: *Ten Lectures on Wavelets.* Society for Industrial and Applied Mathematics, 1992

[24] Duda, Richard O. ; Hart, Peter E.: *Pattern Classification and scene analysis.* John Wiley & Sons, Inc., 1973

[25] Duda, R.O. ; Hart, P.E.: Use of the Hough Transformation to Detect Lines and Curves in Pictures. 1971. – Forschungsbericht. SRI Project 8259 Comm. ACM, Vol 15, No. 1

[26] Feller, William: *An introduction to probability theory and its applications. Vol. II.* New York : John Wiley & Sons Inc., 1971 (Second edition)

[27] Fischler, Martin A. ; Bolles, Robert C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In: *Commun. ACM* 24 (1981), Juni, Nr. 6, S. 381–395

[28] Friedman, Jerome H. ; Bentley, Jon L. ; Finkel, Raphael A.: An Algorithm for Finding Best Matches in Logarithmic Expected Time. In: *ACM Transactions on Mathematics Software* 3 (1977), September, Nr. 3, S. 209–226

[29] Frigo, Matteo ; Johnson, Steven G.: The Design and Implementation of FFTW3. In: *Proceedings of the IEEE* 93 (2005), Nr. 2, S. 216–231. – Special issue on "Program Generation, Optimization, and Platform Adaptation"

[30] Gelb, A.: *Applied optimal estimation.* MIT Press, 1974

[31] Gionis, Aristides ; Indyk, Piotr ; Motwani, Rajeev: Similarity Search in High Dimensions via Hashing. In: *Proceedings of the 25th International Conference on Very Large Data Bases.* San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1999 (VLDB '99), S. 518–529

[32] Gool, Luc J. V. ; Moons, Theo ; Ungureanu, Dorin: Affine/ Photometric Invariants for Planar Intensity Patterns. In: *Proceedings of the 4th European Conference on Computer Vision-Volume I - Volume I.* London, UK, UK : Springer-Verlag, 1996 (ECCV '96), S. 642–651

[33] Guttman, Antonin: R-Trees: A Dynamic Index Structure for Spatial Searching. In: Yormark, Beatrice (Hrsg.): *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, ACM Press, 1984, S. 47–57

[34] Hanke-Bourgeois, Martin: *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens.* 1. Teubner Verlag, 2002

[35] Harris, C. ; Stephens, M.: A Combined Corner and Edge Detector. In: *Proceedings of the 4th Alvey Vision Conference*, 1988, S. 147–151

[36] Hartley, R. I. ; Zisserman, A.: *Multiple View Geometry in Computer Vision.* Second. Cambridge University Press, 2004

[37] Holschneider, Matthias: *Wavelets. An Analysis Tool.* Oxford Science Publications, 1995

[38] Ito, Koichi ; Nikaido, Akira ; Aoki, Takafumi ; Kosuge, Eiko ; Kawamata, Ryota ; Kashima, Isamu: A Dental Radiograph Recognition System Using Phase-Only Correlation for Human Identification. In: *IEICE Transactions* 91-A (2008), Nr. 1, S. 298–305

[39] Jolliffe, I.T.: *Principal Component Analysis.* Springer Verlag, 1986

[40] Ke, Yan ; Sukthankar, Rahul: PCA-SIFT: a more distinctive representation for local image descriptors. In: *Proceedings of the 2004 IEEE computer society conference on Computer vision and pattern recognition.* Washington, DC, USA : IEEE Computer Society, 2004 (CVPR'04), S. 506–513

[41] Kemeter, Matthias: *Effizientes Alignment von Stahlband-Fingerprints*, Johannes Gutenberg-Universitaet Mainz, Diploma Thesis, 2008

[42] Keogh, E. ; Ratanamahatana, A.: Everything you know about dynamic time warping is wrong. In: *3rd Workshop on Mining Temporal and Sequential Data, in conjunction with 10th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD-2004), Seattle, WA* (2004)

[43] Keogh, Eamonn ; Chakrabarti, Kaushik ; Pazzani, Michael ; Mehrotra, Sharad: Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. In: *JOURNAL OF KNOWLEDGE AND INFORMATION SYSTEMS* 3 (2000), S. 263–286

[44] Keogh, Eamonn ; Ratanamahatana, Chotirat A.: Exact indexing of dynamic time warping. In: *Knowl. Inf. Syst.* 7 (2005), Nr. 3, S. 358–386

[45] Leutenegger, Stefan ; Chli, Margarita ; Siegwart, Roland: BRISK: Binary Robust Invariant Scalable Keypoints. In: *Proceedings of the IEEE International Conference on Computer Vision*, 2011

[46] Lin, Jessica ; Keogh, Eamonn ; Wei, Li ; Lonardi, Stefano: Experiencing SAX: a novel symbolic representation of time series. In: *Data Min. Knowl. Discov.* 15 (2007), Oktober, Nr. 2, S. 107–144

[47] Lipowsky, Constanze ; Dranischnikow, Egor ; Göttler, Herbert ; Gottron, Thomas ; Kemeter, Mathias ; Schömer, Elmar: *Alignment of Noisy and Uniformly Scaled Time Series*

[48] Lowe, David G.: *Distinctive Image Features from Scale-Invariant Keypoints.* 2003

[49] Maes, Frederik ; Collignon, AndrÃ© ; V, Dirk ; Marchal, Guy ; Suetens, Paul: Multimodality Image Registration by Maximization of Mutual Information. In: *IEEE transactions on Medical Imaging* 16 (1997), S. 187–198

[50] Mallat, Stephane: *A Wavelet Tour of Signal Processing*. 3rd. Academic Press, 2009

[51] Mikolajczyk, Krystian ; Schmid, Cordelia: *A PERFORMANCE EVALUATION OF LOCAL DESCRIPTORS*. 2005

[52] Muja, Marius ; Lowe, David G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: *In VISAPP International Conference on Computer Vision Theory and Applications*, 2009, S. 331–340

[53] Neubeck, Alexander ; Gool, Luc V.: Efficient Non-Maximum Suppression. In: *Proceedings of the 18th International Conference on Pattern Recognition - Volume 03*. Washington, DC, USA : IEEE Computer Society, 2006 (ICPR '06), S. 850–855

[54] Nussbaumer, H. J.: *Fast Fourier Transform and Convolution Algorithms*. 2nd. Springer Verlag, 1990

[55] Ortiz, Raphael: FREAK: Fast Retina Keypoint. In: *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Washington, DC, USA : IEEE Computer Society, 2012 (CVPR '12), S. 510–517

[56] Rakthanmanon, Thanawin ; Campana, Bilson ; Mueen, Abdullah ; Batista, Gustavo ; Westover, Brandon ; Zhu, Qiang ; Zakaria, Jesin ; Keogh, Eamonn: Searching and mining trillions of time series subsequences under dynamic time warping. In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA : ACM, 2012 (KDD '12), S. 262–270

[57] Rublee, Ethan ; Rabaud, Vincent ; Konolige, Kurt ; Bradski, Gary: ORB: An Efficient Alternative to SIFT or SURF. In: *International Conference on Computer Vision*. Barcelona, 2011

[58] Sakoe, Hiroaki ; Chiba, Seibi: *Dynamic programming algorithm optimization for spoken word recognition*. 1990

[59] Silpa-Anan, Chanop ; Hartley, Richard: An Algorithm for Finding Best Matches in Logarithmic Expected Time. In: *CVPR* (2008)

[60] Smith, T. F. ; Waterman, M. S.: Identification of common molecular subsequences. In: *Journal of molecular biology* 147 (1981), Nr. 1, S. 195–197

[61] Venter, J. C. ; Adams, M. D. ; Myers, E. W. ; Li, P. W. ; Mural, R. J. ; Sutton, G. G. ; Smith, H. O. ; Yandell, M. ; Evans, C. A. ; Holt, R. A.: *The Sequence of the Human Genome, , 291: 1304-*. 2001

[62] Viola, Paul ; Wells, William M.: Alignment by Maximization of Mutual Information. In: *Int. J. Comput. Vision* 24 (1997), September, Nr. 2, S. 137–154

[63] Wang, Avery L.: An industrial-strength audio search algorithm. In: *Proceedings of the 4 th International Conference on Music Information Retrieval*, 2003

[64] Witkin, Andrew P.: Scale-space filtering. In: *Proceedings of the Eighth international joint conference on Artificial intelligence - Volume 2*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1983 (IJCAI'83), S. 1019–1022

[65] Xie, Jierui ; Beigi, Mandis S.: A scale-invariant local descriptor for event recognition in 1D sensor signals. In: *Proceedings of the 2009 IEEE international conference on Multimedia and Expo*. Piscataway, NJ, USA : IEEE Press, 2009 (ICME'09), S. 1226–1229

Glossar

$\mathcal{L}^2$-Abstand **37**

1-precision **123**, 137

alignierende Metrik 22

BRIEF Binary Robust Independent Elementary Features 119, 139

Cross-Correlation **38**, 114, 135

CWT continuous wavelet transform **67**

Decimation 88, **232**

Deskriptor 44, 73, 76, 97, **113**

DoG Difference-of-Gaussian **65**, 90, 148, 175

DTW Dynamic Time Warping **47**, 176

Faltung 86, **219**, 227

Feature 56, 76

Fingerprint 56, 75

Frame 57, 76

Gaußfunktion 224

Hough-Transformation **100**

i.i.d. independent and identically distributed 52

Inlier **102**

Interpolation 26, 223

Koordinatentransformation 20

LoG Laplacian-of-Gaussian **61**, 69, 89, 148

LSH Local Sensitive Hashtable **158**

mehrspuriger Fingerprint 193

Messprozessoperator 28

Messung 25

Mutual Information 40

Nachbarschaft 95, 170

normalisierte Zeitreihe 48

Observable 19

Orientierung 76, 98, 115, 117, 142

PAA Piecewise Aggregate Approximatio. **46**

PCA Principal Component Analysis 118, 139

Problem `A` 29, 31, 175

Problem `B` 29, 31, 175, 195

Quantisierung 119, 142, 189

R*-Baum 162

Randomisierte k-d-Bäume 164

Randomisierte linked cells 161

RANSAC-Verfahren RANdom SAmple Consensus **100**, 196

recall 123, 137

SAX Symbolic Aggregate approXimation **49**, 176

Scale-Space-Filtering 58

Scale-Space-Funktion 58, 86, 87, 97, 113

Scatter-Plot 134, 138

Shotgun **51**, 176

SIFT Scale-Invariant-Feature-Transform 116, 135, 175

Spiegelung 21, 22

statistische Methode (zur Reduktion der Anzahl der Features) **168**, 170, 183, 189

SURF Speeded Up Robust Features 117

Translation 21, 22

Wavelet **67**, 89

X2 Auf der Dichte $\phi = x^2 \cdot G(x, \sigma)$ basiertes Wavelet/Detektor 71, 148, **176**

Zielscheibenfehler 145