technische universität
dortmund

fakultät für informatik

Endbericht

PG 548: Cool-IP

8. Juli 2011

Teilnehmer:
Kevin Adrian, Fabian Bürger, Jens Bürger, Marcel Fitzner, Hanno Jaspers, Jan Kleemann, Patrick Norder, Christina Tillmann, Kai Winnekens, Alper Yegenoglu, Sebastian Zach

Betreuer:
Dr. Frank Weichert
Dipl.-Inform. Constantin Timm

Lehrstuhl Informatik VII
(Graphische Systeme)
Technische Universität Dortmund

Lehrstuhl Informatik XII
(Eingebettete Systeme)
Technische Universität Dortmund

fakultät für
informatik

Inhaltsverzeichnis

Mathematische Notation

Notation	**Bedeutung**
$\mathbb{N}$	Menge der natürliche Zahlen
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}^+$	Menge der positiven reellen Zahlen
$\mathbf{M}$	Matrix
$\mathbf{M}^\mathsf{T}$	Matrix $\mathbf{M}$ transponiert
$\boldsymbol{v}$	Vektor
$\boldsymbol{v}_i$	i-tes Element des Vektors $\boldsymbol{v}$
$\|\boldsymbol{v}\|$	Euklidische Norm des Vektors $\boldsymbol{v}$
$A = \{A_1, A_2, ..., A_n\}$	n-elementige Menge A, bestehend aus den Elementen A_1 bis A_n
$\|A\|$	Anzahl der Elemente der Menge A
$\|a\|$	Betrag des Skalars a
$x = \lfloor a \rfloor$	a abgerundet; die größte Zahl $x \in \mathbb{N}$ mit $x \leq a$
$x = \lceil a \rceil$	a aufgerundet; kleinste Zahl $x \in \mathbb{N}$ mit $x \geq a$
$[a, b]$	geschlossenes Intervall von a bis b, mit $a \leq b$
$\operatorname{sgn}(v)$	Signumfunktion
$\operatorname{dom}(f)$	Definitionsbereich der Funktion f
$\measuredangle(\boldsymbol{p}, \boldsymbol{q})$	Winkel zwischen den Vektoren $\boldsymbol{p}$ und $\boldsymbol{q}$
$O(g)$	Groß-O-Notation; $f(n) \in O(g(n))$, wenn $f(n)$ asymptotisch nicht schneller wächst als $g(n)$

1. Einführung

Das Leben beschleunigt sich immer mehr. Geschäftsleute nehmen Termine auf drei Kontinenten in zwei Tagen wahr. Die immer öfter auftretenden viralen Infektionen können sich so auch immer schneller ausbreiten. In Asien werden an Flughäfen fiebrige Menschen mit Wärmebildkameras identifiziert, um eine Ausbreitung von Krankheiten zu verhindern [Ste09] (siehe Abbildung 1.1). Das verdeutlicht, dass eine schnelle Virusdetektion benötigt wird, um Pandemien zu bekämpfen und vermeiden zu können. Doch herkömmliche Verfahren bieten keine Sicherheit. Der Nachweis vom Humanen-Immundefizienz-Virus (HIV) hat beispielsweise eine diagnostische Lücke[1] von 3 Monaten. Der bisher verwendete enzymgekoppelte Immunadsorptionstest (ELISA, Akronym für *enzyme-linked immunosorbent assay*) basiert auf dem Nachweis der zu den Viren passenden Antikörper. Diese müssen sich nach der Infektion erst im Körper bilden. Das Verfahren an sich ist zeitaufwendig: Der Test ist nur in bestimmten Labors möglich, zu denen die Probe transportiert werden muss. Dann muss die Probe erst aufbereitet und anschließend das Verfahren zum Nachweis der Antikörper durchgeführt werden. Einzelheiten zu diesem Verfahren können bei Engvall und Perlmann [EP71] oder Goldsby et al. [GKOK03] nachgelesen werden.

1.1. Das PAMONO-Verfahren

Mit dem PAMONO-Verfahren (*Plasmon Assisted Microscopy of Nano-Size Objects*) ist der direkte Nachweis einzelner Viren in wässrigen Lösungen wie Blut oder Speichel möglich. Das

[1] Der Zeitraum zwischen Infektion und dem Zeitpunkt, zu dem diese mit Tests sicher nachgewiesen werden kann.

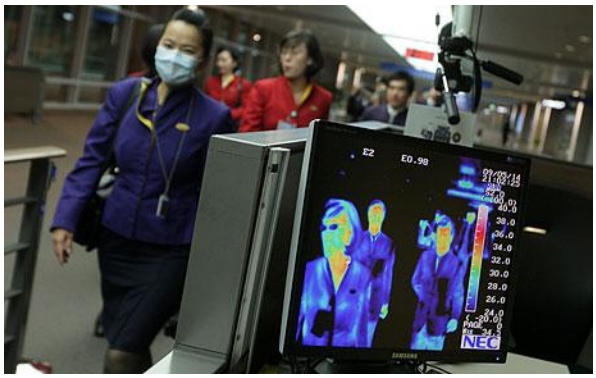


Abbildung 1.1.: Gesundheitskontrolle am Flughafen aus [The09]

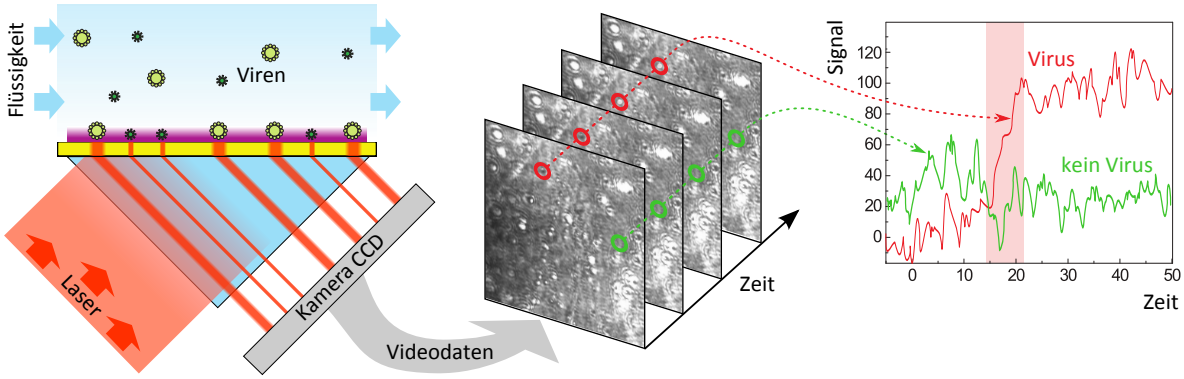


Abbildung 1.2.: Schematische Darstellung des Biosensors und der resultierenden Bildfolge mit den zu den markierten Stellen gehörigen Zeitreihen nach [WGZ+10]

Verfahren basiert auf der Erkennung von markierungsfreien[2] biomolekularen Bindungsreaktionen an einer Goldoberfläche, die mit einer CCD-Kamera (*Charge-Coupled Device*) als Bildfolge aufgenommenen werden. Zur Detektion von Bindungsereignissen wird der Effekt ausgenutzt, dass polarisiertes Licht (Laser), das über ein Prisma auf eine Metallschicht trifft, reflektiert wird und dieses zu einer Anregung der Oberflächenplasmonen innerhalb der Metallschicht führt. Charakteristisch für die Anhaftung eines Virus ist ein sprunghafter Anstieg der Lichtintensität. Den Aufbau des Verfahrens zeigt Abbildung 1.2.

Der Biosensor, der der Plasmonen-unterstützten Mikroskopie von Nanoobjekten zu Grunde liegt, wurde vom ISAS (Leibnitz-Institut für Analytische Wissenschaften) entwickelt. Die Beschreibung erfolgt auf Grundlage des Artikels „SPR detection of single nanoparticles and viruses“ [ZGWN09]. Die weiteren Merkmale des PAMONO-Verfahren sind dem Artikel „Plasmonen-unterstützte Mikroskopie zur Detektion von Viren“ [WGZ+10] entnommen.

Bei der Sensoroberfläche handelt es sich um eine auf Glas gedampfte dünne Goldschicht (ca. 50nm). Die dünne Goldschicht ist auf Seite der Durchflusszelle mit spezifischen Bindungspartnern funktionalisiert. Zu Anhaftungen kommt es durch Bindungsreaktionen zwischen diesen Bindungspartnern und den Partikeln in der Flüssigkeit. Bei Viren sind die zugehörigen Antikörper die Bindungspartner. In der Durchflusszelle, die ca. 20μm dick ist, herrschen dynamische Flussbedingungen. Das bedeutet, dass aufgrund der Wärmebewegung der Teilchen in der Flüssigkeit (brownsche Bewegung) die Wahrscheinlichkeit einer Anhaftung eines Partikels bei 50% liegt. Dafür müssen beispielsweise bei einem Sensorgebiet von 2 x 2 mm^2 0,1 μL der Probe 120 Sekunden lang durch die Durchflusszelle gepumpt werden. Ein Partikel mit einem Durchmesser von 100nm bewegt sich somit in ca. 100 Sekunden 20μm entlang einer bestimmten Achse. Die Goldschicht wird mittels eines Lasers mit Licht der Wellenlänge 670nm, das durch ein Prisma gebündelt wird, beleuchtet. Bei der Anhaftung von Viren steigt das Reflektionssignal an und die Intensitätsänderung wird durch Plasmonenanregungen verstärkt. Dabei wird die Lichtenergie in kollektive Oszillation von Elektronen auf der Goldschicht konvertiert. Der Einfallswinkel des von der Diode kommenden Lichts wird gemäß dem steilsten Teil der Resonanzkurve fest gewählt. Die Abbildung der Reflexion wird von einer digitalen Kamera aufgenommen. Auf einem Bild zeigt sich ein Virus durch eine fleckartige Anhaftung

[2] Die zu untersuchenden Nanopartikel werden nicht mit Markern versehen [WGZ+10].

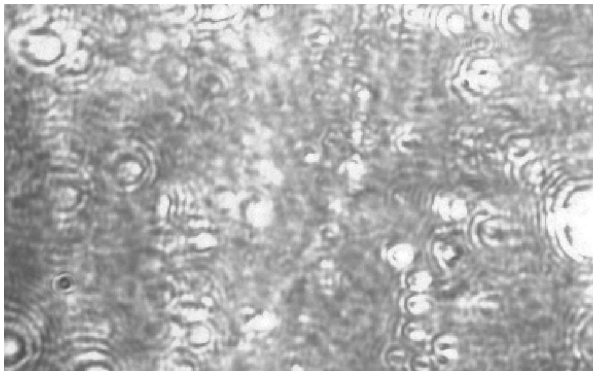


Abbildung 1.3.: Ausschnitt eines unverarbeiteten Bilds, das mit der CDD-Kamera aufgenommen wurde.

und ein kennzeichnendes Zeitverhalten. Abbildung 1.3 zeigt einen Ausschnitt eines solchen Bildes. Es sind helle und dunkle Areale zu erkennen. Das sind meist Störungen, sogenannte Artefakte, die zum Beispiel durch Luftbläschen entstehen. Erst durch Nachbearbeitung werden die Strukturen der Anhaftungen sichtbar. Bei der Verarbeitung der Bilder müssen verfahrensbedingte Schwierigkeiten beachtet werden. Zum einen erhitzt der Laser die Probe, so dass ein stetiger Intensitätsanstieg (Grauwertdrift) zu verzeichnen ist, andererseits erzeugt das gepulste Laserlicht ein Wellenprofil. Zusätzlich beeinflussen die Lichtverhältnisse der Umgebung die Aufnahme. Mängel der verwendeten Komponenten führen zu weiteren Rauschfaktoren. Genannt sei die Rauheit der Sensoroberfläche und der Goldschicht, sowie Inhomogenität. Das Prisma zum Beispiel ist nicht hoch rein.

Trotz dieser Nachteile hat das PAMONO-Verfahren Vorteile, die die Anwendung rechtfertigen. Hauptvorteil ist der erwähnte Nachweis eines einzelnen Virus. Schon ab einer Konzentration von 10^3 Viren pro ml in der Probe können Viren in weniger als fünf Minuten nachgewiesen werden. Die Goldschicht kann so funktionalisiert werden, dass bis zu 400 Virenarten gleichzeitig detektiert werden können, unabhängig von einem Labor.

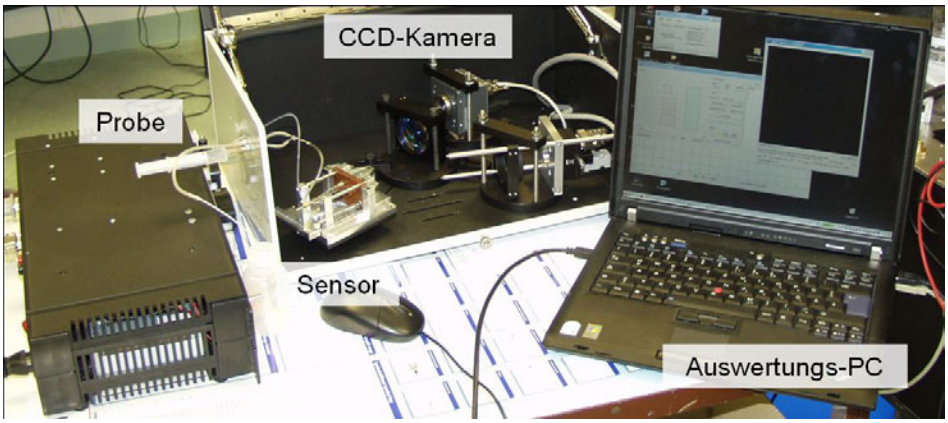


Abbildung 1.4.: Der reale Experimentaufbau

Abbildung 1.4 zeigt den realen Experimentaufbau am ISAS. In der Abbildung ist die Spritze zu sehen, mit der die Probe in den PAMONO-Sensor gegeben wird. Der PAMONO-Sensor befindet sich zusammen mit der CCD-Kamera und dem Laser in einem Koffer, um die Apparatur vor Fremdlicht zu schützen. Bisher geschieht die Analyse der Bilder manuell am Auswertungs-PC durch Mitarbeiter des ISAS, was langwierig ist und so den Vorteil der schnellen Detektion zunichte macht. Hier setzt die Aufgabe der Projektgruppe COOL-IP, eine Kooperation zwischen dem Lehrstuhl 7 (Grafische Systeme), dem Lehrstuhl 12 (Eingebettete Systeme) und dem ISAS, an. Die Entwicklungen in der PG sollen helfen, die Analyse der PAMONO-Daten zu automatisieren und die Analysedauer drastisch zu reduzieren. Durch eine Umsetzung auf eingebetteten Systemen wird zudem eine Mobilität des Systems ermöglicht, um es direkt vor Ort zur Probenauswertung nutzen zu können.

1.2. Aufgabe der Projektgruppe

Das im Sommersemester 2010 und Wintersemester 2010/11 zu erreichende Ziel der PG COOL-IP ist die Erstellung eines Workflow-orientierten Verarbeitungssystems für die automatisierte Echtzeitanalyse der Bilddaten des PAMONO-Sensors. Dazu gliedern sich die Aufgabengebiete der PG in verschiedene Teilgebiete auf. Ein Teilgebiet fokussiert sich dabei auf die digitale Bildverarbeitung und bildbasierte Mustererkennung, das zweite beinhaltet den Entwurf und die Implementierung eines eingebetteten Systems. Letzteres bezieht sich darauf, dass die Vorverarbeitung der aufgenommenen Bilder mithilfe einer Kamera mit FPGA (*Field-programmable Gate Array*) umgesetzt werden soll. Dieser Teilschritt soll unter Verwendung der Systementwicklungsumgebung Xilinx erfolgen. Seitens der Bildverarbeitung steht die Entwicklung verschiedener effizienter Bildverarbeitungsalgorithmen zur Bildanalyse und Mustererkennung an. Für die Lokalisierung der Viren können verschiedene Verfahren, wie zum Beispiel Template-Matching und Zeitreihenanalysen, eingesetzt werden. Nach der Lokalisierung müssen die Viren Klassifiziert werden. Dies soll auf Basis der im vorhergehenden Schritt extrahierten Merkmalen geschehen. Als Herausforderung sind die große Datenmenge (Datenaufkommen von ca. 50 MB/s) mit kleinen gesuchten Strukturen im Verhältnis zur Bildgröße und die prozessbedingten Artefakte zu nennen. Daher ist zusätzlich die Performance-Optimierung zur Verarbeitung und Analyse großer Datenmengen auf CPU respektive Grafikprozessoren als Aufgabe zu nennen. Grundsätzlich soll eine Software, die mittels einer intuitiven Benutzeroberfläche eine übersichtliche Darstellung und Analyse der Ergebnisse erlaubt, entwickelt werden. Bei der Umsetzung dieser Anforderungen in ein Hardware-/Softwaresystem sollen verschiedene Entwurfsphasen beachtet werden und adäquate Werkzeuge (Quellcodeverwaltung, Wiki System, Bugtracking) eingesetzt werden.

1.3. Themenverwandte Arbeiten

Auf themenverwandte Arbeiten wird in den entsprechenden Kapiteln verwiesen. Hier seien explizit einige Arbeiten zu Bildverarbeitung auf eingebetteten Systemen genannt: Martin Brückners Studienarbeit „Embedded Linux auf FPGA-Hardware zur Bildverarbeitung“, im Rahmen des

Diplomstudiengangs Informatik am Institut für Informatik der Humboldt-Universität zu Berlin eingereicht, beinhaltet unter anderem einen Ressourcenvergleich verschiedener FPGAs [Brü07]. Peter H. Dillingers Forschungsbericht „Einsatz von FPGA Prozessoren als Datenreduktionshardware in der Bildverarbeitung“ hat als Ziel die Tauglichkeit der FPGAs auf Einsatz in der Bilddatenreduktion zu untersuchen [Dil]. In seiner Doktorarbeit „Wavelet based image compression using FPGAs“, vorgelegt der Mathematisch-Naturwissenschaftlich-Technischen Fakultät der Martin-Luther-Universität Halle-Wittenberg, hat Jörg Ritter die damaligen besten verfügbaren, meist auf Wavlet-Transformation basierenden, Bildkompressionsverfahren analysiert [Rit02]. In ihrer Veröffentlichung „FPGA-Based Discrete Wavelet Transforms System“ präsentieren M. Nibouche, A. Bouridane, F.Murtagh und O. Nibouche ein Arbeitsgerüst für ein FPGA-basiertetes System für eine diskrete Wavlet-Transformation [NBMN].

1.4. Aufbau des Endberichts

Die weitere Struktur des Endberichts sieht wie folgt aus: Kapitel 2 behandelt die Schritte der Bildverarbeitung eines Bildes in der Analysesoftware. Darauf folgen die Aspekte der Zeitreihenanalyse (Kapitel 3) und Segmentierung (Kapitel 4). Die aus diesen Kapiteln resultierenden Erkenntnisse werden genutzt, um die zentrale Aufgabe der automatisierten Detektion in Form einer Klassifikation anzugehen (Kapitel 5). Der Fokus des Kapitels „Stream-Verarbeitung“ (Kapitel 6) liegt auf Verfahren, mit denen die vorhergehenden Berechnungen effizient ausgeführt und parallelisiert werden können. In Kapitel 7 wird die verwendete Kamera und die damit zusammenhängenden Möglichkeiten der eingebetteten Bildverarbeitung vorgestellt. Kapitel 8 thematisiert die objektorientierte Realisierung der vorgestellten Techniken und Verfahren. Anschließend werden die Möglichkeiten der schnellen Verarbeitung der großen Datenmengen durch Nutzung der Rechenleistung von GPUs in Kapitel 9 erörtert. Es folgt die Evaluation des Pamono-Explorers (Kapitel 10). Abschließend werden die wichtigen Punkte der Arbeit der Projektgruppe mit Fazit und Ausblick zusammengefasst (Kapitel 11). Im Anhang des Endberichts befindet sich neben den Evaluationsdaten (Anhang A) das Handbuch (Anhang B) zum Pamono-Explorer und das am Anfang der Projektgruppe erstellte Pflichtenheft (Anhang C).

2. Bildverarbeitung

Die PAMONO-Technik ist ein bildbasiertes, teilautomatisches Verfahren zur Detektion von Partikeln mit Hilfe einer digitalen Kamera. Die wichtigen Informationen der Bilddaten sind ursprünglich durch Störungen und andere Einflüsse überlagert. Die Bildverarbeitung soll die Bilddaten so verbessern, dass die relevanten Informationen hervorgehoben werden, während die störenden Einflüsse reduziert werden. Das ermöglicht die Analyse der Daten durch die Software oder einen Benutzer.

Die Struktur dieses Kapitels ist dem „Lebenszyklus“ eines Bildes in der Analysesoftware im Hinblick auf die bildverarbeitenden Verfahren nachempfunden. Am Anfang steht die Eingabe der Bilddaten in unterschiedlichen Datei- und Bildformaten, die in ein für die Analyse vorteilhaftes Format übertragen werden (Kapitel 2.1). Verfahren zur Rauschminderung (Kapitel 2.2) und Bildnormalisierung (Kapitel 2.3) verbessern die Bilder, sodass die anschließende Analyse vereinfacht wird. Nach der Analyse müssen die Bilder zur Verifikation der Ergebnisse für den Benutzer aufbereitet werden.

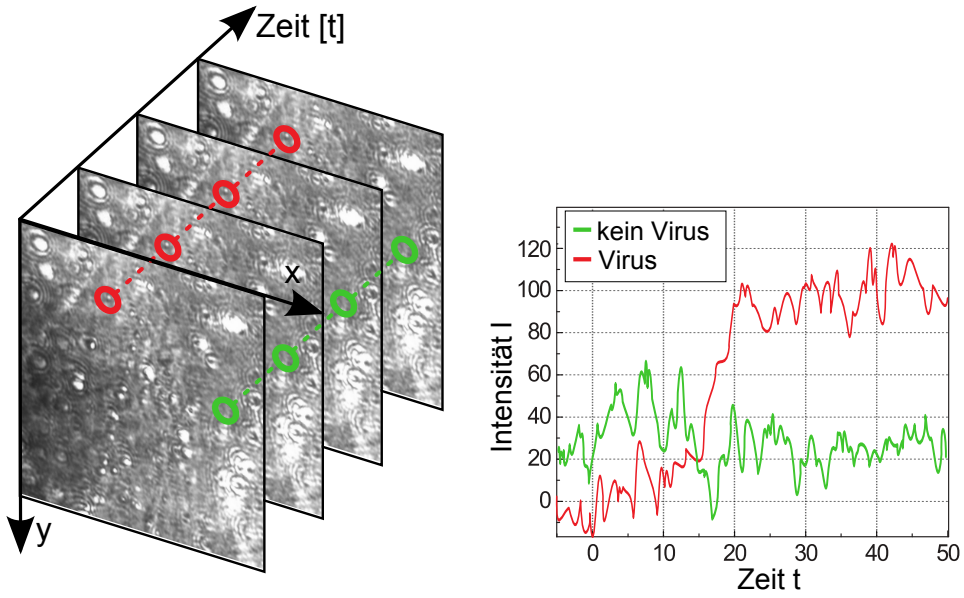
2.1. Bilddaten

Die Bilddaten der PAMONO-Analyse werden mit der in Kapitel 7 beschriebenen Digitalkamera aufgenommen. Die Kamera besitzt einen CMOS-Bildsensor[1], der aus rasterförmig angeordneten, lichtempfindlichen Sensorelementen besteht.

Die Sensorelemente (*Pixel*, von *picture element*) haben meist eine quadratische Form, mit einer Seitenlänge im Bereich von $2-20\mu m$. Ein Pixel misst die Lichtintensität des einfallenden Lichts, indem eine elektrische Spannung erzeugt wird, die proportional zur Menge der eintreffenden Lichtteilchen ist. Die Spannung wird anschließend verstärkt und digitalisiert.

Die **Bit-Tiefe** bestimmt die Anzahl der Quantisierungsstufen bei der Digitalisierung. Für eine Bit-Tiefe B kann die Lichtintensität diskrete Werte im Bereich $[0, 2^B - 1]$ annehmen. Die in der PAMONO-Analyse benutzten Bilddaten haben eine Bit-Tiefe von 12- bis 16-Bit.

[1] Die in der Projektgruppe benutzte Kamera besitzt einen CMOS-Sensor, während die zur Verfügung gestellten Bilddaten mit einer Kamera mit CCD-Sensor aufgenommen wurde. Die beschriebenen Eigenschaften sind dennoch für beide Sensorarten gleichermaßen gültig.



(a) (b)

Abbildung 2.1.: Ein Ausschnitt aus einer Bildreihe (a) und die zu den markierten Stellen zugehörigen Zeitreihen (b).

2.1.1. Definitionen

Ein **Bild** ist ein $M \times N$ großes Pixelraster $f(x, y)$, mit dem Reihenindex $y = 1, 2, \ldots, M$ und dem Spaltenenindex $x = 1, 2, \ldots, N$. M wird als Höhe, N als Breite bezeichnet. Es kann eine Metrik angegeben werden, die ein Pixel mit einer definierten Länge auf dem PAMONO-Bildsensor in Verbindung setzt. Mit einem passenden Vergrößerungsfaktor entspricht ein Pixel im Bild beispielsweise $1\mu m$ auf der PAMONO-Sensoroberfläche [ZGWN09].

Eine geordnete Folge von K Bildern gleicher Größe wird als **Bildreihe** bezeichnet. Jedem Bild wird ein Index $t = 1, 2, \ldots, K$ zugewiesen, welcher der zeitlichen Abfolge der Bilder entspricht. Die Kamera nimmt Zeitreihen mit einer Bildwiederholrate von 30 bis 50 Bildern pro Sekunde auf.

Eine **Zeitreihe** $f_{(x,y)}(t)$ ist eine Folge der Intensitätswerte der Bilder einer Bildreihe für eine konstante Pixelposition (x, y) und einen variablen Bildreihenindex t. Abbildung 2.1 zeigt beispielhaft eine Bild- und Zeitreihe.

2.1.2. Rauschen

Während der Bildaufnahme und Übertragung entstehen technisch bedingte Störungen im Bildsignal, die unter dem Oberbegriff **Rauschen** zusammengefasst werden. Drei wichtige Rauschformen sind [PLH10]:

- **Dunkelrauschen:** Dunkelrauschen tritt auf, ohne dass Licht auf den Sensor trifft. Es besteht aus zwei Anteilen, dem Ausleserauschen, das von der Kameraelektronik abhängt, und dem thermischen Rauschen, das abhängig ist von der Temperatur der Umgebung und des Bildsensors. Das thermische Rauschen entsteht dadurch, dass Pixel nicht nur durch Licht, sondern auch durch thermische Einflüsse angeregt werden. Dunkelrauschen tritt besonders bei langen Belichtungszeiten auf.

- **Schrotrauschen:** Das Verhalten von Photonen ist nach der Quantentheorie nichtdeterministisch, weswegen das Auftreffen von Photonen auf ein Pixel ein Zufallsprozess ist. Die gemessene Photonenzahl ist gemäß der Poisson-Statistik verteilt. Dadurch entstehen zufällige Intensitätsschwankungen, die als Schrotrauschen bezeichnet werden und bei kurzen Belichtungszeiten besonders ausgeprägt sind.

- **Quantisierungsrauschen:** Bei der Diskretisierung von kontinuierlichen Helligkeitswerten entstehen Rundungsfehler, die als Quantisierungsrauschen bezeichnet werden.

Für eine genaue Beschreibung der technischen Prozesse, die zu Rauschen führen, sei auf [Jan01] verwiesen.

Ein verrauschtes Bild $f(x, y)$ kann als Überlagerung des tatsächlichen Signals $g(x, y)$ mit dem Rauschsignal $q(x, y)$ beschrieben werden [Bov05]:

$$f(x, y) = g(x, y) + q(x, y).$$

In den PAMONO-Bilddaten ist das poissonverteilte Schrotrauschen am stärksten ausgeprägt [ZGWN09]. Für eine hohe erwartete Photonenzahl an einem Pixel nähert sich die Poisson-Verteilung der Gauß-Verteilung, wodurch das Rauschen als Gaußsches Rauschen modelliert werden kann. Dieses Rauschmodell wird im folgenden Abschnitt beschrieben.

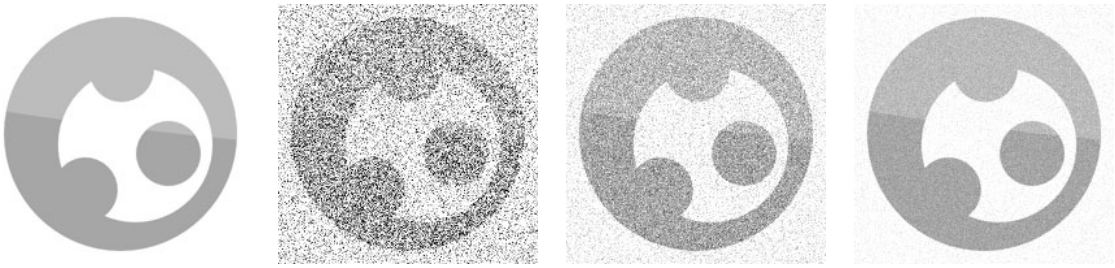
2.1.2.1. Gaußsches Rauschen

Die Gaußsche Glockenkurve ist sowohl im Zeit- als auch im Frequenzbereich gut lokalisiert, weswegen Rauschmodelle auf Basis der Normalverteilung oft eingesetzt werden. Nach [Bov05] ist die Dichtefunktion für eine Zufallsvariable z

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(z-\bar{z})^2}{2\sigma^2}}. \tag{2.1}$$

z entspricht in diesem Fall der Intensität, $\bar{z}$ ist der Mittelwert, und σ die Standardabweichung von z.

Das sogenannte *Weiße Rauschen* zeichnet sich durch ein konstantes Frequenzspektrum aus und kann meist gut durch Gaußsches Rauschen modelliert werden. Viele Rauschprozesse, beispielsweise auch das thermische Rauschen, lassen sich durch *additives weißes Gaußsches Rauschen* modellieren. Es ist ein weißes Rauschen bei dem die Signalwerte gaußverteilt sind, und bei dem das resultierende Signal als Addition des Nutzsignals mit dem Rauschsignal entsteht.



(a) Rauschfrei (b) SNR=10 dB (c) SNR=20 dB (d) SNR=30 dB

Abbildung 2.2.: Ein Originalbild (a) überlagert mit additivem weißen Gaußschen Rauschen verschiedener *signal-to-noise ratios* (b – d).

Für weitere wichtige Dichtefunktionen sei auf [GW08] verwiesen.

2.1.2.2. Signal-Rausch-Verhältnis

Das Signal-Rausch-Verhältnis (*Signal-to-noise ratio*, SNR) ist ein Maß für die Qualität eines Bildsignals und somit für die Stärke des Rauschens. Ist $\hat{f}$ das rauschfreie, und f das verrauschte Bild, so lässt sich nach [GW08] das SNR für Bilder der Größe $M \times N$ ausdrücken als

$$\mathrm{SNR} = \frac{\sum_{x=1}^{N}\sum_{y=1}^{M} \hat{f}(x,y)^2}{\sum_{x=1}^{N}\sum_{y=1}^{M} \left[f(x,y) - \hat{f}(x,y)\right]^2}. \tag{2.2}$$

Bei starkem Rauschen ist das SNR klein, ein hohes SNR lässt auf ein Rauschsignal schließen, das gegenüber dem Nutzsignal schwach ist. Da bei wenig Rauschen sehr große Werte für das SNR entstehen können, wird das SNR oft im logarithmischen Maßstab mit der Einheit Dezibel (dB) angegeben:

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \log(\mathrm{SNR})\ \mathrm{dB}. \tag{2.3}$$

Als „gute“ Signale können solche mit einem SNR ab 25 – 30 dB bezeichnet werden. Für *Wireless LAN* beispielsweise ist laut [Gei05] ab 25 dB sehr guter Empfang möglich, die Untergrenze liegt bei 10 dB. In Abbildung 2.2 ist der Einfluss der Rauschens für verschiedene SNR-Werte exemplarisch dargestellt.

2.2. Denoising

Die Bildreihen, die beim PAMONO-Verfahren aufgenommen werden, weisen sowohl innerhalb einzelner Bilder als auch in den Zeitreihen einzelner Pixel ein hohes Rauschniveau auf, mit einem $\mathrm{SNR}_{\mathrm{dB}}$ Wert im Bereich von 10 – 15 dB. Zur rechnergestützten Erkennung sehr kleiner Strukturen, wie sie in der PAMONO-Analyse auftreten (etwa 5×5 Pixel bei einer Gesamt-

bildgröße im Megapixel-Bereich), sind daher Verfahren zur Rauschminderung (*denoising*) notwendig. Eine Auswahl solcher Verfahren soll im Folgenden näher betrachtet werden.

2.2.1. Bildmittelung

Da das Bildrauschen zufällig gleichverteilt in den Bilddaten auftritt, bietet sich eine Rauschminderung durch Mittelung der Bilder an. Dazu werden n aufeinander folgende Bilder $f_1, \ldots, f_n$ der gleichen Größe addiert und die Summe durch die Anzahl der Bilder dividiert. Für ein Bild f, wie es in Kapitel 2.1.1 definiert wurde, liefert Gleichung 2.4 für jeden Bildpunkt $(x, y) \in \mathrm{dom}(f)$ die mittlere Intensität

$$f_{\mathrm{avg}}(x, y) = \frac{1}{n} \sum_{i=1}^{n} f_i(x, y). \tag{2.4}$$

Beim Arbeiten mit den Bildreihen zeigt sich zudem, dass sich viele der zeitlich aufeinanderfolgenden Bilder bis auf das Rauschen nicht voneinander unterscheiden und somit keine neue Information enthalten. Mit Hilfe der Bildmittelung von jeweils n Bildern lässt sich die Anzahl der Bilder um den Faktor n reduzieren, wenn die Mengen der zu mittelnden Bilder disjunkt sind.

2.2.2. Denoising mit Wavelets

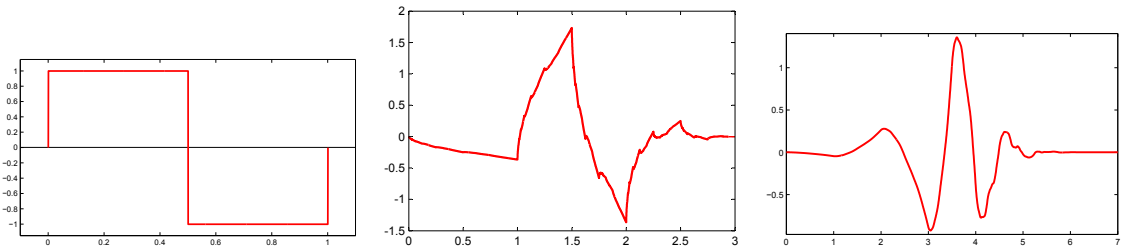
Die *Wavelet-Transformation* (WT) ist ein Verfahren, mit dem die Frequenzen eines Signals auf eine umfassende Art analysiert werden können. Die „Wavelets“, nach außen abklingende Funktionen, deren Integral 0 ist [Nie03], sind Basisfunktionen, mit denen beliebige andere Funktionen dargestellt werden können. Für ein fest definiertes „Mutter-Wavelet“ $\psi(x)$ kann durch Skalierungen a und Verschiebungen b eine Familie von „Kinder-Wavelets“

$$\psi_{a,b} = \frac{1}{\sqrt{a}}\, \psi\left(\frac{x-b}{a}\right), \text{ mit } (a, b) \in \mathbb{R}^+ \times \mathbb{R}$$

erzeugt werden, die als Basis für die Repräsentation des Signals dienen. Abbildung 2.3 zeigt verschiedene Mutter-Wavelets.

Die Koeffizienten der WT entsprechen den Skalierungen und Verschiebungen des Wavelets und werden als Faltung des Signals mit den Kinder-Wavelets berechnet. Durch den Wellencharakter der Wavelets sind die so berechneten Skalierungskoeffizienten eine Repräsentation der in dem Signal vorkommenden Frequenzen. Im Gegensatz zu den Sinus- und Kosinusfunktionen der Fourier-Transformation, die im Frequenzbereich exakt, im Zeitbereich jedoch überhaupt nicht lokalisierbar sind, sind Wavelets durch die Skalierung genau im Frequenz-/Skalenbereich und durch die Verschiebung genau im Zeitbereich lokalisierbar.

Die Wavelet-Koeffizienten bestimmen das Aussehen des Signals. Große Koeffizienzen entsprechen großen Änderungen im Signal und geben so den ungefähren Signalverlauf wieder. Kleine Koeffizienten entsprechen kleinen Änderungen, den Details des Signals, die keinen, oder nur



(a) Haar-Wavelet (b) Daubechies db2-Wavelet (c) Daubechies db4-Wavelet

Abbildung 2.3.: Verschiedene Beispiele für Wavelets

geringen Einfluss auf das gesamte Signal haben. Durch „Abschneiden“ (*Thresholding*) von kleinen Wavelet-Koeffizienten wird das Signal von solchen Details befreit. Da oftmals Rauschen für die hochfrequenten Details im Signal verantwortlich ist, kann die WT mit Thresholding zur Rauschminderung von Daten eingesetzt werden. Dies wird in Abschnitt 2.2.2.1 näher beschrieben.

In der Praxis wird die *Diskrete Wavelet-Transformation* (DWT) benutzt, da die *kontinuierliche* WT aus mehreren Gründen schwierig einzusetzen ist, darunter redundante Wavelet-Koeffizienten und fehlende analytische Lösungen bei der Berechnung [Val10]. Die DWT berechnet die Wavelet-Koeffizienten für konkrete Positionen und Skalierungen, die gemäß dyadischen Samplings ausgewählt werden (also eine Potenz von 2 sind). Trotz der diskreten Werte lässt sich das ursprüngliche Signal durch die inverse DWT vollständig wiederherstellen.

Aufgrund des Umfangs der formalen Grundlagen der (D)WT und DWT sei diesbezüglich auf die Literatur verwiesen: [VM94] und [Nie03].

2.2.2.1. Thresholding

Das Denoising von Signalen mit Wavelets kann in drei Schritte unterteilt werden. Als erstes wird eine DWT bis zu einer bestimmten Auflösungsstufe (*level*) berechnet und in einem zweiten Schritt ein Thresholding der berechneten Wavelet-Koeffizienten durchgeführt. Als drittes wird das Signal mit den modifizierten Koeffizienten reproduziert. Thresholding bezeichnet die Elimination oder Reduktion der Wavelet-Koeffizienten.

Thresholding kann *level-abhängig* sein, mit verschiedenen Grenzwerten je Auflösungsstufe, oder *global*, mit einem einzigen Grenzwert für alle Auflösungsstufen. Zusätzlich kann zwischen *hard* und *soft thresholding* unterschieden werden [HP06]. Bei *hard thresholding* werden alle Waveletkoeffizienten v, die kleiner als, oder gleich dem Grenzwert δ sind, auf 0 gesetzt:

$$v_{hard} = \begin{cases} v \text{ für } |v| > \delta, \\ 0 \text{ für } |v| \leq \delta \end{cases} .$$

Soft thresholding erweitert das *hard thresholding* um eine Subtraktion um δ von den Koeffizi-

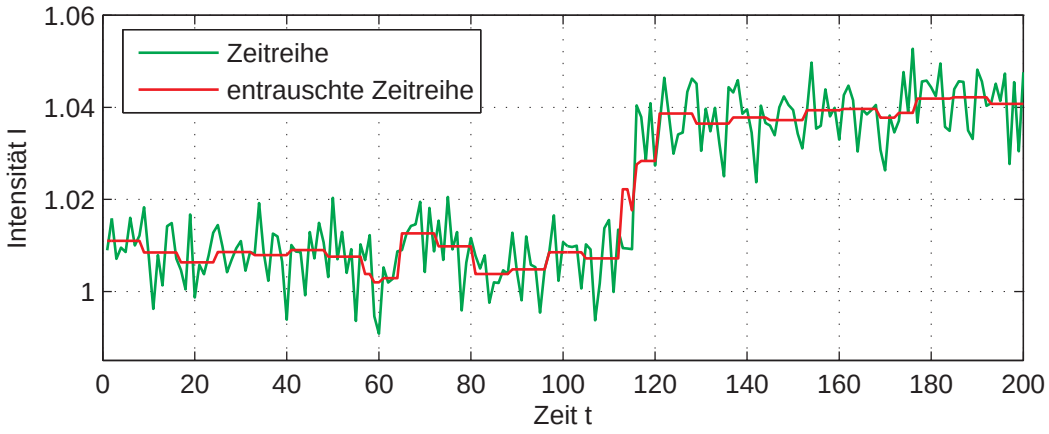


Abbildung 2.4.: Originale Zeitreihe mit Rauschen (grün). Die rote Kurve zeigt die mit einem Haar-Wavelet bis Auflösungsstufe 3 zerlegte, und per globalem Soft-Thresholding entrauschte Zeitreihe.

enten, die größer als δ sind:

$$v_{soft} = \begin{cases} \text{sgn}(v) \cdot (|v| - \delta) \text{ für } |v| > \delta, \\ 0 \text{ für } |v| \leq \delta \end{cases} .$$

Im Kontext der Projektgruppe gibt es zwei Möglichkeiten, Wavelet-Denoising einzusetzen, das Denoising von Zeitreihen und das Denoising von Einzelbildern.

Denoising von Zeitreihen

Damit die Zeitreihenanalyse (siehe Kapitel 3) bessere Ergebnisse erreicht, ist es sinnvoll, die Zeitreihe vorher zu entrauschen. Die Zeitreihe ist ein eindimensionales Signal, daher wird eine 1D-DWT berechnet. Die Zeitreihe wird bis zu einer bestimmten Auflösungsstufe zerlegt, die berechneten Koeffizienten mit Methoden des Thresholdings verändert und abschließend das Signal mit den modifizierten Koeffizienten rekonstruiert. Abbildung 2.4 zeigt ein Beispiel für eine per DWT entrauschte Zeitreihe.

Denoising von Bildern

Die Segmentierung (Kapitel 4) und Klassifikation (Kapitel 5) arbeiten nicht nur auf den Zeitreihen, sondern auch auf den Bildern. Daher kann es von Vorteil sein, das Rauschen in den Bildern zu reduzieren. Dazu wird eine 2D-DWT durchgeführt, indem eine 1D-DWT zuerst auf alle Reihen des Bildes angewendet wird und so jeweils ein Mittelwert- und ein Differenzanteil ermittelt wird. Anschließend wird eine 1D-DWT auf den Spalten des resultierenden Koeffizientenbildes berechnet [VM94]. Wie in Abbildung 2.5 zu erkennen ist, wird das Bild dadurch in einen Tiefpass- bzw. Mittelwertanteil und drei Hochpass- bzw. Differenzanteile zerlegt. Dieser

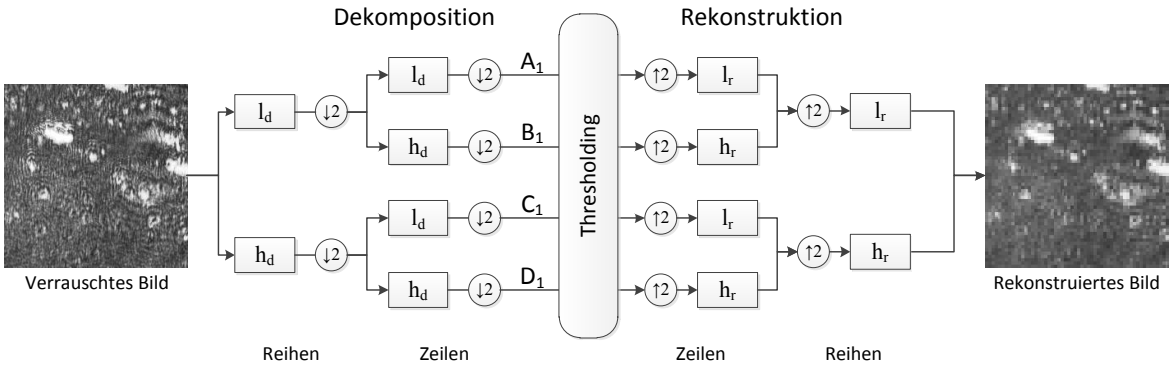


Abbildung 2.5.: Denoising eines Bildes mit Wavelets. Das Bild wird in der ersten Auflösungsstufe durch die Hochpassfilter h_d und die Tiefpassfilter l_d in den Mittelwertanteil A_1 und die Differenzanteile B_1, C_1 und D_1 zerlegt. Das rekonstruierte Bild ist nach Thresholding der Differenzanteile rauschärmer als das Originalbild. Modifiziert nach [HP06].

Vorgang kann für das Tiefpasskoeffizientenbild erneut durchgeführt werden.

2.3. Bildnormalisierung

Die Bildnormalisierung ist eine Transformation der Grauwertintensitäten, so dass die statistische Verteilung der Grauwerte des Bildes zwischen den Werten 0 und 1 liegen. Eine Normalisierung ist sinnvoll, da durch verlustbehaftetes Komprimieren des Bildes und bedingt durch unterschiedliche Einstellungsmöglichkeiten der Kamera, wie sie in Kapitel 7 beschrieben worden sind, Artefakte (siehe Kapitel 7.2.1) und Intensitätsänderungen der Grauwerte entstehen. Die Bilder einer Bildreihe sollen mittels der Normalisierung einfacher miteinander verglichen werden können. Damit werden insbesondere verschiedene Bittiefen und relative Veränderungen der Intensität berücksichtigt. Anwendungen der Bildnormalisierung werden im Pipeline-Abschnitt *Hintergrunddivision* und dem kommenden Abschnitt *Template-Matching-Klassifikatoren* (siehe dazu Kapitel 2.3.2, respektive 5.7) vorgestellt.

2.3.1. Hintergrundsubtraktion

Bei der Hintergrundsubtraktion wird das aktuelle Bild von einem festgelegtem Hintergrundbild abgezogen. Ähnlich der Hintergrunddivision (Abschnitt 2.3.2) wird für das Hintergrundbild das erste Bild gewählt oder nach n Bildern der Hintergrund neu definiert. Sei das aktuelle Bild f und das Hintergrundbild h, so ergibt sich die Subtraktion für jeden Bildpunkt (x, y):

$$f_{\text{sub}}(x, y) = f(x, y) - h(x, y). \tag{2.5}$$

Da einzelne Pixelwerte (x, y) bei der Subtraktion negative Werte annehmen können, wird die im nächsten Abschnitt erläuterte Hintergrunddivision der Hintergrundsubtraktion vorgezogen. Der Wertebereich der Intensitäten ist auf positive Werte festgelegt, dadurch müssen die negativen Werte auf 0 aufgerundet oder positiviert werden. Somit lässt sich bei der Zeitreihen-Analyse (Kapitel 3, Abschnitt 3.1) nicht herausstellen ob es eine positive oder negative Intensitätsänderung eines Pixels (x, y) bei aufeinanderfolgenden Bildern gegeben hat. Durch die Hintergrunddivision ist es ebenfalls einfacher die Pixelwerte in ein prozentuales Maß zu überführen. Dies ist wichtig für die Zeitreihen-Analyse bzw. den Intensitätssprung, um eine Aussage über eine prozentuale Intensitätsänderung machen zu können (siehe Kapitel 3).

2.3.2. Hintergrunddivision

Um das signifikante Rauschen (siehe Kapitel 2.2), welches bei der Aufnahme auftritt, auszugleichen bzw. zu verringern, wird eine Hintergrunddivision durchgeführt. Dazu wird nach jeweils n Bildern ein Bild als Hintergrund definiert und alle nachfolgenden Bilder werden durch den so definierten Hintergrund dividiert. Alternativ bietet es sich an, nur das erste Bild als Hintergrund festzulegen, da während der Analyse bereits angehaftete Partikel bei erneutem Setzen des Hintergrundes verschwinden

$$f_{\mathrm{div}}(x, y) = \begin{cases} \frac{f(x,y)}{h(x,y)} & \text{wenn } h(x, y) \neq 0 \\ 1 & \text{sonst} \end{cases}. \tag{2.6}$$

Das Bild wird zusätzlich normiert, indem zuerst der Mittelwert der Intensitätswerte des aktuellen Bildes berechnet wird:

$$\bar{I_f} = \frac{1}{MN} \sum_{x=1}^{N} \sum_{y=1}^{M} f(x, y). \tag{2.7}$$

Sei weiterhin die mittlere Intensität des Hintergrundbildes gespeichert als

$$\bar{I_h} = \frac{1}{MN} \sum_{x=1}^{N} \sum_{y=1}^{M} h(x, y). \tag{2.8}$$

Ein Normierungsfaktor

$$\eta = \frac{\bar{I_h}}{\bar{I_f}} \tag{2.9}$$

wird berechnet und anschließend die Werte des Bildes f_{div} mit dem Normierungsfaktor multipliziert:

$$f_{\mathrm{norm}}(x, y) = \eta \cdot f(x, y)_{\mathrm{div}}. \tag{2.10}$$

Dieser Faktor berücksichtigt die ansteigende Helligkeit der Bilddaten im Laufe des Prozesses, indem durch die Multiplikation eine gleichwertige Helligkeitsstufe (siehe Kapitel 8.1.6) der Bilder erreicht wird. Die mittlere Gesamtintensität der normierten Bilder, die in den weiteren Analyseprozess weitergegeben werden, bleibt konstant.

3. Zeitreihenanalyse

In diesem Kapitel wird der Aspekt der Zeitreihenanalyse in der automatisierten PAMONO-Analyse methodisch diskutiert. Zunächst finden sich in Abschnitt 3.1 Erläuterungen zur Bedeutung der Zeitreihe und deren Charakteristik in Hinblick auf die Erkennung von Anhaftungen. Anschließend werden zwei Verfahren zur Auswertung des Zeitverhaltens vorgestellt (Abschnitt 3.2 und 3.4), welche im darauf folgenden Abschnitt 3.5 referenziert und für weitere Betrachtungen verwendet werden. Die aus der Funktionsanpassung ergebende Methodik des Differenzbildes auf Basis von geglätteten Zeitreihen befindet sich in Abschnitt 3.3.

3.1. Bedeutung des Zeitverhaltens

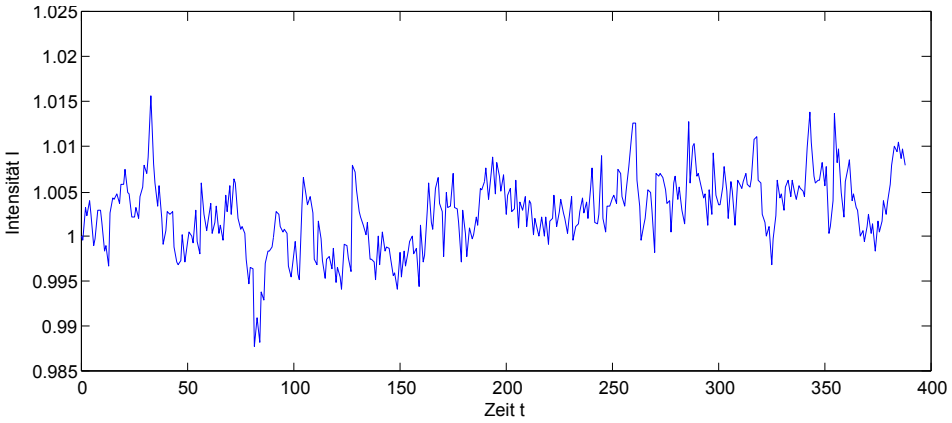
Aus dem Kenntnisstand über das PAMONO-Verfahren (siehe Kapitel 1) kristallisiert sich der Aspekt des zeitlichen Verhaltens von Bildregionen, welche Anhaftungen zeigen, als zentrale Charakteristik heraus. So ist experimentell nachzuweisen, dass im Moment der Anhaftung in einer Region ein sprunghafter Anstieg des Intensitätssignals festzustellen ist. In Abbildung 3.1 sind beispielhaft verschiedene Zeitreihen von drei verschiedenen Punkten eines Datensatzes zu sehen. Dabei sind in den Graphen (b) und (c) die zeitlichen Verläufe von Punkten mit Anhaftungen zu sehen. Charakteristisch ist der signifikante Anstieg der mittleren Intensität im Bereich nach dem Sprungzeitpunkt im Vergleich zur mittleren Intensität vor dem Sprung. Um dieses Verhalten in eine mathematische Form zu überführen, gilt es, ein gut passendes Modell für den zeitlichen Verlauf mit Hilfe einer Funktion zu bestimmen. Sei dazu die folgende **idealisierte Zeitreihenfunktion** definiert: Der Verlauf sei durch zwei Zeitpunkte t_a und t_b bestimmt, wobei $t_a \leq t_b$ sei. Dabei seien t_a der Startzeitpunkt und t_b das Ende eines Intensitätssprungs. Vor Sprungbeginn sei die Intensität konstant gleich I_a, nach dem Sprung nehme sie den konstanten Wert I_b an. Während des Sprungs werden die Werte linear interpoliert. Dazu sei die passende Funktion wie folgt definiert:

$$f_{\text{ideal}}(t) := \begin{cases} I_a & \text{für } 1 \leq t < t_a \\ m \cdot t + n & \text{für } t_a \leq t \leq t_b \\ I_b & \text{für } t_b < t \leq K \end{cases}, \tag{3.1}$$

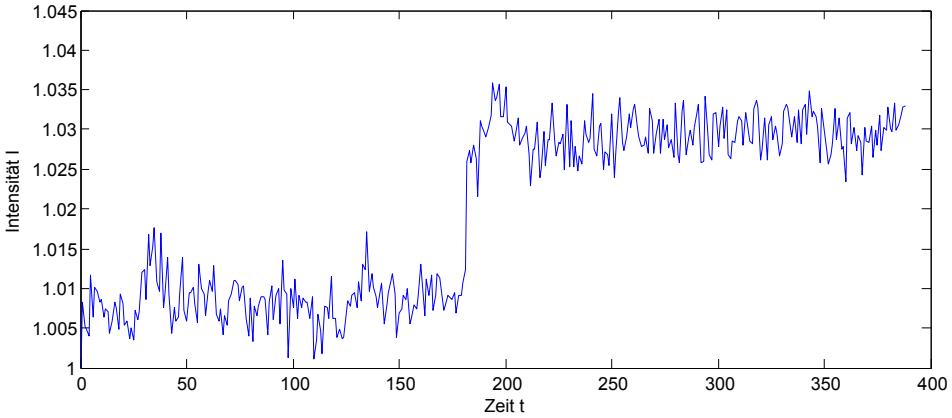
wobei die Koeffizienten der linearen Funktion wie folgt zu wählen sind:

$$m := \frac{I_b - I_a}{t_b - t_a} \quad \text{und} \quad n := I_a - m \cdot t_a. \tag{3.2}$$

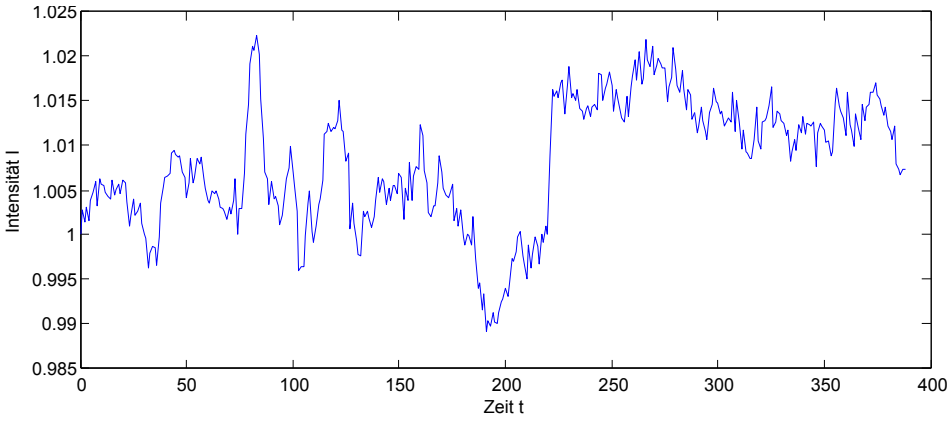
Der Verlauf der Funktion ist in Abbildung 3.2 dargestellt. Die Differenz zwischen t_a und t_b



(a) Zeitreihe eines Punktes ohne Anhaftung



(b) Zeitreihe eines Punktes mit Anhaftung bei $t = 180$



(c) Zeitreihe eines Punktes mit Anhaftung bei $t = 220$ und stärkeren Störungen

Abbildung 3.1.: Zeitreihen aus einem Datensatz mit Anhaftungen von 200 nm großen Partikeln. Der Graph (a) zeigt den Verlauf der (durch Hintergrunddivision) normierten Intensität an einem Punkt, an dem im Zeitfenster der Analyse kein Partikel angehaftet ist. In den Graphen (b) und (c) sind Anhaftungen durch Intensitätssprünge sichtbar.

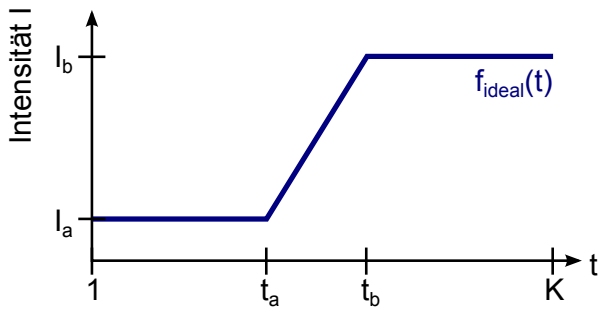


Abbildung 3.2.: Idealisierte Zeitreihenfunktion einer Anhaftung

bestimmt die Flankensteilheit des Sprungs. Wenn $t_a = t_b$ gewählt wird, entsteht ein direkter Sprung ohne linearen Übergang. Die Möglichkeit der Modellierung der Sprungübergangszeit ist sinnvoll, da sich der Prozess der Anhaftung bei geeignet hoher zeitlicher Abtastfrequenz des Intensitätssignals als kontinuierlicher Anstieg manifestiert. Der **Intensitätssprung** ΔI in einer idealen Zeitreihenfunktion sei wie folgt definiert:

$$\Delta I := I_b - I_a. \tag{3.3}$$

Unter der Voraussetzung, dass die Intensitätswerte vor der Untersuchung durch Hintergrunddivision normiert sind (siehe Kapitel 2.3), beschreibt diese Kennzahl die prozentuale Änderung zwischen den Intensitäten vor und nach dem Sprung. Bei der Untersuchung der Intensitätssprünge in verschiedenen Datensätzen hat sich eine Größenordnung der Werte von 2 bis 7% ergeben. Dieser Wert wird im weiteren Verlauf des Verfahrens als Merkmal zur Abgrenzung der Anhaftungen von Rauschen (Sprung kleiner als Schwellwert) und Störungen (Sprung größer als Grenzwert) herangezogen. In den folgenden Abschnitten werden Methoden beschrieben, die Zeitreihensignale auf diesen charakteristischen Sprung hin untersuchen.

3.2. Verfahren auf Basis von Funktionsanpassung

Um den im vorherigen Abschnitt definierten Intensitätssprung ΔI aus einer gegebenen Zeitreihenfunktion $f_{(x,y)}(t)$ zu berechnen, ist eine Anpassung der idealisierten Funktion an die vorliegende Zeitreihe notwendig. Hierbei ist der Einfluss des in Kapitel 2.2 beschriebenen Signalrauschens zu beachten. Im Folgenden soll ein gegenüber Rauschen robustes Verfahren zur Parameterschätzung vorgestellt werden.
Die in Abschnitt 3.1 beschriebene idealisierte Zeitreihenfunktion hat vier Parameter: die Zeitpunkte für den Sprungstart t_a, das Spungende t_b, die Anfangsintensität I_a und die Intensität nach dem Sprung I_b. Ziel ist es, die beiden Intensitätswerte robust zu schätzen. Dies geschieht durch Betrachtung der mittleren Intensität von zwei Bereichen der Zeitreihe, welche in folgender Weise um einen zu wählenden Sprunganfangszeitpunkt t_a angeordnet sind. Sei $B \neq 0$ die Länge der Intervalle, in welchen gemittelt wird. Zwischen den Mittelungsbereichen befindet sich ein Abstandsfenster der Länge A. Seien nun die Schätzungen der Intensitätswerte

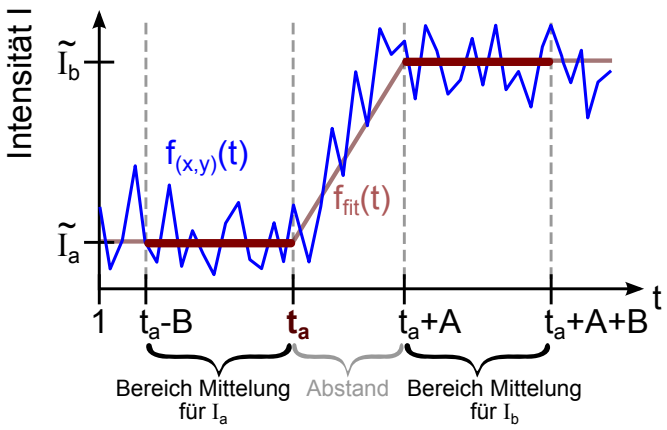


Abbildung 3.3.: Anpassung der idealisierten Funktion f_{fit} an eine gegebene Zeitreihe $f_{(x,y)}$ zum Zeitpunkt t_a

wie folgt definiert:

$$\widetilde{I}_a = \frac{1}{B} \sum_{t=t_a-B}^{t_a} f_{(x,y)}(t) \quad \text{und} \tag{3.4}$$

$$\widetilde{I}_b = \frac{1}{B} \sum_{t=t_a+A}^{t_a+A+B} f_{(x,y)}(t) \; . \tag{3.5}$$

Der Wertebereich für den Zeitpunkt t_a ergibt sich aus den Größen A und B sowie der Länge der Zeitreihe K:

$$B \leq t_a \leq K - A - B. \tag{3.6}$$

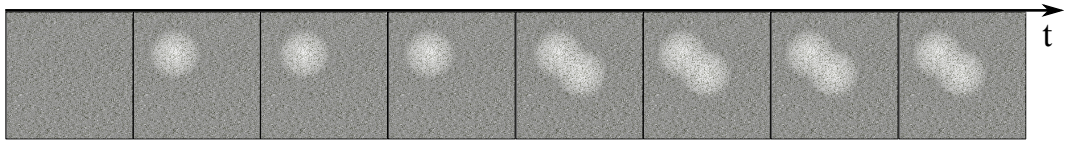
Die Schätzung des Intensitätssprungs für den Zeitpunkt t_a hat demnach folgende Gestalt:

$$\Delta\widetilde{I} = \widetilde{I}_b - \widetilde{I}_a. \tag{3.7}$$

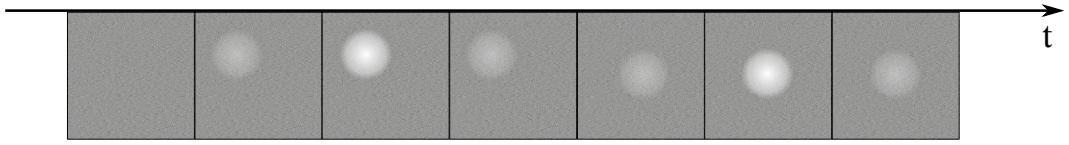
In Abbildung 3.3 ist diese Funktionsanpassung anschaulich dargestellt. Die Länge des Intervalls B gibt die Stärke der Mittelung an und ist abhängig von der Stärke des Rauschens. Der Abstand der beiden Intervalle A ist von der erwarteten maximalen Flankensteilheit der Sprünge abhängig. Die Werte im Bereich zwischen den Mittelungen haben keinen Einfluss auf die Schätzung, ein (langsamer) Anstieg in diesem Bereich verändert das Ergebnis anschaulich nicht, solange der Abstand A groß genug gewählt wird.

3.3. Zeitreihengeglättete Differenzbilder

Aus der im vorherigen Abschnitt vorgestellten Funktionsanpassung zur Sprunghöhendetektion auf Basis einer einzelnen Zeitreihe lässt sich ein zeitreihengeglättetes Differenzbild gewinnen, welches im Verlauf der weiteren Analyse, insbesondere der Klassifikation (siehe Kapitel 5),



(a) Verrauschte Eingangsbildfolge $f_t(x, y)$



(b) zeitreihengeglättetes Differenzbild $f_{\text{diff},t}(x, y)$

Abbildung 3.4.: Wirkung der Zeitreihenglättung auf verrauschte Eingangsbilder. In (a) sind zwei Anhaftungen in Form weißer Flecken zu sehen, welche nacheinander auftreten und sich dabei partiell verdecken. Im Differenzbild (b) ist das Rauschen gemindert und die Flecken überdecken sich nicht mehr, allerdings erscheinen und verschwinden diese mit einer Verzögerung, die durch die zeitliche Mittelung zu erklären ist.

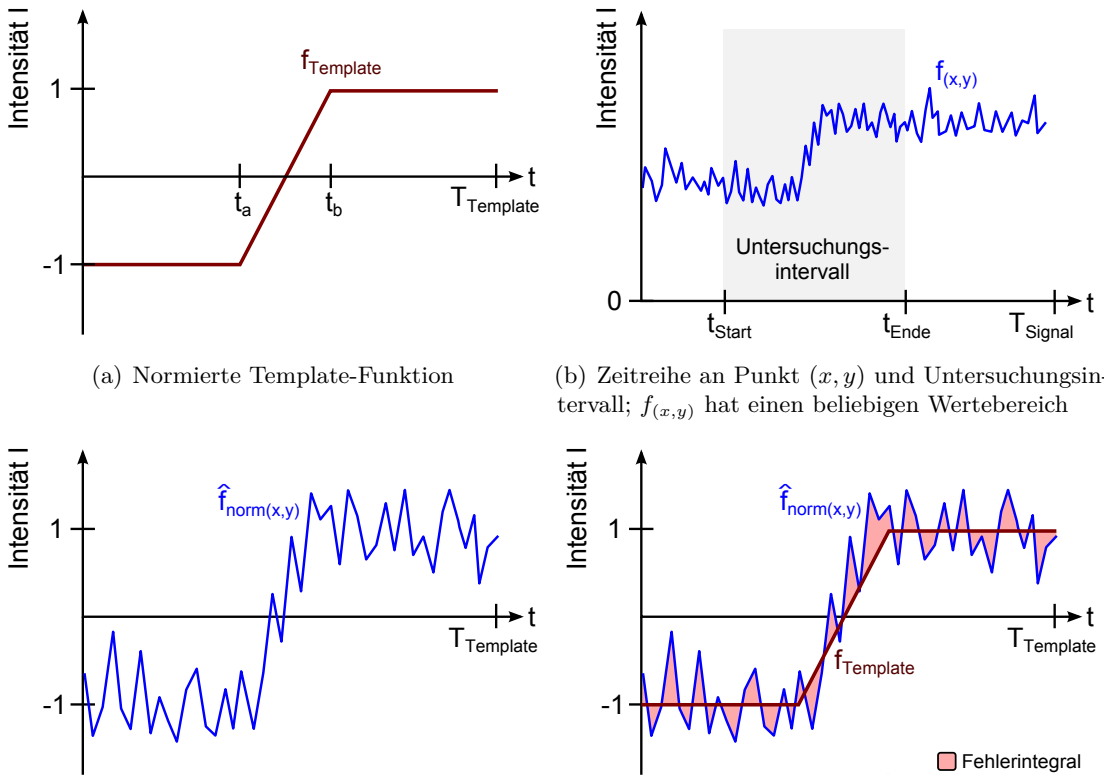
Verwendung findet. Dieses Bild sei wie folgt definiert:

$$f_{\text{diff},t}(x, y) := \Delta\tilde{I}(x, y, t) \ , \quad 1 \leq x \leq N \ , \quad 1 \leq y \leq M, \tag{3.8}$$

wobei die Definition des Intensitätssprungs aus der Berechnung für *einen* Zeitpunkt auf *einer* Zeitreihe in Abschnitt 3.2 auf das gesamte Bild (x- und y-Variable) und alle Zeitpunkte (t-Variable) ausgeweitet ist. Dieses Differenzbild ist durch die Mittelung geglättet und weniger rauschanfällig als die direkte Differenz zwischen zwei Bildern. Die Betrachtung der Differenzbilder ermöglicht auch die Trennung von Anhaftungen, welche hintereinander an einer Stelle auftreten. Auch werden dauerhafte Stöungen, die nach dem Festlegen des Hintergrundbildes auftreten, nur einmal in diesem Differenzbild erscheinen. In Abbildung 3.4 ist die Wirkung dieses Differenzbildes an einem Beispiel erklärt. Die dort nacheinander auftretenden und mit Rauschen überlagerten Flecken überscheiden sich in der Eingabebildfolge teilweise. Im zeitreihengeglätteten Bild überschneiden sich die Flecken nicht, was durch die differentielle Charakteristik des Filters zu erklären ist.

3.4. Templatebasierte Verfahren

Im Folgenden soll ein normiertes Template-Matching auf Zeitreihen vorgestellt werden, mit dem Ziel, Eingabezeitreihen mit einem idealisierten Verlauf – dem Template – vergleichen zu können. Zunächst gilt es, eine geeignete Vorlage (Template) in Form einer Funktion $f_{\text{Template}}(t)$ zu definieren. Dies kann beispielsweise die idealisierte Zeitreihenfunktion aus Abschnitt 3.1 sein. Diese Template-Funktion sei auf dem Intervall $[1, T_{\text{Template}}]$ definiert, wobei $T_{\text{Template}} \leq T_{\text{Signal}}$, der Länge der zu untersuchenden Zeitreihe entspricht. Somit ist es möglich, Abschnitte der Zeitreihe auf ein Muster hin zu untersuchen, und mit *einem* Template verschieden lange Zeitreihen behandeln zu können. Der Prozess des Template-Matchings läuft wie folgt ab:



(a) Normierte Template-Funktion

(b) Zeitreihe an Punkt (x, y) und Untersuchungsintervall; $f_{(x,y)}$ hat einen beliebigen Wertebereich

(c) Normierter Ausschnitt aus Zeitreihenfunktion

(d) Vergleich der normierten Zeitreihe mit dem Template und Fehlerintegral

Abbildung 3.5.: Ablauf des Zeitreihen-Template-Matchings: Zunächst wird eine Template-Funktion als Vorlage definiert (a). Die zu untersuchende Zeitreihe (b) wird in einem Ausschnitt normiert (c) und mit dem Template verglichen (d).

Aus der Zeitreihe $f_{(x,y)}(t)$ wird mit Hilfe eines Intervalls (Startindex t_{Start} und Endindex t_{Ende}) ein Ausschnitt der Länge T_{Template} gewählt. Diese Funktion habe die Bezeichnung $\widehat{f}_{(x,y)}(t)$. Anschließend wird eine **Normierung der statistischen Merkmale** Mittelwert und Standardabweichung durchgeführt, um den Wertebereich des Ausschnitts mit dem der (ebenfalls zu normierenden) Template-Funktion anzugleichen. Zunächst werden die statistischen Merkmale Mittelwert

$$\mu = \frac{1}{T_{\text{Template}}} \sum_{t=1}^{T_{\text{Template}}} \widehat{f}_{(x,y)}(t) \tag{3.9}$$

und Standardabweichung

$$\sigma = \sqrt{\frac{1}{T_{\text{Template}}} \sum_{t=1}^{T_{\text{Template}}} \left(\widehat{f}_{(x,y)}(t) - \mu\right)^2} \tag{3.10}$$

der Funktion $\widehat{f}_{(x,y)}(t)$ bestimmt. Sei weiterhin die Normierung der Funktion wie folgt definiert:

$$\widehat{f}_{\mathrm{norm}(x,y)}(t) = \frac{\widehat{f}_{(x,y)}(t) - \mu}{\sigma} \quad \text{mit} \quad 1 \leq t \leq T_{\mathrm{Template}} \quad \text{und} \quad \sigma > 0 \tag{3.11}$$

Diese Transformation normiert den Mittelwert der Zeitreihenfunktion $\widehat{f}_{\mathrm{norm}(x,y)}(t)$ auf den Wert 0 und die Standardabweichung auf den Wert 1. Dies ist durch die Annahme einer Normalverteilung und Ausnutzung der Linearität der selbigen zu rechtfertigen [BBK08]. Abschließend wird ein Fehlermaß (ein Maß für das Fehlerintegral) definiert, um die Abweichung der untersuchten Zeitreihe vom Template zu bestimmen:

$$\epsilon_{\mathrm{Fehler}}(t_0) = \sum_{t=1}^{T_{\mathrm{Template}}} \left(f_{\mathrm{Template}}(t) - \widehat{f}_{\mathrm{norm}(x,y)}(t + t_0) \right)^2 \quad \text{mit} \quad 1 \leq t_0 \leq T_{Signal} - T_{Template} \tag{3.12}$$

Dieses ist für einen Zeitpunkt t_0 definiert, da die Eingabezeitreihe typischerweise länger ist, als die Templatefunktion. Um entscheiden zu können, ob der Verlauf einer Zeitreihe einem über das Template vorgegebenem Muster entspricht, sei ein Schwellwert $\epsilon_{\max}$ definiert, der den maximal akzeptierten Fehler angibt. Wenn für einen Zeitpunkt t_0 der Schwellwert unterschritten wird, also $\epsilon_{\mathrm{Fehler}}(t_0) < \epsilon_{\max}$, wird von einem (abschnittsweisen) *ähnlichen* Verlauf der Zeitreihe im Vergleich zum Template ausgegangen. Dieses Verfahren lässt sich auch mit vorhergehendem Denoising (siehe Kapitel 2.2) kombinieren, um den Einfluss des Rauschens auf den Fehler (und somit die Entscheidung der Ähnlichkeit) zu minimieren. In Abbildung 3.5 ist der Verlauf des normierten Zeitreihen-Template-Matchings an einem Beispiel dargestellt.

3.5. Zustandsmodelle

Um Anhaftungen in den Bildreihen der PAMONO-Analyse zu detektieren, gilt es, das zeitliche Verhalten der Intensitätswerte von Bildregionen zu erfassen und mit dem erwarteten Verhalten eines positiven Befundes zu vergleichen. Das erwartete Verhalten von Partikeln ist eine dauerhafte Anhaftung an einer bestimmten Position auf dem Sensor, welche sich als fleckartige Struktur in den Bilddaten an einer entsprechenden Region manifestiert. Innerhalb dieser Regionen ist in der Zeitreihe ein sprunghafter Anstieg (oder auch Abnahme[1]) der mittleren Intensität zu verzeichnen (siehe Abbildung 3.1). Wie in Kapitel 1 erwähnt, gibt es allerdings auch Störungen und Artefakte, die im Laufe der Analyse auftreten. Die entsprechenden Zeitreihen zeichnen sich durch einen signifikanten und dauerhaften Sprung der mittleren Intensität aus. Im Folgenden wird eine Methode vorgestellt, die das „Verhalten“ von den Zeitreihen abstrahiert, um die Datendimension und -menge deutlich zu verringern.

In Abbildung 3.6 ist beispielhaft eine Zeitreihe einer Störung gegeben. Dort sind deutlich drei Bereiche zu erkennen: Der erste Bereich zeigt unauffälliges Rauschen, bei $t = 60$ findet ein sprunghafter Anstieg der mittleren Intensität statt, welche bis $t = 115$ auf dem Niveau verbleibt. Danach findet wieder ein starker Abfall der Intensität statt – was in Zusammenhang

[1] Neben dem Anstieg von Intensitätswerten kann bei einigen Partikeln auch eine „Invertierung“ dieses Verhaltens festgestellt werden.

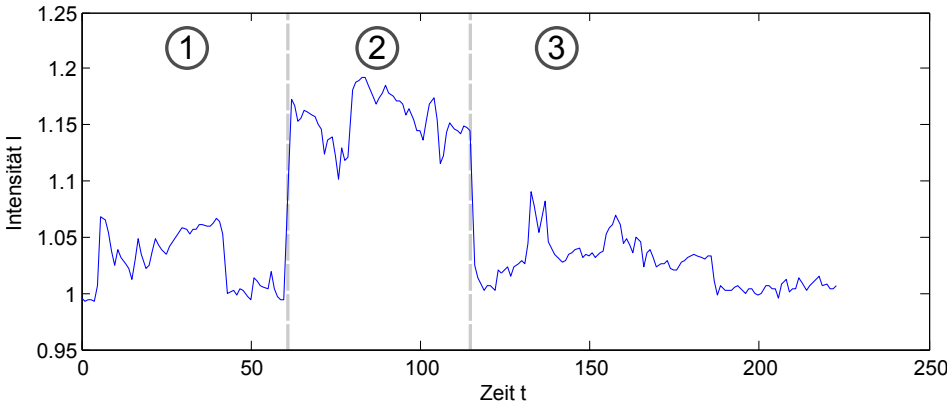


Abbildung 3.6.: Zeitreihe aus einem Bildbereich mit Störung in Form eines Anstiegs mit nachfolgendem Abfall der mittleren Intensität. Die Zeitreihe ist in drei Phasen einzuteilen.

mit dem ersten Anstieg dem erwarteten Verhalten einer Anhaftung widerspricht und als Störung zu werten ist.

Offenbar lassen sich die Verläufe der Zeitreihen durch ein geeignetes Modell mit wenigen Zuständen beschreiben. Aus dieser Erkenntnis stammt die Idee für einen **Zeitreihen-Zustandsautomaten**, der den zeitlichen Verlauf in wenige, aber aussagekräftige Zustände überführt. Hier werde ein deterministischer endlicher Automat verwendet, wie er in [EP08] definiert ist. Das Tupel

$$A_{\text{Zeitreihe}} = (K, \Sigma, \delta, s_0, F) \tag{3.13}$$

beschreibt den Automaten, wobei K eine endliche Menge an Zuständen, Σ ein endliches Eingabealphabet, δ die Zustandsübergangsfunktion, $s_0 \in K$ den Startzustand und $F \subseteq K$ die Menge der akzeptierenden (finalen) Zustände bezeichnet. Die zeitliche Dimension werde sukzessive in jedem Zeitschritt t mit $1 \leq t \leq K$ betrachtet und eine Zeitreihenanalyse mittels der in Abschnitt 3.2 vorgestellten Funktionsanpassung vorgenommen. Der dort berechnete Intensitätssprung sei zusammen mit zwei Schwellwerten δ_{positiv} und δ_{negativ} das Kriterium für den **Sprungindikator**. Die Schwellwerte definieren die minimale prozentuale Änderung der mittleren Intensität, die als signifikanter Sprung gewertet werden. Die Schwellwerte sind je nach Datensatz unterschiedlich und als Parameter anzusehen. Typischerweise liegen diese zwischen 1 bis 4 %. Der Sprungindikator bestimmt weiterhin die Zustandsübergangsfunktion:

$$g(\Delta I) := \begin{cases} \delta^+ & \text{für } \Delta I \geq \delta_{\text{positiv}} \\ \delta^- & \text{für } \Delta I \leq \delta_{\text{negativ}} \\ \delta^0 & \text{sonst.} \end{cases} \tag{3.14}$$

Dabei steht das Symbol δ^+ für den einen positiven Sprungindikator und δ^- für einen negativen Sprungindikator. Das Symbol δ^0 bezeichnet den Fall, dass kein signifikanter Sprung vorliegt. Um das Sprungverhalten von Zeitreihen erfassen zu können, ist die folgende Abbildung auf die Automatenelemente vorzunehmen:

- **Zustandsmenge** $K = \{s_0, s_+, s_-, s_a\}$: Hier werden die möglichen Zustände, die eine

Zeitreihe im Laufe der Analyse annehmen kann, kodiert:

  - Zustand s_0 bezeichne den Zustand „noch kein Sprung detektiert“,
  - Zustand s_+ bezeichne den Zustand „signifikanter positiver Sprung detektiert“,
  - Zustand s_- bezeichne den Zustand „signifikanter negativer Sprung detektiert“,
  - Zustand s_a bezeichne den Zustand „ausgeschieden“, welcher bei nicht passendem Sprungverhalten gewählt wird.

- **Alphabet** $\Sigma = \{\delta^o, \delta^+, \delta^-\}$: Hier werden die Eingaben des Automaten in Form von Informationen über das aktuelle Sprungverhalten beschrieben, das sich aus dem Intensitätssprung ΔI für einen Zeitpunkt t bestimmen lässt. Dabei bestimmt das Ergebnis der Funktion $g(\Delta I)$ die Eingabe für jeden Zeitpunkt t:
  - Eingabe δ^o bezeichnet keinen signifikanten Sprung,
  - Eingabe δ^+ bezeichnet positiven Sprung,
  - Eingabe δ^- bezeichnet negativen Sprung.
- **Zustandsübergangsfunktion** δ: Die Übergänge sind aus der graphischen Notation des Automaten in Abbildung 3.7 zu entnehmen. Sobald zwei signifikante Sprünge in unterschiedliche Richtungen erfolgen, wird das Zeitverhalten der Zeitreihe als „ausgeschieden“ zurückgewiesen, was dem aktuellen Forschungsstand entspricht.
- **Startzustand** s_0: Der Automat startet im Zustand „noch kein Sprung detektiert“.
- **Finale Zustände** $F = \{s_+, s_-\}$: Die beiden Zustände, die gültige (im Sinne von erwarteten) Sprung-Verhaltensweisen der Zeitreihe beschreiben, seien als final deklariert. Für diese Zeitreihen werden weitere (in den nächsten Kapiteln vorgestellte) Untersuchungen vorgenommen.

Durch den Einsatz eines Zustandsautomaten für jeden Punkt (x, y) in der Bildfolge lässt sich das zeitliche Verhalten aller Zeitreihen von der Menge an einzelnen Intensitätswerten abstrahieren. Eine Bewertung für eine Klassifikation (siehe Kapitel 5) ist so ohne wiederholte Betrachtung der Zeitreihen (und aller dort enthaltenen Werte) möglich. Zudem lässt sich das Verhalten des Automaten durch Änderung der Zustandsübergangsfunktion oder durch Einfügen neuer Zustände an neue Kenntnisse über die PAMONO-Analyse flexibel anpassen.

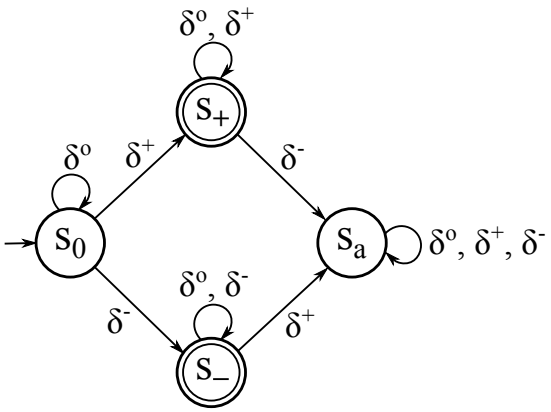


Abbildung 3.7.: Zustandsautomat zur Einordnung von Zeitreihen in „positive“ (Zustand s_+) und „negative“ (Zustand s_-) Intensitätssprünge sowie abzuweisende Sprungkombinationen (Zustand s_a).

4. Segmentierung

Im Folgenden wird die Segmentierung auf der Basis von bildbasierten Merkmalen näher erläutert. Die Darstellung beschränkt sich auf die Verfahren *Region Growing* (Kapitel 4.2) und *Gruppierung zu Regionen* (Kapitel 4.3), die beide im PAMONO-Explorer Verwendung finden.

4.1. Einleitung

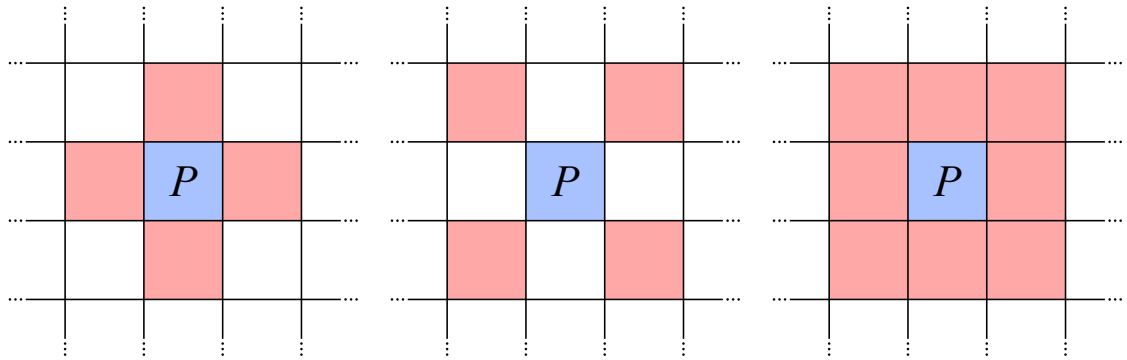
Wie in Kapitel 3 dargestellt, berechnet die Zeitreihenanalyse genau die Punkte einer Bildreihe, die einen sprunghaften Intensitätsanstieg bzw. -abfall aufweisen. Aus Kapitel 2.2 ist außerdem bekannt, dass Anhaftungen von Nanopartikeln sich in den aufgenommenen Bildern, verglichen mit ihrer Umgebung, durch hellere Gebiete (bzw. dunkelere Gebiete bei abfallender Intensität) mit einer Größe von ca. 5×5 Pixel auszeichnen. Daraus ergibt sich die Notwendigkeit, die mit Hilfe der Zeitreihenanalyse identifizierten Punkte zu zusammenhängenden Regionen (Segmenten) zusammenzufassen.

Da bei der Beschreibung der Segmentierungsverfahren in diesem Abschnitt der Begriff der *Nachbarschaft eines Pixels* eine tragende Rolle spielt, soll dieser nachfolgend definiert werden [GW08]. Seien x und y die Koordinaten eines Pixels $P = (x, y)$, so lassen sich für P die folgenden Nachbarschaften definieren (siehe auch Abbildung 4.1):

- $N_4(P) = \{(x+1, y), (y-1, x), (x-1, y), (y+1, x)\}$
- $N_d(P) = \{(x+1, y+1), (x+1, y-1), (x-1, y-1), (x-1, y+1)\}$
- $N_8(P) = N_4(P) \cup N_d(P)$

4.2. Region Growing

Region Growing ist ein Verfahren zur iterativen Segmentierung von Bildern oder Bildausschnitten. Der Algorithmus startet mit dem kleinst möglichen Segment, einem Pixel, als Anfangswert (*Seed*), sodass das Segment $S = \{P\}$ anfangs nur P enthält. Die Zugehörigkeit eines Pixels zu einem Segment bestimmt eine Zugehörigkeitsfunktion $Z : N \times M \mapsto U$ mit $U = \mathcal{P}(N \times M)$ (Menge aller Segmente), wobei für alle Pixel $P \in N \times M$ anfangs $Z(P) = U$ gilt. Die Vergrößerung des Segments geschieht durch Betrachtung der Nachbarschaft $N_8(P)$: Für jeden Pixel $Q \in N_8(P)$ wird überprüft, ob Q bezüglich S einer bestimmten Ähnlich-



(a) 4er-Nachbarschaft $N_4(P)$ (b) d-Nachbarschaft $N_d(P)$ (c) 8er-Nachbarschaft $N_8(P)$

Abbildung 4.1.: Nachbarschaften eines Pixels P

keitsanforderung genügt und ob $Z(Q) = U$. Als Beispiel für eine Ähnlichkeitsanforderung sei an dieser Stellte eine Differenz der Intensität von Q und der mittleren Intensität von S von maximal 5% genannt. Trifft dies zu, so werden $S = S \cup Q$ und $Z(Q) := S$ gesetzt. Für die zu S neu hinzugewählten Pixel beginnt das Verfahren von vorne. Dieses Vorgehen wird solange iteriert, bis keine weiteren Pixel mehr die Ähnlichkeitsanforderung erfüllen oder $|S|$ ein definiertes Maximum erreicht hat.

Als Datenstruktur für die neu hinzugewählten Pixel bietet sich eine Queue an, die anfangs nur P enthält. Soll das Verfahren für einen Pixel aus der Queue gestartet werden, so wird dieser Pixel aus der Queue entnommen. Benachbarte Pixel, die während dieses Vorgangs dem Segment hinzugefügt werden, werden wieder in die Queue eingefügt. Wenn keine weiteren Pixel hinzugewählt werden können, ist die Queue leer und das Verfahren endet. Algorithmus 4.1 veranschaulicht das Vorgehen bei *Region Growing*.

Um die Größe einer mit *Region Growing* berechneten Punktmenge zu visualisieren, bietet sich die Berechnung der konvexen Hülle an. Wie dieses Ziel effizient erreicht werden kann, ist Thema des folgenden Abschnitts.

4.3. Gruppierung zu Regionen (Clustering)

In Kapitel 4.2 wurde mit *Region Growing* ein Segmentierungsverfahren beschrieben, das Segmente, ausgehend von einem Startpunkt, selbstständig iterativ vergrößert. Durch dieses Vorgehen enthalten die Segmente auch Pixel, die gemäß der Zeitreihenanalyse kein für Partikel-Anhaftungen typisches Sprungverhalten aufweisen. Das nachfolgend beschriebene Verfahren beschränkt sich darauf, die Ergebnispixel der Zeitreihenanalyse zu Regionen (im Folgenden als „Segmente“ bezeichnet) zusammenzufassen, die ausschließlich Pixel mit einer typischen Zeitreihe enthalten, dafür aber oft kleiner ausfallen, als die Partikel-Anhaftungen in den Bildern tatsächlich ist.

Die Eingabe bei der „Gruppierung zu Regionen“ bilden die Ergebnispixel der Zeitreihenanalyse.

Algorithmus 4.1 Algorithmische Vorgehensweise bei der Segmentierung mittels *Region Growing* für einen Anfangswert P_0, einer maximalen Segmentgröße `MAXSIZE` und einer Ähnlichkeitsfunktion $s(S, P)$ die genau dann *true* liefert, wenn P ähnlich ist zu S.

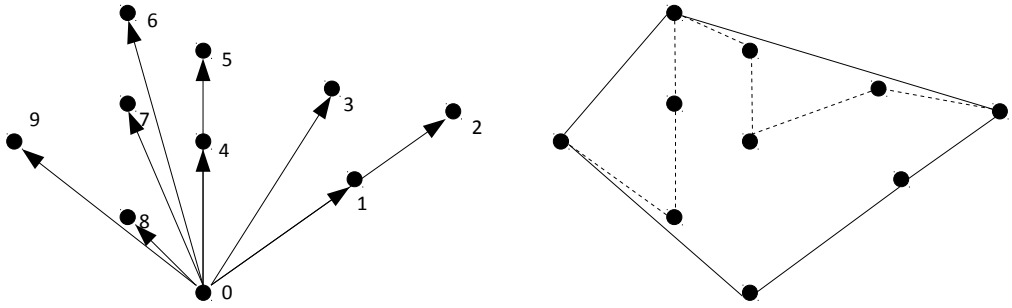
```
Queue Q
Füge initialen Punkt P_0 in Q ein
Segment S = {P_0}
while (not Q ist leer) and (Anzahl Elemente in Q < MAXSIZE) do
  P̃ ← Entnehme vorderstes Element aus Q
  for all P_N ∈ N_8(P̃) do
    if P_N noch keinem Segment zugeteilt and s(S, P_N) ist true then
      Q = Q ∪ {P_N}
      S = S ∪ {P_N}
      Markiere P_N als zu S zugehörig
    end if
  end for
end while
```

Partikel-Anhaftungen können unterschiedlich deutlich ausfallen, weshalb die Zeitreihenanalyse für Partikel-Anhaftungen nur wenige Pixel bishin zu fast allen die Partikel-Anhaftung repräsentierenden Pixel liefert. Um entscheiden zu können, ob zwei Pixel zu derselben Anhaftung gehören, muss daher der Abstand, den zwei solche Pixel maximal voneinander haben können (die maximale Größe der Regionen), vorab bekannt sein. Als Vereinfachung bietet es sich an von quadratischen Regionen auszugehen. Gestartet wird der Algorithmus also mit einer maximalen Seitenlänge $d_{\max}$. Die zu gruppierenden Pixel, die der Algorithmus als Eingabe erhält, werden in einer Queue gespeichert und in der Reihenfolge der Eingabe abgearbeitet. Für jeden der Queue entnommenen Pixel wird das nächste Nachbarsegment bestimmt. Dazu wird der euklidische Abstand d_{neu} zwischen dem neuen Pixel und diesem Segment (falls es eins gibt) berechnet. Ist $d_{\text{neu}} \leq d_{\max}$, so wird der neue Pixel dem Segment hinzugefügt und der Mittelpunkt dieses Segments als arithmetisches Mittel aller zu dem Segment gehörenden Pixel neu berechnet. Ist der neue Pixel jedoch mehr als $d_{\max}$ von dem nächsten Nachbarsegment entfernt, so bildet er ein neues Segment, dessen Mittelpunkt der neue Pixel selbst bildet.

4.4. Berechnung der konvexen Hülle einer Punktmenge

Um die mit dem *Region-Growing-Verfahren* berechneten Segmente zu visualisieren, bietet sich die Berechnung der konvexen Hülle an. Eine Punktmenge A wird als *konvex* bezeichnet, wenn die Verbindungslinie zwischen je zwei Punkten aus A ebenfalls in A liegt. Die *konvexe Hülle* von A bezeichnet die kleinste konvexe Menge, die A umfasst [OW93].

Ein recht einfach zu implementierender und doch gleichzeitig effizienter Algorithmus zur Berechnung der konvexen Hülle einer Punktmenge in der Ebene ist der *Graham-Scan-Algorithmus* [Lan06]. Sei $A = \{P_0, P_1, \ldots, P_{n-1}\}$ eine Menge von n Punkten in der Ebene, zu der die konvexe Hülle zu berechnen ist. In einem ersten Schritt wird der Punkt mit der kleinsten y-Koordinate bestimmt. Gibt es mehrere Punkte mit der gleichen y-Koordinate, so wird der



(a) Sortierung der Punkte (b) Konvexe Hülle einer 2D-Punktmenge

Abbildung 4.2.: Zur Funktionsweise des Graham-Scan-Algorithmus zur Berechnung der konvexen Hülle einer Punktmenge in der Ebene.

Punkt mit der kleinsten x-Koordinate gewählt. Sei P_k mit $k \in \{0, \ldots, n-1\}$ dieser Punkt. Anschließend werden die Koordinaten aller Punkte relativ zu P_k berechnet (P_k wird zum Ursprung) und alle Punkte nach ihrem Winkel zu P_k sortiert. Dabei kann auf rechenintensive trigonometrische Funktionen verzichtet werden: Seien $P_l \neq P_k$ und $P_m \neq P_k$ zwei Punkte, deren Winkel bezüglich P_k miteinander verglichen werden sollen. Der doppelte Flächeninhalt des Dreiecks (P_k, P_l, P_m) in einem Koordinatensystem mit P_k als Ursprung ist negativ, genau Null oder positiv, je nachdem ob P_m bzgl. P_k einen kleineren, genau den gleichen oder einen größeren Winkel bildet als P_l bzgl. P_k und berechnet sich nach

$$T(P_l, P_m) = x_l \cdot y_m - x_m \cdot y_l \begin{cases} < 0 & \measuredangle(P_m, P_k) < \measuredangle(P_l, P_k) \\ = 0 & \measuredangle(P_m, P_k) = \measuredangle(P_l, P_k) \\ > 0 & \measuredangle(P_m, P_k) > \measuredangle(P_l, P_k) \end{cases}. \tag{4.1}$$

In dem Fall, dass die Winkel von P_l und P_m gleich sind, gilt $P_l < P_m$, falls P_l einen geringeren Abstand zu P_k hat als P_m. Hier reicht es aus, den Manhattan-Abstand $|x| + |y|$ [Lan06] zu betrachten. Sind die Punkte einmal nach ihrem Winkel sortiert, können ihre ursprünglichen Koordinaten wiederhergestellt werden, sodass nicht mehr P_k, sondern der Punkt $(0, 0)$ den Ursprung bildet. Abbildung 4.2(a) zeigt beispielhaft eine Punktmenge in der Ebene, die nach dem beschriebenen Verfahren sortiert wurde. Die konvexe Hülle derselben Punktmenge ist in Abbildung 4.2(b) zu sehen.

Sei $A = \{P_0, P_1, \ldots, P_{n-1}\}$ die Menge der Punkte in sortierter Reihenfolge (sortiert nach aufsteigendem Winkel bezüglich P_0). Im Falle $|A| \leq 2$ können die Punkte direkt in der sortierten Reihenfolge ausgegeben werden. Im Falle $|A| > 2$ werden P_0 und P_1 auf einem *Stack* (Stapelspeicher) abgelegt. Diese beiden Punkte gehören auf Grund der Sortierung in jedem Fall zur konvexen Hülle von A. Für P_2 muss getestet werden, ob dieser Punkt kollinear mit P_1 und P_0 ist. In diesem Fall wird P_1 vom Stack entfernt und P_2 auf dem Stack abgelegt. Für jeden anderen Punkt hilft eine Betrachtung der zwei obersten Elemente P_{S0} (oberstes Element) und P_{S1} (direkt unter P_{S0}) auf dem Stack. Liegt der neue Punkt linksseitig des Vektors $P_{S1} \rightarrow P_{S0}$ (Ecke ist konvex) oder auf dem Stack befinden sich nur noch P_1 und P_0, dann wird der neue Punkt auf dem Stack abgelegt. Andernfalls wird P_{S0} vom Stack entfernt und der neue Punkt auf dem Stack abgelegt. Am Ende des Durchlaufs liefert ein Abräumen

Algorithmus 4.2 Der Graham-Scan-Algorithmus berechnet die konvexe Hülle unter Benutzung eines Stacks in Zeit $O(n \log n)$.

input Array $A(0, \ldots, n-1)$ mit n Punkten in der Ebene
$min \leftarrow$ A-Index von Punkt mit kleinster y- bzw. x-Koordinate
Vertausche in A die Elemente mit den Indizes 0 und min
Mache alle Punkte relativ zu $A(0)$
Sortiere $A(1, \ldots, n-1)$ aufsteigend nach Winkel und Entfernung zu $A(0)$
Rekonstruiere die ursprünglichen Koordinaten der Punkte
if Es gibt ≤ 2 Punkte **then**
  Lege die Punkte auf den Stack
end if
for $i = 2$ **to** n **do**
  if $A(i)$ liegt linksseitig von $P_{S1} \rightarrow P_{S0}$ **then**
    Lege $A(i)$ auf dem Stack ab
    $i = i + 1$
  else
    Entferne das oberste Element vom Stack
  end if
end for
return Stack in umgekehrter Reihenfolge

des Stacks die konvexe Hülle im Uhrzeigersinn. Algorithmus 4.2 verdeutlicht dieses Vorgehen.

Die Laufzeit des Graham-Scan-Algorithmus ergibt sich wie folgt: Den Punkt mit der kleinsten y- bzw. x-Koordinate zu bestimmen dauert $O(n)$. Um die Punkte anschließend zu sortieren werden $O(n \log n)$ Schritte benötigt. Das Durchlaufen der Punkte in sortierter Reihenfolge dauert nochmals $O(n)$. Somit ergibt sich als Gesamtlaufzeit $O(n \log n)$.

5. Klassifikation

In den vorhergehenden Kapiteln 2, 3 und 4 wurden Methoden aus den Bereichen Bildverarbeitung, Zeitreihenanalyse und Segmentierung im Kontext der PAMONO-Analyse vorgestellt. In diesem Kapitel sollen diese Erkenntnisse genutzt werden, um die zentrale Aufgabe der automatisierten Detektion in Form einer Klassifikation zu lösen. In Abschnitt 5.1 wird dabei ein genereller Überblick über Mustererkennung und Klassifikation gegeben. Im folgenden Abschnitt 5.2 wird eine an die PAMONO-Analyse angepasste Klassifikationsstrategie vorgestellt. Anschließend wird die verwendete Klassenstruktur in Abschnitt 5.3 erläutert, um im nächsten Abschnitt 5.4 die Einführung einer Vorauswahlstufe begründen zu können. In Abschnitt 5.5 wird der Prozess des Trainings beschrieben. Abschließend werden drei Klassifikatoren vorgestellt: Ein klassischer Bayes-Ansatz (Abschnitt 5.6) und zwei Template-Matching-Ansätze, welche im letzten Abschnitt 5.7 diskutiert werden.

5.1. Mustererkennung und Klassifikation

Bei der Problematik der Mustererkennung geht es darum, aus Eingabedaten wiederkehrende Gesetzmäßigkeiten zu detektieren. Diese Gesetzmäßigkeiten werden dabei als Muster definiert. Neben der Identifizierung von Mustern ist die Einordnung und Klassifikation dieser in verschiedene Kategorien von Bedeutung. Dabei geht es darum, Objekte mit zugehörigen Eigenschaften in abstrakte Klassen einzuordnen. Diese Klassen sind dabei untereinander abgegrenzt und haben typischerweise eine zugewiesene Bedeutung. Es gilt, die Klasseneinteilung für die zu Grunde liegende Problemstellung jeweils adäquat zu definieren. Um diese Einteilung und Erkennung zu ermöglichen, müssen die Klassen zuvor beschrieben oder *angelernt* werden. Dabei wird grundsätzlich zwischen überwachten und unüberwachten Lernverfahren unterschieden [Alp08]. Der Lernvorgang (oder auch das *Training*) besteht dabei aus einer Referenzzuordnung von Objekten – den sogenannten *Samples* – zu Klassen. Bei den überwachten Verfahren entscheidet ein menschlicher Betrachter über diese Zuordnung. Die Anzahl und die zugewiesenen Bedeutungen der Klassen sind dabei ebenfalls durch die menschliche Instanz definiert. Bei unüberwachten Lernverfahren werden diese Schritte automatisiert durchgeführt, wobei die Eigenschaften der Objekte selbst zur Abgrenzung verwendet werden. Die automatisierte Einordnung in Klassen wird nach Abschluss des Trainings von einem *Klassifikator* durchgeführt, der hierfür die durch Merkmale repräsentierten Eigenschaften der (neu) zu klassifizierenden Objekte verwendet. Ein Merkmal ist dabei eine mess- oder wahrnehmbare Eigenschaft eines betrachteten Objekts. Ein Merkmalsraum ist typischerweise ein Vektorraum $\mathbb{R}^d$ mit Dimensionalität d, wobei hier d die Anzahl der verwendeten skalaren Merkmale darstellt. Die Auswahl der relevanten Merkmale der Objekte ist ein entscheidender Aspekt für einen Klassifikator, da nur eine ausreichende Unterscheidbarkeit von Merkmalsausprägungen eine zuverlässige Klassifikation ermöglicht.

Für weitergehende Einführungen in die Theorie der Mustererkennung sei an dieser Stelle auf [Nie03] und [TK06] verwiesen. Der Schwerpunkt in den folgenden Ausführungen liegt auf dem Anwendungsaspekt – es sollen Methoden der Mustererkennung auf die Problematik dieser Projektgruppe angewandt werden.

5.2. Klassifikationsstrategie im Rahmen der Projektgruppe

Das zentrale Ziel der PAMONO-Analyse ist die automatisierte Detektion von Regionen in den Eingabebildreihen, die Anhaftungen auf dem PAMONO-Sensor entsprechen (siehe Kapitel 1). Dabei ergeben sich verschiedene Fragestellungen:

1. Wie unterscheidet sich eine Region, die eine Anhaftung repräsentiert, von ihrer Umgebung?

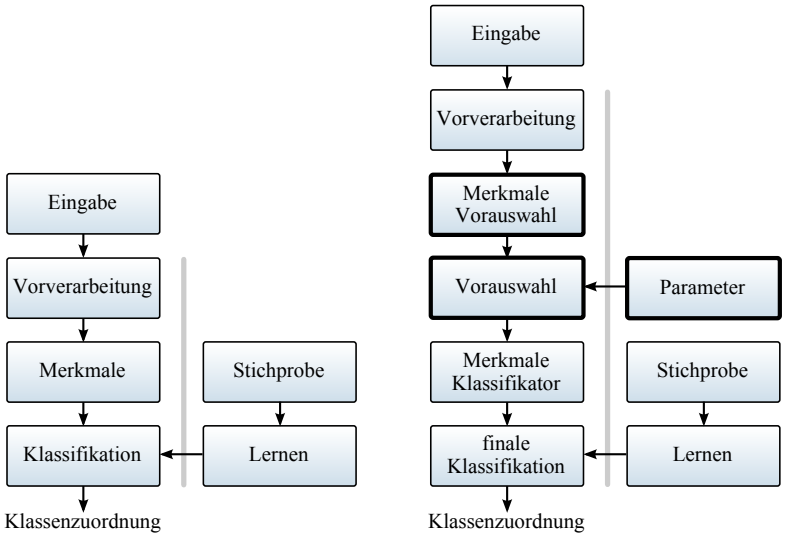
2. Welche mess- oder berechenbaren Merkmale lassen sich als Kriterium verwenden?

3. Gibt es verschiedene Erscheinungsformen dieser Anhaftungen und welche Einflüsse spielen eine Rolle?

4. Welche Anforderungen werden an den Analyseprozess gestellt?

Um die ersten beiden Fragen zu beantworten, gilt es zunächst, Forschung über die zu erkennenden Strukturen zu betreiben, um geeignete Methoden zur Bestimmung von Merkmalen zu entwickeln. Dies ist in den letzten drei Kapiteln geschehen – dort finden sich Verfahren aus Bildverarbeitung, Zeitreihenanalyse und Segmentierung, um Merkmale aus zur Verfügung stehenden Daten zu extrahieren. Frage drei betrifft die Einteilung in die zu unterscheidenen Klassen. Es muss (manuell oder automatisiert) entschieden werden, wie granular die Einteilung von Mustern der Anhaftungen in Klassen vorzunehmen ist. Dies wird in Abschnitt 5.3 genauer beleuchtet. Die letzte Frage nach den Anforderungen an den Prozess selber hängt von den Rahmenbedingungen ab, unter denen die automatisierte Erkennung stattfinden soll. So zeigt sich die Anforderung an die Geschwindigkeit der Analyse als Herausforderung – was die Motivation zur Abwandlung des Standardmodells des Klassifikationsablaufs begründet.

Der klassische Klassifikationsablauf nach Niemann [Nie03] ist in Abbildung 5.1 (a) dargestellt. Dort wird nach der Eingabe der (Bild-)Daten zunächst eine Vorverarbeitung durchgeführt. Aus diesen vorverarbeiteten Daten werden Merkmale extrahiert, die an einen Klassifikator weitergereicht werden. Bevor klassifiziert werden kann, muss der Klassifikator mit einer Stichprobe (einer Menge an Samples) angelernt werden. Die Klassifikationskette muss überarbeitet werden, wenn die Merkmalsextraktion und Klassifikation aufwändig und damit teuer (im Sinne von rechenintensiv) sind, die Analyse aber in einer gewissen Zeitvorgabe stattfinden soll. So sei hier der Ansatz von zwei kaskadierten Klassifikatorsystemen vorgestellt, wobei die erste (weniger aufwändige) Stufe eine Vorauswahl für die eigentliche, finale Klassifikation darstellt. Für die Vorauswahl werden zunächst einfach (und damit schnell) zu berechnende Merkmale für *alle* Eingabedaten extrahiert. Die Wahl der weiter zu analysierenden Kandidaten wird anschließend durch parametrisierbare Schwellwerte realisiert, die die Wertebereiche



(a) Klassischer Ansatz nach Niemann (b) Erweiterung durch Vorauswahlelemente

Abbildung 5.1.: In Bild (a) ist der klassische Modulaufbau einer Klassifikationskette nach [Nie03] dargestellt. Bild (b) zeigt eine Abwandlung mit Erweiterung durch eine Vorauswahlstufe (Änderungen sind hervorgehoben).

der (Vorauswahl-)Merkmale bestimmen. Dieser Einschub in die Verarbeitungskette ist in Abbildung 5.1 (b) zu sehen und wird in Abschnitt 5.4 näher erläutert. Zunächst werden die Klassendefinition und -einteilung diskutiert, um diese Abwandlung begründen zu können.

5.3. Klassen

Ein entscheidender Aspekt bei der Klassifizierung ist die Wahl der Klassen und deren Abgrenzung untereinander. Diese Entscheidung ist inhärent von den Eigenschaften des zu untersuchenden Prozesses abhängig. Bei der PAMONO-Analyse geht es darum, die messbaren Ausprägungen von Anhaftungen auf dem Sensor zu klassifizieren. Der Begriff der Anhaftung ist stellvertretend für Viren und die zunächst verwendeten *virus-like particles* (Partikel mit virenähnlichen Eigenschaften) zu verstehen (siehe Kapitel 1). Diese Bereiche, welche Anhaftungen zeigen, sind von großer Bedeutung bei der Bewertung einer Probe. Im Gegensatz dazu stehen die restlichen Bereiche der Bilddaten, die für eine Beurteilung irrelevant sind. Auf dieser Tatsache beruht die Entscheidung, die Klassenstruktur in einen relevanten und einen nicht relevanten Teil aufzuspalten. Dabei enthalte der nicht relevante Teil die statische Ablehnungsklasse „kein Fund“, während die Unterteilung der relevanten Gruppe weiterer Untersuchung bedarf. So zeigen Versuche mit unterschiedlichen Partikelgrößen verschiedene räumliche Ausprägungen. In Abbildung 5.2 sind verschiedenartige Formen von Flecken bei 80 nm und 200 nm großen Partikeln abgebildet. Bei letzteren zeigen sich charakteristische,

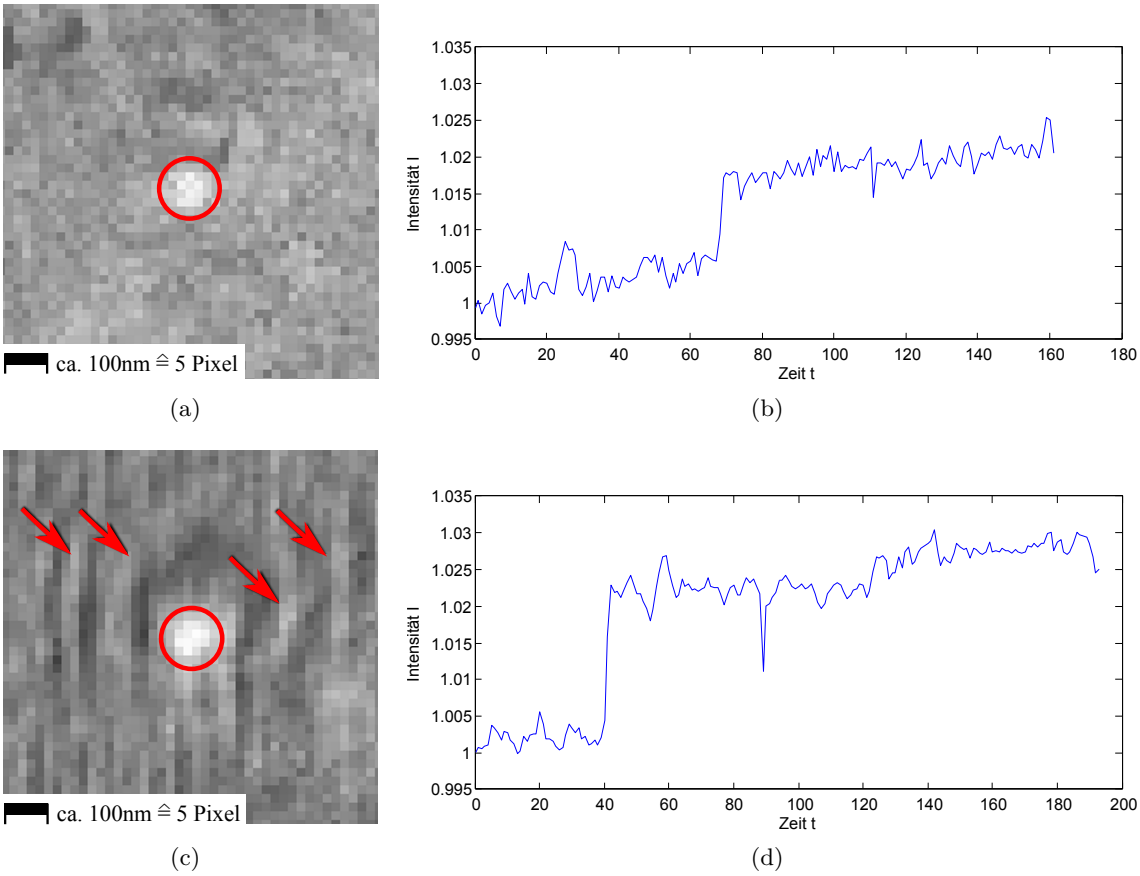


Abbildung 5.2.: Detailaufnahme und Zeitreihe von Anhaftungen verschiedener Größe. Abgebildet sind zentrierte Bildausschnitte sowie die gemittelten Zeitreihen eines 80 nm (siehe (a), (b)) und 200 nm großen Partikels (siehe (c), (d)). Auffällig sind der Größenunterschied und die beim 200 nm großen Partikel zu erkennende Ringstruktur (Pfeilmarkierungen) in der Umgebung des Mittelpunkts (siehe (c)). Die Zeitreihen zeigen bis auf unterschiedliche Sprungzeitpunkte ähnliche Charakteristik.

ringartige Strukturen in der Umgebung der Anhaftungen. Daten aus Versuchen mit realen Viren liegen dabei noch nicht vor, allerdings zeigt sich bereits bei den Kriterien Form und Größe die Notwendigkeit, verschiedene „Fund“-Klassen unterscheiden zu können. Daher ist es nötig ein Multiklassensystem zu verwenden. So gebe es neben der immer vorhandenen Ablehnungsklasse ω_0 noch $k \geq 1$ weitere Klassen $\omega_1, \omega_2, \ldots \omega_k$. Abbildung 5.3 zeigt eine mögliche Klasseneinteilung mit Ablehnungsklasse und zwei Partikelklassen.

5.4. Parametergesteuerte Vorauswahl

Ein parametergesteuerter Vorauswahlklassifikator nutzt Merkmale, um einen Teil der zu untersuchenden Eingabedaten von weiteren (typischerweise mit größerem Aufwand verbundenen)

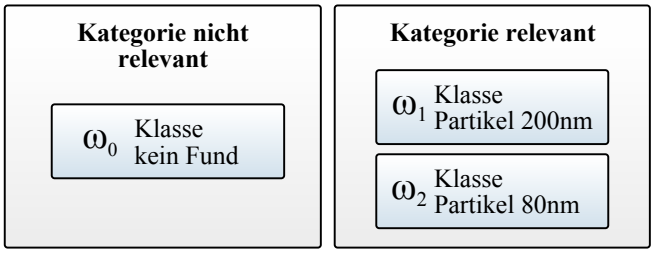


Abbildung 5.3.: Mögliche Klassenstruktur bei der PAMONO-Analyse. Die Klasse ω_0 ist die statische Ablehnungsklasse.

Untersuchungen auszuschließen. Diese Vorauswahl-Klassifikation nutzt „Ausschluss-Merkmale“, einen Teil des Merkmalsraums, in dem sich eine *deutliche* Einteilung in relevante und nicht relevante Kandidaten vornehmen lässt. Dies ist damit zu rechtfertigen, dass ein Experte diese Kandidaten in die Klasse „kein Fund“ (ω_0) einteilen würde. Im Prozess der PAMONO-Analyse lassen sich drei dieser Kriterien ausmachen:

- Sprungindikator
- Zustandsautomat
- räumliche Segmentierung.

Die in Kapitel 3.5 eingeführten Kriterien Sprungindikator und Zustandsautomat zeigen für einzelne Zeitreihen an, ob diese dem erwartetem Verlauf einer Anhaftung entsprechen. Durch Rauschen können diese Kriterien allerdings bei vereinzelten Pixeln auch fälschlicherweise ausgelöst werden. Durch das Wissen der Größe der Pixelstrukturen (siehe Abbildung 5.2) lässt sich die Aussage treffen, dass nur größere und zusammenhängende Bereiche von mehr als 4 miteinander verbundenen Pixeln mit dem gleichen Sprungverhalten für eine Anhaftung in Frage kommen. Um eine Vorauswahl treffen zu können, wird die Fusion der Merkmale Sprungindikator, Zustand und räumliche Segmentierung (siehe Kapitel 4) durchgeführt. Abbildung 5.4 verdeutlicht das Prinzip der Fusion: Die ausreichend große Region mit gleichem Sprungverhalten ist hervorgehoben – nur dort wird eine weitere Untersuchung durch die anschließende Klassifikationsstufe durchgeführt. Vereinzelte Bildpunkte mit „korrektem“ Zeitverhalten werden aussortiert.

Die Abgrenzung der Vorauswahl-Klassifikation von der finalen Klassifikationsstufe sei dabei wie folgt festgelegt: Die **Vorauswahl** K_{vor} klassifiziert Regionen entweder in die Ablehnungsklasse ω_0 oder leitet sie zur **finalen Klassifikation** K_{final} weiter, wo eine Einteilung in Klassen $\omega_0, \omega_1, \ldots \omega_k$ stattfindet[1]. Die Vorauswahl K_{vor} kann durch Parameter gesteuert werden und bestimmt die Menge der weiter zu klassifizierenden Bereiche. Als Parameter dienen die Schwellwerte des Sprungindikators (die Mindestgröße der prozentualen Änderung der mittleren Intensitätswerte einer Zeitreihe, siehe Kapitel 3.5) sowie die Mindestfläche eines zusammenhängenden Areals gleicher Zeitreihencharakteristik. Sind die Parameter so eingestellt, dass nur ein geringer Teil der Regionen vom finalen Klassifikator K_{final} verworfen werden (als

[1] Die Ablehnungsklasse kann natürlich auch von der finalen Klassifikation gewählt werden.

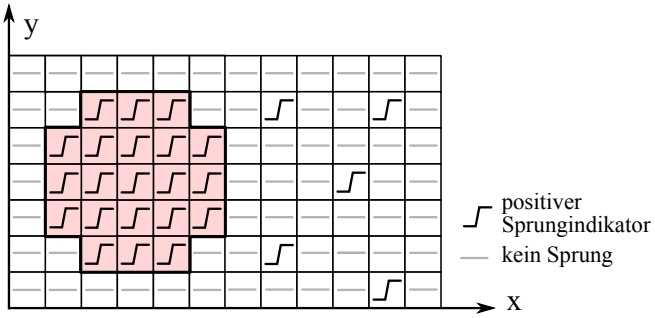


Abbildung 5.4.: Beispielhafter Ausschnitt aus dem Zeitverhalten einer Bildreihe. Bildpunkte, an denen ein Sprung festgestellt wurde, sind mit dem Sprungindikator gekennzeichnet. Erkennbar ist links eine größere Region (hervorgehoben) von Sprüngen. Auf der rechten Seite befinden sich einige einzelne (beispielsweise durch Rauschen verursachte) Sprungindikatoren, welche keine Anhaftung zeigen.

„kein Fund“ (ω_0) klassifiziert werden), so verringert sich auch der Aufwand der Gesamtanalyse, da insgesamt eine geringere Anzahl an aufwändigen Untersuchungen in K_{final} durchgeführt werden muss.

5.5. Lernverfahren und Samples

Die in der Projektgruppe verwendete Klassifikationskette besteht aus einer Vorauswahl-Klassifikation K_{vor} und einer finalen Klassifikation K_{final}, welche die Zuordnung von Kandidaten zu den in Abschnitt 5.3 definierten Klassen durchführt. Während K_{vor} durch Parameter gesteuert wird, basiert K_{final} auf dem Prinzip des *überwachten* Lernens. Dies ist durch die Vorgabe begründbar, verschiedenartige Partikel mit unterschiedlichen Größen und Formen nach Expertenvorgabe zu erkennen. Da sich das Projekt derzeit noch in einer aktiven Forschungsphase befindet, sind nicht alle Eigenschaften von Partikeln und Viren, wie sie später einmal erkannt und gezählt werden sollen, vollständig bekannt.

Generell lassen sich die Anhaftungen durch fleckartige Regionen beschreiben, die nach einer Zeit erscheinen und an einer Stelle verbleiben. Die zeitliche Charakteristik wird bereits in der Vorauswahl-Klassifikation verwendet – es verbleibt die Information der fleckartigen Struktur innerhalb der Bilder. Wie in Abbildung 5.2 zu sehen, weisen die (unterscheidbaren) Partikel visuelle Unterschiede in Form von Größe, Kontur und Ringstruktur in der Umgebung dieser Anhaftungen auf. Aus diesen bildhaften Ausprägungen in Form von Intensitätsverteilungen werden (klassifikatorspezifische) Merkmale extrahiert. Dieser Vorgang wird für die drei verwendeten Klassifikatoren in den folgenden Abschnitten jeweils beschrieben.

Jeder Klasse ω_l sei eine nichtleere Menge an Samples S_l zugeordnet. Das r-te Sample (von insgesamt n_l Samples) der Klasse ω_l sei dabei ein quadratischer Bildausschnitt $S_{l,r}(i,j)$ mit Kantenlänge d um den Mittelpunkt der Anhaftung im zeitreihengeglätteten Differenzbild (siehe Kapitel 3.3) zum Zeitpunkt nach dem Sprung. Der Grund für die Verwendung des

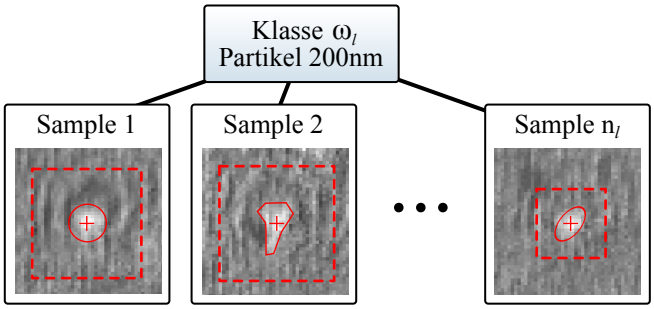


Abbildung 5.5.: Beispielhafte Zuordnung von n_l Samples zu einer Klasse ω_l. Für das Training zur Verfügung gestellt werden die Bildinformationen um eine zu definierende Mitte (Kreuz mit ovaler oder polygonaler Formbegrenzung) sowie eine (von der Mitte abhängigen) Umgebung der Region (hier als gestrichelte Box dargestellt).

Differenzbildes liegt in der differentiellen Dimension, die gegenüber räumlichen Überlagerungen von Anhaftungen (bei hinreichend großem zeitlichen Abstand) robust ist. Ein Experte hat zur Erstellung eines Samples zunächst die Bildpunkte zu bestimmen, welche das erwartete Zeitverhalten aufweisen. Danach wird eine Mitte der für ihn charakteristisch erscheinenden Region bestimmt, sowie eine Ausdehnung der Struktur. Die im PAMONO-Prozess auftretenden relevanten Regionen weisen eine Fläche von 5×5 (nur die helle Mitte ist sichtbar) bis 40×40 (mit großräumigen Ringstrukturen) Pixeln auf. Abbildung 5.5 zeigt beispielhaft die Zuordnung von Bildregionen zu einer Klasse. In den folgenden Abschnitten werden drei Klassifikatoren vorgestellt, welche diese Samples verwenden, um das Training durchzuführen.

Beim Training ist die explizite Definition einer „kein Fund“-Klasse ω_0 nicht vorgesehen, da sie sowohl von der Vorauswahl automatisch verwendet wird, als auch vom finalen Klassifikator als Label ausgegeben wird, wenn dort keine Übereinstimmung gefunden wurde. Genaue Definitionen dazu folgen in den nächsten beiden Abschnitten über den Bayes-Klassifizierer und die auf Template-Matching basierenden Klassifikatoren.

5.6. Bayes-Klassifikator

Der in der Projektgruppe eingesetzte Bayes-Klassifikator berechnet aus jedem Differenzbild-Sample mehrere Merkmale. Mit diesen Merkmalen werden für jede Klasse die Parameter einer mehrdimensionalen Normalverteilung geschätzt. Durch diese Normalverteilungen und die a-priori Wahrscheinlichkeiten der Klassen können neu beobachtete Samples klassifiziert werden.

Im folgenden Abschnitt werden kurz die Grundlagen des Bayes-Klassifikators erläutert. Eine genauere Darstellung der Grundlagen und Hintergründe des Bayes-Klassifikators findet sich in [Nie03].

5.6.1. Grundlagen

Der Bayes-Klassifikator basiert auf der Entscheidungsregel, dass ein Muster $\boldsymbol{c}$, also eine Bildregion mit sprunghaftem Intensitätsanstieg, der Klasse ω_k zugeordnet wird, für die $p(\omega_k)p(\boldsymbol{c}|\omega_k)$ maximal ist. Hierbei ist $p(\omega_k)$ die a-priori Wahrscheinlichkeit der Klasse und $p(\boldsymbol{c}|\omega_k)$ die klassenbedingte Dichte, also die Wahrscheinlichkeit, dass das Muster $\boldsymbol{c}$ beobachtet wird, wenn die Klasse ω_k vorgegeben ist.

Es gilt die Bayes-Regel für die a-posteriori Wahrscheinlichkeit:

$$p(\omega_k|\boldsymbol{c}) = \frac{p(\omega_k)p(\boldsymbol{c}|\omega_k)}{p(\boldsymbol{c})}. \tag{5.1}$$

Da $p(\boldsymbol{c})$ von ω_k unabhängig ist, wählt der Bayes-Klassifikator die Klasse, für welche die a-posteriori Wahrscheinlichkeit am größten ist. Bei gleichen a-priori Wahrscheinlichkeiten $p(\omega_k)$ erhält man den *Maximum-Likelihood-Klassifikator*, der ein Sample der Klasse zuweist, für die die klassenbedingte Dichte $p(\boldsymbol{c}|\omega_k)$ maximal ist.

Der hier beschriebene Bayes-Klassifikator nutzt als Wahrscheinlichkeitsverteilung die Normalverteilung. Die n-dimensionale Normalverteilung (Gauß-Verteilung) für eine Klasse k ist definiert als:

$$p(\boldsymbol{c}|\omega_k) = \frac{1}{\sqrt{|2\pi\boldsymbol{\Sigma}_{(k)}|}} e^{-\frac{1}{2}(\boldsymbol{c}-\boldsymbol{\mu}_{(k)})^{\mathsf{T}}\boldsymbol{\Sigma}_{(k)}^{-1}(\boldsymbol{c}-\boldsymbol{\mu}_{(k)})}, \tag{5.2}$$

mit dem Mittelwertvektor $\boldsymbol{\mu}$ und der Kovarianzmatrix $\boldsymbol{\Sigma}$.

Die Parameter $\boldsymbol{\mu}_{(k)}$ und $\boldsymbol{\Sigma}_{(k)}$ einer Klasse ω_k können mit den Maximum-Likelihood-Schätzern $\hat{\boldsymbol{\mu}}_{(k)}$ und $\hat{\boldsymbol{\Sigma}}_{(k)}$ aus der N-elementigen Stichprobe $\{\boldsymbol{c}_{(1)}, \boldsymbol{c}_{(2)}, \ldots, \boldsymbol{c}_{(N)}\}$ dieser Klasse ermittelt werden:

$$\hat{\boldsymbol{\mu}}_{(k)} \simeq \boldsymbol{\mu}_{(k)} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{c}_{(i)} \tag{5.3}$$

$$\hat{\boldsymbol{\Sigma}}_{(k)} \simeq \boldsymbol{\Sigma}_{(k)} = \frac{1}{N}\sum_{i=1}^{N} (\boldsymbol{c}_{(i)} - \boldsymbol{\mu}_{(k)})(\boldsymbol{c}_{(i)} - \boldsymbol{\mu}_{(k)})^{\mathsf{T}}. \tag{5.4}$$

5.6.2. Auswahl der Merkmale

Es gibt viele Möglichkeiten, aus den klassifizierten Samples Merkmale zu extrahieren. Eine einfache Methode ist es, das komplette Sample (ein $d \times d$ großes Differenzbild) als d^2-dimensionalen Merkmalsvektor zu benutzen. Das führt zu einem schwerwiegenden Problem: Die per Maximum-Likelihood-Methode geschätzte Kovarianzmatrix wird singulär und ist somit nicht mehr invertierbar, wenn weniger als d^2 Samples in der Stichprobe sind. Die invertierte Kovarianzmatrix $\boldsymbol{\Sigma}^{-1}$ wird jedoch für die Berechnung der Wahrscheinlichkeit gebraucht. Es

gibt zwei Lösungen für diese Problematik:

1. Es kann eine Dimensionsreduktion, zum Beispiel per Hauptkomponentenanalyse (PCA) durchgeführt werden, damit die Dimension des Merkmalsvektors kleiner als der Umfang der Stichprobe wird. Durch die PCA werden die Merkmale auf eine neue Basis abgebildet, die aus den sogenannten Hauptkomponenten besteht. Die Merkmalsdimension wird reduziert, indem die Merkmale als Linearkombination der aussagekräftigsten Hauptkomponenten genähert werden[2]. Auf welche Dimension reduziert wird, kann entweder fest vorgegeben oder adaptiv gewählt werden. Dieses Vorgehen wird oft dann gewählt, wenn die Charakteristiken der Samples unbekannt sind.

2. Es können aussagekräftige Eigenschaften manuell aus den Samples ausgewählt werden. Dies ist das bevorzugte Vorgehen wenn die Merkmale der Samples bekannt sind, und wird daher auch hier eingesetzt.

Die Differenzbilder angehafteter Partikel zeigen einen „hellen Fleck“ in der Mitte, der langsam nach außen abnimmt. Diese Charakteristik kann durch eine Normalverteilung modelliert werden. Abbildung 5.6 zeigt, wie die Parameter der 2D-Normalverteilung bestimmt werden. Davon ausgehend, dass der hellste Punkt in der Mitte des $d \times d$ großen Bildes liegt, werden die Intensitäten der mittleren Zeile $\boldsymbol{r}$ und die der mittleren Spalte $\boldsymbol{s}$ ausgewählt. Die Intensitätsverläufe werden durch 1D-Normalverteilungen approximiert. Dazu werden die kumulierten Summenvektoren $\boldsymbol{R} = (\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_d)^\mathsf{T}$ und $\boldsymbol{S} = (\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots, \boldsymbol{S}_d)^\mathsf{T}$ berechnet, mit

$$\boldsymbol{R}_k = \sum_{i=1}^{k} \boldsymbol{r}_i \quad \text{und} \tag{5.5}$$

$$\boldsymbol{S}_k = \sum_{i=1}^{k} \boldsymbol{s}_i \quad \text{für } k = 1, \ldots, d. \tag{5.6}$$

$\boldsymbol{R}$ und $\boldsymbol{S}$ entsprechen den Verteilungsfunktionen der Normalverteilung. Die Verteilungsfunktion $N(x)$ einer Normalverteilung nimmt für den Mittelwert μ den Wert $N(\mu) = 0{,}5$ an. Für die Standardabweichung gilt $N(\mu - \sigma) \approx 0{,}5 - 0{,}341$ und $N(\mu + \sigma) \approx 0{,}5 + 0{,}341$. Bei den kumulierten Summen $\boldsymbol{R}$ und $\boldsymbol{S}$ gibt es im Allgemeinen keinen exakten Treffer wie $\boldsymbol{R}_i = 0{,}5$. Daher wird i so gewählt, dass $\boldsymbol{R}_{i-1} < 0.5$ und $\boldsymbol{R}_i \geq 0{,}5$. Der Mittelwert wird dann durch lineare Interpolation als

$$\mu_r = \frac{0{,}5 - \boldsymbol{R}_i}{\boldsymbol{R}_i - \boldsymbol{R}_{i-1}} + i \tag{5.7}$$

berechnet. Analog werden $\sigma_{r,1}$ für $\boldsymbol{R}_i \approx 0{,}159$ und $\sigma_{r,2}$ für $\boldsymbol{R}_i \approx 0{,}841$ berechnet. $\sigma_{r,1}$ und $\sigma_{r,2}$ stimmen in realen Daten meist nicht überein, daher wird die Standardabweichung σ_r als Mittelwert von $\sigma_{r,1}$ und $\sigma_{r,2}$ gebildet:

$$\sigma_r = \frac{\sigma_{r,1} + \sigma_{r,2}}{2}. \tag{5.8}$$

Die Berechnung der Parameter μ_s und σ_s für $\boldsymbol{S}$ funktioniert gleichermaßen.

[2] Aussagekräftige Hauptkomponenten der PCA sind die Komponenten, die einen großen Anteil der Gesamtstreuung der Stichprobe enthalten.

Aus den Parametern der beiden 1D-Normalverteilungen werden folgendermaßen die Parameter für die 2D-Normalverteilung ermittelt:

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_r \\ \mu_s \end{pmatrix}, \quad \text{und} \quad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_r & 0 \\ 0 & \sigma_s \end{pmatrix}. \tag{5.9}$$

Um den Approximationsfehler zwischen der so geschätzten 2D-Normalverteilung und einem Sample zu berechnen, müssen die Intensitäten des Samples passend skaliert werden. Der Maximalwert des Samples wird der Normalverteilung angeglichen. Der **(quadratische) Approximationsfehler** für das skalierte Sample $X(x, y)$ und die Normalverteilung $N(y, x)$ wird berechnet als

$$\varepsilon = \sum_{i=1}^{d} \sum_{j=1}^{d} \left(X(i, j) - N(i, j) \right)^2. \tag{5.10}$$

Es werden fünf Merkmale aus den Samples extrahiert:

- Die Intensität an Position $\boldsymbol{\mu}$,
- der quadratische Abstand von $\boldsymbol{\mu}$ zum Mittelpunkt des Bildes,
- die Standardabweichung in horizontaler Richtung σ_r,
- die Standardabweichung in vertikaler Richtung σ_s und
- der quadratische Approximationsfehler.

Der Fokus des Bayes Klassifikators liegt auf der Unterscheidung der Klassen „Virus“ und „Kein Virus“. Die „Kein Virus“-Klasse wird mit zufälligen Werten angelernt. Wie Abbildung 5.2(c) zeigt, können auch ringhafte Strukturen um die Anhaftung herum auftreten. Diese Eigenschaft wird bei der Unterscheidung der Virus-Klassen nicht berücksichtigt. Stattdessen müssen sich die Virus-Klassen durch die Ausdehnung der Strukturen unterscheiden, die durch die Standardabweichungen mit einbezogen werden.

5.7. Template-Matching-Klassifikatoren

In den vorherigen Abschnitten dieses Kapitels ist deutlich geworden, dass sich die Anhaftungen verschiedener Partikel deutlich visuell unterscheiden und klassenspezifische, charakteristische Ausprägungen in den Bilddaten besitzen. Ein menschlicher Experte ist durch Erfahrung in der Lage, Anhaftungen durch ihre optische Erscheinung zu klassifizieren. Die Tatsache, dass die räumliche Verteilung der Intensitätswerte einer Region direkt zur Klassifizierung dienen kann, rechtfertigt die Verwendung von Template-Matching-Verfahren. Dabei handelt es sich um das Problem der Lokalisierung eines Referenzbildes (Templates) in einem Eingabebild. Es gilt, die Position des Templates auf der (x, y)-Ebene[3] mit der größten Übereinstimmung

[3] Hier sei nur der Fall von zweidimensionaler Translation betrachtet – es existieren Verallgemeinerungen des Verfahrens mit weiteren Freiheitsgraden bei der Suche der größten Übereinstimmung, beispielsweise die

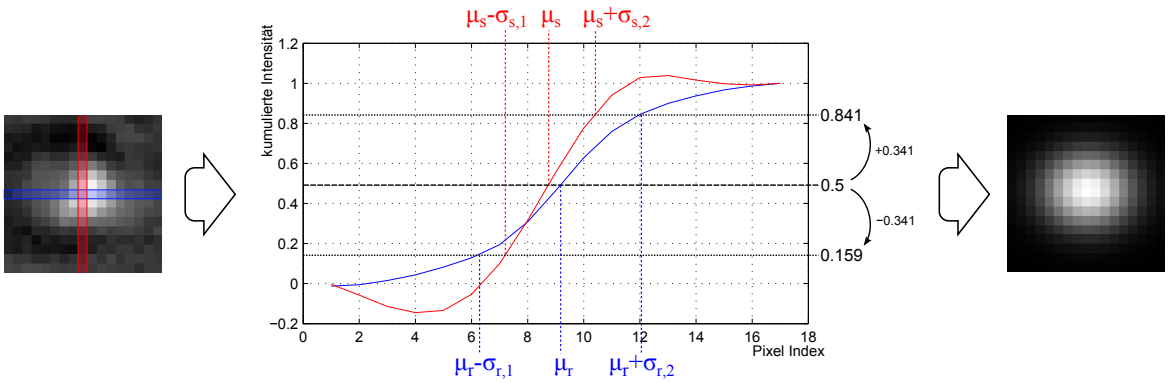


Abbildung 5.6.: Schätzung der Parameter einer 2D-Gauss-Verteilung aus einem Partikel-Sample. Der Mittelwert ist $\boldsymbol{\mu} = (\mu_r, \mu_s)^\mathsf{T}$ und die Kovarianzmatrix ist $\boldsymbol{\Sigma} = \mathrm{diag}\left((\sigma_{r,1} + \sigma_{r,2})/2, (\sigma_{s,1} + \sigma_{s,2})/2)\right)^\mathsf{T}$.

zum Eingangsbild zu finden [BB06]. Die Übereinstimmung des Referenzbildes wird durch ein Ähnlichkeitsmaß auf eine Zahl abgebildet, welche beim klassischen Ansatz in einer Ähnlichkeitsmatrix an den Koordinaten der Translation eingetragen wird. Dieses wird für alle möglichen Positionen durchgeführt, um anschließend mit Hilfe der vollständigen Ähnlichkeitsmatrix die wahrscheinlichsten Positionen mit größter Übereinstimmung angeben zu können. Dieser Ansatz soll in den folgenden Unterabschnitten verwendet werden, um Anhaftungen durch ihre bildhaften Ausprägungen zuvor definierten Klassen zuzuordnen.

5.7.1. Rigides Template-Matching

Zunächst sei das rigide Template-Matching mit mittelpunktbasierten Templates vorgestellt. Ausgehend davon, dass über ein Distanzmaß die Ähnlichkeit einer Region mit verschiedenen Templates gemessen werden kann, sei ein Klassifikator definiert, der für jede Klasse ω_l mit $1 \leq l \leq k$ eine quadratische Template-Matrix $R_l(i,j)$ der Größe $d \times d$ Pixel besitzt, mit $1 \leq i \leq d$, $1 \leq j \leq d$, wobei d ungerade sei. Die Ausrichtung auf die Mitte des Templates ist durch die Charakteristik der zu suchenden Strukturen begründet. So werden durch den Sprungindikator die hellen Bereiche in der Mitte markiert, deren Umgebung es zu untersuchen gilt. Die untersuchte Region ist symmetrisch um einen Bildpunkt (x, y) angeordnet und die Größe der Fläche ist vom Durchmesser[4] des Templates abhängig. Eine quadratische und achsenparallele Umgebung mit der Seitenlänge d um eine Position (x_0, y_0) im Eingabebild ist dabei wie folgt definiert:

$$(x, y) \in [x_0 - \lfloor d/2 \rfloor, x_0 + \lfloor d/2 \rfloor] \times [y_0 - \lfloor d/2 \rfloor, y_0 + \lfloor d/2 \rfloor] \tag{5.11}$$

Der Suchbereich für die Bestimmung der Ähnlichkeit ist offensichtlich durch diese Definition der Umgebung kleiner als das Bild $f(x, y)$. In Abbildung 5.7 findet sich eine anschauliche

zusätzliche Beachtung verschiedener Größen und Rotationen.

[4] Diese Symmetrie begründet, dass d ungerade sein muss, damit eine Mitte im Template existiert.

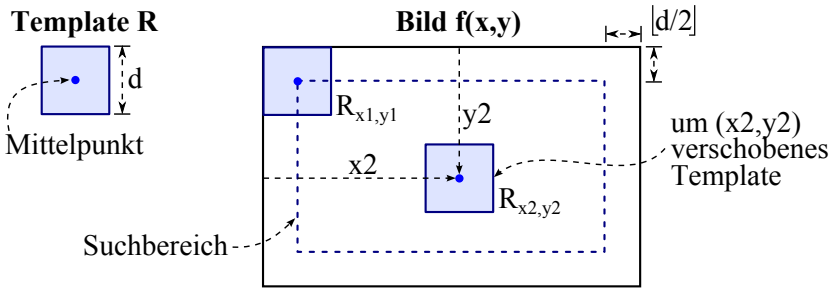


Abbildung 5.7.: Veranschaulichung des Suchbereiches eines mittelpunktbasierten Template-Matchings

Darstellung des Suchbereiches; der nicht behandelte äußere Rand hat eine Größe von $\lfloor d/2 \rfloor$.

Es gibt eine Vielzahl von Ansätzen, um Ähnlichkeits- und Abstandsmaße zweier Bildmuster zu definieren [BB06]. Hier werde ein Maß auf Basis des Betrags der Differenz verwendet. Der Ausdruck

$$d_{\text{abs}}(l,x,y) = \sum_{(i,j)\in R_l} \left\| f\left(x+i-\left\lceil\frac{d}{2}\right\rceil, y+j-\left\lceil\frac{d}{2}\right\rceil\right) - R_l(i,j)\right\| \tag{5.12}$$

bezeichne die Distanz eines Templates R_l von der gleichgroßen Region um eine Position (x,y) des Bildes $f(x,y)$, wobei $\left\lceil\frac{d}{2}\right\rceil \leq x \leq N - \left\lfloor\frac{d}{2}\right\rfloor$ und $\left\lceil\frac{d}{2}\right\rceil \leq y \leq M - \left\lfloor\frac{d}{2}\right\rfloor$.

Abbildung 5.8 zeigt das Ergebnis dieser Distanztransformation in Form einer Distanzmatrix, in der die Positionen mit geringer Distanz (entsprechend hoher Ähnlichkeit) mit kleinen Werten markiert sind (dunkle Bereiche).

Eigene Untersuchungen zeigen, dass verschiedene Anhaftungen der gleichen Klasse unterschiedliche Wertebereiche besitzen. Abbildung 5.9 zeigt die Analyse der Wertebereiche von drei unterschiedlichen Anhaftungen, die der gleichen Klasse zugeordnet werden sollen. Aus diesem Grund erscheint die Anwendung einer Normierung der Wertebereiche sinnvoll, da beim Template-Matching die Werte von Bild und Template über eine Differenz direkt miteinander verglichen werden.

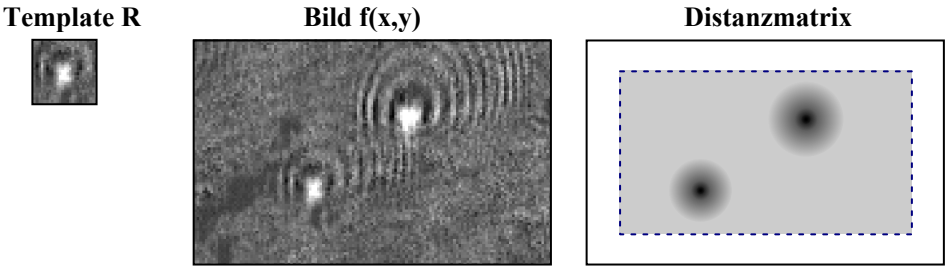


Abbildung 5.8.: Symbolisch dargestelltes Ergebnis einer Distanzmatrix eines Templates auf einem Bild. Dunkle Bereiche in der Matrix sind ein Indikator für kleine Abstände an der entsprechenden Stelle.

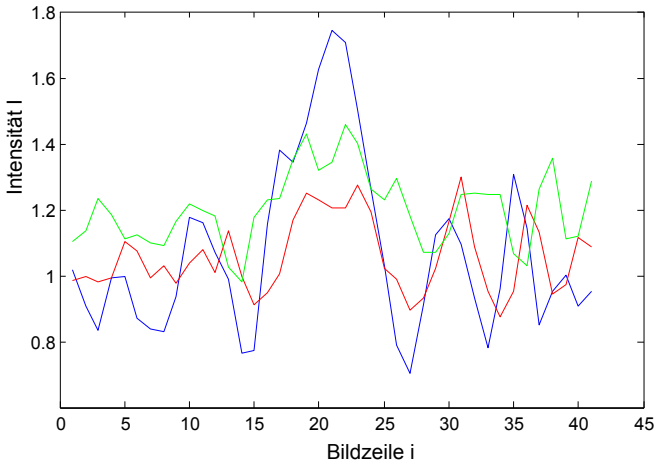


Abbildung 5.9.: Betrachtung der Intensitätsverteilung innerhalb einer mittigen Bildzeile von drei verschiedenen Anhaftungen derselben Klasse. Der Verlauf ähnelt sich, der Wertebereich weist aber unterschiedliche Skalierungen auf.

Um diese Normierung zu erreichen, werde eine Anpassung der statistischen Werte *Mittelwert* und *Standardabweichung*, ähnlich wie in Kapitel 3.4 (dort allerdings eindimensional) beschrieben, verwendet. Sei dazu der Mittelwert der Intensität eines Bildes $f(x, y)$ in einem symmetrischen Fenster der Größe d um eine Position (x, y) als

$$\mu(x,y) = \frac{1}{d^2} \sum_{i=1}^{d} \sum_{j=1}^{d} f\left(x + i - \left\lceil \frac{d}{2} \right\rceil, y + j - \left\lceil \frac{d}{2} \right\rceil\right) \tag{5.13}$$

definiert sowie die Standardabweichung der Intensitäten in jenem Fenster als

$$\sigma(x,y) = \sqrt{\frac{1}{d^2} \sum_{i=1}^{d} \sum_{j=1}^{d} \left[f\left(x + i - \left\lceil \frac{d}{2} \right\rceil, y + j - \left\lceil \frac{d}{2} \right\rceil\right) - \mu(x,y) \right]^2}. \tag{5.14}$$

Ziel ist die Normierung der Intensitätswerte in diesem Fenster im Bezug auf den Mittelwert (nach Normierung gleich 0) und die Standardabweichung (nach Normierung gleich 1). Dies wird durch folgende Transformation erreicht:

$$\widehat{f}_{(x,y)}(i,j) = \frac{f\left(x + i - \left\lceil \frac{d}{2} \right\rceil, y + j - \left\lceil \frac{d}{2} \right\rceil\right) - \mu(x,y)}{\sigma(x,y)}. \tag{5.15}$$

Die Definitionsbereiche von (x, y) entsprechen dabei denen der Distanzmatrix, für die Laufvariablen des normierten Bildausschnittes gelte $1 \leq i \leq d$ und $1 \leq j \leq d$. Um die Distanz des Templates von dem normierten Bildbereich zu ermitteln, wird die oben definierte Funktion verwendet in

$$d_{\text{norm}}(l,x,y) = \sum_{(i,j) \in R_l} \left\| \widehat{f}_{(x,y)}(i,j) - R_l(i,j) \right\|. \tag{5.16}$$

Im Folgenden wird beschrieben, wie das Verfahren mehrere Klassen unterscheidet und die für jede Klasse anzulegenden Templates R_l zu bestimmen sind. Zunächst sei die Erstellung der

Templates für eine Klasse diskutiert. Da alle Samples einer Klasse erkannt werden sollen, ist es sinnvoll, die Distanz des Templates zu allen Samples in der Klasse zu minimieren. Dies geschieht durch die Wahl eines punktweisen Mittelwertes. Um dies zu erreichen werde aus den zur Verfügung stehenden Samples aus Klasse ω_l (siehe Abschnitt 5.5) ein Template-Bild

$$R_l(i,j) = \frac{1}{n_l} \sum_{r=1}^{n_l} S_{l,r}(i,j) \tag{5.17}$$

gemittelt, wobei $1 \leq i \leq d$ und $1 \leq j \leq d$. Das Template ist mit der gleichen Methode statistisch zu normieren[5] wie der Bildausschnitt, um die Wertebereiche anzugleichen. Das Verfahren verläuft analog wie jenes zur Bestimmung von $\widehat{f}$.
Um eine Klassenzugehörigkeit zu bestimmen, welche auch eine Einteilung in die Ablehnungsklasse ω_0 beinhaltet, sei ein Schwellwert $\varepsilon_{\max}$ für die maximal akzeptierte Distanz eines Bildbereiches zu vorhandenen Templates definiert. Der Template-Matching-Klassifikator klassifiziere nach folgenden Regeln: Eine Region um (x, y) werde der Klasse ω_l zugeordnet, wenn $d_{\text{norm}}(l, x, y) < \varepsilon_{max}$ und die Distanz im Vergleich zu allen Templates den kleinsten Wert annimmt:

$$d_{\text{norm}}(l, x, y) < d_{\text{norm}}(l', x, y) \;\; \forall l \in \{1...k\} \;, \;\; l \neq l' \tag{5.18}$$

Ist die kleinste Distanz größer als der Schwellwert ε_{max}, so wird die Ablehnungsklasse ω_0 gewählt.

5.7.2. Statistisches Sigma-Template-Matching

Neben dem klassischen Template-Matching mit der Erweiterung um die Normierung der Wertebereiche zwischen Bild und Template soll eine weitergehende Abwandlung vorgestellt werden, welche auf dem Prinzip eines statistischen Hypothesentests basiert. In Abbildung 5.10 befinden sich mehrere Bilder, welche Ähnlichkeiten mit Partikelanhaftungen aufweisen. Angenommen, ein menschlicher Betrachter bekäme die Aufgabe, sich die acht Bilder als Ausprägung einer Klasse „Virus“ zu merken und davon zu lernen, um diese später in Bilddaten wiederzuerkennen. Das typische Vorgehen ist dabei die Suche nach Gemeinsamkeiten und Unterschieden zwischen den einzelnen gegebenen Samples. Die Gemeinsamkeiten spielen eine zentrale Rolle im Prozess der Abstraktion von Mustern, da diese eine *Informationskompression* ermöglichen. Bereiche, in denen die Samples wenig Gemeinsamkeiten besitzen (bei der PAMONO-Analyse die unterschiedlichen Ausprägungen der Ringstrukturen) werden dabei als „verschiedenartig“ eingeprägt. Dies hat zur Folge, dass dort bei der Entscheidung, ob ein „Fleck“ zur Klasse „Virus“ gehört, die Toleranz größer ist („Es können Ringe vorhanden sein“). In Bereichen mit Gemeinsamkeiten wird dagegen weniger Streuung akzeptiert („Die Mitte ist immer hell und rund“). Diese Erkenntnisse über die kognitive Abstraktion lassen sich auf eine Methode zur automatisierten Klassifikation übertragen.

Das sogenannte **Sigma-Template-Matching** greift diese Eigenschaft der unterschiedlich hohen Toleranz der Wertebereiche in Bildern als Kriterium zur Klassifikation auf und realisiert dies durch einen statistischen Hypothesentest auf einen erwarteten Intensitätsbereich an jedem

[5] Nach der statistischen Normierung des Templates ist der Mittelwert aller Pixel gleich 0, die Standardabweichung gleich 1.

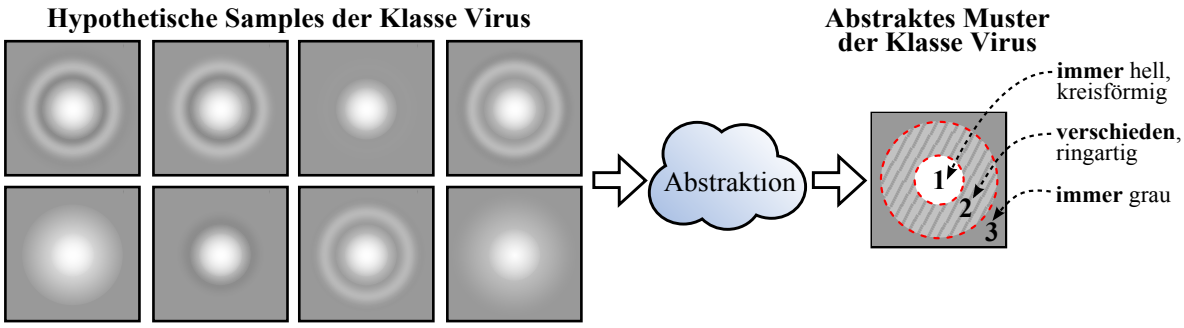


Abbildung 5.10.: Kognitive Abstraktion von gegebenen zusammengehörigen Bildern (links) zu einem komprimierten Muster (rechts). Die Bilder lassen sich in Bereiche mit Gemeinsamkeiten (Flächen 1 und 3) und Unterschieden (Fläche 2) einteilen.

Punkt in der Umgebung einer zu untersuchenden Region. Die Information über die Streuung der Bildbereiche lässt sich dabei automatisiert aus den Samples bestimmen. Dazu sei zunächst das Prinzip eines statistischen Hypothesentests näher erläutert. Die folgenden Ausführungen dazu sind an [Sac02] angelehnt.

Eine Hypothese trifft eine Aussage über die Verteilung von Zufallsvariablen[6] im Hinblick auf messbare Ausprägungen. Diese Aussagen können den Verteilungstyp betreffen oder konkrete Aussagen über Parameter einer Verteilung beinhalten. Beim Hypothesentest werden zwei Hypothesen voneinander abgegrenzt:

- Die sogenannte **Nullhypothese** H_0 sagt aus, dass die festgelegten Eigenschaften zutreffen, was dem „gewünschten“ Normalfall entspricht.
- Die **Alternativ-Hypothese** H_A sagt aus, dass die Aussagen nicht zutreffen. Dieser Fall wird als „problematisch“ angesehen.

Der Anwendungsfall für einen Hypothesentest ist hier der Test eines zweidimensionalen Bildbereichs auf eine erwartete Intensitätsverteilung, die sich aus trainierten Samples ergibt. Der Ansatz besteht darin, einen Test für jeden Bildpunkt durchzuführen und das Ergebnis anschließend auf die Gesamthypothese zu übertragen. Für jeden Bildpunkt ist der Test der erwarteten Intensität eindimensional und es gilt, Aussagen über die Verteilung der Intensitäten an einer Position zu definieren. Es wird dazu angenommen, dass die Intensitätswerte an einer festgelegten Position (x, y) über die Menge der Samples normalverteilt sind. Diese Annahme ist gerechtfertigt, da die Werte der Samples an der selben Position unabhängig voneinander, aber (idealerweise) identisch verteilt sind[7], wenn sie aus der gleichen Objektklasse ω_l stammen. Für solche eindimensionalen Probleme mit Normalverteilung können Tests mit Hilfe von Konfidenzintervallen konstruiert werden, welche ein akzeptierendes Intervall für eine Stichprobe x_I angeben. Sei H_0 für eine Position (x, y) die Annahme, dass die Werte in allen

[6] Grundlegende Bausteine der Stochastik und Statistik sind z.B. in [Vie03] zu finden.

[7] Die Summe und der Durchschnitt von n Zufallsvariablen, welche unabhängig voneinander sind und aus derselben Wahrscheinlichkeitsverteilung stammen, sind für $n \to \infty$ normalverteilt. Dies ist durch den zentralen Grenzwertsatz begründet [EKT05].

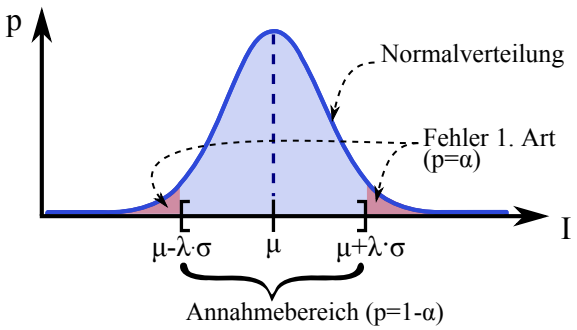


Abbildung 5.11.: Symmetrisches Konfidenzintervall der Breite $2\lambda \cdot \sigma$ um einen Mittelwert μ

Ausprägungen der Samples an jener Stelle $N(\mu, \sigma)$-verteilt[8] sind. Hier werde ein beidseitiger Test verwendet, da sowohl auf Abweichungen nach oben als auch nach unten getestet werden soll. Das Konfidenzintervall für Hypothese H_0 (Es werde eine $N(\mu, \sigma)$-Verteilung angenommen) bei einer Stichprobe x_I ist wie folgt definiert:

$$H_0 \text{ werde akzeptiert} \quad \Leftrightarrow \quad x_I \in [\mu - \lambda \cdot \sigma, \mu + \lambda \cdot \sigma]. \tag{5.19}$$

Der Annahmebereich ist durch ein symmetrisches Intervall um den Mittelwert μ gebildet, wobei die Intervallbreite durch die Standardabweichung σ und den Faktor λ beeinflusst wird. Die Intervallbreite definiert auch die Größe α des Fehlers erster Art (auch Signifikanzniveau genannt). Dieser bezeichnet die Wahrscheinlichkeit, die *wahre* Hypothese H_0 abzulehnen, obwohl diese zutrifft [Vie03]. Sei beispielsweise das Signifikanzniveau 5% (also $\alpha = 0{,}05$), so wird in 95% der Fälle die Hypothese H_0 richtigerweise akzeptiert ($p = 1 - \alpha = 0{,}95$). Ein solches Konfidenzintervall ist in Abbildung 5.11 dargestellt.

Der Zusammenhang von λ und α ist Tabellen mit Normalverteilungsquantilen zu entnehmen (z.B. in [Geo09]), beispielsweise ergibt sich für ein Signifikanzniveau von 95% ein Wert für λ von 1,96. Sei zunächst das „Training“ der Samples und die Bestimmung der Konfidenzintervalle beschrieben. Analog zum rigiden Template-Matching (siehe Abschnitt 5.7.1) wird bei diesem Verfahren auch die Umgebung um eine Position (x, y) untersucht. Dabei müssen pro Position im Template ein Wert für den Mittelwert und die Standardabweichung bestimmt werden. Das Signifikanzniveau α sei dabei ein globaler Parameter für alle Intervalle. Für jede Klasse ω_l seien zwei Matrizen der Größe $d \times d$ definiert. Dabei sei $R_{l,\mu}(i, j)$ die **Mittelwertmatrix** und $R_{l,\sigma}(i, j)$ die **Standardabweichungsmatrix** mit jeweils $1 \leq i \leq d$, $1 \leq j \leq d$ und d ungerade (analog zum rigiden Template-Matching). Dabei werden diese Verteilungsparameter für jede Position (i, j) der Template-Umgebung mit

$$R_{l,\mu}(i, j) = \frac{1}{n_l} \sum_{r=1}^{n_l} S_{l,r}(i, j) \tag{5.20}$$

[8] Für eine Normalverteilung mit Erwartungswert μ und Standardabweichung σ kann abkürzend $N(\mu, \sigma)$ verwendet werden.

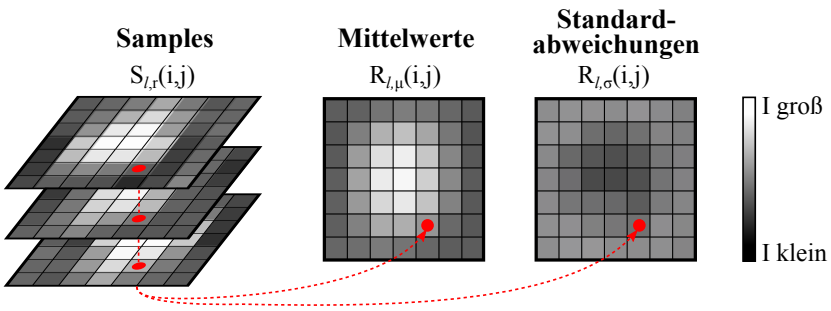


Abbildung 5.12.: Training der Mittelwert- und Standardabweichungsmatrix

und

$$R_{l,\sigma}(i,j) = \sqrt{\frac{1}{n_l}\sum_{r=1}^{n_l}\left(S_{l,r}(i,j) - R_{l,\mu}(i,j)\right)^2} \tag{5.21}$$

durch Maximum-Likelihood-Schätzer bestimmt [BBK08]. Das Tupel $\langle R_{l,\mu}(i,j), R_{l,\sigma}(i,j)\rangle$ werde dabei als **Konfidenzintervall-Template** der Klasse ω_l bezeichnet. Diese Berechnung ist in Abbildung 5.12 dargestellt. Dabei muss die Anzahl n_l der Samples pro Klasse mindestens zwei betragen, da sonst keine Standardabweichung geschätzt werden kann[9]. Dabei sind die Bildausschnitte der Samples wie beim rigiden Template-Matching statistisch zu normieren, um Wertebereiche vergleichen zu können. Die Mittelwertmatrix entspricht dabei dem Template R_l des rigiden Template-Matchings.

Die Distanz eines Konfidenzintervall-Templates zu einem Bildbereich um einen Punkt (x, y) sei auf folgende Weise zu ermitteln. Der zu untersuchende Bildausschnitt aus $f_{(x,y)}(i,j)$ mit $\left\lceil \frac{d}{2} \right\rceil \leq x \leq N - \left\lfloor \frac{d}{2} \right\rfloor$ und $\left\lceil \frac{d}{2} \right\rceil \leq y \leq M - \left\lfloor \frac{d}{2} \right\rfloor$ sowie $1 \leq i \leq d$ und $1 \leq j \leq d$ werde zwecks Normierung mit dem gleichen Verfahren wie beim rigiden Template-Matching normiert und habe fortan die Bezeichung $\widehat{f}_{(x,y)}(i,j)$. Es wird punktweise die Distanz zwischen erwartetem Mittelwert und der Intensität des normierten Bildausschnitts $\widehat{f}_{(x,y)}(i,j)$ gemessen:

$$d_{\mathrm{abs}(l,x,y)}(i,j) = \left\| \widehat{f}_{(x,y)}(i,j) - R_{l,\mu}(i,j) \right\|. \tag{5.22}$$

Diese absolute Distanz wird anschließend relativ zur Breite des Konfidenzintervalls an dieser Stelle betrachtet:

$$d_{\mathrm{rel}(l,x,y)}(i,j) = \frac{d_{\mathrm{abs}(l,x,y)}(i,j)}{\lambda \cdot R_{l,\sigma}(i,j)}. \tag{5.23}$$

So lässt sich für jede Position (i, j) innerhalb des Templates ein Hypothesentest durchführen. Die Hypothese H_0 besagt, dass der Intensitätswert an jener Stelle im erwarteten Intervall liegt. Dies ist offenbar erfüllt, wenn der relative Fehler $d_{\mathrm{rel}(l,x,y)}(i,j)$ kleiner oder gleich 1 ist. Für Werte größer eins (Mehr als 100% Fehler im Bezug auf die halbe Intervallbreite), ist die Hypothese für diese Position abzulehnen. Abbildung 5.13 visualisiert die Berechnung der Differenzen zu den Konfidenzintervallen einer Bildzeile.

Da das Template aus d^2 Punkten besteht, können einige davon innerhalb und der Rest außerhalb

[9] Wie bei allen Lernverfahren ist für das Training tendenziell ein Stichprobenumfang von mehr als zwei sinnvoll.

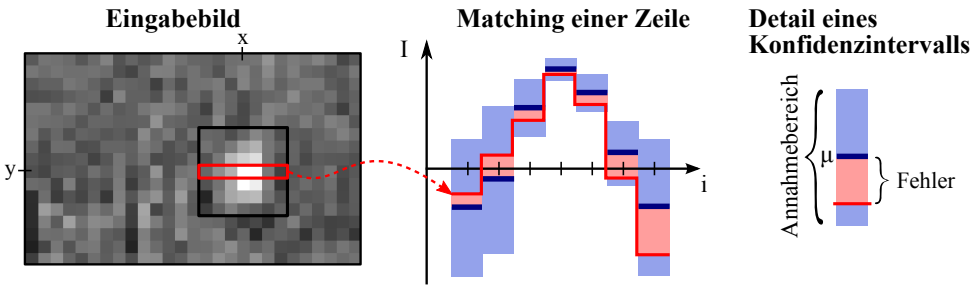


Abbildung 5.13.: Beispielhafter Verlauf eines Matchings einer Zeile. Der Fehler pro Intervall wird dabei relativ zur Intervallbreite betrachtet.

des jeweiligen Intervalls liegen. Es gilt, eine Vorschrift zur Bewertung der Gesamtheit aller Konfidenztests zu finden. Dafür hat sich eine Gewichtung der relativen Differenzen als sinnvoll herausgestellt. Zu große Abweichungen sollen mit Hilfe eines Exponenten $\gamma \geq 1$ bestraft werden:

$$d_{\mathrm{eval}(l,x,y)}(i,j) = \left[d_{\mathrm{rel}(l,x,y)}(i,j)\right]^{\gamma} \tag{5.24}$$

Abbildung 5.14 zeigt den Effekt der Bestrafung. Je größer die Potenz γ gewählt wird, desto größer wird die Differenz bei Ausreißern ($d_{\mathrm{rel}} > 1$) gewichtet.

Sei die Bewertungsfunktion als der Mittelwert aller mit Gewichtung versehenen relativen Distanzen definiert als

$$d_{\mathrm{norm}}(l,x,y) = \frac{1}{d^2}\sum_{i=1}^{d}\sum_{j=1}^{d} d_{\mathrm{eval}(l,x,y)}(i,j)\ . \tag{5.25}$$

Weiter sei ein Schwellwert $\varepsilon_{\max}$ definiert, der den maximalen Wert der mittleren relativen Abweichungen beschreibt. Hier wird dieselbe Grenze von $\epsilon_{\max} = 1$ wie für einen einzelnen Punkt gewählt. Der Fall $d_{\mathrm{norm}}(l,x,y) \leq \epsilon_{\max}$ bedeutet anschaulich, dass die Intensitätswerte in der Umgebung um (x,y) im Mittel (und mit eventuell nichtlinearer Bestrafung $\gamma > 1$) in den erwarteten Intervallen des gewählten Signifikanzniveaus liegen. Die Hypothese H_0 wird akzeptiert. Trifft dies für mehrere Klassen zu, so wird die Klasse mit der kleinsten mittleren Distanz als Klassifizierungsergebnis gewählt. Die quadratische Region um eine Position (x,y)

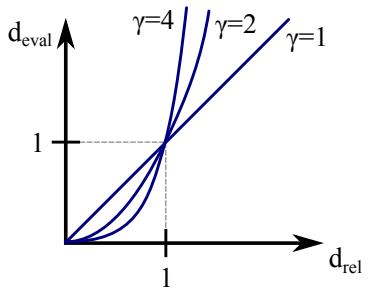


Abbildung 5.14.: Auswirkungen der Bestrafungspotenz γ auf die Differenzbewertung von d_{rel}. Die Potenzierung des relativen Fehlers d_{rel} mit $\gamma > 1$ bewirkt dabei bei $d_{rel} > 1$ eine überproportionale „Bestrafung“ in der Bewertung d_{eval}.

wird dabei der Klasse ω_l zugeordnet, wenn

$$d_{\text{norm}}(l, x, y) < d_{\text{norm}}(l', x, y) \;\; \forall l \in \{1...k\} \; , \;\; l \neq l'. \tag{5.26}$$

Wenn die Hypothesen von allen Klassen $\omega_1, \ldots, \omega_k$ verworfen werden, so werde das Klassenlabel „kein Fund“ (ω_0) ausgegeben. Dieser Klassifikator besitzt insgesamt zwei Parameter – die Höhe des Signifikanzniveaus α (welche die Intervallbreiten beeinflusst) und den Bestrafungsfaktor γ, welcher Regionen, die nicht der Klassenvorlage entsprechen, überproportional in der Fehlersumme gewichtet. Verringert sich das Signifikanzniveau α, so verringert sich auch das Risiko, Bildregionen mit (korrekten) Anhaftungen wegen zu großen Distanzwerten zu verwerfen (*false negatives*). Allerdings werden dadurch auch unter Umständen Störungen als positive Funde deklariert (*false positives*). Eine Untersuchung der Leistungen des Klassifikators findet sich in Kapitel 10.

Durch die verwendeten statistischen Modelle mit adaptiven Akzeptanzbereichen kann die Idee der „erlernten Unschärfe“, wie sie in der Motivation beschrieben wurde, umgesetzt werden. Die Tatsache, dass die Standardabweichung einen Einfluss bei der Bewertung hat, hat die Namensgebung des Verfahrens (*Sigma*-Template-Matching) geprägt. Die Normierung der Wertebereiche sorgt zusätzlich für eine Erkennung von „ähnlich“ geformten Bildregionen, wenn die relative Verteilung der Intensitätswerte dem vorgegebenen Muster entspricht. Zusammenfassend gesehen verwendet das Sigma-Template-Matching eine ähnliche Erkennungsstrategie wie ein menschlicher Betrachter.

6. Stream-Verarbeitung

In den vorherigen Kapiteln wurden Techniken besprochen, mit denen Bilddaten verarbeitet und analysiert werden können. Der Fokus dieses Kapitels liegt auf Verfahren, mit denen diese Berechnungen effizient ausgeführt und parallelisiert werden können. Dazu wird in Kapitel 6.1 auf die grundlegende Problematik und die daraus entstehenden Herausforderungen eingegangen. In Teil 6.2 werden Threads beschrieben. Dabei handelt es sich um Teile eines Programms, die parallel ausgeführt werden können. Die Kommunikation zwischen Threads per Shared Memory und die Probleme bei der Synchronisation der Daten werden in Kapitel 6.3 erläutert. Die gewonnenen Erkenntnisse führen zu der in Kapitel 6.4 beschriebenen Pipeline-Architektur, die eine parallele Verarbeitung des Datenflusses erlauben.

6.1. Einleitung

Das Pamono-Verfahren erzeugt umfangreiche Bilddatensätze mit mehreren Gigabyte Daten und einigen tausend Einzelbildern[1]. Daher ist eine effiziente Verarbeitung für die schnelle Analyse der Daten wichtig.

Die Bildreihen werden mit den in den vorherigen Kapiteln beschriebenen Verfahren analysiert. Die Folge der einzelnen Operationen wird sequentiell abgearbeitet, wie Abbildung 6.1 darstellt. Jedes Bild des Datensatzes durchläuft dieselben Verarbeitungsschritte: Bildverarbeitung, Zeitreihenanalyse, Segmentierung und Klassifikation. Anstatt an jedem Zeitpunkt nur ein Bild zu verarbeiten, kann die Analyse ähnlich der Fließbandfertigung in der Industrie parallelisiert werden: Die Elemente der Verarbeitungskette arbeiten parallel an je einem Bild. Sobald ein Element ein Bild fertig verarbeitet hat, gibt es dies weiter und fängt mit der Verarbeitung des nächsten Bildes an. Die gleichzeitige Ausführung der Elemente ermöglicht eine hohe Verarbeitungsgeschwindigkeit. Dies wird als *Stream-Verarbeitung* bezeichnet.

Parallele Berechnungen können technisch durch Prozesse oder Threads realisiert werden. Eine Schwierigkeit ist, den Zugriff auf gemeinsam genutzte Daten von gleichzeitig ablaufenden Prozessen zu synchronisieren und die Kommunikation der Prozesse zu ermöglichen.

[1] In Kapitel 10.1 sind mehrere Datensätze genau beschrieben.

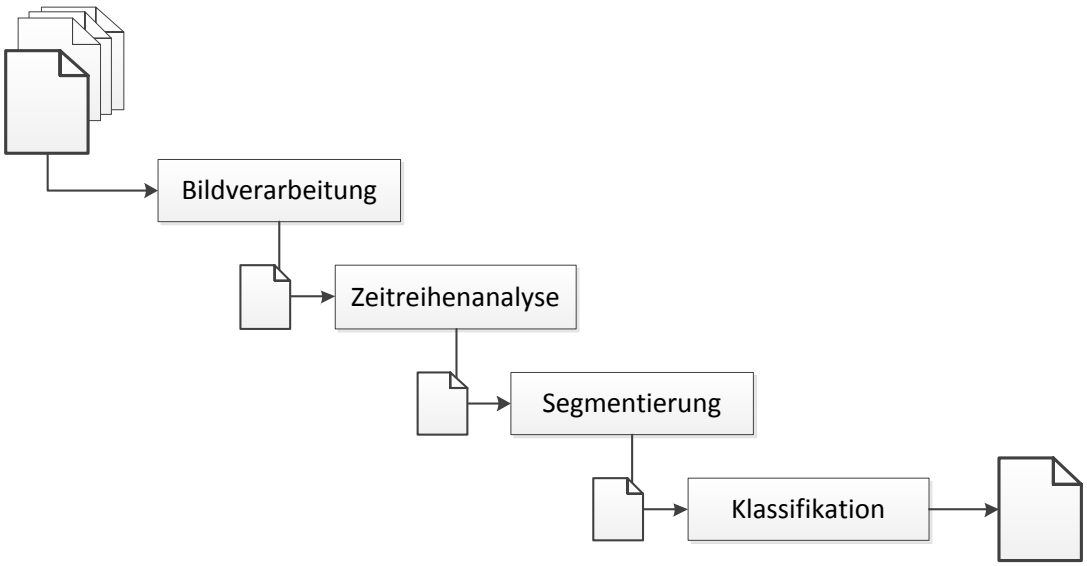


Abbildung 6.1.: Sequentielle Verarbeitung von Daten in der Pipeline

6.2. Threads

Um eine parallel arbeitende Verarbeitungskette, wie sie im vorangegangenen Abschnitt 6.1 thematisiert wurde, zu realisieren, kann auf Prozesse oder Threads zurückgegriffen werden. Beide Konzepte werden heute von den meisten Betriebssystemen unterstützt.

Jeder Prozess bekommt vom Betriebssystem seinen eigenen Speicherbereich zugeteilt und besitzt mindestens einen Thread. Verschiedene Prozesse greifen also auf unterschiedliche Speicherbereiche zu. Dabei überwacht das Betriebssystem, dass ein Prozess nicht auf den Speicherbereich eines anderen Prozesses zugreift. Ein Thread hingegen kann nur im Kontext eines Prozesses erstellt werden und benutzt den gleichen Speicherbereich wie der übergeordnete Prozess. Jeder Thread führt jedoch eigenen Programmcode aus, deshalb besitzt jeder Thread einen Programmzähler, eigene Register, einen Stack und einen Zustand (laufend, blockiert, bereit oder terminiert) [Tan07]. Besitzt ein Prozess mehrere Threads, so arbeiten alle Threads auf einem gemeinsamen Speicherbereich. Damit lässt sich die Kommunikation zwischen Threads desselben Prozesses über gemeinsam genutzten Speicher abwickeln. In Parallsystemen mit gemeinsamem Speicher kann dieser zur Kommunikation der einzelnen Prozesse bzw. Threads genutzt werden. Für den Programmierer bedeutet dies eine Vereinfachung, da ein Nachrichtenaustausch nicht mehr explizit implementiert werden muss. Außerdem kann der Speicher effizienter genutzt werden, da keine Daten repliziert werden müssen [RR07].

6.3. Interprozesskommunikation

Gleichzeitig laufende Prozesse oder Threads müssen miteinander kommunizieren, wenn sie kooperieren sollen. Die Probleme der sogenannten *Interprozesskommunikation* können in zwei Gebiete eingeteilt werden [SGG08]:

- Kommunikation (Übertragung von Daten) und
- Synchronisation (Regelung des Zugriffs auf gemeinsame Daten).

6.3.1. Kommunikation

Damit Prozesse miteinander interagieren können, müssen sie Daten und Informationen austauschen. Dies kann durch das Austauschen von Nachrichten funktionieren, das sogenannte *Message Passing*. Message Passing ist eine zuverlässige, gegenüber anderen Methoden jedoch langsame Kommunikationsmethode, da die Nachrichten mithilfe des Betriebssystemkerns verteilt werden.

Eine schnellere Methode zur Kommunikation zwischen Prozessen/Threads ist die Nutzung eines gemeinsamen Speichers (*Shared Memory*). Threads innerhalb eines Prozesses teilen sich den gleichen Speicherbereich. Shared Memory ist für sie daher die bevorzugte Technik zum Datenaustausch. Der Speicher- und Adressraum von Prozessen ist im Allgemeinen getrennt. Laufen die Prozesse auf dem gleichen lokalen System kann Shared Memory dennoch eingesetzt werden. Für örtlich getrennte, verteilte Systeme bieten sich andere Kommunikationsmethoden an, die in [SGG08] und [Tan07] beschrieben sind.

Der Umstand, dass bei einem Shared Memory mehrere Prozesse gleichzeitig auf die gleichen Daten zugreifen können, kann zu Fehlern bei den Berechnungen führen. Der Schutz von Daten, die gerade bearbeitet werden, vor dem Zugriff durch andere Prozesse ist Aufgabe der Synchronisation.

6.3.2. Synchronisation

Wie vorangehend beschrieben, kann der Einsatz von mehreren Prozessen (oder auch Threads), die auf einen gemeinsamen Speicher zugreifen, zu einer Inkonsistenz des Speichers führen. Das Problem kann an einem einfachen Beispiel aufgezeigt werden. Seien A und B zwei Prozesse, die während ihrer Ausführung eine gemeinsam genutzte Integer-Variable v um 1 erhöhen möchten. Prozess A führt den Code zuerst aus und liest dabei den Wert in v. Anschließend kommt, entweder weil A und B parallel laufen oder etwa durch einen *Interrupt*, Prozess B zum Zuge. B liest ebenfalls den Wert in v und speichert sofort den um 1 erhöhten Wert in v wieder ab. Schließlich speichert auch A den um 1 erhöhten Wert in v ab. A hat jedoch nicht den von B schon erhöhten Wert gelesen und nochmals erhöht, sodass v insgesamt nur um 1 erhöht wurde. Dieses Problem wird als *Race Condition* bezeichnet und ein Code-Abschnitt, in

dem eine Race Condition auftreten kann, als *kritischer Bereich*.

Eine Möglichkeit um *Race Conditions* zu vermeiden und den Speicher konsistent zu halten, sind *Semaphore*. Darunter ist im Wesentlichen eine Integer-Variable zu verstehen, die mit $n > 0$ initialisiert und für jeden Prozess, der auf die begrenzt verfügbare Ressource zugreift, um 1 verringert wird. Indem jeder Prozess vor Betreten des kritischen Bereichs prüft, ob der Wert der Semaphore größer als 0 ist, kann sichergestellt werden, dass zu jedem Zeitpunkt immer höchstens n Prozesse auf die kritische Ressource zugreifen. Gibt ein Prozess die Ressource wieder frei, so wird die Semaphore um 1 erhöht. Doch ohne Hardware-Unterstützung würde auch die Synchronisierung mit einer Semaphore nicht funktionieren, denn es würde zu denselben Race Conditions kommen wie im Beispiel beschrieben. Um dieses Problem zu lösen bieten Prozessoren den `TSL`-Befehl (Test and Set Lock) [Tan07] an. Dieser Befehl stellt sicher, dass Operationen auf der Semaphore nicht unterbrechbar sind und verhindert so Race Conditions.

Der klassische wechselseitige Ausschluss (*Mutual Exclusion*), wo zu jeder Zeit nur ein Prozess einen kritischen Bereich betreten darf, ist der Spezialfall einer Semaphore, die für „kritischer Abschnitt ist frei“ maximal den Wert 1 und für „kritischer Abschnitt ist nicht frei“ den Wert 0 annimmt (binär). Eine binäre Semaphore wird auch als „Mutex“ bezeichnet.

6.4. Pipeline-Architektur

Die in der Projektgruppe erarbeitete Pipeline-Architektur ermöglicht die Stream-Verarbeitung der PAMONO-Daten über die Verkettung von parallel arbeitenden Verarbeitungselementen. Es gibt drei verschiedene Elementarten:

Normales Element Ein normales Element der Pipeline besitzt genau ein Vorgängerelement und ein oder mehrere Nachfolgerelemente.

Source Ein Source-Element ist der Anfang einer Pipeline. Es besitzt keinen Vorgänger und ein oder mehrere Nachfolger. Eine Pipeline hat genau ein Source-Element.

Sink Ein Sink-Element ist das Ende einer Pipeline. Es besitzt genau einen Vorgänger und keinen Nachfolger.

Es werden zwei Klassen von Pipelines unterschieden: Einfache Pipelines und komplexe Pipelines. Eine **einfache Pipeline** ist eine verkettete Liste von Pipeline-Elementen. Normale Elemente und Source-Elemente haben genau einen Nachfolger, sodass es nur einen Datenfluss gibt. Abbildung 6.2(a) stellt eine einfache Pipeline dar.
Bei einer **komplexen Pipeline** können normale Elemente und Source-Elemente mehr als einen Nachfolger haben. Aufspaltungen des Datenflusses sind möglich, sodass die Pipeline eine eine baumartige Struktur besitzt. Abbildung 6.2(b) zeigt eine komplexe Pipeline.

Pipeline-Elemente müssen parallel ausgeführt werden und können daher als Threads mit Shared Memory realisiert werden. In einer einfachen Pipeline können die Daten per Referenz auf den Speicherbereich weitergegeben werden. Das bedeutet bei großen Datenobjekten wie

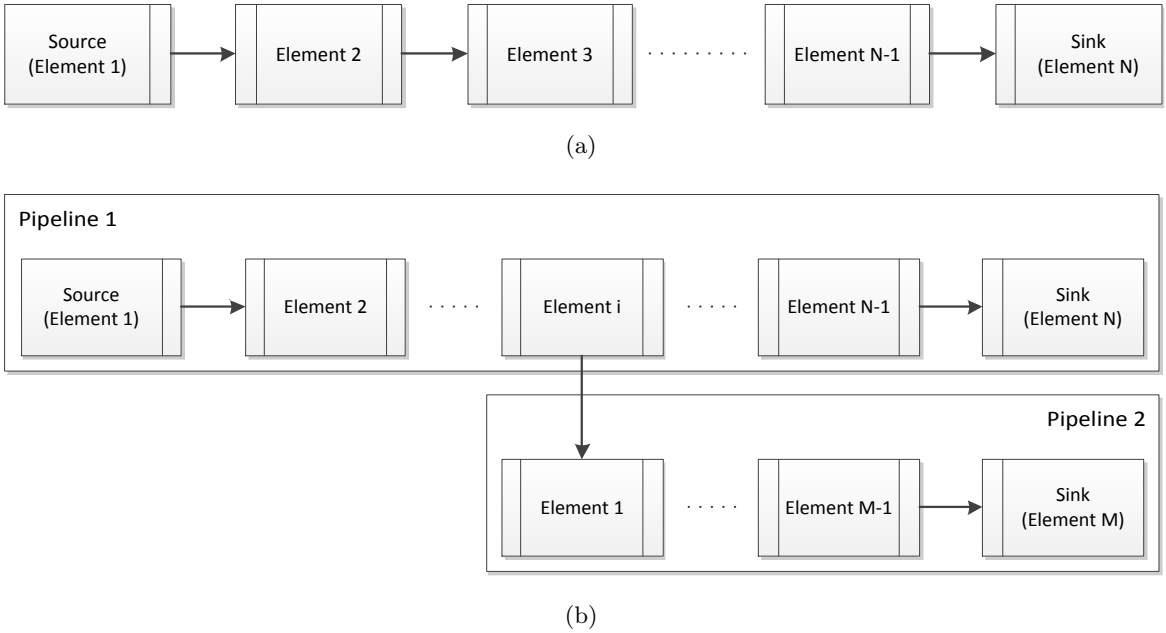


Abbildung 6.2.: In einer einfachen Pipeline (a) hat jedes Element genau 0 oder 1 Vorgänger und Nachfolger. Bei der komplexeren Variante (b) können Elemente mehrere Nachfolger besitzen, wodurch eine baumartige Pipeline-Struktur entsteht.

Bildern einen Geschwindigkeitsvorteil, da die Daten nicht kopiert werden müssen. Bei einer komplexen Pipeline müssen die Daten an jeder Abzweigung kopiert werden. Ansonsten treten Inkonsistenzen bei der Berechnung auf, wenn zwei Pipeline-Zweige auf denselben Daten arbeiten.

Im Anfangszustand wartet ein Pipeline-Element auf Daten, die vom Vorgänger bereitgestellt werden. Sobald diese bereitstehen, holt sich das Element die Daten und fängt an, sie zu verarbeiten. Ist die Verarbeitung abgeschlossen werden die Nachfolgerelemente benachrichtigt. Anschließend wird gewartet, bis alle Nachfolger die Daten abgeholt haben. Erst dann kann das Element auf neue Daten warten. Ein Pipeline-Element kann somit immer nur ein Datenobjekt gleichzeitig verarbeiten. Abbildung 6.3 veranschaulicht diesen Ablauf.

Im Gegensatz zu zeitkritischen Systemen, bei denen es eine Grenze für die Verarbeitungszeit der Pipeline gibt, werden bei dieser Pipeline-Architektur keine Daten verworfen. Die Pipeline arbeitet also blockierend und immer nur so schnell, wie das langsamste Element.

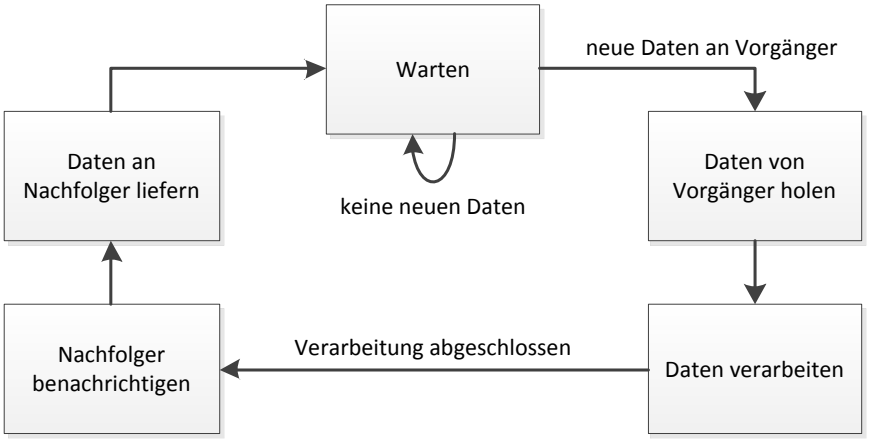


Abbildung 6.3.: Der interne Ablauf in einem Pipeline-Element. Liegen am Vorgängerelement neue Daten bereit, so werden diese geholt, verarbeitet und der Nachfolger benachrichtigt. Sobald dieser die Daten abholt wird auf neue Daten gewartet.

7. Smart Camera

Dieses Kapitel beschäftigt sich mit der im Rahmen der Projektgruppe verwendeten Smart Camera, einer Elphel NC353L. Innerhalb dieses Kapitels werden dabei die Anforderungen, die im Rahmen des Projekts an die Funktionalität der Kamera gestellt worden sind, beschrieben. Desweiteren werden der Hardware-Aufbau und die in den programmierbaren Logikbausteinen (FPGAs) realisierten Funktionen sowie die eigenen Erweiterungen erläutert. Kapitel 7.1 ordnet die Kamera in den Verarbeitungsfluss des Gesamtsystems ein und beschreibt ihre Rolle als Quell-Element in der Pipeline. Aus der Einordnung in den Verarbeitungsfluss und die zu erreichenden Ziele innerhalb der PG ergeben sich Anforderungen an die zu leistenden Arbeiten der Kamera, auf die in Kapitel 7.2 eingegangen wird. Die als Ausgangspunkt für die Arbeiten dienende Smart Camera wird in ihrer Struktur und dem inneren Aufbau in Kapitel 7.3 beschrieben. Hier zeigen sich Ansatzpunkte, an denen sich die Funktionalität erweitern lässt. In Kapitel 7.4 wird detailliert darauf eingegangen, an welchen Stellen im Signalfluss zusätzliche Module integriert werden können. In Kapitel 7.5 wird die Umsetzung der Funktionserweiterungen der Kamera erläutert, wie sie in Kapitel 7.2 erarbeitet wurden.

7.1. Die Smart Camera als Quell-Element der Pipeline

Wie im Datenflussmodell in Abbildung 7.1 dargestellt wird, ist die Smart Camera eine mögliche Datenquelle für die Verarbeitungspipeline der Analysesoftware, wie sie in Kapitel 6.4 beschrieben wird. Die vom Sensor aufgenommenen Bilder werden JPEG-komprimiert und können über den Ethernet-Anschluss per Stream abgerufen werden. Als Streaming-Protokoll wird das RTSP eingesetzt und dient dazu, den Datenstrom von der Kamera zu steuern. Zur Realisierung der Steuerung sind typische Befehle wie *stop*, *play* und *pause* implementiert. Für weiterführende Informationen über das RTSP wird auf [IET98] verwiesen. Der Bilddatenstrom

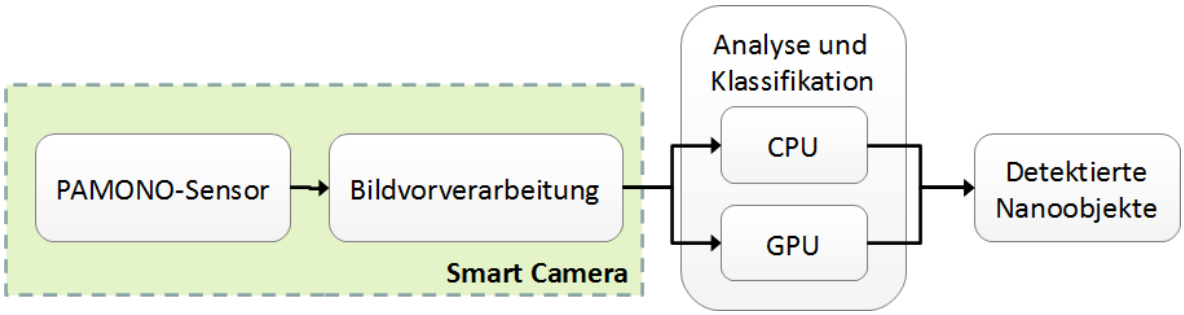


Abbildung 7.1.: Einordnung der Smart Camera in die Verarbeitungspipeline

selbst ist ein Motion-JPEG-Stream (MJPEG). Der Stream besteht dabei aus der Konkatenation der einzelnen JPEG-Bilddaten, jeweils unterbrochen durch einen Header.

7.2. Anforderungen an die Kamerafunktionalität

Da die Kamera als eigenständiges Gerät vorhanden und durch eine Ethernet-Schnittstelle mit dem Auswertungs-PC verbunden ist, muss eine Schnittstelle zur Steuerung und Statusübermittlung zwischen Analyse-Software und Kamera geschaffen werden. Welche Anforderungen an diese Schnittstelle gestellt werden und wie diese Schnittstelle realisiert worden ist, ist in Kapitel 7.2.1 beschrieben. Die Kamera nimmt durch ihre Eigenschaft als Quell-Element in der Verarbeitungspipeline eine wichtige Rolle hinsichtlich der Menge der zu verarbeitenden Daten ein. Durch die Menge der Daten, die von der Kamera erzeugt werden, wird maßgeblich die Grundlast auf der gesamten Pipeline bestimmt. Es ist daher wünschenswert, die Menge der von der Kamera erzeugten Daten möglichst gering zu halten. Aus diesem Grund sind im Rahmen der Projektgruppe mehrere Möglichkeiten zur Datenreduktion mit ihrer Eignung zur Umsetzung auf dem eingebetteten FPGA untersucht worden.

Wenn von der Kamera viele Bilder pro Sekunde erzeugt werden und diese jeweils eine große Datenmenge beanspruchen, sind alle Elemente der Verarbeitungspipeline entsprechend stark ausgelastet. Zur Verminderung der Last wäre es dann notwendig, auf dem PC in der Pipeline ein Filter-Element einzusetzen, das für die Analyse irrelevante Informationen entfernt. Dies bedeutet aber zusätzliche Rechenlast und bedarf einer Verbesserung, da die Bilder bereits bei ihrer Entstehung in der Kamera reduziert werden können. In der Kamera ist ein FPGA vorhanden, der im Auslieferungszustand der Kamera mitunter die Kompression und eine allgemeine Vorverarbeitung der Bilddaten vornimmt, sodass es nahe liegt, diesen FPGA ebenfalls zur Reduktion der über die Ethernet-Schnittstelle übertragenen Daten einzusetzen. Als Möglichkeit zur Reduktion der zu übertragenen Daten ist die Subtraktion des Hintergrundes eruiert worden. Die Grundlagen hierzu sind in Kapitel 2.3.1 beschrieben. Da sich FPGAs besonders für die echtzeitkonforme Bildverarbeitung eignen, sind neben Maßnahmen zur Reduktion der übertragenen Daten ebenso Möglichkeiten zur Bildvorverarbeitung auf der Kamera untersucht worden. Wie sich aus Kapitel 2.1.2 ergibt, sind die vom PAMONO-Sensor erzeugten Bilddaten stark verrauscht, sodass im Vorfeld zur Analyse eine rauschmindernde Bildvorverarbeitung sinnvoll erscheint. Zur Implementierung auf dem FPGA bietet sich hier einerseits die Rauschminderung durch eine Bildmittelung an, wie sie in Kapitel 7.2.3 erläutert wird. Darüber hinaus gibt es noch die Möglichkeit zur Rauschminderung unter Anwendung eines Waveletdenoising, wie es in Kapitel 7.2.4 beschrieben wird.

7.2.1. Steuerschnittstelle für die Analyse-Software

Die Analyse-Software mit ihrer Verarbeitungspipeline muss auf vielfältige Weise gesteuert werden. In Abbildung 7.2 ist zu sehen, wie die Software und die Kamera miteinander verbunden sind. Auf Basis der Ethernet-Verbindung gilt es, eine Möglichkeit zu schaffen, die Kamera anzusteuern. Neben der Anforderung des Bilddaten-Stroms (als MJPEG-Stream) muss die

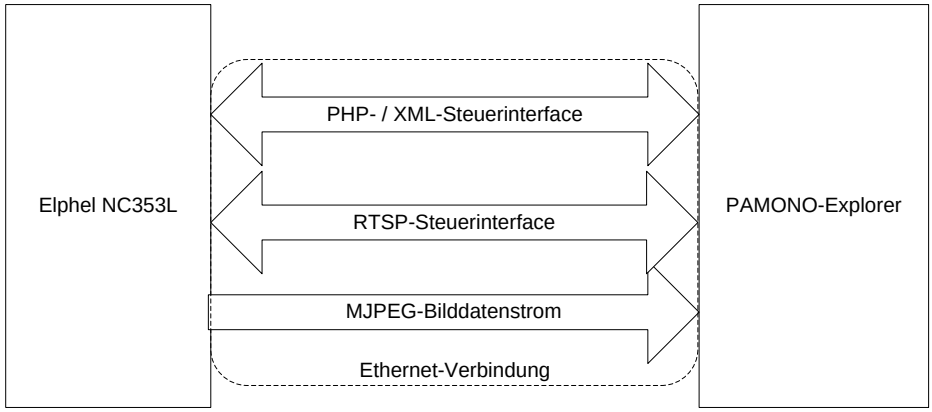
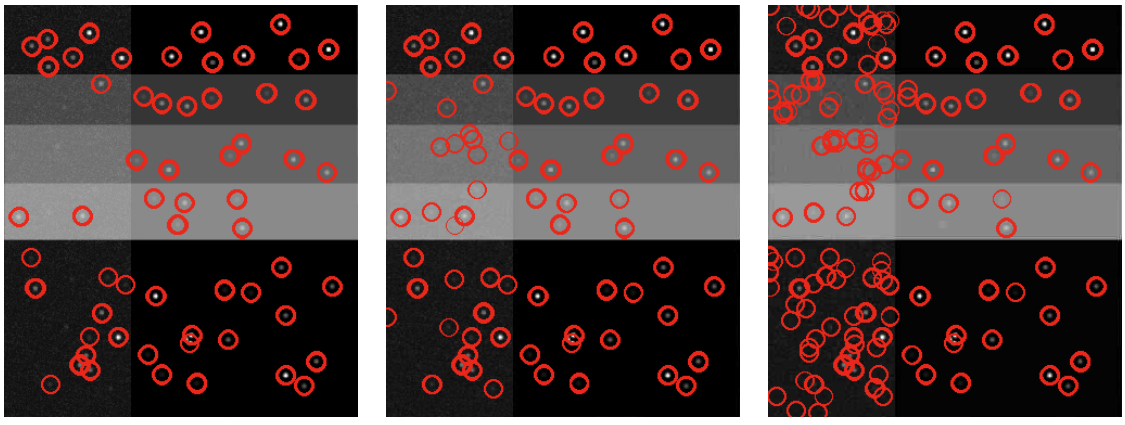


Abbildung 7.2.: Die Elphel-Kamera, die Analyse-Software und die Schnittstellen für Steuerung und Datenaustausch

Kamera auch vor der Durchführung einer Analyse konfiguriert und kalibriert werden können. Zunächst muss der aktuelle Zustand der Kamera abgefragt werden können. Da die Kamera und die Analyse-Software unabhängig voneinander beendet/abgeschaltet bzw. neu gestartet werden können, ist es wichtig, dass die Analyse-Software den aktuellen Status der Kamera erfragen kann, um diesen programmintern verwenden zu können. Die Elphel-Kamera bestimmt im Ausgangszustand (also insbesondere nach einem Neustart) die Belichtungszeit automatisch. Im Versuchsaufbau ist aber eine konstante Belichtungszeit notwendig, damit die Helligkeitswerte der aufgenommenen Bilder nicht durch Änderungen der Belichtungszeit beeinflusst werden. Ein weiterer Aspekt ist hierbei auch die Bildrate. Da der Zeitabstand zwischen zwei aufgenommenen Bildern konstant sein sollte, ist es wichtig, eine konstante Bildrate zu erreichen. Die Belichtungszeit muss konfiguriert werden können, um stabile Bedingungen für die Analyse zu schaffen. Dabei soll die Möglichkeit vorhanden sein, die Belichtungszeit vor Durchführung einer Analyse individuell an die vorhandene Versuchssituation anzupassen.



(a) JPEG-Qualität 100% (b) JPEG-Qualität 70% (c) JPEG-Qualität 55%

Abbildung 7.3.: Anstieg von *false positives* bei der Partikelerkennung bei sinkender JPEG-Qualität

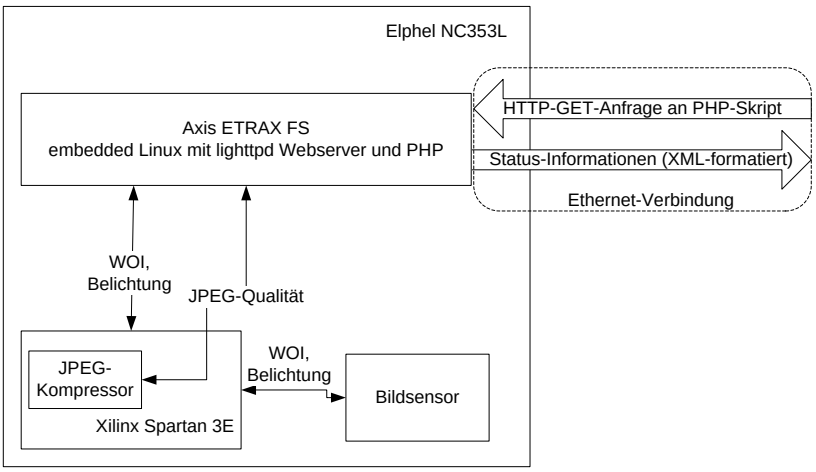


Abbildung 7.4.: Abstraktion der Hardware-Ansteuerung durch das PHP-/XML-Interface

Der „Qualität“-Wert der JPEG-Kompression hat einen großen Einfluss auf die Rate der übertragenen Bilder und natürlich auch auf das Datenvolumen pro Bild. Mit sinkender Qualität werden die Bilder kleiner und damit steigt die Anzahl der erzeugten Bilder pro Sekunde. Allerdings nehmen bei sinkender Qualität die aus der verlustbehafteten Kompression resultierenden typischen Artefakte zu. In Abbildung 7.3 wird ein Testmuster bestehend aus Quadraten mit unterschiedlichen Grauwerten, das manuell mit einigen Partikel-Abbildungen, wie sie für ein PAMONO-Experiment typisch sind, versehen wurde, gezeigt. Die Ausgangsbilder sind in unterschiedlichen JPEG-Qualitätsstufen komprimiert und anschließend mit der Analyse-Software (mit identischen Analyseparametern) untersucht worden. Hierbei sind erkannte Partikel mit einem roten Kreis markiert zu sehen. Deutlich erkennbar steigt die Rate der *false positives* (Region wurde fälschlicherweise als gefundener Partikel markiert) bei sinkender JPEG-Qualitätsstufe.

Die Kamera und der PAMONO-Sensor sind im Versuchsaufbau starr aufeinander ausgerichtet. Auch bei möglichst präziser Vorausrichtung ist selten das komplette von dem Sensor erfasste Bild relevant. Die Elphel-Kamera bietet die Möglichkeit, nur einen Ausschnitt der Sensor-Fläche aufzunehmen und als Bilddaten weiterzureichen. Diese als *Window of interest* (WOI) bezeichnete Funktion hat ebenfalls Einfluss auf die erzeugten Bilder pro Sekunde. Dem Anwender soll die Möglichkeit gegeben werden, das WOI zu setzen. Daraus ergibt sich, dass, zusätzlich zur Anforderung des Bilddatenstroms eine Möglichkeit geschaffen werden muss, um die beschriebenen Parameter der Kamera steuern zu können. Auf der Elphel-Kamera werden die benötigten Systemparameter als Variablen innerhalb der PHP-Umgebung des integrierten Webservers repräsentiert. Als Abstraktionsebene ist in der PG eine in XML-formatierte Zustandsseite entwickelt worden, die mithilfe des in der Kamera integrierten Webservers abrufbar ist. Zusätzlich dazu sind die gleichen Parameter über einen parametrisierten Aufruf der PHP-Interface-Seite einstellbar. In Abbildung 7.4 ist dargestellt, in welchen Abstraktionsebenen die Steuerung und die Rückmeldung der Parameterkonfiguration auf der Kamera stattfinden. Die Analyse-Software kann zur Konfiguration der Kamera eine HTTP-GET-Anfrage über das Ethernet-Interface an die Kamera senden. Der Webserver mit seinen Erweiterungen wird auf der Kamera mit dem Axis-ETRAX-Chip, einem SoC (*system on a chip*) mit eingebettetem Linux,

realisiert. Auf dem eingebetteten System wiederum sind die Funktionsaufrufe Abstraktionen von der konkreten Implementierung im FPGA. Das eingebettete System kommuniziert mit dem FPGA, im Speziellen mit dem darauf implementierten JPEG-Kompressor. Über einen internen Datenbus steuert das FPGA die Einstellungen für Belichtung und WOI des Sensors. Dabei ist die Kommunikation zwischen Bildsensor, FPGA und SoC bidirektional, sodass das SoC neben der Änderung der Parameter zu WOI, Belichtung und JPEG-Qualität auch die jeweils aktuellen Werte kennt. Aus diesen Werten wird eine XML-formatierte Status-Seite generiert, die der anfragenden Instanz als Antwort auf die HTTP-GET-Anfrage gesendet wird. Da dieser Konfigurationsmechanismus weitestgehend von den hardwarespezifischen Daten der Kamera abstrahiert ist, ist an dieser Stelle die Unterstützung weiterer Kameratypen denkbar.

7.2.2. Datenreduktion durch Hintergrundsubtraktion

Die Hintergrundsubtraktion eignet sich zur Reduktion der zu übertragenen Daten. Wie in Kapitel 2.3.1 beschrieben wird, ist es möglich, die Bilddaten so aufzufassen, dass das erste bzw. ein definiertes Bild als Ausgangsbild benutzt wird und alle Folgebilder nur noch als eine Reihe von Abstandswerten bezüglich der Lichtintensität aufgefasst werden. Wie Abbildung 3.1 zeigt, ändern sich die Intensitätswerte einzelner Pixel über die Zeit nur geringfügig. Um diese geringfügigen Änderungen abbilden zu können, werden keine großen Bitbreiten benötigt. Aus diesem Grunde sind pro Pixel weniger als die 12 Bit im Ausgangszustand notwendig, sodass Bits bei der Datenübertragung eingespart werden können.

7.2.3. Rauschminderung durch Bildmittelung

Die Bildmittelung, wie sie in Kapitel 2.2.1 beschrieben ist, kann dazu beitragen, die Verarbeitungspipeline (s. hierzu Kapitel 6.4) zu entlasten. Die Kamera hat als Quellelement der Pipeline maßgeblichen Einfluss auf dessen Grundlast. Wenn von der Kamera viele Bilddaten erzeugt werden, müssen auch alle verarbeitenden Elemente viele Daten analysieren. Von Bild zu Bild sind immer nur geringe Änderungen der einzelnen Intensitätswerte vorhanden, sodass durch das kontinuierliche Übertragen von Vollbildern viel nicht benötigte Information übertragen wird. Desweiteren erfolgen die Intensitätssprünge relativ langsam gegenüber der Bildserie, wie durch Abbildung 3.1 verdeutlicht wird. Die Bildmittelung sorgt für eine Rauschminderung auf den Bilddaten. Diese Rauschminderung mindert geringfügige Änderungen in den Intensitätswerten und sorgt für eine verbesserte Ausgangslage bei der Analyse. Da der beschränkende Faktor bei der Datenübertragung der Kamera die 100 MBit-Netzwerkkarte ist, erfüllt die Bildmittelung darüber hinaus den Zweck, die zu übertragenden Daten zu reduzieren. Dadurch ist es möglich, die bei der Übertragung von Standard-Vollbildern theoretisch maximale Bildrate von ca. 30 fps zu überschreiten und eine echtzeitkonforme Analyse wird so ermöglicht.

7.2.4. Rauschminderung durch Waveletdenoising

In Kapitel 2.2.2 ist beschrieben, dass sich Wavelets, dazu geeignet sind, um eine Rauschminderung auf Signalen durchzuführen. Zur einfacheren Berechnung eignet sich die im gleichen Kapitel vorgestellte DWT. Die Durchführung einer DWT auf einem Bilddatenstrom stellt eine in sich geschlossene Aufgabe dar. Aus diesem Grund ist nach möglichen einsetzbaren FPGA-Kernen recherchiert worden. Die Recherche hat ergeben, dass es keinen frei verfügbaren Kern gibt, der diese Aufgabe bewältigt, in einem funktionsfähigen Zustand ist und für die in der Elphel-Kamera verwandte Architektur implementiert worden ist. Allerdings existiert die Implementierung einer DWT auf VHDL-Basis, die relativ unabhängig von der zugrundeliegenden Hardwarearchitektur implementiert wurde und sich daher als Grundlage für die Integration in das Verilog-Projekt auf der Elphel-Kamera eignet.

7.2.5. Entlastung der Pipeline durch Region of Interest

In Kapitel 7.1 wurde erläutert, dass die Smart Camera das Quell-Element der Verarbeitungspipeline ist. Daraus ergibt sich, dass die Kamera einen maßgeblichen Einfluss auf die Datenmenge hat, die von der Pipeline verarbeitet werden muss. Wie die Forschung ergeben hat, ist nicht die komplette Sensorfläche interessant für die Bildaufnahmen. Das bedeutet, dass nur ein Ausschnitt der aufgenommenen Bilder überhaupt relevant für die Analyse ist. Die hier notwendige Ausschnittselektion (Region of Interest, ROI) wird durch die Kamera nativ unterstützt. Es kann ein Rechteckausschnitt der Gesamtsensorfläche im Bild definiert werden. Das Definieren einer ROI (im Sprachgebrauch des Herstellers Window of Interest, WOI, genannt) wird der Analyse-Software mit Hilfe des in Kapitel 7.2.1 beschriebenen Steuerinterfaces ermöglicht. Durch die Festlegung des ROIs ergibt sich eine möglicherweise drastische Reduktion der zu übertragenden Daten. Dadurch wird die gesamte Verarbeitungspipeline entlastet, indem die Grundmenge der zu verarbeitenden und zu analysierenden Daten reduziert wird. Durch diese Entlastung der Verarbeitungspipeline wird dafür gesorgt, dass die besonders rechenintensiven Operationen in der Analyse vermindert werden. Dadurch ergibt sich eine Verringerung des Energieverbrauchs des Gesamtsystems.

7.3. Aufbau der Hardware

In der Projektgruppe wird eine Kamera vom Typ Elphel NC353L-369-HDD verwendet. Die Kamera besteht aus mehreren, auf Platinen realisierten Modulen. Allen Kamera-Modellen gemein ist das Mainboard 10353. In Abbildung 7.5 werden im oberen Bereich das Mainboard 10353, links das Sensor-Board 10338 sowie unten drei mögliche Erweiterungsboards dargestellt. Das Board 10353 trägt mitunter das Ethernet-Interface, das FPGA und ein SoC (*system on a chip*), auf dem ein eingebettetes Linux läuft, um unter anderem ein Web- und ein Programmierinterface bereitzustellen. Auf dem Sensorboard 10338 ist der Bildsensor verbaut, bei der vorliegenden Kamera ein Aptina MT9P001. Dieser Bildsensor ermöglicht eine Auflösung von maximal 2592×1944 Pixeln. Der Signalfluss auf der Kamera stellt sich dabei im Wesentlichen wie folgt dar:

Die aufgenommenen Sensordaten werden auf dem Xilinx Spartan 3E 1200k FPGA verarbeitet. Über das *System interface* werden die Bilder an einen Axis ETRAX FS SoC geleitet. Dieser Chip beinhaltet einen 32 Bit MIPS-RISC Prozessor mit 200 MHz Systemtakt sowie ein Ethernet-Interface mit 10/100 MBit. Das Kamera-Modell der PG besitzt zusätzlich zum 10353- und 10338-Board auch das HD-Modul 103692, das zur nicht-flüchtigen Speicherung von Bildreihen und Videos geeignet ist. In der Regel werden die aufgenommenen Bilder direkt über die Netzwerkkarte übertragen, um sie möglichst latenzarm weiter verarbeiten zu können (zur Notwendigkeit einer echtzeitkonformen Datenübertragung siehe auch Kapitel 7.1).

In Abbildung 7.5 ist die Verarbeitung der Daten auf dem FPGA zu sehen. Beachtenswert hierbei ist, dass sämtliche bildverarbeitenden und bildvorverarbeitenden Elemente der Kamera in Verilog programmiert und auf dem FPGA realisiert worden sind. Der Sensor überträgt die Rohdaten mit 12 Bit Datenbreite zunächst an die *FPN correction*. Diese reicht die Daten an die *Vignetting compensation* und die *Gamma correction* weiter, bis die Bilddaten am Kanal 0 des *SDRAM Controllers* empfangen werden. Dieser verwaltet alle Speicherzugriffe auf die 64 MB des DDR-SDRAM und gruppiert die Pixel für die weitere Verarbeitung. In Blöcken von je 20×20 Pixeln werden diese Teilbilder über den Kanal 2 an die Kompressionseinheit übertragen. Die Kompression erlaubt es, die Bilder in das JPEG-Format zu konvertieren. Die JPEG-Qualität der Kompression kann dabei im laufenden Betrieb angepasst werden. In Abbildung 7.6 ist der Aufbau für das Erweiterungsboard 10359 zu sehen. Dieses Board wurde als Multiplexer-Board entwickelt, um an einer Elphel-Kamera drei Bildsensoren betreiben zu können. Auch hier ist ein FPGA vom Typ Spartan 3E 1200K der Firma Xilinx verbaut, ebenso mit 64 MB DDR-SDRAM. Die werksseitig ausgelieferte Firmware bietet die Möglichkeit, die Bilddaten der verschiedenen Sensor-Anschlüsse auf unterschiedliche Weise zu kombinieren. Detailliert wird hierauf in Kapitel 7.3.3 eingegangen. Da das Board zwischen Bildsensor(en) und Prozessor-Board geschaltet wird, ergibt sich die Möglichkeit, das FPGA für die Bildvorverarbeitung zu nutzen. Der Verilog-Code beider FPGAs ist frei verfügbar und kann somit mit beliebigen Verilog-Entwicklungsumgebungen, wie zum Beispiel der Xilinx ISE [Xil], weiterentwickelt werden.

7.3.1. Besonderheiten beim 10353-Board

Im Rahmen der Projektgruppe war es zunächst das Ziel, eine Bildvorverarbeitung auf dem FPGA des 10353-Boards umzusetzen. Hierbei stand die Umsetzung einer Bildmittelung und einer Hintergrundsubtraktion im Fokus, wie sie in den Kapiteln 7.2.2 und 7.2.3 theoretisch und in Kapitel 7.5 methodisch bzw. technisch beschrieben werden. Die Notwendigkeit eines Waveletdenoising (s. Kapitel 7.2.4 und Kapitel 7.5.2) ergab sich im späteren Verlauf der Projektgruppe. Es ist daher zunächst untersucht worden, welche Wege existieren, in das bestehende FPGA weitere Module zu integrieren. Wie in Kapitel 7.3 beschrieben ist, gibt es im FPGA einen Signalweg mit einigen Elementen und zeitlich aufeinander abgestimmten Elementen. Dabei kann die Signalverarbeitung auf dem FPGA grob in drei Abschnitte, orientiert an der zeitlichen Abhängigkeit, unterteilt werden:

1. Der Abschnitt zwischen Sensor und Memorycontroller verläuft zeitlich linear. Über eine Uhr wird gesteuert, wann der nächste Bildpunkt weitergegeben wird. Jedes Modul, das

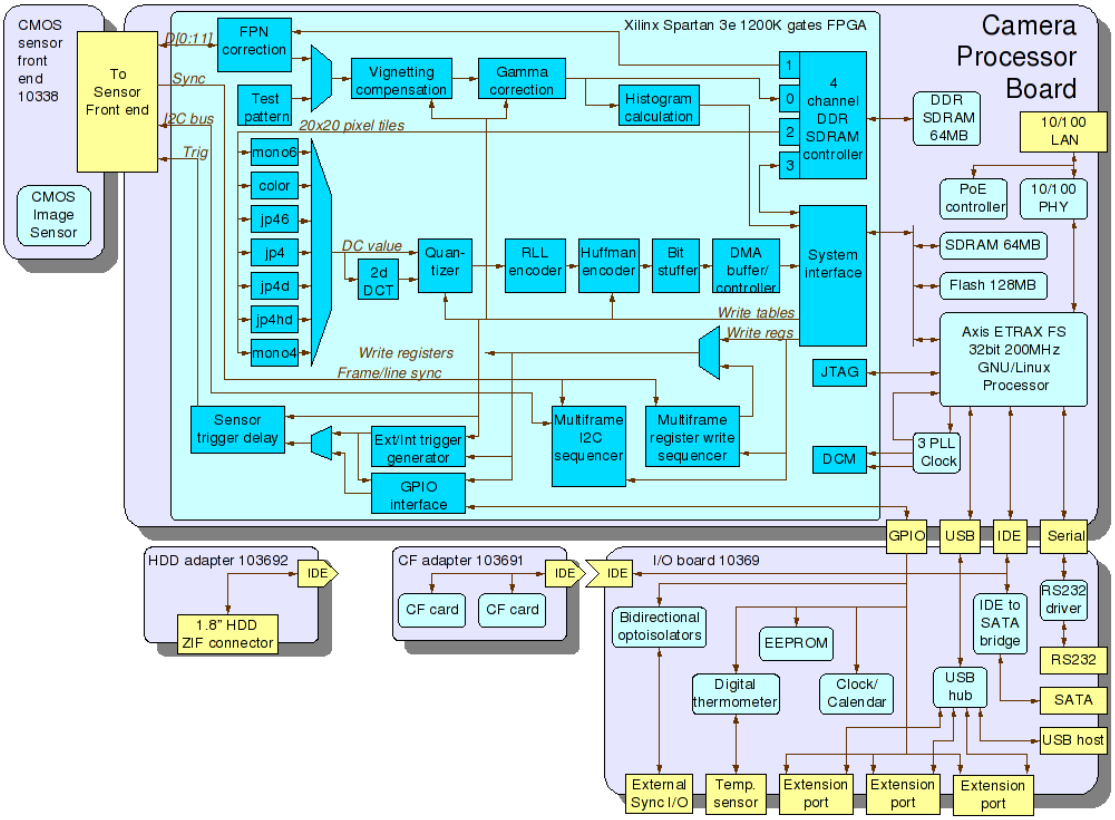


Abbildung 7.5.: Funktionseinheiten und Erweiterungen der Elphel-Kamera (aus [Fil])

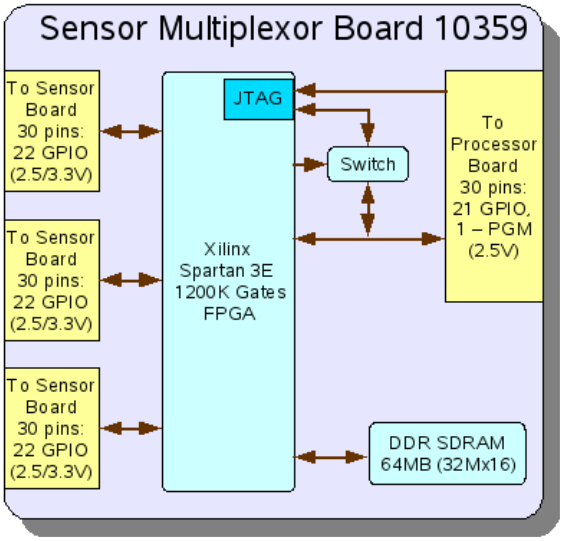


Abbildung 7.6.: Funktionseinheiten des Sensor-Boards 10359 der Elphel-Kamera (aus [Fil])

in diesem Abschnitt hinzugefügt wird, stört den zeitlichen Ablauf und erzeugt Fehler. Eine Anpassung an die unvermeidbare Verzögerung eines zusätzlichen Moduls zieht weitere schwere Bildfehler nach sich, die darauf zurückzuführen sind, dass Uhren und Trigger an mehreren Punkten zum Einsatz kommen. Dadurch kann dieser Abschnitt für Erweiterungen nicht oder nur unter hohem Zeitaufwand genutzt werden.

2. Zwischen Memorycontroller und Kompression erfolgt der Datenaustausch nicht über den gegebenen Takt und Uhren, sondern über Trigger, die zwischen den beiden Modulen ausgetauscht werden. Das Kompressor-Modul schickt nach jedem 20×20 Pixel großen Fenster ein Signal an den Memorycontroller, um den Versand des nächsten Fensters anzufordern. Dies ist notwendig, da für den Kompressor keine feste Laufzeit festgelegt ist. Die schwankende Laufzeit resultiert hauptsächlich aus der Möglichkeit für den Benutzer, die Einstellungen der Kompression dynamisch zu verändern. Dabei benötigt das Kompressor-Modul bei einer schlechteren Qualitätsstufe weniger, und bei einer höheren Qualitätsstufe mehr Zeit, um ein 20×20 Pixel-Fenster zu komprimieren.

3. Die Bildpunkte liegen nach der Kompression im JPEG-Bildformat vor. Eine Bildverarbeitung kann zu diesem Zeitpunkt nicht mehr durchgeführt werden, da sich die Daten in einem komprimierten Format befinden und dazu erst dekomprimiert werden müssten. Neben einem beträchtlichen Zeitaufwand wäre durch die verlustbehaftete JPEG-Kompression ein Informationsverlust in Kauf zu nehmen, da die Bilddaten für die Übertragung erneut komprimiert werden müssten. Daher wurde auch dieser dritte Abschnitt bei den Arbeiten der PG nicht berücksichtigt.

In Anbetracht der gegebenen Umstände bietet sich für eine sinnvolle Erweiterung der Funktionalität der Kamera das FPGA auf dem Multiplexer-Board 10359. Dieses Modul kann zusätzlich eingebaut und verwendet werden.

7.3.2. Besonderheiten beim 10359-Board

Das Zusatz-Sensor-Board 10359 ist, wie in Kapitel 7.3 erwähnt, als Erweiterung des Prozessor-Boards 10353 für die Kamera erhältlich. Mit dem 10359 ist es möglich, drei Sensoren an einer Kamera zu betreiben. Für die Projektgruppe bietet dieses FPGA die Möglichkeit, Module zu entwickeln, ohne in den Ablauf auf dem 10353 einzugreifen. Da in der Projektgruppe nur ein Sensor an der Kamera benötigt wird, können die Ressourcen für die anderen beiden Sensoren für eigene Zwecke verwendet werden. Da mit einer Kamera auf diesem FPGA keine Änderungen an den Bildpunkten vorgenommen werden, gibt es keine Angriffspunkte die unpassend sind oder Störungen verursachen. Wegen der besseren Überprüfbarkeit ist die sinnvollste Stelle allerdings direkt vor der Übergabe der Bildpunkte an das 10353. Im folgenden Abschnitt wird erläutert, wie die beiden FPGAs verbunden sind.

Abbildung 7.7.: Dies ist das 10359-FPGA-Board von oben gesehen. J1 ist die Verbindung zum 10353, J2 bis J4 werden verwendet, um Sensoren zu verbinden [Fil].

7.3.3. Verbindung zwischen 10353 und 10359

Das 10359-Board wird nachträglich in die Kamera eingebaut, wodurch die Erweiterbarkeit erheblich verbessert wird. Wie in Abbildung 7.7 zu sehen ist, können drei Sensoren mit dem FPGA verbunden werden. Der Steckverbinder J1 dient zur Verbindung mit dem 10353-Board. Welcher Steckplatz mit welchen Signalnamen verbunden ist, kann aus Tabelle 7.1 entnommen werden. Der vorhandene Quelltext für das 10359 verwendet nur jene Signale, für die ein entsprechender Sensor angeschlossen ist.

Bezeichnung des Pins	Signalname im Quellcode	Funktion
J1	PXD	Datentransfer von 10359 nach 10353
J2	PXD1	Datentransfer von erstem Sensor nach 10359
J3	PXD2	Datentransfer von zweitem Sensor nach 10359
J4	PXD3	Datentransfer von drittem Sensor nach 10359

Tabelle 7.1.: Zusammenhang zwischen Pinname und Signalname auf dem 10359-Board

Auf dem FPGA des 10359-Boards stehen ab Werk vier Betriebsarten zur Verfügung[Fil]:

1. Wird verwendet um die Sensoren zu programmieren. Entweder einzeln, oder alle vorhandenen gemeinsam.

2. Wechselt zwischen den vorhandenen Sensoren. Die Daten werden dabei in das SDRAM des 10359 gespeichert.

3. Die Bilder von 2 Sensoren alternieren direkt.

4. Wechselt zwischen zwei Sensoren wie Betriebsart drei, speichert aber das Bild von einem Sensor im SDRAM zwischen.

Da die Projektgruppe nur einen Sensor benötigt, bietet es sich an, den vierten Modus zu benutzen und zu überarbeiten, um die Bilder sichern zu können, die für eine Vorverarbeitung benötigt werden. Im folgenden Abschnitt wird ausgeführt, warum ein Wechsel des Zielbausteins auf den FPGA des 10359-Boards nötig war.

7.3.4. Erweiterung mit 10359

Weder ein Waveletdenoising unter Nutzung einer Diskreten Wavelet Transformation (DWT, siehe Kapitel 7.5.2) noch die Verwendung des Quelltextes aus [App07] (siehe Kapitel 7.2.4) ist auf dem 10353-Board möglich gewesen. Die DWT erfordert es, Bilder im SDRAM zu puffern; da aber für das Kompressor-Modul auf dem 10353-Board keine garantierte Abholrate der Bilder existiert, kann nicht sicher gestellt werden, dass auf dem SDRAM des 10353 genügend Speicherplatz zur Verfügung steht, um die Bilder für die DWT temporär zu sichern. Darüber hinaus erfordert der Quelltext, der für die Erweiterung mit einer DWT verwendet wird, mehr Platz auf dem FPGA, als für programmierbare Logik auf dem 10353-Board zur Verfügung steht. Dies bedeutet, dass selbst bei genügend freiem Speicherplatz im SDRAM, die DWT nicht auf dem 10353 implementiert werden kann.

7.4. Einbindung eigener Erweiterungen in den Signalfluss

Um die in Kapitel 7.2 beschriebenen Vorverarbeitungsschritte auf einem FPGA der Kamera implementieren zu können, muss die gegebene Architektur der Smart Camera berücksichtigt werden. Die eigenen funktionalen Module für das FPGA müssen in den Signalfluss auf dem gegebenen FPGA eingepasst werden. Abbildung 7.5 zeigt den Signalfluss auf dem Haupt-FPGA im Ausgangszustand.

7.4.1. Implementierung auf dem Haupt-FPGA

Wie die Rücksprache mit dem Hersteller der Kamera ergeben hat, ist der Code der *FPN-Correction* in keinem funktionsfähigen Zustand, sodass das entsprechende Modul als Rahmen für eine eigene Entwicklung genutzt werden kann. FPN steht hierbei für *Fixed Pattern Noise* und beschreibt ein bei digitalen Bildsensoren oftmals auftretendes, charakteristisches Rauschen auf den Bilddaten. In Abbildung 7.8 sind für die PG relevante Ausschnitte aus dem Signalfluss inklusive notwendiger Modifikationen zur Umsetzung einer Bildmittelung oder Hintergrundsubtraktion dargestellt. Direkt am Sensor angesiedelt ist das Modul `sensorpix`. Ein Teilmodul dieses Moduls ist jenes für die FPN-Correction. An dieser Stelle können nun verarbeitende Schritte durchgeführt werden. Durch die Signalleitung zu dem Modul für die Bildmittelung des Kanals `Channel 1` des Memory Controllers besteht die Möglichkeit, bspw. das Hintergrundbild für die Hintergrundsubtraktion im DDR-RAM zu puffern.

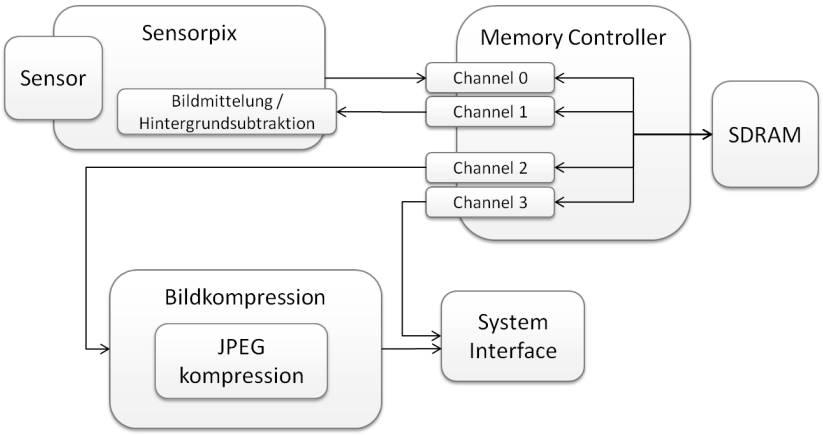


Abbildung 7.8.: Integration eigener Implementierungen in den Signalfluss auf dem 10353

7.4.2. Implementierung auf dem Zusatz-FPGA

Für weitergehende Vorverarbeitungsschritte muss eine Position gewählt werden, die nicht durch existierende feste Takte eingeschränkt ist. Als einzige Möglichkeit ist die Stelle zwischen dem *Memory controller* und der Kompression zu sehen. Die Kommunikation beider Elemente findet nicht über Takte statt, sondern wird von der Kompression gesteuert. Diese steuernden Signale können verwendet werden, um ungewollte Auswirkungen von Verzögerungen eigener Module zu verhindern. Auf der Erweiterungsplatine stellen getaktete Abläufe kein Problem dar. Wegen der Übersichtlichkeit, der besseren Überprüfbarkeit und einem Vorteil bei der Simulation, wird auf dem 10359-Board die eigene Implementierung kurz vor der Übergabe an die Verarbeitungspipeline des 10353 gestellt. Hier kann ohne Beeinträchtigung der existierenden Umgebung mit der Realisierung eigener Erweiterungen begonnen werden.

7.5. Realisierung der eigenen Erweiterungen

In Kapitel 7.2 wird eruiert, welche zusätzlichen Funktionalitäten auf der Kamera implementiert werden sollten, um den Analyseprozess, wie er im Rahmen der Projektgruppe entwickelt worden ist, zu unterstützen bzw. zu entlasten. Hierbei sind vor allem drei Funktionalitäten sinnvoll: die Hintergrundsubtraktion, die Bildmittelung und das Waveletdenoising. In Kapitel 7.4 wird dargestellt, welche Stellen im Signalfluss auf den FPGAs der Kamera sich zur Integration zusätzlicher Module eignen. In diesem Abschnitt wird beschrieben, wie die benötigten Funktionen umgesetzt werden sollen. Aufgrund großer inhaltlicher Überschneidung werden beide Erweiterungen *Hintergrundsubtraktion* und *Bildmittelung* in einem Abschnitt, dem Kapitel 7.5.1 erläutert. In Kapitel 7.5.2 wird beschrieben, wie die Rauschunterdrückung mithilfe der Wavelettransformation realisiert werden soll.

7.5.1. Hintergrundsubtraktion und Bildmittelung

Das größte Problem bei dem Entwurf dieses Vorverarbeitungsschrittes ist die Speicherung von Bildern. Um eine deutliche Datenreduzierung zu erhalten, ist angestrebt, eine Mittelung über 16 Bilder durchzuführen. Hierfür müssen diese Bilder im vorhandenen SDRAM zwischengespeichert werden. Neben dem in Kapitel 7.3.4 aufgezeigten Platzproblem auf dem FPGA des 10353-Boards ergeben sich zusätzliche Herausforderungen, da der Memorycontroller an diese neue Architektur angepasst werden muss. Der Memorycontroller besitzt vier Kanäle, jeweils mit eigenem Puffer. Die Pufferung erfolgt mit Block-RAM-Modulen von Xilinx. Für jede dieser Schnittstellen wird im Modul *chArbit* die entsprechende Adresse im SDRAM berechnet. Es gibt zwei Varianten, den Memorycontroller anzupassen:

1. Der Memorycontroller wird um einen zusätzlichen Kanal erweitert. Dadurch wird die vorhandene Konfiguration nicht beeinträchtigt, doch die Berechnung der Adressen für den SDRAM muss um einen weiteren Puffer erweitert werden. Diese Erweiterung führt zu Überlappungen im SDRAM. Damit werden die gespeicherten Daten unvorhersehbar geändert und unbrauchbar.

2. Einer der vorhandenen Kanäle wird so verändert, dass er die notwendigen Aufgaben übernimmt. Hier entsteht das Problem, dass bei keinem Kanal die Adressberechnung darauf ausgelegt ist, mehr als 2 Bilder in Serie zu speichern. Hier ist eine Änderung der Adressberechnung erforderlich, die zu dem gleichen Problem wie bei Punkt eins führen würde.

Die Bildvorverarbeitung hat als primäres Ziel die Reduktion der zu übertragenden Datenmenge. Eine Herangehensweise ist die Implementierung einer Subtraktion des Hintergrundes. Durch diese Methode wird die maximale Amplitude des Signals verkleinert und die Anzahl der pro Pixel zu übertragenden Bits wird minimiert. Für die Durchführung dieser Subtraktion wird ein Referenzbild benötigt. Um aktuelle Fehler der Linse und des Versuchs in diesem Referenzbild zu berücksichtigen, bietet es sich an, das erste Bild zu verwenden, das von dem Sensor weitergeleitet wird. In Abbildung 7.9 ist der Ablauf während der Subtraktion zu sehen. Das erste Bild wird im RAM gesichert und bei jedem weiteren Bild wird dieses jeweils vom Referenzbild aus dem Speicher abgezogen.

Bei der Bildmittelung wird ein identischer Aufbau verfolgt. Bei der Mittelung wird jedoch nicht nur das erste Bild gesichert, sondern jedes zweite Bild muss im RAM gesichert werden. Für eine Bildmittelung über 16 oder mehr Bilder müssen diese Bilder separat im RAM gesichert und abgerufen werden. Dabei bietet sich eine Lösung über einen Ringpuffer an, wie er zum Beispiel in Kapitel 9.3.2.1 beschrieben wird.

7.5.2. Waveletdenoising

Wie in Kapitel 7.2.4 bereits angesprochen wird, dient der Projektgruppe als Ansatz zur Durchführung eines Waveletdenoising ein abgeschlossenes Projekt, in dessen Rahmen eine VHDL-basierende Implementierung einer DWT unter Nutzung eines Haar-Wavelets (vgl.

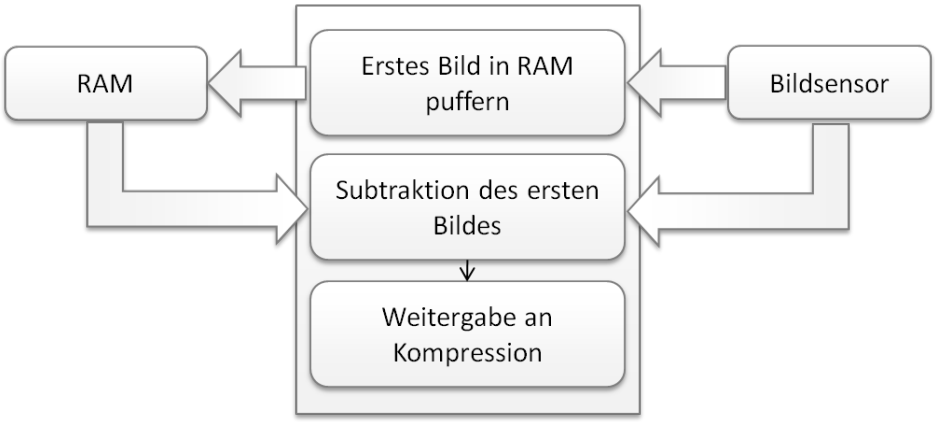


Abbildung 7.9.: Methodischer Ablauf der Hintergrundsubtraktion

Abbildung 2.3(a)) entstanden ist. Bei diesem Projekt handelt es sich um eine Studienarbeit an der Humboldt-Universität zu Berlin. Teil dieser Studienarbeit ist die Implementierung einer 2D-Wavelet-Transformation auf der XUP-Plattform von Xilinx[App07] gewesen. Die Ausführungen zur DWT in diesem Abschnitt basieren auf den Erkenntnissen aus dieser Studienarbeit. Das FPGA-Projekt aus der Studienarbeit realisiert eine zweidimensionale, diskrete Haar-Wavelet-Transformation. Sie wird in der vorliegenden Arbeit auf 2×2 Pixeln ausgeführt. Hierfür wird eine Mittelwert- und Differenzbildung durchgeführt, wobei das Mittelwertsignal die Bildinformation in halber Auflösung beinhaltet und im Differenzensignal – im zweidimensionalen Fall den drei Differenzen in horizontaler, vertikaler und diagonaler Richtung – Kanten in der entsprechenden Richtung sichtbar gemacht werden.

In Abbildung 7.10 ist ein Beispiel für die implementierte Wavelet-Transformation dargestellt. Im ersten Transformationsschritt wird das Ausgangsbild in zwei Bilder aufgeteilt, die in x-Richtung die halbe Auflösung haben. In der Abbildung haben je zwei Pixel, die für die Berechnung benutzt werden, das gleiche Muster und jeweils die gleiche Farbe wie das entsprechende Ergebnis-Pixel. Nach dem ersten Transformationsschritt erfolgt ein zweiter Transformationsschritt, der die beiden Bilder nun in y-Richtung transformiert. Dies führt somit zu vier Teilbildern,

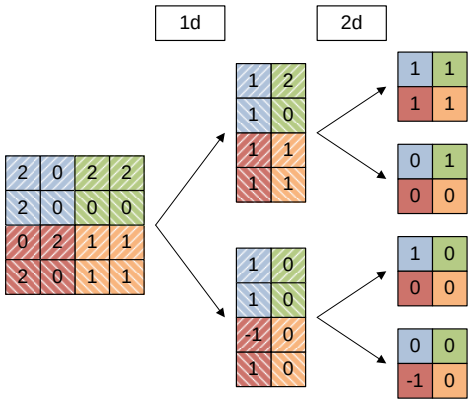


Abbildung 7.10.: 2D-Wavelet-Transformation (aus [App07])

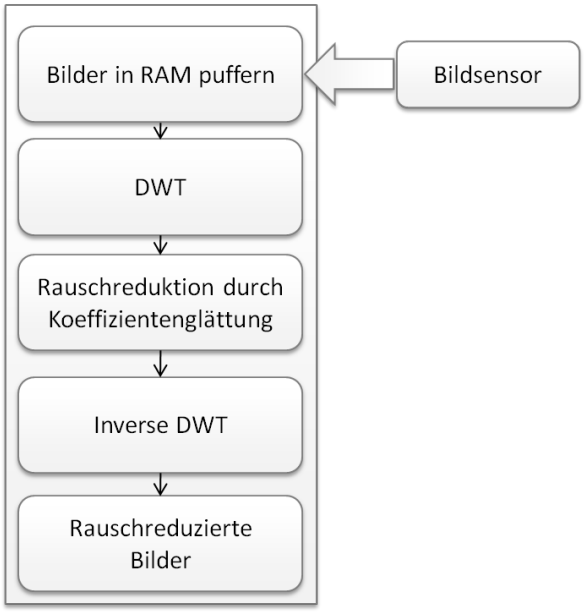


Abbildung 7.11.: Methodischer Ablauf der Diskreten Wavelet Transformation

jeweils mit der halben Größe in x- und y-Richtung. Das oberste Bild ist das Summenbild. In diesem sind die Kanten aus den berechneten Gebieten geglättet. Die anderen drei Bilder sind Differenzenbilder, bei denen immer dann von Null verschiedene Zahlen auftreten, wenn Kanten vorlagen. Die Vorzeichen in den Bildern mit den Differenzen zeigen die Richtung der jeweiligen Kante an. Außerdem werden durch die Transformationen nicht alle Kanten im Bild gefunden, sondern nur die, die sich innerhalb der 2×2 Kästchen befinden. Dies ist jedoch nur im ursprünglichen Zustand des Projektes so, nach der Übertragung auf das Elphel-Projekt sind hier Erweiterungen möglich. Mittels einer inversen Wavelet-Transformation kann das Ausgangsbild aus den Teilbildern wieder zurückgewonnen werden. Das Projekt aus der Studienarbeit soll zunächst in das FPGA-Projekt der Elphel-Kamera integriert werden, um dann entsprechend den im Rahmen der Bildvorverarbeitung benötigten Funktionen erweitert zu werden.

Wie in Kapitel 2.2.2 beschrieben ist, kann das Waveletdenoising einen wertvollen Beitrag zur Bildvorverarbeitung leisten. Die Entwicklung erfolgt direkt auf dem 10359. Probleme, die bei der Bildmittelung aufgetreten sind, werden dadurch umgangen. Das integrierte und erweiterte Projekt soll kurz vor der Übergabe der Daten vom 10359- an das 10353-Board integriert werden. Dadurch kann in einer Simulation besser überprüft werden, welche Daten das 10359 verlassen. Für die Integration wurden alle für die Projektgruppe irrelevanten Bestandteile des bestehenden Projektes entfernt, und nur die DWT weiter verwendet.

Der vorgesehene Ablauf des Waveletdenoising wird in Abbildung 7.11 gezeigt. Für die Anpassung der Transformation an eine Zeitreihe wird eine entsprechende Anzahl Bilder im RAM gesichert. Der *Memorycontroller* wird modifiziert, damit die Pixel aller gesicherten Bilder mit identischen Koordinaten nacheinander geladen werden können. Auf dieser Reihe wird die DWT, wie in Kapitel 2.2.2 beschrieben, ausgeführt. Über den daraus entstandenen Differenzen

wird eine Koeffizientenglättung ausgeführt, um so eine Rauschminderung herbeizuführen. Damit der Verarbeitungspipeline der Analyse-Software Bilder im JPEG-Format zur Verfügung gestellt werden können, werden die transformierten und koeffizientengeglätteten Daten über eine inverse DWT in Bilddaten zurückgeführt. Die so rauschreduzierten Bilder werden weitergeleitet in das Verarbeitungsboard 10353, wo sie durch die vorhandene Signalverarbeitung JPEG-komprimiert und schließlich über das Netzwerk an die Analyse-Software übergeben werden.

8. Objektorientierte Realisierung

Dieses Kapitel beschreibt die objektorientierte Realisierung der in den vorherigen Kapiteln beschriebenen Techniken und Verfahren. Kapitel 8.1 gibt einen Überblick über den Aufbau und die Unterteilung der Software. In Kapitel 8.1.4 wird die Realisierung der Pipeline aus Kapitel 6 sowie die unterschiedlichen Verarbeitungselemente beschrieben. Kapitel 8.1.6 erläutert das Benutzerinterface und die Arten der Visualisierung der Ergebnisse. In Kapitel 8.2 schließlich werden die während des Softwareentstehungsprozesses benutzten Hilfsmittel und Bibliotheken kurz dargestellt. Für eine detaillierte Beschreibung aller Funktionen sei an dieser Stelle auf den Anhang verwiesen, in dem sich das Benutzerhandbuch befindet. Dort sind alle Funktionen und Fenster der Software PAMONO-Explorer ausführlich mit Screenshots erklärt.

8.1. Struktur der Software

Die in der Projektgruppe entstandene Software PAMONO-Explorer ist aus mehreren Paketen zusammengesetzt, die nach Funktionalität getrennt sind. Diese Pakete heißen:

- *Analysis* (Klassen zur Analyse)
- *ImageServer* (Klassen zur Datenakquise)
- *Classification* (Klassen zur Klassifikation und für die Samples)
- *Pipeline* (Klassen für die Verarbeitungspipeline)
- *ProjectManager* (Klassen der Projektverwaltung)
- *GUI* (Klassen für die Benutzerinteraktion und das Fenstersystem)

Abbildung 8.1 zeigt die Pakete und deren Abhängigkeiten untereinander. In den folgenden Abschnitten werden die einzelnen Pakete und deren Inhalte in Form von Klassen näher erläutert.

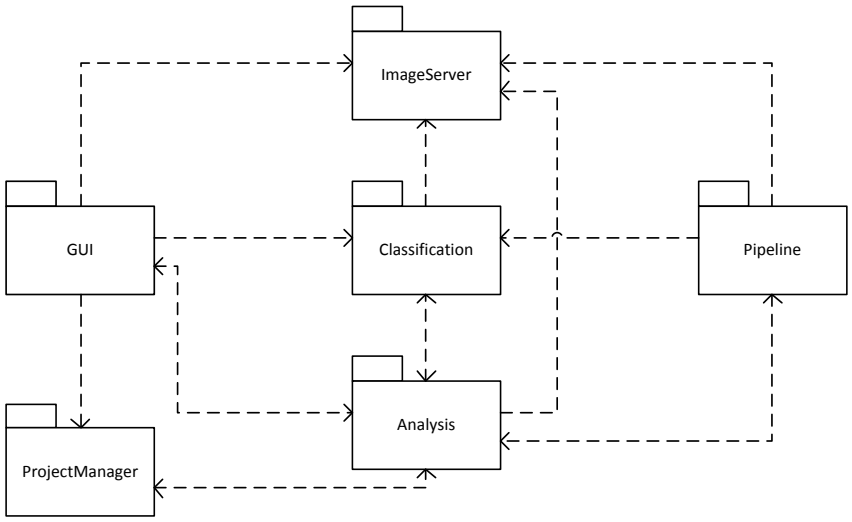


Abbildung 8.1.: Die Softwarepakete aus denen der PAMONO-Explorer aufgebaut ist.

8.1.1. Das Paket *Analysis*

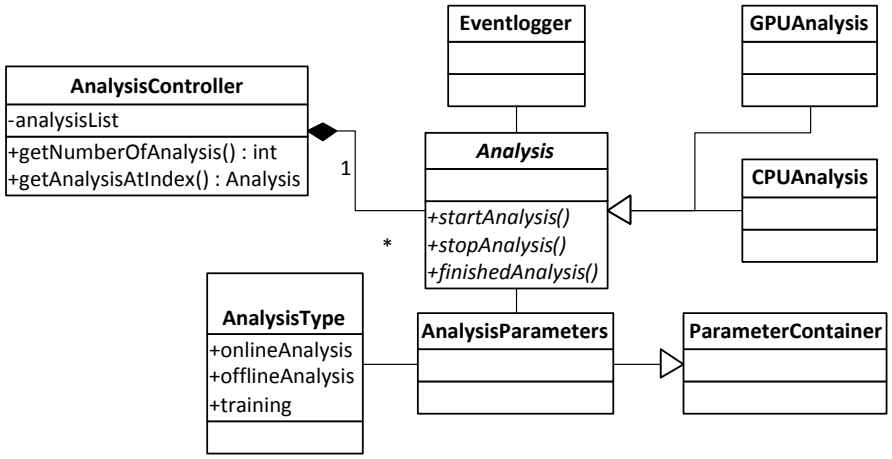


Abbildung 8.2.: Klassendiagramm des Analysis-Pakets

Im Softwarepaket *Analysis* sind die Klassen gruppiert, die die notwendigen Aufgaben für die Analyse kapseln. Die Übersicht und die Verbindungen zwischen den Klassen sind in Abbildung 8.2 zu sehen. Die Klasse *AnalysisController* übernimmt die Verwaltung von mehreren Analysen, die parallel existieren können. Die Klasse *Analysis* ist eine abstrakte Klasse, von der die Unterklassen *GPUAnalysis* (Verwendung der Grafikkarte des Systems für Berechnungen) und *CPUAnalysis* (Verwendung des Hauptprozessors) abgeleitet sind. Die Implementierung der Grafikkartenvariante benötigt dabei eine abweichende Pipelinestruktur, hat aber eine große funktionale Schnittmenge mit der CPU-Variante. Die Beschreibung der Pipeline-Verarbeitung befindet sich in Abschnitt 8.1.4. Die abstrakte Klasse *Analysis* bietet eine einheitliche Schnitt-

stelle zu den unterschiedlichen Methoden der Analysen, unter anderem das Starten und Stoppen eines Vorgangs sowie immer benötigte Elemente. Darunter fallen beispielsweise die einstellbaren Analyseparameter, welche in der Klasse *AnalysisParameters* verwaltet werden. Diese Klasse erbt von der Klasse *ParameterContainer*, die eine Reihe an Datenstrukturen zur Speicherung von Parametern bereitstellt. Dort lassen sich Zeichenketten (QString und Listen von QString), Zahlen (Integer und Float-Werte) sowie Boolesche Werte unter einem Schlüssel vom Typ QString speichern und referenzieren. Die Unterklasse *AnalysisType* fügt den Typ der Analyse (Online, Offline oder GPU-Analyse, siehe hierzu Abschnitt 8.1.4) in Form der Aufzählung (Enum) *AnalysisType* als Feld hinzu. Zusätzlich besitzt die Klasse *Analysis* noch ein Objekt vom Typ *Eventlogger*, die Ereignismeldungen verwaltet und an die GUI weiterleitet.

8.1.2. Das Paket *ImageServer*

Das Klassenpaket des *ImageServers* (siehe Abbildung 8.3) beinhaltet die für die Bildakquise notwendigen Klassen. Bei den Analysen wird grundsätzlich zwischen der Offline- und Online-Analyse unterschieden. Online bedeutet dabei, dass ein laufender Versuch mit der Kamera aufgenommen wird und dabei die mit dem Rechner verbundene Kamera zur Bereitstellung der Bilder dient. Eine Offline-Analyse bezieht Bilddaten eines bereits aufgenommenen Versuchs aus einem Ordner mit Bilddateien.

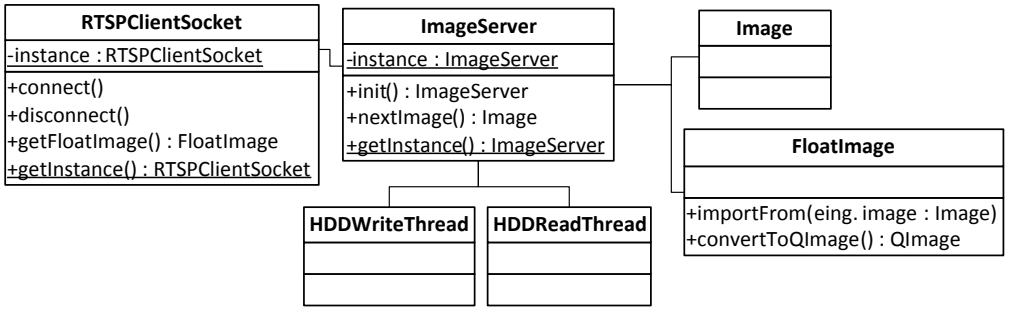


Abbildung 8.3.: Klassendiagramm des ImageServer-Pakets

Diese Unterscheidung nimmt dabei die Klasse *ImageServer* vor, die mit allen nötigen Komponenten verbunden ist. Das interne Format von Bilddaten ist in der Klasse *FloatImage* zu finden. Es handelt sich um eine zweidimensionale Matrix mit Fließkommazahlen (einfacher) Float-Genauigkeit[1], die aus der Bibliothek *FreeImage* stammt und mit sinnvollen Bildfunktionen (wie Konvertierungs- und Matrixrechenfunktionen) erweitert ist. Die Entscheidung, eine Repräsentation der Intensitätswerte intern mit Float-Zahlen durchzuführen, ist in der Flexibilität und Geschwindigkeit begründet. Eingangsbilddaten werden in 8- oder 16-Bit-Ganzzahlen (Integer) angeliefert, aber anstehende Rechnungen (wie etwa die Division durch ein Hintergrundbild) machen eine Konvertierung in Fließkommazahlen unumgänglich. Die einfache Genauigkeit reicht in diesem Fall aus, da keine iterativen Verfahren verwendet werden, bei denen sich Rundungsfehler potenzieren könnten. Zum Aufnehmen von Bildern der Kamera (Online-Analyse) wird die Klasse *RTSPClientSocket* verwendet. Diese kann eine Verbindung

[1]Float-Zahlen werden mit 32 Bit kodiert. Alternativ steht dazu das doppelt genaue Double-Format zur Verfügung, was allerdings auch doppelt so viel Speicher benötigt und Rechenoperationen verlangsamt.

zu einem RTSP-Stream der Elphel-Kamera aufnehmen und konvertiert die eingehenden Bilddaten aus dem JPG-Format des Kamera-Streams in FloatImage-Objekte, die an die Klasse *ImageServer* weitergeleitet werden.

Weiterhin gibt es über die Verbindung zu den beiden Klassen *HDDWriteThread* und *HDDReadThread* eine asynchrone Verbindung zur Festplatte. Dabei können bei der Bildaufnahme mit der Klasse *HDDWriteThread* eingehende FloatImage-Objekte (über den Dateinamen) durchnummeriert als TIF-Bilddateien in einem Ordner der Festplatte gespeichert werden. Bei der Offline-Analyse werden mit der Klasse *HDDReadThread* Bilddateien aus Ordnern auf der Festplatte gemäß Dateisortierung gelesen und in *FloatImage*-Objekte konvertiert.

8.1.3. Das Paket *Classification*

Das Klassenpaket *Classification* (siehe Abbildung 8.4) beinhaltet alle Klassen, die mit der Klassifikation zusammenhängen. Dazu zählen Datenquellenverwaltung, Sample-Datenbank und die Klassifikatoren. Die übergeordnete Klasse ist der *ClassificationController*, über den alle Klassen des Pakets erreichbar sind. Wie in Kapitel 5 beschrieben, sind Samples zum Training der Klassifikatoren notwendig. Diese Aufgabe wird von der Klasse *ClassificationDatabase* übernommen. Sie hat eine Verbindung zum *DataSourceController*, der Pfade zu Ordnern mit Bildern verwaltet. In einem Objekt der Klasse *DataSource* wird je ein Pfad mit Bilddateien verwaltet und mit einem eindeutigen Bezeichner (Identifier) in Verbindung gebracht. Dieser Bezeichner bleibt auch nach Umbenennen des Bildordners erhalten[2] und sorgt systemweit für eine konsistente Datenquellen-Zuordnung. Die Klasse *ClassificationDatabase* besitzt eine Liste von Objekten der Klasse *ClassificationSet*, die einer Datenquelle (*DataSource*) eine Menge an Samples zuordnet.

Diese von Benutzereingaben abhängigen Samples werden von der Klasse *PatternSample* dargestellt. Jedes Sample erhält einen eindeutigen Bezeichner sowie eine Klassenzuordnung in Form von zwei Zeichenketten (Haupt- und Unterklassenbezeichner, kurz „Hauptbezeichner.Subbezeichner“). So lassen sich unterschiedliche Klassen hierarchisch gruppieren, beispielsweise können Samples der Klassen „Partikel.80nm“ und „Partikel.200nm“ auch zusammengefasst betrachtet werden als alle Samples der Klasse „Partikel“. Darüber hinaus werden Anhaftungszeitpunkt und prozentuale Sprunghöhe (der Zeitreihen) der betreffenden Anhaftung festgehalten. Die geometrische Umrandung eines Samples ist eine entscheidende Information, die in der Klasse *SelectionRegion* erfasst wird. Diese wiederum erweitert die Klasse *GeometryInformation* für eine Benutzerinteraktion (Selektionsfunktionen). Die Klasse *GeometryInformation* ist dabei eine abstrakte Klasse mit einheitlicher Schnittstelle für die Unterklassen *GeometryPoint*, *GeometryRectangle*, *GeometryCircle* und *GeometryPolygon*. Hier werden je nach gewählter Geometrie die notwendigen Daten gespeichert (Mittelpunkt bei Typ Punkt, Mittelpunkt und Radius bei Typ Kreis, Eckkoordinaten bei Typ Rechteck und eine Punktliste bei Typ Polygon). Ein *PatternSample* besitzt ein Objekt der Klasse *SampleFeatures*, das eine Menge an Bildmatrizen (vom Typ *FloatImage*) aufnehmen kann. Dort werden die bildhaften Ausprägungen der Anhaftung gespeichert.

[2] Der Identifier wird in einer XML-Datei im entsprechenden Pfad mit den Bilddateien abgelegt.

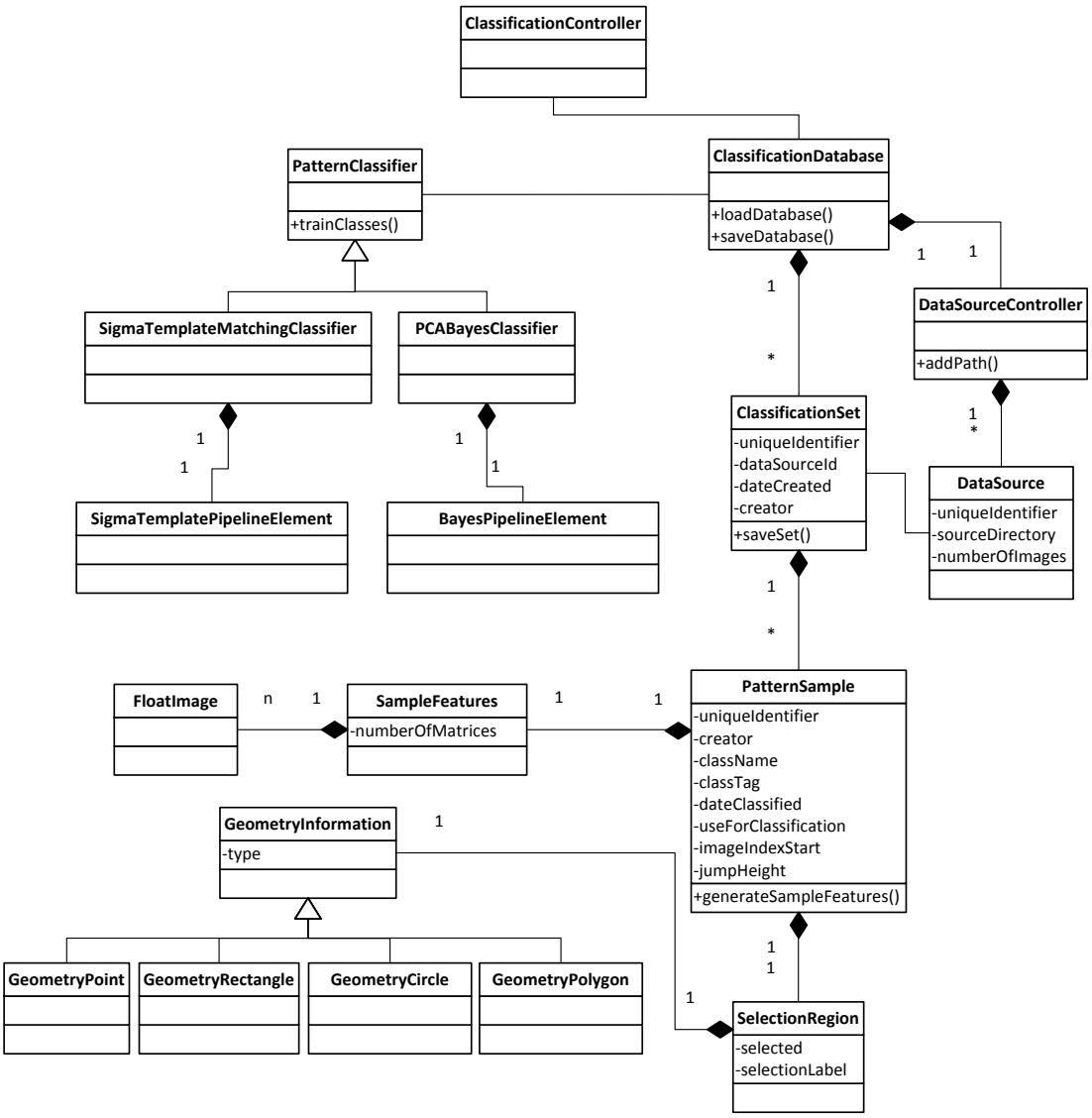


Abbildung 8.4.: Klassendiagramm des *Classification*-Pakets

Die Klasse *PatternClassifier* ist eine abstrakte Klasse und Basis für alle im PAMONO-Explorer implementierten Klassifikatoren. Über die Verbindung zur Klasse *ClassificationDatabase* wird der Zugriff auf alle Samples ermöglicht. Die abgeleiteten Klassen *SigmaTemplateMatchingClassifier* und *PCABayesClassifier* implementieren die Klassifikatoren, die in Kapitel 5 beschrieben sind. Die Unterklassen setzen dabei die Methoden *trainClasses()* und *classify()* jeweils unterschiedlich um. Durch die Vorgabe des Benutzers lassen sich die zu erkennenden Klassen und die letztendlich verwendeten Samples vor dem Training bestimmen. Es ist außerdem möglich, die Subklasseneinteilung zu deaktivieren, um die unterschiedlichen Ausprägungen von Partikeln zu verallgemeinern. Die Verbindung zur Analyse-Pipeline besteht in der Hinsicht, dass die CPU- und GPU-Pipeline-Elemente (siehe nächster Abschnitt) jeweils das passende

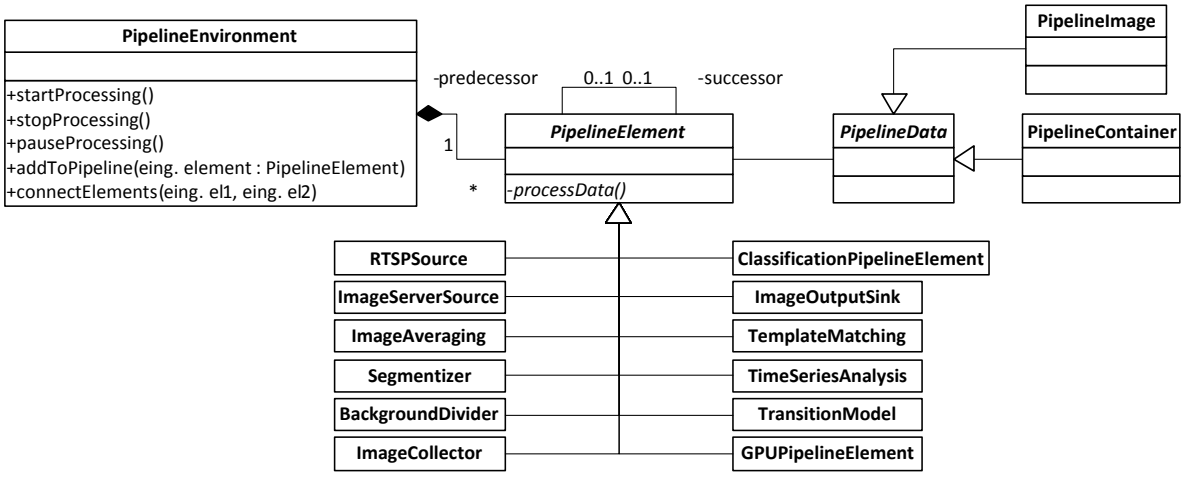


Abbildung 8.5.: Reduziertes Klassendiagramm der Pipeline-Architektur, das die für die Pipeline-Verarbeitung wichtigen Klassen zeigt.

Klassifikatorobjekt verwalten und zur Klassifikation nutzen.

8.1.4. Das Paket *Pipeline*

Die softwaretechnische Realisierung der Pipeline ist sehr nah an der in Kapitel 6.4 beschriebenen Pipeline-Architektur gehalten. Die Klasse *PipelineElement* ist eine abstrakte Oberklasse, von der die konkreten Implementierungen der Pipeline-Elemente erben (Abbildung 8.5). Die abstrakte Klasse *PipelineData* bietet einen einheitlichen Datentyp für die Datenobjekte der Pipeline. In der im Pamono-Explorer implementierten Pipeline sind die Datenobjekte *FloatImages*, die in *PipelineImage* gekapselt sind. Das *PipelineEnvironment* ermöglicht den Aufbau und die Kontrolle der Pipeline. Es bietet Funktionen, um Pipeline-Elemente hinzuzufügen und zu verbinden, sowie die Verarbeitung zu starten, stoppen und zu pausieren.

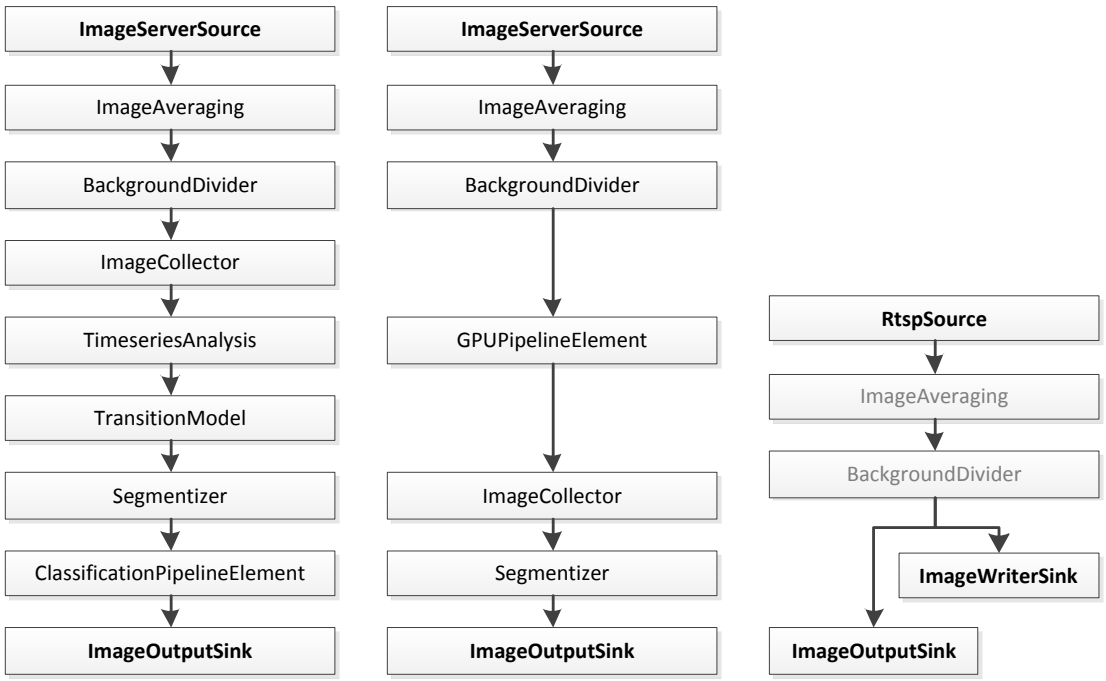
Jedes Pipeline-Element läuft in einem eigenen Thread. Dazu erbt die Klasse *PipelineElement* von *QThread*, einer QT-Klasse, die plattformunabhängige Thread-Funktionalität bereitstellt. Threads laufen parallel und können auf mehrere Rechenkerne verteilt werden. Daher nutzt der Pamono-Explorer Multicore-Prozessoren sehr gut aus und erreicht eine hohe Verarbeitungsgeschwindigkeit.

Die folgenden Pipeline-Elemente wurden implementiert:

- **ImageServerSource:** Source-Element für Festplattenordner.
- **RtspSource:** Source-Element für die Elphel-Kamera und andere RTSP-Streams.
- **ImageAveraging:** Element für die Bildmittelung (Kapitel 2.2.1).
- **BackgroundDivider:** Element für die Hintergrunddivision (Kapitel 2.3.2).

- **TimeSeriesAnalysis:** Sprungerkennung in den Zeitreihen (Kapitel 3).
- **TransitionModel:** Realisiert das Zustandsmodell der Zeitreihenanalyse (Kapitel 3.5).
- **ClassificationPipelineElement:** Klassifizierung mit Bayes-Klassifikator oder per Sigma-Template-Matching (Kapitel 5).
- **ImageCollector:** Puffer-Element für die Anzeige von Zeitreihen.
- **Segmentizer:** Partikel-Segmentierer (Kapitel 4).
- **GPUPipelineElement:** Pipeline-Element für die Analyse auf der GPU.
- **ImageOutputSink:** Sink-Element, das die GUI benachrichtigt, wenn ein Bild fertig verarbeitet wurde.
- **ImageWriterSink:** Sink-Element, das ankommende Bilder auf der Festplatte speichert.

Im PAMONO-Explorer kommen drei verschiedene Pipelines zum Einsatz: Es gibt zwei Pipelines zur Analyse der PAMONO-Bilddaten, eine ist CPU-basiert (Abbildung 8.6(a)), die andere führt die Berechnungen auf der GPU aus (Abbildung 8.6(b)), wie in Kapitel 9 beschrieben wird. Die dritte Pipeline (Abbildung 8.6(c)) ermöglicht die Aufnahme von Bildern mit der Elphel-Kamera und besitzt optional zuschaltbare Elemente zur Bildmittelung und Hintergrunddivision.



(a) CPU-Pipeline (b) GPU-Pipeline (c) Kamera-Aufnahmepipeline

Abbildung 8.6.: Der Aufbau der Pipeline zur CPU- und GPU-basierten Partikeldetektion, sowie zur Aufnahme von Bildreihen mit der Kamera.

8.1.5. Das Paket *ProjectManager*

Die zentrale Klasse bei der Realisierung der Projektverwaltung des PAMONO-Explorer ist die Klasse *Project*. Neben dem Projektnamen, dem Pfad, in dem das Projekt gespeichert ist, und dem Erstellungsdatum verwaltet ein Objekt der Klasse *Project* die aktuellen Einstellungen des jeweiligen Projekts, den aktuellen Inhalt des Projektordners (insbesondere aller Ordner, in denen Analyseergebnisse gespeichert sind) sowie die Ergebnisdateien selbst. Die Klasse *Project* stellt eine Reihe von Methoden bereit, die es erlauben, den Namen und Pfad des Projekts zu ändern. Ferner können die aktuellen Einstellungen des Projekt gespeichert (*saveSettings*) und geladen (*loadSettings*) werden. Außerdem stellt die Klasse *Project* weitere wichtige Funktionalitäten in Bezug auf die Bilddatenanalyse durch Methoden bereit, mit denen neue Analyseergebnisse in einem separaten Ordner in das aktuelle Projekt eingefügt (*addAnalysisResult*) und vorhandene Ordner mit Analyseergebnissen (*getAnalysisResultFolders*), die Analyseergebnisse eines Ordners (*getAnalysisResults*) sowie eine Datei mit Analyseergebnissen (*getAnalysisResult*) abgefragt werden können. Des Weiteren bietet die Klasse *Project* die Möglichkeit, den Zielpfad innerhalb des aktuellen Projekts abzufragen, in dem die Analyseergebnisse in Form von CSV-Dateien exportiert werden.

Um Projekte laden und speichern zu können, stellt das Paket *ProjectManager* die Klasse *ProjectIO* zur Verfügung. Diese Klasse implementiert neben der Lade- und Speicherfunktion (*read* und *write*) auch zwei weitere Methoden, mit denen ein neues Projekt angelegt (*create*) und ein bestehendes Projekt umbenannt (*rename*) werden kann.

Zusätzlich existiert in der Klassenstruktur der Projektverwaltung die Klasse *ProjectManager*, die lediglich den Pfad zu dem Projekt speichert, das zuletzt geöffnet worden ist. Auf diese Weise kann der Benutzende nach dem Start des PAMONO-Explorer das Projekt öffnen, an dem er zuletzt gearbeitet hat.

Aufgrund der Tatsache, dass viele Elemente eines Projekts wie die die Projektdatei und die Dateien, in denen die Projekteinstellungen gespeichert sind, in Form von XML-Dateien verwaltet werden, stellt das Paket *ProjectManager* eine Klasse *XMLDocument* zur Verwaltung von XML-Inhalten mittels *Document Object Model* (DOM) bereit. DOM ist eine Spezifikation einer Schnittstelle, über die auf XML-Inhalte zugegriffen werden kann.

Mit Hilfe der Klasse *XMLIO* können die Inhalte der Objekte vom Typ *XMLDocument* gespeichert (*write*) und geladen (*read*) werden.

Viele der bisher aufgezählten Klassen aus dem Paket *ProjectManager* verwenden Methoden, die nach ihrem Aufruf über Rückgabewerte signalisieren, ob die Methode erfolgreich abgearbeitet werden konnte oder ob Fehler aufgetreten sind. Als Datentyp für diese Rückgabewerte wird der Aufzählungstyp *ReturnCode* verwendet. Dieser definiert möglichen Werte, die bestimmte Fehler repräsentieren, wie zum Beispiel das Öffnen einer nicht vorhandenen Datei.

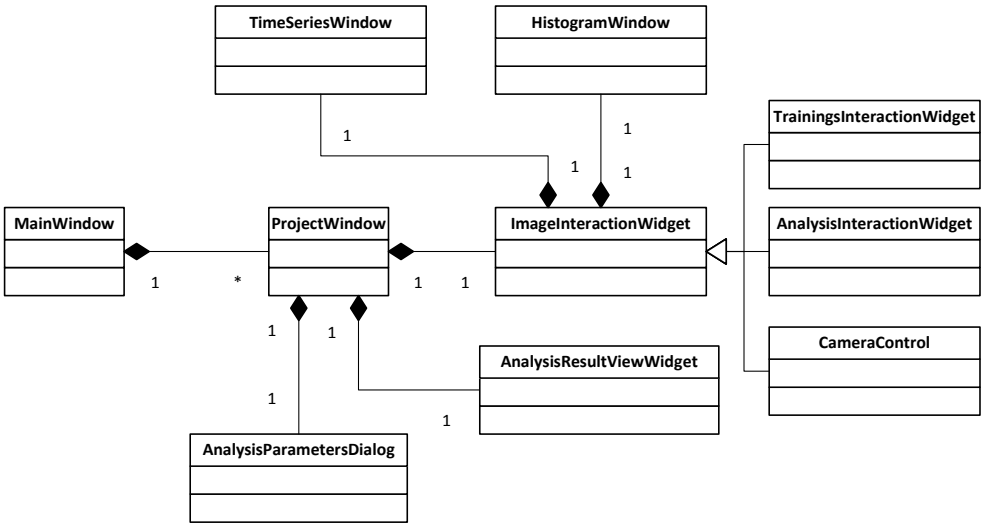


Abbildung 8.7.: Reduziertes Klassendiagramm der wichtigsten Klassen der grafischen Benutzeroberfläche

8.1.6. Das Paket *GUI*

Die Klassen für die grafische Benutzeroberfläche des PAMONO-Explorers befinden sich im Paket *GUI* (Graphical User Interface). In Abbildung 8.7 sind die wichtigsten Klassen und ihre Verbindungen untereinander zu sehen. Jede Instanz des PAMONO-Explorers besteht aus einem Objekt der Klasse *MainWindow*, welches eine Menge an aktiven Projektfenstern verwaltet. Die Klasse *ProjectWindow* repräsentiert ein Projektfenster und enthält die zentrale Benutzeroberfläche für den gesamten Prozess der PAMONO-Analyse. Dazu gibt es die abstrakte Klasse *ImageInteractionWidget*, die alle grafischen Interaktionsmöglichkeiten für die PAMONO-Bilddaten bereitstellt. Dazu zählen Funktionen zum Anzeigen der Bilddaten, zur Selektion von Regionen innerhalb der Bilder, für die Zeitreihen- und Histogramm-Anzeige. Die beiden Klassen *TimeSeriesWindow* und *HistogramWindow* übernehmen dabei die letzten beiden Funktionen. Von der Oberklasse *ImageInteractionWidget* sind Unterklassen abgeleitet, die die wichtigen Programmfunktionen des PAMONO-Explorer implementieren:

- ***TrainingInteractionWidget:*** Um Samples anzutrainieren, müssen Bildregionen markiert und Klassenlabels zugeordnet werden. Die dafür nötigen Funktionen sind in der Klasse *TrainingInteractionWidget* zu finden. Als weitere Funktion ermöglicht es diese Klasse auch, durch alle Bilder einer Bildfolge zu navigieren, um die Anhaftungen manuell erfassen zu können.

- ***AnalysisInteractionWidget:*** Um einen laufenden Analyseprozess beobachten zu können, müssen Bilddaten und Fundstellen sichtbar gemacht werden. Diese Aufgabe übernimmt die Klasse *AnalysisInteractionWidget*, die zusätzlich noch eine Übersicht über die aktuell gezählten Funde zu allen verwendeten Klassenlabels anzeigt.

- ***CameraControl:*** Diese Klasse erweitert die Oberklasse um Funktionen für die Kamerasteuerung, -vorschau und -aufnahme. Es ist möglich, den Status der Kamera einzusehen und Parameter über das Netzwerk zu senden. Die Aufnahmefunktion bestimmt einen Pfad, in dem die Bilddaten der Kamera gespeichert werden sollen sowie eine maximale Aufnahmezeit.

Die Klasse *AnalysisResultViewWidget* übernimmt die Funktion der Anzeige der Ergebnisstatistiken einer Analyse. Dort werden die von jeder Analyse angelegten Ergebnisordner angezeigt (sofern diese im selben Projekt durchgeführt wurden) sowie Export- und Plot-Funktionen von wichtigen Daten bereitgestellt (Anzahl Anhaftungen über die Zeit, Anzahl Anhaftungen über die Größe der Anhaftungen).

Weiterhin ist an das Projektfenster (Klasse *ProjectWindow*) die Klasse *AnalysisParameters-Dialog* angeschlossen. Hierbei handelt es sich um ein Fenster, in dem alle wichtigen Parameter für die Analysevorgänge eingestellt, gespeichert und geladen werden können.

8.2. Bibliotheken und Hilfsmittel

Während der Entwicklungsphase eines Softwareprojekts werden neben einer Programmiersprache Bibliotheken und weitere Hilfsmittel, wie zum Beispiel Editoren für den Entwurf grafischer Benutzeroberflächen (GUI), eingesetzt.

In diesem Kapitel werden die Bibliotheken und Hilfsmittel vorgestellt, die während der Entwicklung des Pamono-Explorers eingesetzt wurden. Sie sind in die Bereiche *Software* (Kapitel 8.2.1), *Kamera* (Kapitel 8.2.2) und *GPU* (Kapitel 8.2.3) eingeteilt.

8.2.1. Software

Die folgende Aufzählung umfasst alle im Softwarebereich eingesetzten Bibliotheken und Hilfsmittel. Der Softwarebereich umfasst die Entwicklung des Pamono-Explorers ohne die Unterstützung durch die GPU und ohne Implementierungen *auf* der Kamera.

- **C++**

  Für die konkrete Realisierung des Pamono-Explorers wurde die Programmiersprache C++ eingesetzt. Da die digitale Bildverarbeitung großer Bilddatenmengen einen wesentlichen Teil des Projekts darstellt, sind Ausführungsgeschwindigkeit des Programms sowie dessen Platzverbrauch im Hauptspeicher wichtige Aspekte. Die Programmiersprache C++ eignet sich für die Entwicklung des Pamono-Explorers, da Quellcode vom Compiler in plattformspezifischen Maschinencode übertragen wird. In C++ erzeugter Code ist aus diesem Grund von der jeweiligen Plattform direkt ausführbar und es sind keine zusätzlichen Programme wie zum Beispiel Interpreter oder virtuelle Maschinen notwendig, die die Ausführungsgeschwindigkeiten reduzieren und zusätzlichen Speicherplatz erfordern

würden. C++ zeichnet sich insbesondere durch hohe Ausführungsgeschwindigkeiten sowie durch Kompaktheit des erzeugten Maschinencodes aus.

Des Weiteren stellt C++ Sprachelemente für die objektorientierte Programmierung bereit. Dies ist ein Vorteil für die Realisierung eines Softwareprojekts, da die objektorientierte Softwareentwicklung eine hohe Modularität ermöglicht.

- **Qt**

  Bei Qt handelt es sich um eine umfassende Bibliothek für die C++, die neben vielen Funktionsbibliotheken auch der Programmierung von grafischen Benutzeroberflächen (GUI) dient. Sie stellt eine große Anzahl an Klassen bereit, die die grundlegenden GUI-Elemente, wie zum Beispiel Schaltflächen, Listen, Menü sowie Fensterrahmen, zur Verfügung stellen. Des Weiteren ermöglicht die Bibliothek unter anderem die Netzwerkwerkprogrammierung sowie Multithreading für nebenläufige Anwendungen.

  Für Qt sind neben der Entwicklungsumgebung (QtCreator) inklusive eines GUI-Editors eine Reihe von zusätzlichen Werkzeugen erhältlich, die zum Beispiel das Entwickeln von mehrsprachigen Anwendungen ermöglichen.

  Qt ist ferner für eine Vielzahl unterschiedlicher Betriebssysteme verfügbar und zeichnet sich durch ihre Portabilität aus. Dies bedeutet, dass der Quellcode eines mit Qt geschriebenen Programms auf ein anderes Betriebssystem übertragen und übersetzt werden kann, ohne das Änderungen am Quellcode vorgenommen werden müssen.

- **Qwt**

  Qwt ist eine Bibliothek für C++, die auf der GUI-Bibliothek von Qt basiert. Sie stellt neben Qt einen Satz von zusätzlichen Elementen für grafische Benutzeroberflächen bereit. Das Erzeugen und Visualisieren von Diagrammen, wie zum Beispiel in den Anzeigen von Zeitreihen, ist das Hauptanwendungsgebiet von Qwt im Rahmen des Softwareprojekts Pamono-Explorer.

- **FreeImage**

  Bei FreeImage handelt es sich um eine C++ Bibliothek , mit deren Hilfe Bilder geladen, verarbeitet und gespeichert werden können. FreeImage unterstützt eine Vielzahl unterschiedlicher Bildformate.

  Einen wesentlichen Teil des Pamono-Explorers stellt die digitale Verarbeitung von Bildern dar. Daher wird FreeImage im Rahmen der Verarbeitung von Bildern im JPEG- und TIFF-Format eingesetzt und um Bilder auf die Festplatte zu schreiben und von ihr zu lesen.

- **GStreamer**

  Das Framework GStreamer stellt grundlegende Funktionen zur Verfügung, mit denen Multimedia-Datenströme wie zum Beispiel Video- oder Audioströme verarbeitet werden

können. Somit kann GStreamer als Basis für die Entwicklung eigener Wiedergabe- oder Bearbeitungsprogramme für Videos und Musik verwendet werden. Ferner baut GStreamer auf einer Plug-in Architektur auf, so dass GStreamer durch Plug-ins auf einfache Art und Weise erweitert werden kann. Im Rahmen des PAMONO-Explorers wird GStreamer für die Aufnahme von Bildreihen, die von der Kamera mittels RTSP übermittelt werden, eingesetzt.

- **Armadillo**

  Die Bibliothek Armadillo stellt einen Satz von Operationen für die lineare Algebra zur Verfügung, wie zum Beispiel die Rechnung mit Matrizen. Unterstützt werden dabei ganze Zahlen, Gleitkommazahlen einfacher und doppelter Präzision, sowie komplexe Zahlen. Armadillo zeichnet sich durch eine hohe Ausführungsgeschwindigkeit und einfaches API aus.

8.2.2. Kamera

Die Programmierung des FPGA sowie die Parametereinstellung der Kamera sind dem Kamerabereich zugeordnet. Nachfolgend werden alle in diesem Rahmen angewendeten Werkzeuge vorgestellt.

- **Verilog**

  Verilog ist eine Sprache zur Beschreibung von Hardware auf einer höheren Abstraktionsebene. Sie kann zum Beispiel eingesetzt werden, um das Verhalten eines FPGA zu beschreiben. Die Hardwarebeschreibung geschieht textbasiert und ist vergleichbar mit der Programmierung in einer Programmiersprache wie beispielsweise C++. Verilog wurde zur Programmierung des FPGAs der Kamera verwendet (siehe Kapitel 7).

- **VHDL**

  Neben Verilog existiert zur Beschreibung von Hardware die Sprache VHDL. Mit VHDL lässt sich das Verhalten von Hardware genau wie bei Verilog textbasiert beschreiben. VHDL wurde ebenfalls zur Programmierung des FPGAs der Kamera verwendet.

- **Xilinx ISE und ISim**

  Bei der Xilinx ISE handelt es sich um eine Software zur Synthese und Analyse von Hardwareentwürfen, die mittels einer Hardwarebeschreibungssprache wie zum Beispiel Verilog oder VHDL erstellt worden sind. Zur Simulation und Verifikation einer entworfenen Hardware steht zusätzlich das Werkzeug ISim zur Verfügung.

- **GTKWave**

  GTKWave ist ein Programm, welches im Rahmen des Entwurfs elektronischer Schaltungen eingesetzt wird. Es dient dazu, digitale Signalformen wie in einem Oszillographen zu

visualisieren. GTKWave unterstützt auch die Visualisierung von VCD/EVCD-Dateien, den Standarddateiformaten der Hardwarebeschreibungssprache Verilog.

- **PHP**

  PHP ist eine Skriptsprache, die vor allem zur Generierung dynamischer Webseiten verwendet wird. Die Elphel-Kamera, die für den PAMONO-Explorer verwendet wird, bietet eine PHP-basierte Schnittstelle, über die die Kameraparameter sowohl gelesen als auch verändert werden können.

8.2.3. GPU

Für die digitale Bildverarbeitung auf der GPU wird das Framework OpenCL (siehe Kapitel 9) eingesetzt. Es bietet eine einheitliche Bibliothek und Programmiersprache zur parallelen Programmierung von Anwendungen für verschiedene Plattformen und Architekturen (CPU/GPU/Cell).

8.2.4. Sonstige Bibliotheken und Hilfsmittel

Nachfolgend werden alle Bibliotheken und Hilfsmittel aufgezählt, die keinem der Bereiche *Software*, *Kamera* oder *GPU* zuzuordnen sind.

- **Matlab**

  Matlab ist eine Software zur numerischen Lösung mathematischer Probleme. Die eingebaute Skript-Sprache und viele verschiedene Toolboxen schaffen eine Umgebung, in der Algorithmen schnell geschrieben und ausprobiert werden können. Darüber hinaus gibt es umfassende Möglichkeiten zur Auswertung und Visualisierung von Daten. Matlab wurde während der Experimentierphase vor der Implementierung des PAMONO-Explorers eingesetzt, um Verfahren für die Bildverarbeitung zu testen.

- **XML**

  Bei XML (Abkürzung für *Extensible Markup Language*) handelt es sich um einen Regelsatz für die Kodierung von Dokumenten, so dass dabei auf Binärcode verzichtet wird und das Dokument dennoch maschinenlesbar ist. XML wird im Rahmen der Entwicklung des PAMONO-Explorers zur Konfiguration der Kameraparameter über das Netzwerk sowie zur Speicherung von Projekten eingesetzt.

9. GPU

In der Bildverarbeitung sind große Datenmengen zu verarbeiten. Dies ist mit einem hohen Rechenaufwand verbunden. Um die Daten möglichst schnell zu verarbeiten, lohnt sich der Einsatz der hohen Rechenleistung von Grafikkarten (siehe dazu auch Kapitel 9.1).

Dieses Kapitel geht auf die Architektur von Grafikkarten ein (Kapitel 9.1), erläutert die Grundprinzipien von OpenCL (Kapitel 9.2), einer Programmiersprache für Grafikkarten, und beschreibt wie die Bildverarbeitung auf der GPU realisiert werden kann (Kapitel 9.3).

9.1. GPU Architektur

Im Folgenden wird die Architektur einer Grafikkarte beschrieben. Es werden Möglichkeiten und Einschränkungen bei der Nutzung von Grafikkarten für generelle Probleme und die Aufgaben der Projektgruppe aufgezeigt.

Bestimmte Komponenten einer Grafikkarte, wie das Bus-Interface, die Speicherarchitektur oder die GPU, beeinflussen die Berechnungen allgemeiner Probleme sehr stark. So muss bspw. darauf geachtet werden, dass Speicherzugriffe optimiert werden. Die Größe des globalen Speichers aktueller Grafikkarten ermöglicht es, große Datenmengen darin abzulegen, beispielsweise die zu analysierenden Bildreihen.

Die Bandbreite für den Zugriff auf den (internen) Speicher der Grafikkarte ist um ein Vielfaches höher als bei CPUs, sie wird aber durch die Bandbreite des Bus-Interfaces beschränkt. Der schnellste verfügbare Speicher ist der relativ kleine interne Speicher der GPU. Der *Shared Memory* liefert einen Datendurchsatz von 1 TB/s und mehr, der *Global Memory* hingegen nur etwa bis zu 177GB/s. DDR3 Arbeitsspeicher liefert Spitzentransferraten von 17GB/s (PC3-17000 Modul).

Wichtige Bestandteile der GPU sind die *Streaming (Multi-)Processors*. Die Streaming Prozessoren (SPs) werden als „Shader“ bezeichnet. Der Begriff wird auch für Software-Shader verwendet, kleine Programme, die auf den Streaming Prozessoren ausgeführt werden. Die Streaming Prozessoren werden auf der Grafikkarte zu *Streaming Multiprocessors* zusammengefasst. Ein Streaming Multiprocessor besteht (bei NVIDIA Produkten) aus acht oder mehr Streaming Prozessoren, zwei *Special Function Units* und einem lokalen Speicher (Shared Memory).

Streaming Prozessoren sind eigene Recheneinheiten, die Integer- und Floating-Point-Operation durchführen können. Jeder Einzelne hat ein Register, auf das äußert schnell, jedoch nur von dem jeweiligen SP aus zugegriffen werden kann. Bei NVIDIAs G92b Chip ist das Register

der SPs 2 KB groß, wobei die SPs mit einer Frequenz von ca. 1,8 GHz getaktet sind. Für aufwendigere, seltener auf einer Grafikkarte verwendete Operationen, wie der Sinusfunktion, sind die Special Function Units verfügbar.

Der lokale (shared) Speicher kann von allen SPs eines Streaming Multiprocessors aus verwendet werden. Er ist ein Tightly-Coupled-Speicher mit expliziten Load und Store Operationen. Hauptaufgabe dieses Speichers ist der Ersatz des Caches. Beim G92b Chip umfasst der lokale Speicher der Streaming Multiprocessors 16 KB.

Bei der Programmierung ist darauf zu achten, möglichst viele Operationen mit Daten aus dem schnelleren Speicher der Grafikkarte auszuführen.

9.2. OpenCL

Dieses Kapitel erläutert einige wichtige Konzepte des C-Dialektes OpenCL, das Plattformmodell, das Programmiermodell, das Ausführungsmodell und das Speichermodell.

9.2.1. Eigenschaften von OpenCL

Die Spezifikation der *Open Computing Language* (OpenCL) ist offen und wird von der Khronos Group gepflegt. OpenCL beinhaltet keine Annahmen über die Beschaffenheit von Geräten und ist prinzipiell portabel. Mit Hilfe von OpenCL wird spezialisierte Hardware wie eine Grafikkarte für Berechnungen allgemeiner Probleme verfügbar. Die Rechenperformanz von Grafikkarten resultiert nicht aus hohen Taktraten, sondern aus einer großen Anzahl von Kernen, die besonders einfach aufgebaut sind, sowie einer hardwareseitigen Ablaufsteuerung. Zielsetzung ist ein einfaches Berechnungsmodell und ein möglichst einfach zu verwendendes API.

Die Implementation von Berechnungen mit OpenCL bietet sich besonders bei Problemen mit hoher Datenparallelität an, ist aber im Gegensatz zu *Cuda* oder *Stream* auch für Taskparallelität geeignet. Datenparallelität bedeutet, dass eine oder mehrere Operationen in gleicher Weise auf mehreren Datenelementen unabhängig ausgeführt wird. Davon abzugrenzen ist Task-Parallelität. Sie bezeichnet das unabhängige Ausführen von mehreren Tasks. OpenCL bietet keinerlei Vorteile für Probleme sequentieller Art, Berechnungen die viel Kommunikation oder viel Zeigerarithmetik bedürfen. Die Zeitreihenanalyse pro Pixel der Bildreihen ist ein datenparalleles Problem.

9.2.2. Plattformmodell

OpenCL unterscheidet zwischen dem *Host* und den *Compute Devices*. Host bezeichnet den eigentlichen Rechner, der ein oder mehrere Compute Devices enthält bzw. mit ihnen verbunden ist (siehe Abbildung 9.1). Compute Devices können Grafikkarten, CPUs oder Beschleuniger-

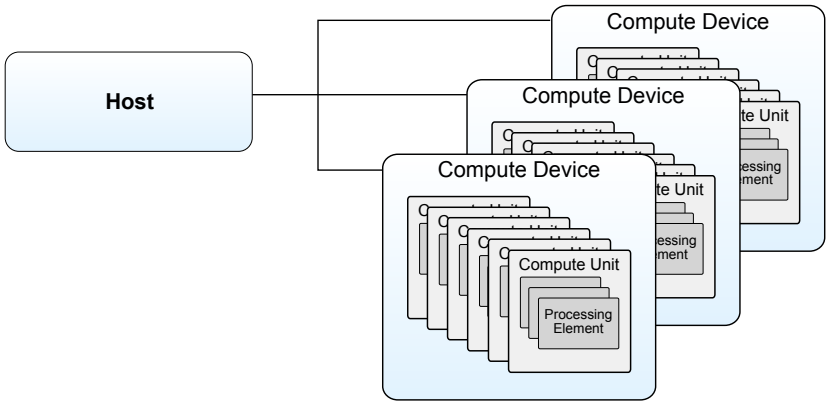


Abbildung 9.1.: Das OpenCL Plattformmodell

karten wie das *Cell Accelerator Board* sein. Ein Compute Device umfasst *Compute Units*, die weiter in *Processing Elements* unterteilt sind. Bei einer Grafikkarte entsprechen die Streaming Multiprocessors den Compute Units, während die Processing Elements die Streaming Processors (Shader) darstellen. 9.1 stellt die Verbindung des Host mit den Compute Devices dar. Sie unterteilen sich in Compute Units, die wiederum in Processing Elements unterteilt sind.

9.2.3. Ausführungsmodell

Ein OpenCL Programm besteht aus zwei Teilen. Zum einen gibt es das Host-Programm: Es entspricht einer normalen C/C++ Anwendung. Das Host-Programm ist für die Verwaltung der Compute Devices zuständig. Es reserviert und befüllt den Speicher der Compute Devices und führt die sogenannten *Kernels* aus.

Die Kernels sind der zweite Teil des OpenCL Programms. Es sind kleine Programme, die auf den Compute Units ausgeführt werden. Kernel werden meist getrennt vom Host-Code in separaten Dateien gespeichert. OpenCL Kernel unterstützen den Sprachstandard ANSI-C99 und bieten darüber hinaus weitere Datentypen und Funktionen. Sie werden mit dem „kernel“ Qualifizierer markiert. Der Source Code für die Kernel wird vom meist vom Host-Programm geladen, für die Compute Devices kompiliert und gestartet. Codeausschnitt 9.1 zeigt ein Beispiel wie der Rahmen eines Kernels aussehen kann.

Im Folgenden wird gezeigt wie ein Kernel geladen wird und was Context, Command Queue und Work Groups sind.

```
1 kernel void kernelName(...)
2 {
3     ...
4 }
```

Codeausschnitt 9.1: Rahmen eines Kernels: Der Kernel wird durch den „kernel“ Qualifizierer markiert.

9.2.3.1. Laden eines Kernels mit C++ Bindings für OpenCL

Im Gegensatz zu herkömmlichen C/C++ Programmen werden OpenCL Kernel üblicherweise zur Laufzeit des Host-Programms kompiliert. Das Host-Programm lädt den Kernel-Code, der mehrere Kernel enthalten kann, aus einer Textdatei in einen String, der über ein *Sources-Objekt* an den Konstruktor für ein *Program-Objekt* gegeben wird. Dieses Program-Objekt kann den Code kompilieren und aus ihm kann das *Kernel-Objekt* erstellt werden .

9.2.3.2. Context

Ein *Context-Objekt* beschreibt die Arbeitsumgebung. Es wird mit einer Liste von Devices (als Konstruktor Argument) erstellt und wird zur Verwaltung von Speicherobjekten, Program-Objekten, Command Queues und Kernel benötigt. Die *Device-Objekte* können über die Plattform geholt werden. Diese werden für die Konstruktion des Context-Objektes benötigt.

9.2.3.3. Command Queue

Die *Command Queue* koordiniert die Ausführung von Kernels, das Lesen bzw. Schreiben aus/in den Speicher und die Synchronisation über *Command Queue Barriers*. Eine Command Queue Barrier bewirkt, dass alle vorher an die Command Queue übergebenen Kommandos ausgeführt werden, bevor das Host-Programm weiter läuft.

9.2.3.4. Work Groups

Instanzen eines Kernels werden in sogenannten *Work Groups* organisiert. Dabei wird jede Instanz als *Work Item* bezeichnet. Eine Work Group enthält in bis zu drei Dimensionen eine Anzahl von Work Items. Der gesamte Indexraum enthält wiederum in ein bis drei Dimensionen eine Anzahl an Work Groups. Das bedeutet, dass die globale Anzahl an Work Items in einer Dimension teilbar sein muss durch die Anzahl der Work Groups in der gleichen Dimension. Der Indexraum wird auch als *NDRange* (*N-Dimensional Range*) bezeichnet. In diesem hat jede Work Group einen Index. Work Items haben einen lokalen Index (innerhalb der Work Group) und einen globalen Index. Jeder Index ist ein N-Tupel. Eine Work Group wird immer in einer einzelnen Compute Unit ausgeführt. Abbildung 9.2 zeigt wie eine zweidimensionale NDRange aussehen kann.

9.2.4. Speichermodell

In OpenCL wird zwischen vier verschiedenen Speicherbereichen unterschieden. *Global Memory* und *Constant Memory* befinden sich im *Compute Device Memory* (dem Arbeitspeicher der Grafikkarte). Work Items können auf beide Speicherbereiche zugreifen. Daten im Constant Memory können von Work Items nicht verändert werden. Sie werden in der GPU gecached

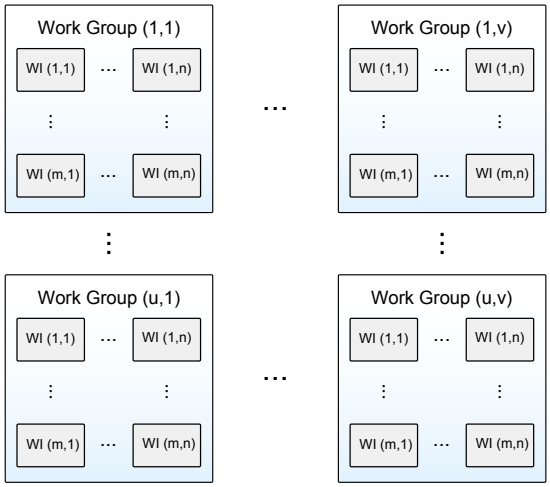


Abbildung 9.2.: Eine 2-dimensionale NDRange: Die Work Items $W(i, j)$ sind in Work Groups (p, q) unterteilt. Jede Work Group hat die gleiche Anzahl an Work Items in beiden Dimensionen.

und sind ähnlich schnell abrufbar wie aus dem lokalen Speicher. Jede Compute Unit verfügt über ein *Local Memory* (Shared Memory eines Streaming Multiprozessors), einem kleinen[1], sehr schnellen Speicherbereich, der nur von seiner Compute Unit aus zugreifbar ist. Weiterhin verfügt jedes Processing Element über einen *Private Memory* (Register eines SPs), der ebenfalls klein aber sehr schnell ist. Abbildung 9.3 zeigt die Aufteilung des Speichermodells. Bei der Verwendung von OpenCL muss besonders darauf geachtet werden, dass es Datenkonsistenz nicht zusichert. Das kann zu Problemen beim Zugriff auf Daten an den Rändern einer Work Group führen.

9.3. Bildverarbeitung auf der GPU

In den vorigen Abschnitten wurde OpenCL und die Architektur von Grafikkarten beschrieben. Dieses Unterkapitel geht auf die Frage ein, wie die Bildverarbeitung auf der Grafikkarte softwaretechnisch realisiert werden kann.

In 9.3.1 wird die Interfaceklasse zur Grafikkarte konzeptuell beschrieben. 9.3.2 erklärt wie die zu analysierenden Daten auf der Grafikkarte verwaltet werden. Kapitel 9.3.3 geht auf die Integration in die Pipeline ein (siehe dazu auch Kapitel 8).

[1] Bei dem G92b Chip zum Beispiel 16k

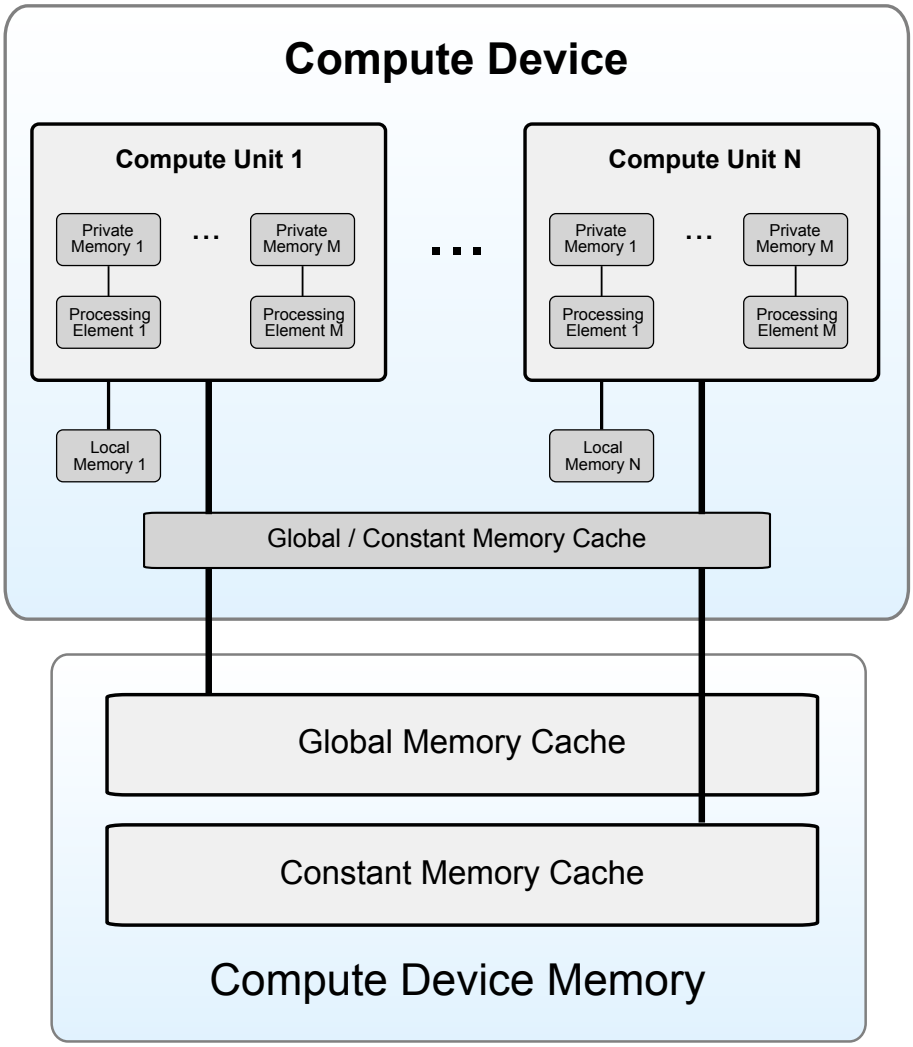


Abbildung 9.3.: Das OpenCL Speichermodell: Global und Constant Memory befinden sich im Compute Device Memory. Jede Compute Unit hat einen Local Memory, das von jedem PE aus zugreifbar ist. Jedes PE hat weiterhin ein Private Memory.

9.3.1. Interface

Ein allgemeines Interface zur Grafikkarte (siehe Abbildung 9.4) bildet die *GenericCL*-Klasse. In ihr werden alle OpenCL Befehle gekapselt. Mit der Klasse können Puffer angelegt und Kernel instanziiert und ausgeführt werden. Auch ist das Herab- bzw. Hochladen von Daten der Puffer auf der Grafikkarte möglich. Um Puffer und Kernel möglichst einfach und generisch erstellen und benutzen zu können, sind zwei Beschreibungsklassen *KernelArgument* und *KernelInfo* implementiert.

Ein KernelArgument-Objekt repräsentiert ein Argument eines Kernels bzw. einen Puffer auf der Grafikkarte. Die Klasse enthält die Typgröße des Arguments in Bytes und einen String, der dem Argument eine Feldgröße zuordnet. Ein KernelArgument-Objekt mit der Typgröße 4 Byte und dem Größenstring „InputImageSize“ könnte z.B. für ein Float-Array von der Größe der Anzahl der Pixel der zu verarbeitenden Bilder stehen. Auf diese Weise werden wichtige Größen nur in der GenericCL-Klasse, in einer assoziativen Datenstruktur (std::map) gespeichert. KernelArgument-Objekte können in der GenericCL-Klasse registriert werden. Dabei werden die entsprechenden Puffer von der Klasse automatisch angelegt. Die Größenstrings müssen vor dem Registrieren des KernelArgument-Objekts von außerhalb der GenericCL-Klasse einer bestimmten Größe zugeordnet werden.

Die KernelInfo-Klasse beschreibt einen Kernel. Die Klasse enthält den Pfad und Namen der Programmdatei eines Kernels, den Kernelnamen, Optionen für das Erstellen des Kernels und einen oder mehrere Zeiger auf KernelArgument-Objekte. Beim Registrieren eines KernelInfo-Objekts werden seine KernelArgument-Objekte registriert, der Kernel selber wird erstellt und die Argumente des Kernels werden auf die entsprechenden Puffer gesetzt. Puffer zu bereits registrierten KernelArgument-Objekten werden dabei nicht neu erstellt. Die Handles, die beim Erstellen von OpenCL-Programm und Kernel zurückgegeben werden, werden zusammen mit dem Kernelnamen in einem KernelInstance-Objekt gespeichert. Dieses ist für die Ausführung des Kernels zuständig und wird von der GenericCL-Klasse kontrolliert.

Beim Auslesen oder Beschreiben von Puffern bzw. beim Ausführen von Kerneln werden der GenericCL-Klasse zur Adressierung nur Zeiger auf die entsprechenden KernelArgument- bzw. KernelInfo-Objekte übergeben. Innerhalb der GenericCL-Klasse werden die Zeiger über assoziative Datenstrukturen eindeutig den entsprechenden Puffern bzw. KernelInstance-Objekten zugeordnet.

Die *PamonoCL*-Klasse ist eine Unterklasse von GenericCL und erweitert diese um die für die für die Bildverarbeitung nötigen Aspekte. Sie verwaltet die Bilder im Ringpuffer (siehe Kapitel 9.3.2.1) und bietet eine Funktion, um wichtige Daten wie Partikel-Kandidaten und Partikel-Funde effizient aus dem Grafikkartenspeicher zu laden.

9.3.2. Speicherkonzepte

Da die Bildverarbeitung eine unbegrenzte Anzahl an Bildern verarbeiten können muss, und der verfügbare Grafikkartenspeicher begrenzt ist, wird für den globalen Grafikkartenspeicher

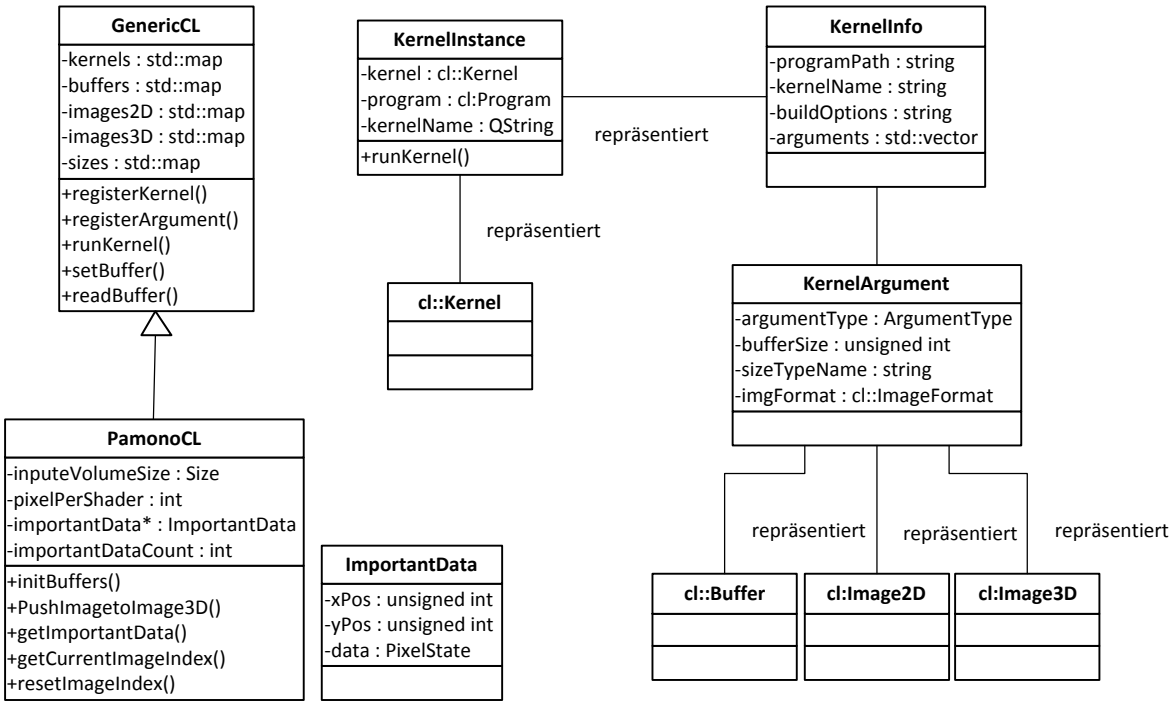


Abbildung 9.4.: Klassendiagramm des GPU-Interfaces

eine geeignete Datenstruktur benötigt.

9.3.2.1. Ringpuffer

Als geeignetes Speicherkonzept wird ein Ringpuffer implementiert, der sich über den OpenCL-Speicherobjekttyp *Image3D* realisieren lässt. Dieser besteht aus einem dreidimensionalen Array, in welches Bilder in Form von Float-Arrays geschrieben werden. Bei Erreichen des Array-Endes wird die Schreibposition an den Anfang des Arrays gesetzt und weitere eingehende Bilder überschreiben die bestehenden Bilder.

Vor der Initialisierung des Ringpuffers muss die Bildgröße angegeben und die insgesamt speicherbare Bildmenge ermittelt. Anschließend wird für den Größe des verfügbaren Grafikkartenspeichers festgestellt werden. Daraus wird die Ringpuffer die Hälfte dieser Speichergröße alloziert. Mit diesen vorbereitenden Schritten wird der Ringpuffer als Bildvolumen in der im Kapitel 9.3.1 angesprochenen assoziativen Datenstruktur registriert und mit den übergebenen Größen erzeugt.

Das Speicherkonzept mittels Ringpuffer bietet den Vorteil, dass ein Kernel einen Fensterbereich zusammenhängender Bilder (d.h. Bilder mit fortlaufendem Index) betrachten kann.

9.3.2.2. Ergebnispuffer

Ein wichtiges Speicherkonzept ist der Ergebnispuffer, in den alle Kernel die Ergebnisse ihrer Pixelverarbeitungen zurückliefern. Dieser Puffer besteht aus einem zweidimensionalen Array, dessen Koordinaten die Positionen der Bildpixel widerspiegeln und an jeder Position der Zustand eines Pixels gespeichert wird. Mit dieser Struktur können sowohl Zwischenergebnisse als auch das Endergebnis der gesamten Bildverarbeitungskette festgehalten werden.

9.3.3. Integration in die Pipeline

In der Pipeline-Architektur des PAMONO-Explorers wird ein Pipeline-Element zur Bildverarbeitung auf der Grafikkarte zur Verfügung gestellt. Dieses wird durch die Klasse *GPUPipelineElement* implementiert. Es nutzt die PamonoCL-Klasse und stellt Kernel sowie den Ergebnispuffer bereit. Dazu enthält es Methoden zur Initialisierung, in der Puffer erzeugt und Kernels registriert werden, zur Übergabe von Parametern und zum Zurücksetzen von Laufzeitvariablen. Die Methode *processData* realisiert die zentrale Aufgabe des GPU-Pipeline-Elements: Die in der Pipeline befindlichen Bilder an die PamonoCL-Klasse zu übergeben, die sie in den Grafikkartenspeicher schreibt.

Abbildung 9.5 veranschaulicht beispielhaft das bereits vorgestellte Speicherkonzept des Ringpuffers, welcher genutzt wird, um eine ausreichend große Anzahl von Bildern in den Grafikkartenspeicher hochzuladen. Es sei beispielsweise angenommen, dass bereits w Bilder in den Puffer geschrieben wurden (0.). Der in eine Richtung fortlaufende Schreibvorgang (1.) überschreibt die ältesten im Speicher befindlichen Bilder. Nach jedem Schreibvorgang von jeweils p Bildern in den Ringpuffer wird die eigentliche GPU-basierte Bildverarbeitung durch Aufruf der Methode *processImages* ausgelöst. Mit Hilfe der PamonoCL-Klasse werden bildverarbeitende Kernel ausgeführt, welche auf p zuletzt geschriebene und w ältere Bilder im Ringpuffer zugreifen können (2.). Dabei darf $p + w$ maximal so groß sein wie die Anzahl der Bilder, die der Ringpuffer umfassen kann. Erst nach Ausführung der Kernel werden wieder neue Bilder in den Ringpuffer geladen.

Die Kernel schreiben ihre Verarbeitungsergebnisse in den gemeinsamen Ergebnispuffer. Dieser wird nach Ausführung aller Kernel aus dem Grafikkartenspeicher ausgelesen, worauf Funde, abhängig vom Status des Partikels, in der Bildausgabe markiert werden.

Ein Nachteil an der Methode ist, dass mit steigender Ringpuffergröße eine niedrigere Aktualisierungsrate der Bildausgabe in der graphischen Benutzeroberfläche ergibt. Allerdings bietet ein größer gewählter Ringpuffer den Vorteil, dass insgesamt langsame Auslesezeiten reduziert werden.

9.3.4. Implementierte Kernel

Für die GPU-Verarbeitung sind einige, verschiedene Kernel implementiert worden. Der Kernel *timeSeriesAnalysis()* implementiert die in Kapitel 3 vorgestellte Zeitreihenanalyse. Mit

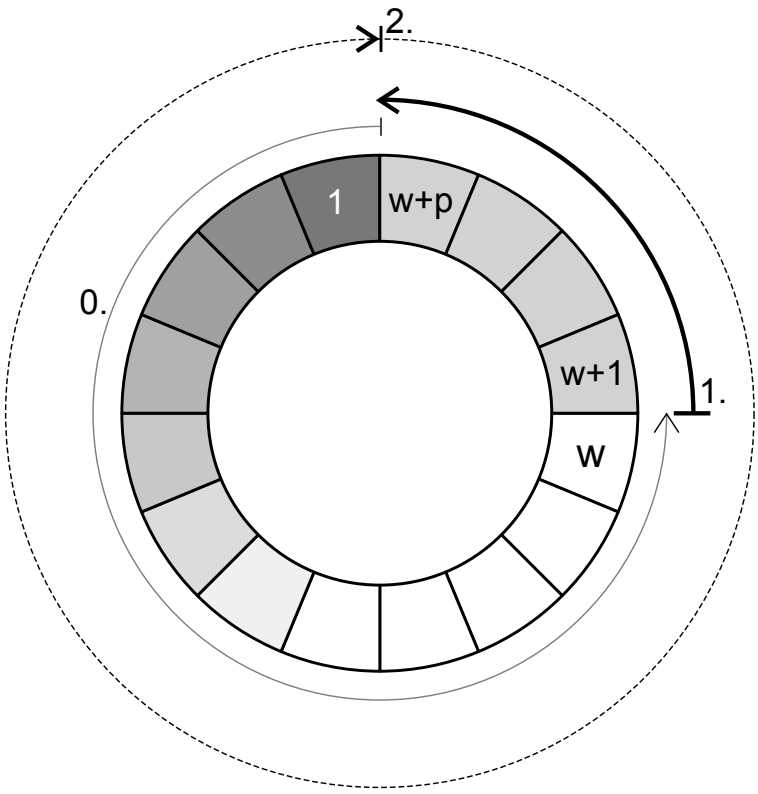


Abbildung 9.5.: Veranschaulichung des Arbeitsweise des GPU-Pipeline-Elements auf dem Ringpuffer

rigidTemplateMatching() und *sigmaTemplateMatching()* sind die beiden Template-Matching-Verfahren aus Kapitel 5.7 implementiert worden. Außerdem sind mit groupPixels() ein Kandidaten-Filter und mit *parallelGatherImportantData()* ein Kernel zum Sammeln relevanter Pixel, d.h. Kandidaten und gefundene Partikel, implementiert worden.

Die Kernel parallelisieren auf Datenebene. Bei den Analyse-Kerneln wird jeweils eine Instanz für jedes Pixel bzw. für jeden Kandidaten gestartet. Da diese Instanzen jeweils isoliert arbeiten, wird der lokale Speicher der Compute Units nicht gebraucht. Auch ist keine Synchronisierung innerhalb einer Kernel-Ausführung und auch keine spezielle Aufteilung der Workgroups notwendig. Der Zeitreihenanalyse-Kernel arbeitet auf den in 9.3.3 erwähnten p Bildern im Bildpuffer und greift auf weitere w Bilder in die Vergangenheit zurück, um die Sprunganalyse durchzuführen. Die beiden Template-Matching-Kernel greifen jeweils nur auf das (zeitlich) letzte Bild im Bildpuffer zu. Der Kernel *groupPixels()* arbeitet direkt auf dem Ergebnispuffer und filtert Randkandidaten bzw. einzeln stehende Kandidaten heraus, indem er die Umgebung eines Kandidaten betrachtet und ihn verwirft, sofern es nicht ausreichend andere Kandidaten gibt.

Für *parallelGatherImportantData()* werden eine Anzahl von Instanzen gestartet, die der Anzahl der Streaming Prozessoren in der GPU entspricht. Jede dieser Instanzen betrachtet einen

Teilausschnitt aus dem Ergebnispuffer und sammelt wichtige Pixel in einer separaten Liste. Im Anschluss werden diese Daten präzise aus dem Speicher der GPU in den Hauptspeicher geladen und dort in einer Liste vereinigt. Auch hier ist keine Synchronisierung notwendig. Die Workgroupgröße beträgt acht, entsprechend der typischen Anzahl von Streaming Prozessoren pro Streaming Multiprozessor in aktuellen Nvidia Grafikkarten.

10. Evaluation

In den vorherigen Kapiteln wurde beschrieben, wie Partikel in PAMONO-Bilddaten erkannt werden können, und wie diese Erkennung schnell ausgeführt werden kann. In diesem Kapitel werden nach Vorstellung der Testdatensätze und des Testsystems (Abschnitt 10.1) die Erkennungsrate des Sigma-Template-Matchings und des Bayes-Klassifikators (Abschnitt 10.2), sowie die Geschwindigkeit (Abschnitt 10.3) des PAMONO-Explorers evaluiert.

10.1. Datensätze und Testumgebung

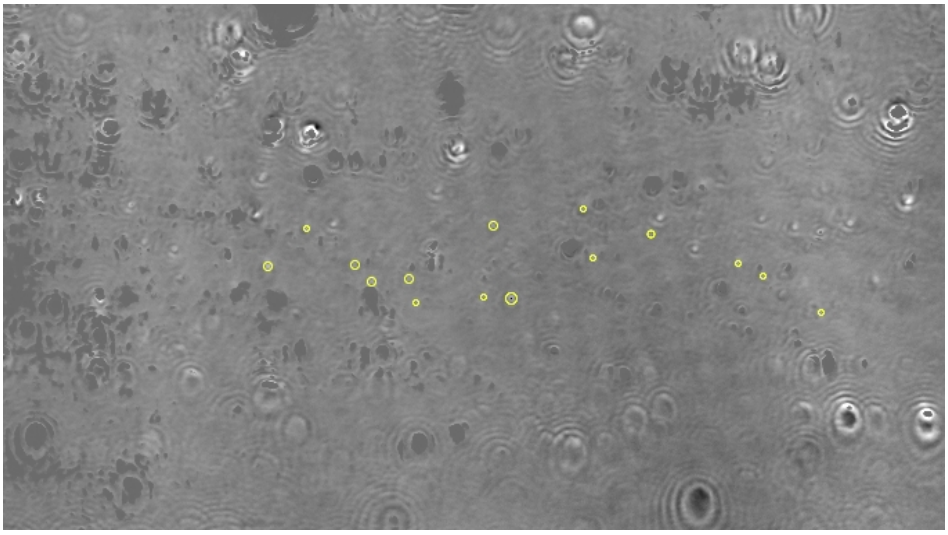
Es wurden zwei Datensätze für die Evaluation benutzt: Datensatz 1 umfasst 2449 Bilder und zeigt 80 nm-Partikel. In ihm wurden 15 Partikel klassifiziert, ohne eine weitere Unterteilung in Unterklassen vorzunehmen. Datensatz 2 ist ein 390 Bilder umfassender Datensatz von 200 nm-Partikeln. Es wurden 40 Partikel einer Partikelklasse klassifiziert. Abbildung 10.1 zeigt Beispielbilder beider Datensätze mit den Markierungen der klassifizierten Partikel.

Der Testrechner hat folgende Spezifikationen:

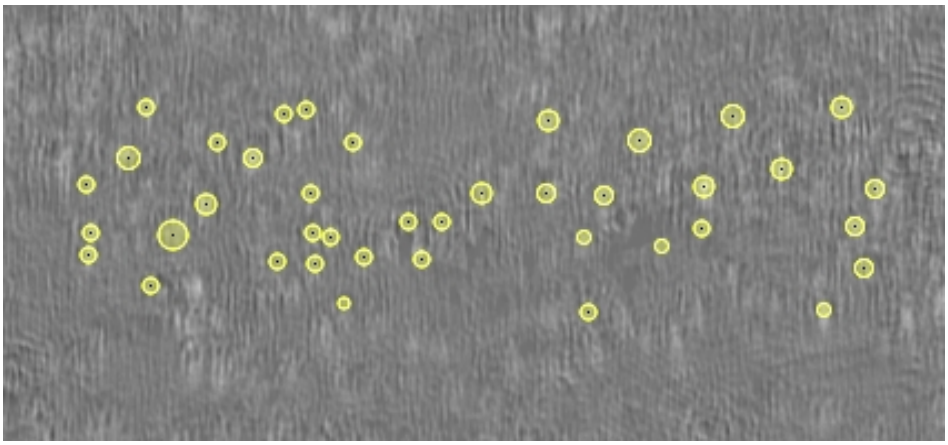
- Betriebssystem: Ubuntu Linux 10.04.1 64bit,
- Arbeitsspeicher: 6GB RAM,
- CPU: Intel Core i7 960 (4 Kerne mit je 3,20Ghz),
- GPU: Geforce GTS 250.

10.2. Evaluation der Erkennungsrate

Bevor die Klassifikatoren evaluiert werden können, müssen einige Begriffe erklärt werden: Als *true positives* (TP) werden Partikel bezeichnet, die korrekt erkannt wurden. *false positives* (FP) sind vom Klassifikator entdeckte Segmente, die keine Partikel sind. Partikel, die der Klassifikator nicht erkannt hat, heißen *false negatives* (FN). Der Begriff der *true negatives* (TN) existiert ebenfalls. Er beschreibt die Segmente, die keine Partikel sind, und nicht vom Klassifikator erkannt wurden. Da nicht klar definiert ist, was ein „Segment, aber kein Partikel“ ist, können *true negatives* nicht sinnvoll verwendet werden und werden in diesem Kontext nicht weiter betrachtet. Die Vierfeldertafel (Abbildung 10.2) verdeutlicht diese Begriffe. Die



(a) Datensatz 1, 80 nm, 15 klassifizierte Partikel



(b) Datensatz 2, 200 nm, 40 klassifizierte Partikel

Abbildung 10.1.: Bilder aus den Evaluationsdatensätzen. Die gelben Markierungen zeigen die klassifizierten Partikelsamples. In (a) ist ein um 90° gekippter Ausschnitt eines Bildes des 80-nm-Datensatzes zu sehen. Abbildung (b) zeigt ein ganzes Bild des 200-nm-Datensatzes.

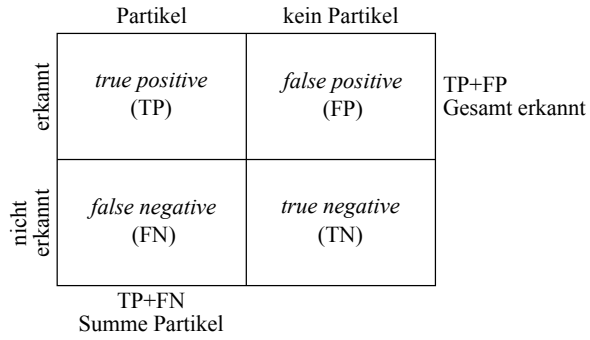


Abbildung 10.2.: Vierfeldertafel

Sensitivität drückt den Anteil der korrekt klassifizierten Partikel aus, sie ist definiert als

$$P_s = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{10.1}$$

Im Kontext der Projektgruppe und der PAMONO-Analyse sind *false positives* tendenziell als unkritisch zu betrachten. Würde die PAMONO-Analyse im Ernstfall zur Virendetektion eingesetzt, würde ein fälschlicherweise gefundenes Virus (also ein *false positive*) nur eine genauere Untersuchung und keine schwerwiegenden Konsequenzen für den Patienten nach sich ziehen. Würde ein Virus jedoch fälschlicherweise nicht gefunden, würde der Patient als gesund entlassen werden, mit möglichen fatalen Konsequenzen. Darüber hinaus kann es sein, dass für die Evaluation nicht alle Partikel in den Daten klassifiziert wurden. Erkennt das Analyseverfahren ein *false positive*, so ist es möglich, dass es ein Partikel ist, das vom Benutzer übersehen wurde.

Für das Training der Klassifikatoren während der Evaluation wurde eine zufällige Teilmenge der klassifizierten Partikel ausgewählt. Der Einfluss der Anzahl an Trainings-Samples auf die Sensitivität wurde untersucht. Die Parameter der Klassifikatoren wurden so eingestellt, dass ungefähr die gleiche Menge an Partikeln gefunden wurde. Anschließend sind die *true positives* gezählt worden. Die Erkennungsrate der GPU-Analyse wurde nicht gesondert betrachtet, da diese ebenfalls das Sigma-Template-Matching nutzt, das mit dem Bayes-Klassifikator verglichen wurde.

Die Ergebnisse der Evaluation sind in Tabelle 10.1 zu sehen. In Datensatz 1 haben beide Verfahren eine Sensitivität von über 50%. Die höchste Sensitivität von 75% wurde mit 20 Trainings-Samples für den Bayes-Klassifikator gemessen. Das heißt, dass von den 40 klassifizierten Partikeln 30 erkannt wurden. Für den Bayes-Klassifikator lässt sich kein eindeutiger Einfluss der Anzahl an Trainings-Samples auf die Sensitivität nachweisen. Zwar ist diese bei 20 Samples am höchsten und sinkt dann bei 10 Samples auf 62,5%, doch steigt die Sensitivität bei nur 5 Samples wieder auf 70%. Beim Sigma-Template-Matching kann ein leichter Einfluss der Trainings-Sample-Zahl auf die Sensitivität festgestellt werden. Von 65% bei 20 Samples sinkt sie bei 5 Samples auf 55%. Die Ergebnisse des zweiten Datensatzes sind vergleichbar. Die Sensitivität des Sigma-Template-Matchings fällt von 66,67% bei 10 von 15 genutzten Samples auf 46,67% bei 5 genutzten Samples. Die Sensitivität des Bayes-Klassifikators ist in beiden

Datensatz			Sigma-Template-Matching				Bayes-Klassifikator			
	Samples total	Samples benutzt	TP	FN	FP	Sensititivät (%)	TP	FN	FP	Sensititivät (%)
200 nm	40	20	26	14	52	65,00	30	10	53	75,00
		10	23	17	46	57,50	25	15	41	62,50
		5	22	18	39	55,00	28	12	33	70,00
80 nm	15	10	10	5	15	66,67	12	3	21	80,00
		5	7	8	28	46,67	12	3	23	80,00

Tabelle 10.1.: Evaluationsergebnisse für die Klassifikationsgenauigkeit von Sigma-Template-Matching und Bayes-Klassifikator.

Fällen bei 80%.

Der Bayes-Klassifikator schneidet insgesamt besser ab als das Sigma-Template-Matching. Er hat bei beiden Datensätzen die höhere Sensitivität. Anhand der Tests lässt sich die Tendenz feststellen, dass eine geringere Zahl an Trainings-Samples auf das Sigma-Template-Matching eine negative Wirkung hat. Beim Bayes-Klassifikator ist dies nicht der Fall. Allerdings können bedingt durch den geringen Umfang der Evaluation keine abschließenden, absoluten Aussagen getroffen werden: Die randomisierte Auswahl der Test-Samples kann den Bayes-Klassifikator zufällig bevorteilt haben. Darüber hinaus wurde nur eine Virenklasse benutzt. Es ist daher nicht möglich zu sagen, wie sich die Verfahren bei mehreren Virenklassen verhalten.

Abschließend ist anzumerken, dass die Erkennungsrate von Sigma-Template-Matching und Bayes-Klassifikator nicht vollständig unabhängig ist. Beide Verfahren klassifizieren nur die Bereiche, die ein ausreichend starkes Sprungverhalten in der Zeitreihe aufweisen. Ist der Sprung eines Partikels unter dem eingestellten Schwellwert der Sprungerkennung, kann er von keinem der beiden Verfahren erfolgreich klassifiziert werden.

10.3. Evaluation der Geschwindigkeit

Die Verarbeitungsgeschwindigkeit wurde für das Sigma-Template-Matching, den Bayes-Klassifikator und die Verarbeitung mit der GPU gemessen. Es wurde Datensatz 1 zur Evaluation ausgewählt, da er mehr Bilder als Datensatz 2 enthält und die Verarbeitung somit länger dauert. Dadurch können die verarbeiteten Bilder pro Sekunde (fps, *frames per second*) genauer gemessen werden. Es wurde jeweils der komplette Verarbeitungsdurchlauf gemessen, mit allen Elementen der Pipeline. Die Evaluationsergebnisse sind in Abbildung 10.3 zu sehen.

Anzumerken ist, dass die Geschwindigkeit einer erstmals gestarteten Analyse durch die Festplattengeschwindigkeit limitiert ist. Auf dem Testsystem ist die Geschwindigkeit dadurch bei dem ersten Durchlauf auf ungefähr 40 fps beschränkt. Beim Lesen von der Festplatte werden die Bilder betriebssystemseitig im Arbeitsspeicher zwischengespeichert. Bei weiteren Zugriffen auf dieselbe Datei wird diese aus dem Arbeitsspeicher geladen und damit die

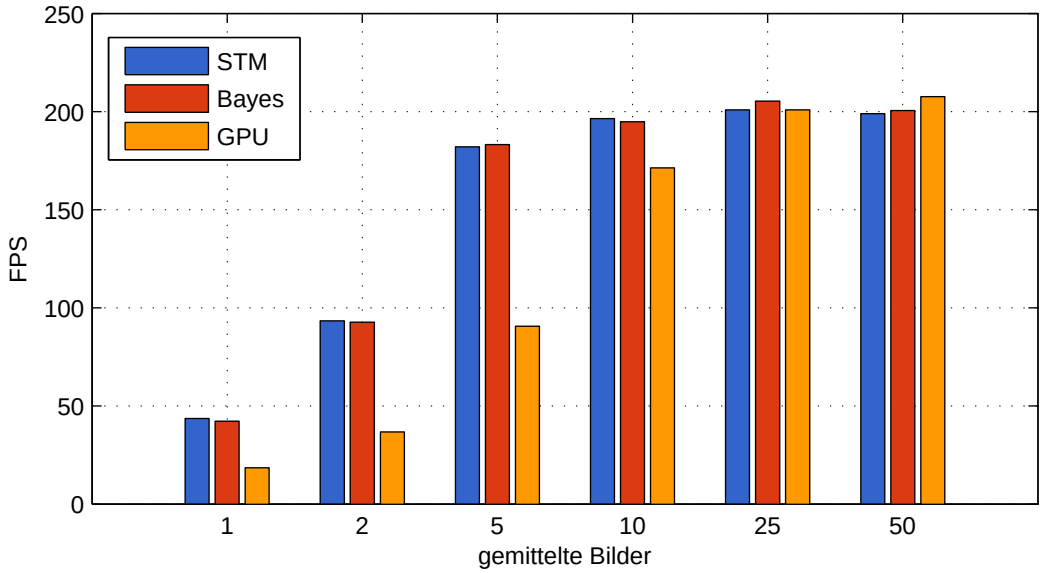


Abbildung 10.3.: Die Verarbeitungsgeschwindigkeiten des Pamono-Explorers für die Analysemethoden mit Sigma-Template-Matching, Bayes-Klassifikator und GPU für verschiedene Anzahlen gemittelter Bilder. Die genauen Daten sind in Anhang A aufgeführt.

Festplatte umgangen. Der Arbeitsspeicher des Testsystems ist groß genug um darin den gesamten Datensatz zwischenzuspeichern. Folgende Verarbeitungsdurchläufe lesen die Bilder direkt aus dem Arbeitsspeicher. Dadurch kann die maximal erreichbare Geschwindigkeit der Analyse getestet werden und so wurde auch die Evaluation mit zwischengespeicherten Daten durchgeführt.

Die Ergebnisse zeigen, dass zwischen Sigma-Template-Matching und dem Bayes-Klassifikator kein signifikanter Geschwindigkeitsunterschied besteht. Sie sind auch bei unterschiedlicher Mittelungszahl etwa gleich schnell. Die GPU-Analyse fällt bei geringer Bildmittelung deutlich hinter den beiden anderen Ansätzen zurück. Das kann an der Kommunikation zwischen GPU und dem Rest des Systems liegen. Die Bilder müssen aus dem Arbeitsspeicher der CPU in den Arbeitsspeicher der GPU kopiert, und nach der Berechnung zurückkopiert werden. Bei geringer Bildmittelung finden diese Kopiervorgänge sehr häufig statt und können die Leistung beeinträchtigen. Die Maximalgeschwindigkeit der Verarbeitung liegt bei etwa 200 fps, unabhängig von der verwendeten Analysemethode.

Aus der Evaluation wird deutlich, dass die Pipeline-Struktur in Kombination mit der Anordnung der Pipeline-Elemente eine sehr hohe Verarbeitungsgeschwindigkeit bietet. Die Vorverarbeitungskette aus Bildmittelung und Zeitreihensegmentierung reduziert die Menge der Daten so weit, dass diese auch auf der CPU sehr schnell mit komplexeren Algorithmen analysiert und klassifiziert werden können. Die Stärken des GPU-basierten Ansatzes, sehr viele Daten parallel verarbeiten zu können, kommen damit in dieser Analysestruktur nicht zur Geltung. Vielmehr überwiegen die Nachteile der erhöhten Kommunikation zwischen CPU und GPU.

11. Zusammenfassung und Ausblick

In diesem Kapitel werden abschließend die wichtigsten Punkte der Arbeit an dem Softwareprojekt PAMONO-Explorer und der Projektgruppe allgemein betrachtet. Neben einer kurzen Übersicht über die einzelnen Arbeitsphasen in einer Zusammenfassung (Kapitel 11.1) wird das erzielte Ergebnis in einr Bewertung (Kapitel 11.2) kritisch untersucht und im Ausblick (Kapitel 11.3) mögliche Verbesserungen und Erweiterungen des Projekts PAMONO-Explorer aufgezählt.

11.1. Zusammenfassung

Das Gesamtziel der Projektgruppe bestand darin, ein softwarebasiertes Analyse-System zur Erkennung von Partikeln, den PAMONO-Explorer, zu entwickeln.

Der Softwareentwicklungsprozess des PAMONO-Explorers begann mit einer Konzeptphase, in der die Ziele festgelegt und präzise definiert wurden. Anschließend wurden sämtliche Funktionen erfasst, über die der PAMONO-Explorer verfügen sollte. Auf dieser Grundlage wurden verbindliche Musskriterien, optionale Wunschkriterien und Abgrenzungskriterien formuliert und in einem Pflichtenheft (siehe Anhang C) festgehalten.

Nach der Konzeptphase folgte eine Experimentierphase, in der verschiedene Verfahren zur Bildverarbeitung, insbesondere der Bildverbesserung und Partikelerkennung, unter den Gesichtspunkten der Eignung und Effizienz getestet wurden. Die Algorithmen, die schließlich gewählt worden sind, wurden anschließend verbessert. Gleichzeitig wurden Überlegungen angestellt, ob eine Bildkompression für die Bildübertragung von der Kamera zur Software notwendig ist. Im Vordergrund dieser Überlegungen standen dabei die möglichen Auswirkungen der Kompression auf die Partikelerkennung. Daher wurde nach einem verlustfreien Kompressionsverfahren gesucht, das auf der Kamera implementiert werden kann.

Für die Realisierung der Bildverarbeitungspipeline und zur Aufnahme der Bilder von der Kamera wurden innerhalb der Projektgruppe Diskussionen über den Einsatz der Bibliothek GStreamer geführt. Ein Grund für die Nutzung von GStreamer war insbesondere die Einsparung von Programmieraufwand, da das Caching der Bilddaten im Speicher und der Multithreading-Mechanismus bereits vorhanden sind. Auf der anderen Seite wurden auch Gründe in die Diskussion eingebracht, die gegen den Gebrauch von GStreamer sprachen. Ein Grund war die Tatsache, dass es sich bei GStreamer um eine Bibliothek für C handelt, die einen hohen Einarbeitungsaufwand aufweist. Des Weiteren war GStreamer vor allem für die Bearbeitung kleiner Bilddatenmengen vorgesehen, da der Haupteinsatzbereich von GStreamer ursprünglich

im Bereich der eingebetteten Systeme lag. Die Diskussion führte schließlich zu der Entscheidung, dass die Gruppe die Pipeline einschließlich der notwendigen Nebenläufigkeit durch Threads und dem Caching selbst implementiert. Für die Aufnahme der Bilddaten von der Kamera stand die Bibliothek LiveMedia zur Verfügung, während für das Speichern und Laden der Bilder auf FreeImage zurückgegriffen werden konnte. Aufgrund der Tatsache, dass der PAMONO-Explorer eine große Menge von Bilddaten verarbeiten muss und dass auf den Gebrauch von GStreamer verzichtet wurde, waren weitere Überlegungen bezüglich der Speicherverwaltung (Caching) notwendig, so dass zusätzlich ein Konzept für das Caching erstellt und ein Prototyp implementiert worden ist.

Auf der Grundlage der während der Experimentierphase gewonnenen Erkenntnisse wurde ein Prototyp des Softwareprojekts PAMONO-Explorer implementiert. Dabei wurden Entscheidungen bezüglich der Implementierung und der verwendeten Bibliotheken teilweise verworfen und auf Alternativen zurückgegriffen. Da die Kompilierung des PAMONO-Explorers auf einigen Plattformen zu Problemen mit der Bibliothek LiveMedia führte, wurde diese durch GStreamer ersetzt. Somit konnte GStreamer das Caching der Bilddaten übernehmen, was dazu führte, dass der durch die Projektgruppe entworfene und implementierte Caching-Mechanismus ebenfalls verworfen wurde.

11.2. Bewertung

In diesem Kapitel wird kritisch beleuchtet, ob die von der Projektgruppe festgelegten Ziele erreicht worden sind und in welchem Maße der PAMONO-Explorer die Zielvorgaben, wie sie im Pflichtenheft niedergeschrieben worden sind, erfüllt. Ferner wird überprüft, ob es seinem Einsatzzweck, der automatischen und schnellen Detektion von Viren, gerecht wird.

Bezüglich der Bilddatenaufnahme, -verbesserung und -auswertung wurden die Ziele der Projektgruppe erreicht. Der PAMONO-Explorer enthält die für den im Pflichtenheft vorgesehenen Einsatzzweck notwendigen Funktionen:

- Datenaufnahme mit Hilfe der Elphel-Kamera
- Einlesen und Speichern von Bildreihen von der lokalen Festplatte
- Anlernen von Virenmustern (Templates)
- Analyse der Bildreihen (Detektion und Klassifikation von Viren)
- Visuelle Ausgabe der Analyse-Ergebnisse
- Projektverwaltung insbesondere Anlegen und Öffnen von Projekten

Ferner besteht die Möglichkeit die Analyseergebnisse zu exportieren, sodass sie mit einem externen Programm wie Microsoft Excel geladen werden können. Es können neue Bildreihen von der Kamera akquiriert und auf der lokalen Festplatte gespeichert werden, wobei der Benutzende jederzeit die Möglichkeit hat, die Datenakquise abzubrechen.

Die Vorverarbeitung, wie zum Beispiel Bildmittelung, der aufgenommenen Bilder durch das FPGA der Elphel-Kamera, wie sie ursprünglich im Pflichtenheft vorgesehen war, konnte dagegen nicht realisiert werden. Eine Vielzahl von Problemen ergaben sich während der Entwicklung auf den FPGAs. Zunächst ist Xilinx ISE, die von Xilinx zur Verfügung gestellte Entwicklungsumgebung, und die Kombination mit den Quelltexten der Elphel-Projekte für die FPGAs der Boards 10359 und 10353 zu nennen. Da die Quelltexte für das Projekt der Kamera mit älteren Xilinx-Versionen erstellt worden sind, für das Arbeiten aber eine aktuelle Version benötigt wird, musste das Projekt aufwändig über mehrere Versionen der IDE migriert werden. Xilinx selbst bietet bei lediglich in Form einer webbasierenden Knowledge Base Unterstützung, die jedoch in vielen Fällen keine passenden Lösungen für die auftretenden Probleme enthielt.

Die Ressourcen auf dem Haupt-FPGA waren durch die Kamera-Funktionalität einerseits bereits weitestgehend ausgeschöpft, andererseits stellte es sich als äußert schwierig heraus, eine Stelle im Signalfluss zu finden, bei der die für eine Vorverarbeitung benötigten Daten vorliegen und das Timing der restlichen Module nicht die Leistung bereitstellt, die die Bildvorverarbeitung erfordert.

Für die Erweiterung der Funktionalität des FPGAs wäre es wünschenswert gewesen, wenn der Hersteller hierfür ein Development Kit zur Verfügung stellen würde. Tatsächlich sind die einzigen Daten, die der Hersteller zur Verfügung stellt, der Verilog-Quellcode selbst. Es ist keine Dokumentation des Projekts an sich verfügbar und auch ein Dokument, das die Struktur und Funktionsweise und das Taktverhalten der auf den FPGAs realisierten Funktionseinheiten realisiert, fehlt völlig.

Für eine vollständige Rückgewinnung aller Strukturinformationen aus dem FPGA-Projekt mangelte es im Umfeld der Projektgruppe an ausreichender Erfahrung mit Verilog-Projekten. Darüber hinaus stellte sich heraus, dass ganz allgemein Verilog im europäischen Raum kaum verbreitet ist.

Die Unterstützung des Herstellers war zwar über die Projektzeit hin vorhanden, aber in Antwortzeit und Ausführlichkeit der Beantwortung von Fragen unzureichend. Obwohl Elphel seit Herbst 2002 Kameras mit frei programmierbarem FPGA anbietet, existiert bis heute kein Projekt, das den FPGA um weitere Module erweitert. Somit war dieser Teil der Projektgruppe zu einem erheblichen Teil von dem einen Entwickler des Codes des Kamera-FPGAs abhängig.

Der Pamono-Explorer unterstützt wie im Pflichtenheft vorgesehen die Offline-Analyse. Für die Offline-Analyse wird zur Analyse eine Bildreihe von der lokalen Festplatte geladen. Die ursprünglich vorgesehene Online-Analyse wurde durch einen zweistufigen Ansatz ersetzt: Zuerst werden Bilddaten von der Kamera aufgenommen und diese mit der Offline-Analyse anschließend verarbeitet. Eine laufende Analyse kann vom Benutzenden jederzeit abgebrochen werden.

Des Weiteren bietet der Pamono-Explorer eine Projektverwaltung. Es können Projektordner für jedes neue Projekt erstellt werden, die einen frei wählbaren Namen an Stelle eines Datums erhalten. Für die Bildreihen und Analysedaten werden Unterordner erzeugt und nach jeder gestarteten und abgeschlossenen Analyse wird ein Statistikreport in dem jeweiligen Ordner abgespeichert. Die Projektdatei wird in einer XML-Datei gespeichert, die den Projektnamen

und das Erstellungsdatum enthält. Projekte können ebenfalls im PAMONO-Explorer geschlossen werden, jedoch ist das Löschen eines Projekts nur außerhalb des Programms möglich, indem der Projektordner entfernt wird. Die Löschfunktion war im Pflichtenheft jedoch lediglich als Wunschkriterium vermerkt worden.

Die grafische Benutzeroberfläche (GUI) wurde nach den Zielvorgaben der Projektgruppe gestaltet. Der Hauptaugenmerk lag dabei vor allem bei der Bedienbarkeit der Software für den Benutzenden. Die GUI ist ergonomisch aufgebaut, sodass die Bedienung des PAMONO-Explorers leicht zu erlernen ist. Zusätzlich erleichtern aussagekräftige Icons und Menüeinträge den Gebrauch der Software.

Die Anforderungen aus dem Pflichtenheft bezüglich der Visualisierung werden vom PAMONO-Explorer ebenfalls erfüllt. Die gefundenen und klassifizierten Partikel werden bereits während der laufenden Analyse unmittelbar angezeigt. Der Benutzende hat die Möglichkeit, die Partikel selbst zu untersuchen, indem er sich die Zeitreihen für die jeweiligen Partikelbereiche anzeigen lassen kann.

Die Effizienz des PAMONO-Explorers bezüglich der Rechenzeit bei den Bildreihenanalysen entspricht ebenfalls den Zielvorgaben der Projektgruppe. Zur weiteren Beschleunigung der Analysegeschwindigkeit steht zusätzlich die Möglichkeit offen, eine dezidierte Grafikkarte mit OpenCL-Unterstützung für die Berechnungen einzusetzen.

Zusammenfassend stellt das Softwareprojekt PAMONO-Explorer eine Verbesserung im Vergleich zum Ist-Zustand dar. Es ist nicht mehr notwendig, Bildreihen, die oft aus einer sehr großen Anzahl von Einzelbildern bestehen, aufwändig manuell auf Viren zu untersuchen. Der PAMONO-Explorer bietet eine automatische Virendetektion mit viel geringerem Zeitaufwand und mit einer hohen Zuverlässigkeit.

11.3. Ausblick

Das von der Projektgruppe 548 entwickelte Softwareprojekt PAMONO-Explorer ist ein lauffähiges Produkt, das alle notwendigen Funktionalitäten bietet. Dennoch bieten sich Möglichkeiten, das Projekt weiterzuentwickeln, zumal die Erweiterbarkeit des Projekts bereits im Pflichtenheft als Qualitätsziel bestimmt war. Vor allem die Steigerung der Aussagegenauigkeit der Bilddatenanalyse gemäß dem Pflichtenheft (siehe Anhang C) stellt dabei ein lohnenswertes Ziel dar, um den Benutzenden in seiner Arbeit noch besser zu unterstützen. Die Aussagegenauigkeit hängt von mehreren Faktoren ab, zu denen unter anderem die Klassifikationsverfahren als auch die Bildverbesserungsverfahren und Lernmethoden zählen. In der nachfolgenden Auflistung werden einige Ansätze zur Verbesserung und zur Erweiterung des PAMONO-Explorers genannt.

- Ein Ansatz zur Verbesserung des Programms kann darin bestehen, die Verfahren zur Bildverbesserung in Bezug auf die Qualität des verbesserten Bildes zu optimieren, da das Ergebnis der Bildverbesserung das Ergebnis der Bildanalyse beeinflusst. Ferner können auch alternative Verfahren getestet und gegebenenfalls in den PAMONO-Explorer eingebettet werden, sofern durch die getesteten Algorithmen Verbesserungen im Vergleich

zum Ist-Zustand erzielt werden können.

- Es existieren neben den implementierten Klassifikationsverfahren andere Verfahren, die realisiert werden können. In Zukunft können neue Klassifikationsverfahren entwickelt werden, die die Klassifikation von detektierten Viren optimieren.

- Ferner bestehen auch im Bereich der ausgewählten Merkmale der Partikel Möglichkeiten zur Verbesserung und Erweiterung. Es können alternative Ansätze getestet werden und gegebenenfalls in den PAMONO-Explorer eingebettet werden.

- Ein weiterer Gesichtspunkt ist die Bildkompression durch das FPGA der Kamera. Da sich die Wavelet-Transformation als ineffizient herausgestellt hat, bietet es sich an, nach alternativen Verfahren zu suchen, die auf dem FPGA programmiert werden können, um Bildreihen verlustfrei zu komprimieren. Denn auch die Qualität der Bildkompression trägt zum Ergebnis der Virendetektion durch den PAMONO-Explorer bei.

Literaturverzeichnis

[Alp08] Alpaydin, E.: *Maschinelles Lernen.* Oldenburg Wissenschaftsverlag, 2008

[App07] Appel, M.: *Entwurf eines eingebetteten Bildverarbeitungssystems mit dem FPGA-Board XUP Virtex-II Pro*, Institut für Informatik der Humboldt-Universität zu Berlin, Diplomarbeit, 2007

[BB06] Burger, W. ; Burge, M. J.: *Digitale Bildverarbeitung.* Springer, 2006

[BBK08] Bamberg, G. ; Baur, F. ; Krapp, M.: *Statistik.* Oldenburg Wissenschaftsverlag, 2008

[Bov05] Bovik, A. C.: *Handbook of image and video processing.* Academic Press, 2005

[Brü07] Brückner, M.: *Embedded Linux auf FPGA-Hardware zur Bildverarbeitung.* August 2007

[Dil] Dillinger, P.H.: *Einsatz von FPGA Prozessoren als Datenreduktionshardware in der Bildverarbeitung.* Zentralinstitut für Elektronik, Forschungszentrum Jülich in der Helmholtz-Gemeinschaft,

[EKT05] Eckey, H. F. ; Kosfeld, R. ; Türck, M.: *Wahrscheinlichkeitsrechnung und induktive Statistik.* Gabler, 2005

[EP71] Engvall, E. ; Perlmann, P.: Enzyme-linked immunosorbent assay (ELISA). In: *Quantitative assay of immunoglobulin G. Immunochemistry* 8(9) (1971), S. 871–874

[EP08] Erk, K. ; Priese, L.: *Theoretische Informatik: Eine umfassende Einführung.* Springer, 2008

[Fil] Filipov, A.: *Elphel Wiki.* `http://wiki.elphel.com`, Abruf: 17.03.2011

[Gei05] Geier, J.: *SNR Cutoff Recommendations.* `http://www.wi-fiplanet.com/tutorials/article.php/3468771/SNR-Cutoff-Recommendations.htm`, Abruf: 13.03.2011

[Geo09] Georgii, H. O.: *Stochastik: Einführung in die Wahrscheinlichkeitstheorie und Statistik.* Walter de Gruyter, 2009

[GKOK03] *Kapitel* Enzyme-Linked Immunosorbent Assay. In: Goldsby, R.A. ; Kindt, T.J.

; Osborne, B.A. ; Kuby, J: *Immunology*. 2003, S. 148–150

[GW08] González, R. C. ; Woods, R. E.: *Digital image processing*. 3rd. Prentice Hall, 2008

[HP06] Hošťálková, E. ; Procházka, A.: Wavelet Signal and Image Denoising. In: *14th Annual Conference Technical Computing Prague*, 2006

[IET98] IETF: *RFC: Real Time Streaming Protocol (RTSP)*. `http://tools.ietf.org/html/rfc2326`, Abruf: 2011.03.20

[Jan01] Janesick, J. R. ; PM83, SPIE Press Monograph V. (Hrsg.): *Scientific Charge-Coupled Devices*. SPIE – The International Society for Optical Engineering, 2001

[Lan06] Lang, H.-W.: *Algorithmen in Java*. 2nd. Oldenbourg Wissenschaftsverlag, 2006

[NBMN] Nibouche, M. ; Bouridane, A. ; Murtagh, F. ; Nibouche, O.: *FPGA-Based Discrete Wavelet Transforms System*. School of Computer Science, The Queen's University of Belfast. 18, Malone Road - Belfast BT7 1NN - UK,

[Nie03] Niemann, H.: *Klassifikation von Mustern*. 2. überarbeitete Auflage. Universität Erlangen-Nürnberg, 2003 `http://www5.informatik.uni-erlangen.de/fileadmin/Persons/NiemannHeinrich/klassifikation-von-mustern/m00-www.pdf`

[OW93] Ottmann, T. ; Widmayer, P.: *Algorithmen und Datenstrukturen*. 2nd. BI Wissenschaftsverlag, 1993

[PLH10] Puente León, F. (Hrsg.) ; Heizmann, M. (Hrsg.): *Forum Bildverarbeitung*. Regensburg : KIT Scientific Publishing, Dezember 2010

[Rit02] Ritter, J.: *Wavelet based image compression using FPGAs*, Mathematisch-Naturwissenschaftlich-Technischen Fakultät der Martin-Luther-Universität Halle-Wittenberg, Diss., 2002

[RR07] Rauber, T. ; Rünger, G.: *Parallele Programmierung*. 2nd. Springer, 2007

[Sac02] Sachs, L.: *Angewandte Statistik*. Springer, 2002

[SGG08] Silberschatz, A. ; Galvin, P. B. ; Gagne, G.: *Operating System Concepts*. 8th. John Wiley & Sons, 2008

[Ste09] Stephen, A.: Swine flu: Border controls 'don't work' warns WHO. In: *The Telegraph* 28.April (2009), 1. `http://www.telegraph.co.uk/news/worldnews/centralamericaandthecaribbean/mexico/5238030/Swine-flu-Border-controls-dont-work-warns-WHO.html`, Abruf: 16.03.2011

[Tan07] Tanenbaum, A. S.: *Modern Operating Systems*. 3rd. Upper Saddle River, NJ,

USA : Prentice Hall Press, 2007. – ISBN 9780136006633

[The09] The Telegraph: Swine flu: essential advice for travellers. In: *The Telegraph* 21.Juli (2009), 1. http://www.telegraph.co.uk/travel/travelnews/5877710/Swine-flu-essential-advice-for-travellers.html, Abruf: 16.03.2011

[TK06] Theodoridis, S. ; Koutroumbas, K.: *Pattern recognition*. Academic Press, 2006

[Val10] Valens, C.: *A Really Friendly Guide to Wavelets*. http://polyvalens.pagesperso-orange.fr/clemens/wavelets/wavelets.html, Abruf: 20.02.2011

[Vie03] Viertl, R.: *Einführung in die Stochastik*. Springer, 2003

[VM94] Vidakovic, B. ; Mueller, P.: Wavelets for kids: A tutorial introduction / Duke University. 1994. – Forschungsbericht

[WGZ+10] Weichert, F. ; Gaspar, M. ; Zybin, A. ; Gurevich, E. ; Görtz, A. ; Timm, C. ; Müller, H. ; Marwedel, P.: *Plasmonen-unterstützte Mikroskopie zur Detektion von Viren*. 2010

[Xil] Xilinx: *Xilinx Webseite*. www.xilinx.com, Abruf: 19.03.2011

[ZGWN09] Zybin, A. ; Gurevich, E. L. ; Weichert, F. ; Niemax, K.: SPR detection of single nano particels and viruses. In: *PHYSCON*, 2009

Abbildungsverzeichnis

A. Evaluationsdaten

gemittelte Bilder	Methode	Zeit (s)	FPS	gemittelte Bilder	Methode	Zeit (s)	FPS
1	STM	56.071	43.68	10	Bayes	12.920	189.55
1	STM	56.278	43.52	10	Bayes	12.242	200.05
1	Bayes	59.444	41.20	10	GPU	14.700	166.60
1	Bayes	56.361	43.45	10	GPU	14.200	172.46
1	GPU	131.879	18.57	10	GPU	14.093	173.77
1	GPU	131.791	18.58	10	GPU	14.073	174.02
2	STM	26.264	93.25	10	GPU	14.403	170.03
2	STM	26.164	93.60	25	STM	12.500	195.92
2	Bayes	26.691	91.75	25	STM	11.895	205.88
2	Bayes	26.159	93.62	25	Bayes	11.800	207.54
2	GPU	66.454	36.85	25	Bayes	12.053	203.19
2	GPU	66.843	36.64	25	GPU	11.858	206.53
5	STM	13.540	180.87	25	GPU	12.500	195.92
5	STM	13.369	183.18	25	GPU	12.239	200.10
5	Bayes	13.505	181.34	50	STM	11.755	208.34
5	Bayes	13.232	185.08	50	STM	12.920	189.55
5	GPU	26.971	90.80	50	Bayes	11.840	206.84
5	GPU	27.044	90.56	50	Bayes	12.609	194.23
10	STM	13.652	179.39	50	GPU	12.262	199.72
10	STM	12.039	203.42	50	GPU	11.524	212.51
10	STM	12.013	203.86	50	GPU	11.621	210.74
10	STM	12.300	199.11				

Tabelle A.1.: Alle Messreihen zur Evaluation der Geschwindigkeit.

gemittelte Bilder	STM [in fps]	Bayes [in fps]	GPU [in fps]
1	43.60	42.33	18.58
2	93.42	92.69	36.75
5	182.03	183.21	90.68
10	196.44	194.80	171.38
25	200.90	205.36	200.85
50	198.94	200.53	207.66

Tabelle A.2.: Testergebnisse der Geschwindigkeitsmessung für das Sigma-Template-Matching (STM), den Bayes-Klassifikator und die GPU-Analyse.

B. Handbuch

B.1. Vorwort

Der PamonoExplorer ist eine Software, die den Workflow im Forschungsprozess mit Pamono-Analysedaten organisiert und Arbeitsschritte automatisiert. Im Folgenden wird angenommen, dass der Benutzer der Software grundsätzliches Wissen über den Prozess der Pamono-Analyse besitzt.
Die Anwendungsgebiete der Software bestehen aus Bilddatenakquise und deren Verwaltung, Funktionen zur Bildverarbeitung der Rohdaten, manuelle Erkundungsmöglichkeit der Bilddaten, Zeitreihen-Analyse, Klassifizierungsdatenbank, automatische Detektion von antrainierten Objekten und Export der statistischen Analyse-Ergebnisse.

B.1.1. Start des Programms

Abbildung B.1.: Das Willkommensfenster des PamonoExplorers.

Abbildung B.1 zeigt das Willkommensfenster des PamonoExplorers. Falls der PamonoExplorer zum ersten Mal benutzt wird oder ein neuer Versuch gestartet werden soll, muss ein neues Projekt angelegt werden. Dies geschieht entweder durch die Auswahl über die Hauptmenüleiste 1 `File->New Project` oder durch Anklicken des Buttons 2 `Create New Project`. Ein Name muss dem neu erstellten Projekt vergeben werden.

Wurde ein Projekt schon vorher angelegt, ist möglich, es entweder durch 1 `File->Open Project` oder per Knopf 3 aufzurufen. Es öffnet sich ein Dialogfenster, in dem das zuletzt bearbeitete Projekt angezeigt wird. Natürlich können andere Projekte auf der Festplatte über die Menüstruktur ausgesucht werden.

B.1.2. Project Dashboard

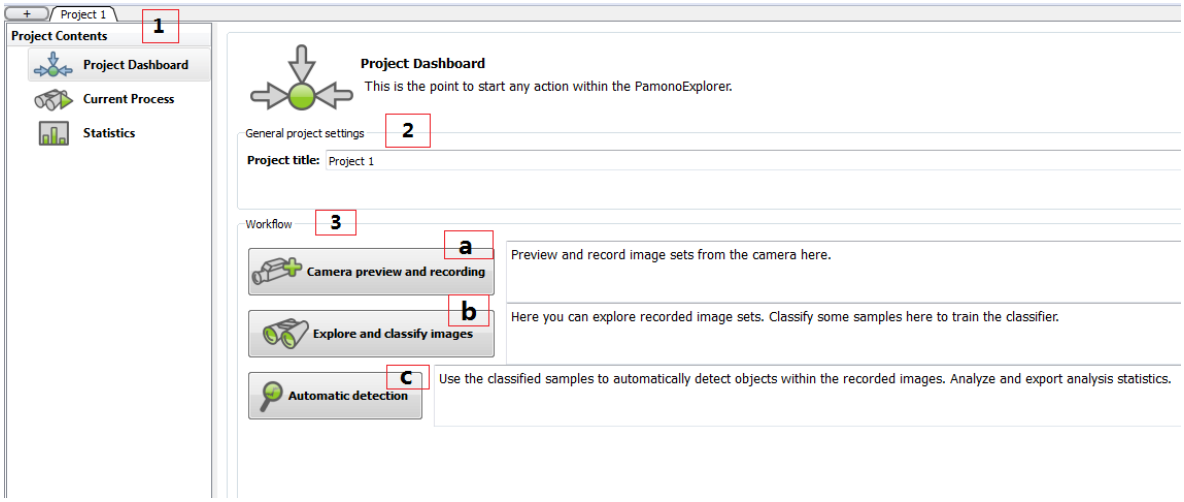


Abbildung B.2.: Das Project Dashboard listet die für eine erfolgreiche Analyse notwendigen Schritte auf.

Nachdem ein Projekt erstellt oder ausgewählt wurde, erscheint das Project Dashboard. Das Dashboard ist maßgebend für den Programmablauf, der sich in drei 3 Schritte a,b,c gliedert. Im ersten Schritt a `Camera preview and recording` werden - falls noch nicht vorhanden - Bilder über die Kamera aufgenommen und wenn gewünscht auf die Festplatte gespeichert.
Der zweite Schritt b `Explore and classify images` ist das Antrainieren des Klassifikators (**Achtung**: Nur für Bilder, die vorher auf die Festplatte gespeichert wurden, siehe auch Kapitel B.2). Die hier vorgenommenen manuellen Klassifizierungen werden im nächsten Schritt c `Automatic Detection` dem Klassifikator als Stichproben zur automatischen Erkennung von Funden übergeben. Einmal klassifizierte Samples werden automatisch gespeichert. Sollen neue Klassen oder Partikel mit unterschiedlichen Charakterisiken erkannt werden, lässt sich die Sample-Bibliothek beliebig erweitern.
Schritt b stellt darüber hinaus einen sogenannten „Explore-Modus“ zur Verfügung, in dem die aufgenommenen Bilder über einen Slider bequem nacheinander betrachtet werden können. Das Interface ist generell mit minimalen Änderungen gleich gehalten, um eine intuitive Bedienung zu ermöglichen.
Näheres zu den einzelnen Programmschritten befindet sich in den entsprechenden Kapiteln (B.3, B.4, B.5).

Das Dashboard ist in drei Teile gegliedert. 1 ist die Ansicht des aktuellen Projekts. Über `Project Dashboard` wird das aktuelle Dashboard aufgerufen. `Current Process` zeigt eine

laufende Analyse an (a, b oder c). `Statistics` öffnet eine Ansicht zur Ausgabe von Statistik. In 2 lässt sich das Projekt umbenennen (dazu mehr in Kapitel B.2). Funktion 3 listet den bereits bekannten Workflow der drei Schritte.

B.2. Projektmanagement

Das Anlegen von Projekten dient der Gruppierung von unterschiedlichen Versuchsreihen. So lassen sich die Ergebnisse von Einzelversuchen innerhalb von Versuchsreihen schnell vergleichen.

B.2.1. Anlegen eines Projektes

Ein Projekt besteht im Wesentlichen aus einem Ordner im Dateisystem, wie in Abbildung B.3 zu sehen.
Der Name des Projekts ist gleich dem des Ordners. Beim Umbenennen des Projekts wird auch der Ordnername entsprechend geändert.

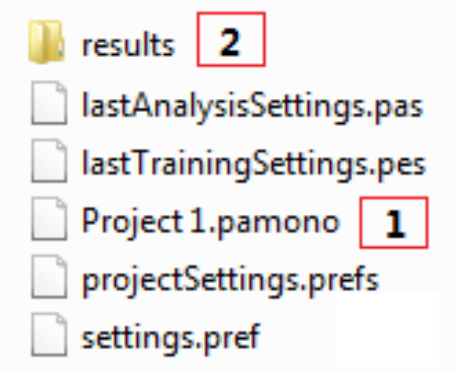


Abbildung B.3.: Projektordner mit abgespeicherten Einstellungen und Ergebnisordner.

Alle Einstellungen zu einem Projekt und den zugehörigen Analysen werden im Projektordner abgespeichert 1. Dazu gehören auch Parametereinstellungen, die während einer Analyse benutzt worden sind. Die Einstellungen werden beim Starten wieder geladen.
Auch Ergebnisse werden im Projektordner abgelegt, diese erhalten jedoch einen weiteren, eigenen Ordner 2 namens `results`. So werden mehrere Analysen von verschiedenen Datensätzen voneinander getrennt, indem die Ergebnisse einer jeden Analyse in einem eindeutig bezeichneten Ordner abgelegt werden. Abbildung B.4 verdeutlicht diesen Vorgang beispielhaft.

Ein neues Projekt kann auch jederzeit aus dem Project Dashboard heraus erstellt werden. Dazu ist auf das kleine Symbol mit dem + - Zeichen in der linken Ecke oder auf das Menü `File -> New Project` zu klicken.
Laufende Projekte können einzeln oder alle zusammen geschlossen werden. Dies ist unter `File -> Close Current Project` bzw. `File -> Close all Projects` zu finden.

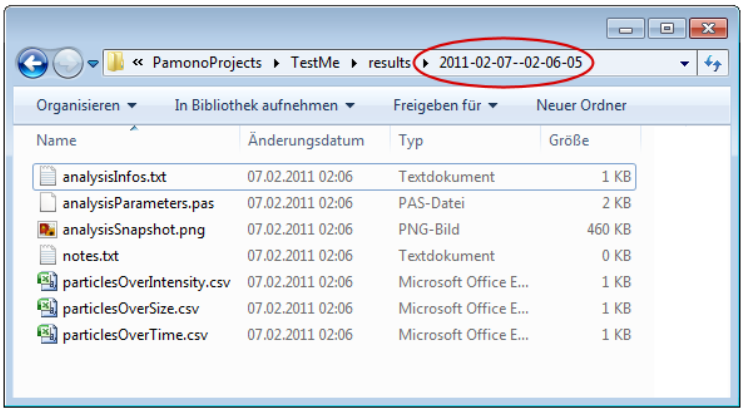


Abbildung B.4.: Die Analyse-Ergebnisse werden in einem Unterordner mit Zeitstempel der Analyse (Jahr-Monat-Tag–Stunde-Minute-Sekunde) abgelegt.

B.2.2. Globale Einstellungen

Aufgenommene Bilddaten werden als Datenquelle in Ordnern abgelegt. Der PamonoExplorer bietet eine bequeme globale[1] Datenquellen-Verwaltung, welche auf Suchpfaden basiert. Die Suchpfade lassen sich im Einstellungsfenster unter `File -> Settings` anpassen. Es erscheint das Fenster wie in Abbildung B.5.

In 1 werden alle Einstellungen vorgenommen, die für den Import nötig sind. 2 öffnet einen Dialog, der den Pfad der zu aufzunehmenden Kamerabilder setzt (Abbildung B.6). Der Kasten 3 zeigt die Ordner an, die für den Import der Bilder durch `add path` genutzt wurden. Hier muss nur ein Ordner angegeben werden, in dem Bilder enthalten sind. Der PamonoExplorer sucht automatisch alle Ordner, welche die Importkriterien erfüllen und trägt sie in die Liste 4 ein. In Abbildung B.5 ist zu sehen, dass jede Datenquelle (ein Ordner mit Bildern) einen eindeutigen Identifikator erhält, den Pfad angibt und die Anzahl der Bilder auflistet.
Zusätzlich können weitere Einstellungen 4 vorgenommen werden. So ist es möglich, eine Mindestanzahl an Bilddateien anzugeben oder die Suchkriterien auf bestimmte Bildertypen zu beschränken. Sollte eine Datenquelle nicht automatisch importiert werden, müssen diese Einstellungen geändert werden.

B.3. Bildakquise

B.3.1. Kameravorschau

Die Kameravorschau wird über das Projekt-Dashboard erreicht, wie in Kapitel B.1.2 aufgezeigt. Abbildung B.7 zeigt die gesamte Ansicht der Kameravorschau. Über 1 `Camerasettings` können Einstellungen bezüglich des Kamerastreams vorgenommen werden.

[1] Alle Projekte können alle Datenquellen verwenden.

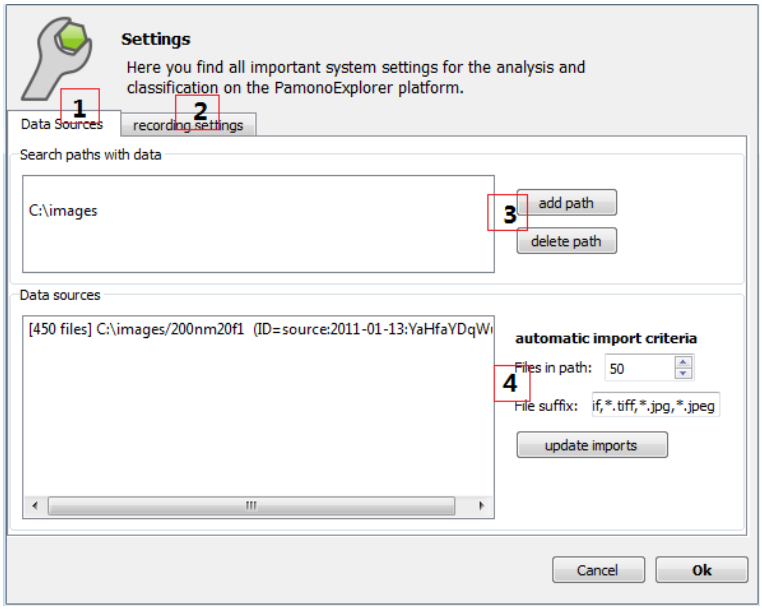


Abbildung B.5.: Das Einstellungsfenster: Hier können Pfade zu Datenquellen gesetzt werden. Der PamonoExplorer importiert automatisch passende Unterordner mit Bilddateien.

Über 2 `Record images` wird der Pfad für die aufzunehmenden Bilder gesetzt. 3 `Further Options` enthält zusätzliche Optionen. Mehr zu 1,2,3 ist im Kapitel B.3.2 erläutert.

Unter 4 befindet sich eine Leiste, die generelle Informationen über das aktuelle Bild zeigt und weitere Einstellungen am Bild vornehmen lässt. Alle Einstellungen sind durch Klicken der Reiter leicht zu erreichen.
Abbildung B.8 zeigt die Intensitätseinstellungen für das aktuelle Bild. Es können jederzeit, auch im laufenden Betrieb, die Parameter Helligkeit, Kontrast, Gamma, sowie die Sichtbarkeit der Markierungen verändert werden.
Die Leiste `Selection + Time series` und `Histogram` werden im Abschnitt B.4.4 und B.4.5 erklärt.

5 beinhaltet die Auswahl-Werkzeugleiste. Diese wird genutzt, um mögliche Partikelfunde oder auch bestimmte Stellen auf dem Bild zu markieren. Hier stehen Punkt-, Kreis-, Rechteck- oder Polygonmarkierungstool zur Verfügung. Für Details sei auf Kapitel B.4.2 verwiesen.

B.3.2. Aufnahme der Bilder

Abbildung B.9 beschreibt die Einstellungsmöglichkeiten der Kamera [2]. Unter 1 wird die IP-Adresse der Kamera eingestellt. Als Standardadresse ist `192.168.0.9` voreingestellt. Das rote Icon rechts wird, wenn eine korrekte Verbindung des PamonoExplorers mit der Kamera

[2] alle Angaben beziehen sich auf die Kamera *Elphel Model 353* und wurden darauf getestet

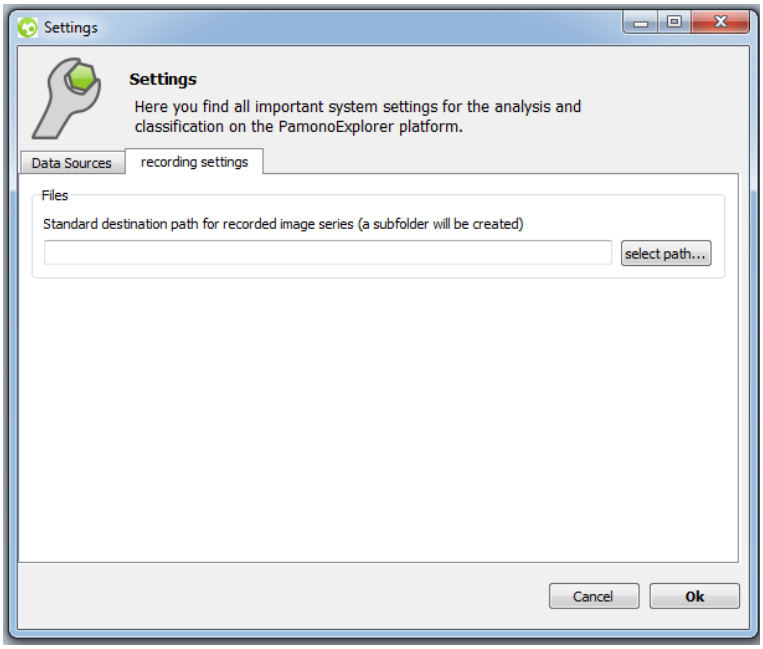


Abbildung B.6.: In diesem Einstellungsfenster wird der Standard-Pfad für die von der Kamera aufgenommenen Bilder gesetzt.

besteht, grün aufleuchten.
Durch Klicken auf `Show Stream` wird das aktuelle Bild der Kamera auf der Anzeigefläche des PamonoExplorers dargestellt. Der Stream kann pausiert oder komplett gestoppt werden 2. Die Verbindung zur Kamera wird beim Stoppen des Streams getrennt.
Unter 3 ist es möglich die automatische Belichtungszeit `Auto exposure`, die Qualität `Quality`, die Belichtungszeit `Exposure Time` und die Window of Interest `WOI` einzustellen. Die WOI kann durch Angabe oder per Maus (auf `WOI` klicken) bestimmt werden.

Mit `Record Images` wird eine Leiste geöffnet, in welcher der Pfad 1 der zu aufzunehmenden Bilder vergeben werden kann, siehe Abbildung B.10. Es wird der globale Standardpfad mit einem Unterordner mit aktuellem Zeitstempel voreingestellt. **Es wird empfohlen, hier einen aussagekräftigen Unterordner zu wählen.**
Die Namen der Bilddateien können mit Feld 2 vergeben werden. Das Namensmuster besteht aus festen Zeichen, einem Zahlenformat (z.B. „%04i“ steht für eine Zahl mit vier führenden Nullen) und der Dateiendung „.tif“. Im gezeigten Beispiel ergibt sich die Dateifolge $image\_0000.tif, image\_0001.tif, image\_0002.tif, \ldots$ Zusätzlich kann ein Timer 3 eingestellt werden, sodass solange aufgenommen wird, bis die in Sekunden angegebene Zeit abgelaufen ist.

Wie in Abbildung B.11 dargestellt, lassen sich weitere Optionen für die Analyse einstellen. Unter 1 kann ein Hintergrundbild ausgewählt werden, welches sich vom aktuellen Bild abziehen lässt, um Störungen bei der Aufnahme auszugleichen. Falls gewünscht, ist es möglich, nach einer bestimmten Anzahl an Frames das ausgwählte Hintergrundbild automatisch abzuziehen. Unter 2 lässt sich die Anzahl der zu mittlenden Bilder konfigurieren. Hohe Mittelungswerte

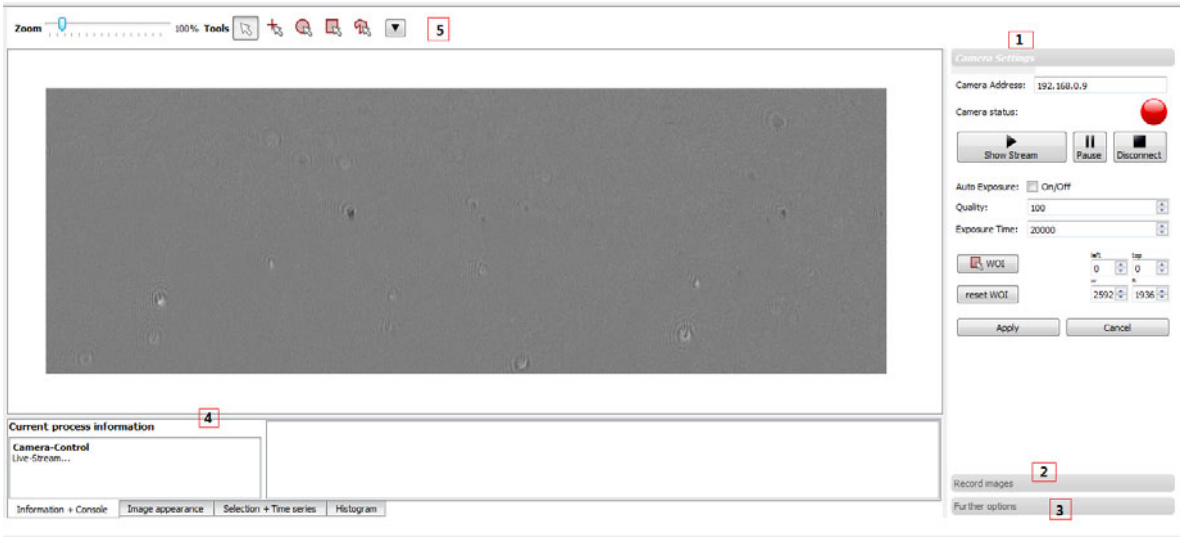


Abbildung B.7.: Die Kameravorschau und Einstellungensmöglichkeiten.

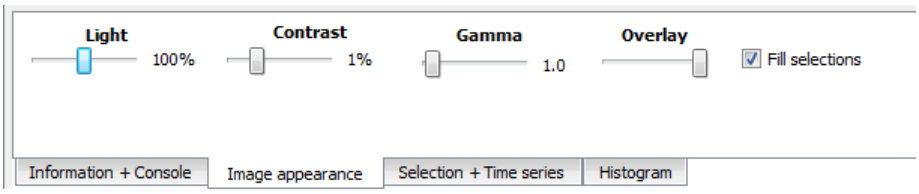


Abbildung B.8.: Intensitätseinstellungen für das aktuelle Bild.

sorgen für geringere Datenmengen und kompensieren das Rauschen. Beide Optionen 1,2 sind standardmäßig deaktiviert.

B.4. Bilddatenerkundung und Training des Klassifikators

Nachdem Bilddaten aufgenommen und in einen Ordner auf der Festplatte gespeichert wurden, wird dieser Ordner als Datenquelle importiert, sofern sich der Ordner innerhalb der Suchpfade befindet. Diese Pfade können in den globalen Einstellungen des PamonoExplorers angepasst werden, mehr dazu in Kapitel B.2.2. Diese Bilddaten können nun erkundet werden und dort befindliche Partikel als solche vermerkt und klassifiziert werden. Trainierte Samples können für die automatische Analyse verwendet werden, um ähnliche Strukturen zu finden.

B.4.1. Trainingsmodus starten

Bevor im Trainingsmodus die Bilder betrachtet und mögliche Partikelfunde markiert werden können, müssen die Bilder eingelesen werden. Dies geschieht mit Hilfe des „Training-Wizards“, welcher in Abbildung B.12 zu sehen ist.

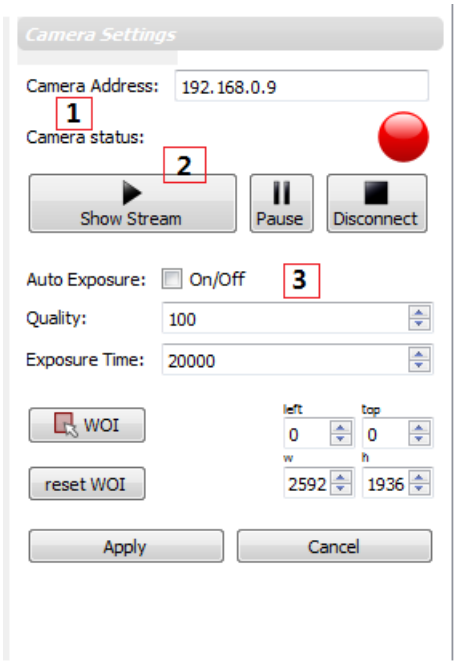


Abbildung B.9.: Einstellungsmöglichkeiten der Kamera.

Zuerst muss der Pfad der Bilder, die genutzt werden sollen, bestimmt werden. Die Auswahl 1 listet alle Pfade auf, die vorher importiert worden sind.
Sollen die Bilder, die eingelesen werden sollen, gemittelt werden, ist dies mit 2 möglich. Wurden bei der Aufnahme schon Bilder gemittelt, sollte hier der Wert eins eingestellt werden. Dadurch werden alle Bilder ohne Mittelung geladen.
Zusätzlich lässt sich unter 3 einstellen, wie viele Bilder im Arbeitsspeicher (RAM) gehalten werden sollen. So entsprechen 1000 Bilder etwa 1GB an RAM. Unter 4 wird der maximal zu verwendende Arbeitsspeicher festgelegt. Es wird nicht mehr an Arbeitsspeicher beim Einlesen verbraucht als angegeben. Falls mehr Bilder vorhanden sind, die die festgesetzte Grenze an Speicher überschreiten, wird der Vorgang vorher abgebrochen. Dadurch wird gewährleistet, dass die Geschwindigkeit des gesamten Rechners von der Arbeit mit dem PamonoExplorer nicht beeinträchtigt wird.
Die voreingestellten Werten sind für die meisten Zwecke sinnvoll.
Durch Klicken auf `Start Exploring` wird der Trainingsmodus und das Einlesen der Bilder gestartet.

B.4.2. Trainingsmodus verwenden

In Abbildung B.13 ist die volle Ansicht des Trainingsmodus zu sehen. Wie schon erwähnt, ist das Interface ähnlich dem Interface der Kameraaufnahme und der automatischen Klassifikation gehalten.

Die Werkzeugleiste 1 enthält die Punkt-, Kreis-, Rechteck- und Polygonmarkierungswerkzeu-

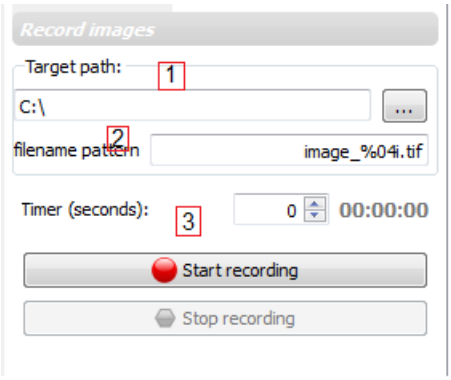


Abbildung B.10.: Der Pfad und das Dateinamen-Muster der zu aufzunehmenden Bilder kann hier vergeben werden.

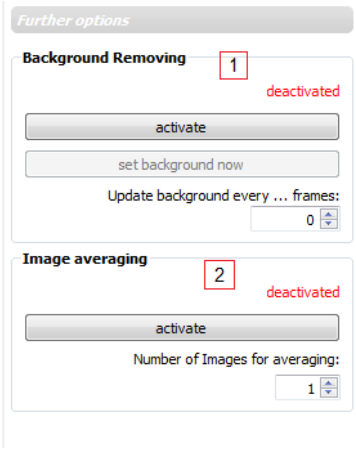


Abbildung B.11.: Zusätzliche Einstellungen für die Aufnahme.

ge (siehe auch Abbildung B.14). Das Punkmarkierungswerkzeug setzt einen Punkt auf die markierte Stelle, zur Veranschaulichung erscheint dort ein Kreuz.
Für das Kreiswerkzeug kann ein Radius eingestellt werden, falls eine größere Markierung gewünscht ist.
Das Rechteckwerkzeug wird per Maus gezogen und es erscheint eine rechteckige Markierung. Schließlich lässt sich noch mit dem Polygonwerkzeug eine polygonale Form zur Markierung erstellen. Jeder Klick erstellt dazu einen neuen Punkt im Polygon. In diesem Modus sind zudem noch Rückgängig-Funktionen enthalten.
Jede Markierung kann einzeln gelöscht werden entweder durch Drücken der Tasten „Entf.“ oder das Auswahlmenü der rechten Maustaste (siehe weiter unten). Sollen alle Auswahlen auf einmal gelöscht werden, so ist dies durch Klicken des Knopfes mit dem schwarzen Dreieck neben den Auswahlwerkzeugen möglich (alle Auswahlen löschen).

Die Leiste 3 enthält Informationen über den aktuellen Prozess und zusätzliche Einstellungen bezüglich des angezeigten Bildes. Zudem wird der aktuelle Status des Einlesens der Bilder unten durch einen Fortschrittsbalken grafisch dargestellt.

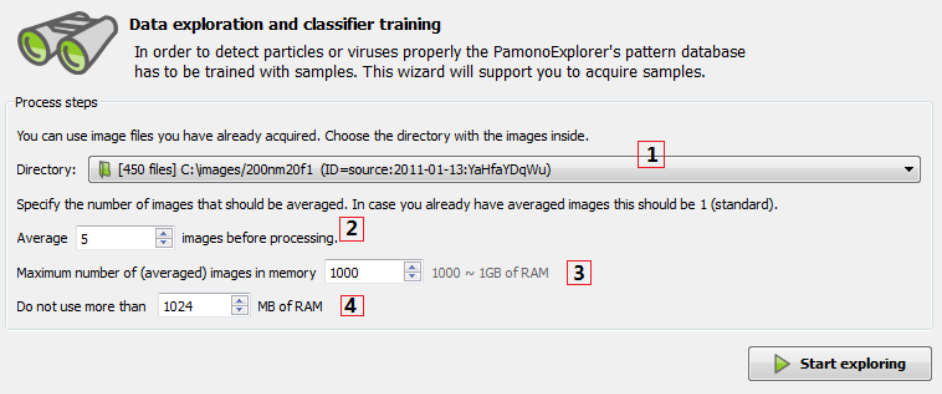


Abbildung B.12.: Im „Training-Wizard“ werden Optionen für das Einlesen der Bilder eingestellt.

Durch Verschieben des Zeit-Sliders (Abbildung B.15) können die bisher eingelesenen Bilder betrachtet werden. Der aktuelle Frame und die Gesamtanzahl der Bilder werden ebenfalls dargestellt.

B.4.3. Anlegen eines Samples

Im Folgenden soll beschrieben werden, wie Samples erfasst und abgespeichert werden. Samples werden benötigt, um Anhaftungen automatisch zu erkennen.

Zuerst wird wie in Abbildung B.16 zu sehen mit dem Zeit-Slider (oder `Shift + Mausrad`) 1 ein Bereich (typischerweise eine fleckartige Struktur) ausgemacht, der in der zeitlichen Dimension charakteristisch erscheint 2. Danach wird mit Hilfe eines Markierungswerkzeugs 3 eine Auswahl **mittig** auf diese Anhaftung gelegt (siehe Abbildung B.17). Die Kreis-Auswahl ist dabei am einfachsten zu verwenden, da hier ein Mittelpunkt auf den hellsten Punkt gesetzt werden kann, siehe 5. Der Radius kann mit 4 auch nachträglich angepasst werden. Es wird automatisch eine ausreichend große Umgebung[3] sowie die Region selbst zur Analyse gewählt. **Die genaue Umrandung muss nicht manuell ausgewählt werden, sondern ergibt sich aus statistischen Streumaßen.** Der Zoom kann behilflich sein, um die betrachtete Fläche zu vergrößern (`Strg + Mausrad`).

Durch Klicken mit der rechten Maustaste über der Auswahl erscheint ein Auswahlmenü, siehe Abbildung B.17 Teil b).

Für die markierte Stelle kann nun ein Histogramm oder eine Zeitreihe angezeigt werden (siehe auch Abschnitt B.4.4 und Abschnitt B.4.5). Die Zeitreihe kann auch als Referenzzeitreihe genutzt werden, um diese mit anderen Zeitreihen zu vergleichen. Dafür muss sie als Referenz im Auswahlmenü markiert werden. Die markierte Stelle erhält Labels für Histogramm, (Referenz-) Zeitreihe und (Sub-) Klassenname, siehe dazu auch Abbildung B.18.

[3] Es wird eine doppelt so große Umgebung des Sprungbildes für statistische Analysen verwendet. Bei einer Punktauswahl wird eine 11x11 Umgebung um das Pixel gewählt.

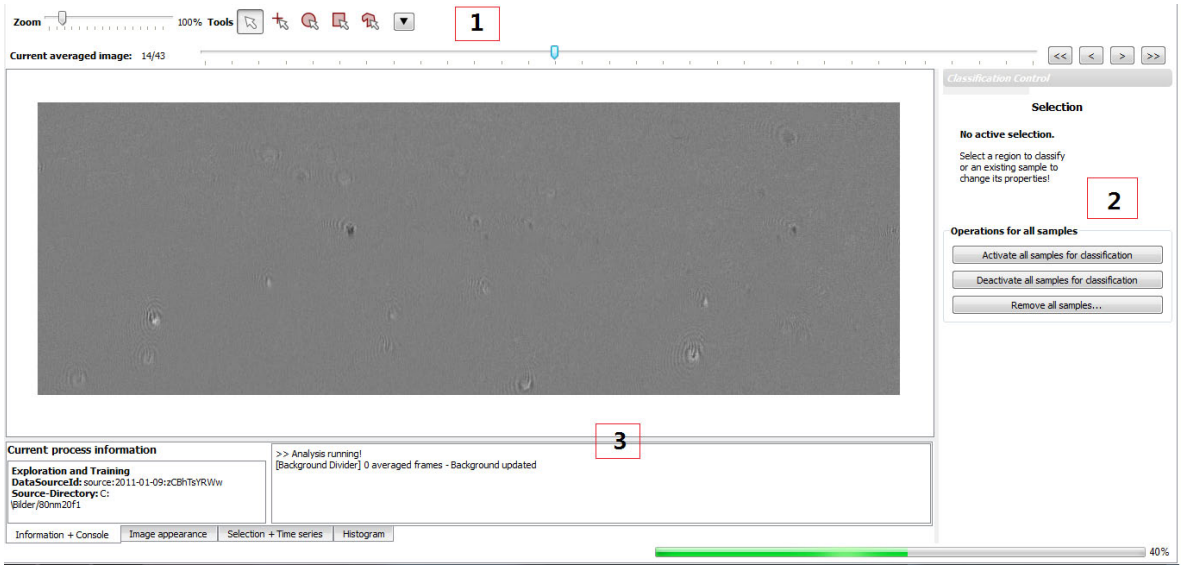


Abbildung B.13.: Der Trainingsmodus in voller Ansicht.

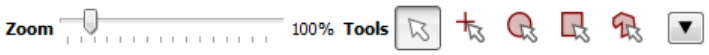


Abbildung B.14.: Die Markierungswerkzeuge und die Zoom-Funktion.

Wurde eine Fundstelle markiert, muss diese klassifiziert werden. Dazu wird auf `Classify selection` 6 geklickt, wie in Abbildung B.17 zu sehen. Der Knopf erscheint nach einer Markierung automatisch auf der rechten Leiste des Programms (in Abbildung B.13 2 nicht zu sehen). Hiernach erscheint ein Menü, in dem die Markierung klassifiziert werden kann. Ein Screenshot dazu findet sich in Abbildung B.19 a).

Die Klassenzuordnung geschieht in drei Schritten:

1. Der Klassenname und Subklassenname werden in den Feldern 7 vergeben. Hier sollen ähnliche Objekte *einer* Klasse zugeordnet werden. Leicht unterschiedliche Charakteristiken können durch Subklassen differenziert werden. So könnte ein Sample etwa das Label „Virus“ und das Sublabel „200nm“ erhalten. Bei der automatischen Analyse können Sublables auch ignoriert werden, die Samples werden dann zu einer Klasse (der des Hauptlabels) gruppiert. Das Sample wird an die Klassifikation weitergegeben, wenn das Häkchen unter `Use sample for classification` angehakt ist.

2. Die charakteristische Sprungstelle wird ermittelt, in dem auf `Automatically find jump` 8 geklickt wird. Das Ergebnis der Zeitreihenanalyse wird in einem Zeitreihenfenster wie in Abbildung B.19 b) gezeigt. Die gefundene Sprungstelle wird markiert. Diese kann auch manuell gesetzt werden, wenn die Automatik nicht das korrekte Ergebnis liefert[4]. Dazu wird der Sprungindex im Zeitreihenfenster mit Hilfe des Mauszeigers ermittelt (Zeitpunkt unter Mauszeiger wird eingeblendet) und in das Feld neben `Jump time index` eingegeben. Bestätigt

[4]Es wird eine Warnung ausgegeben, wenn die Automatik mehrere mögliche Sprungpositionen gefunden hat.

Abbildung B.15.: Der Zeit-Slider ermöglicht ein einfaches Betrachten der Bilder.

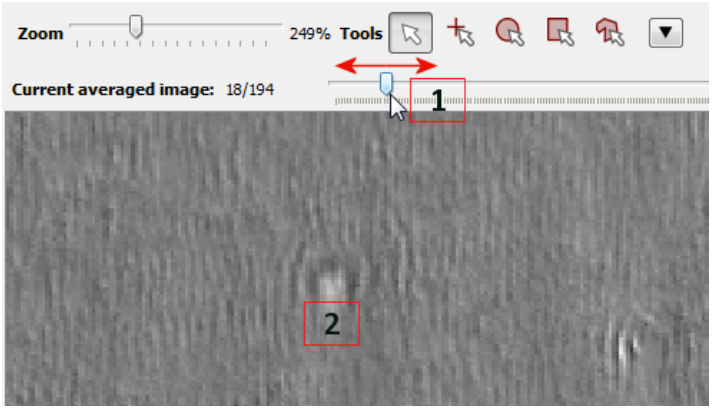


Abbildung B.16.: Durch Verschieben des Zeit-Sliders können Anhaftungen lokalisiert werden.

wird mit Klick auf den Knopf `Set manually`. Der Bildindex, in dem die Sprungstelle auftritt, kann angezeigt werden mit `Go verify frame`.

Es werden zusätzlich weitere nützliche Informationen über die Sprungstelle angezeigt, wie die gemittelte Intensität vor und nach dem Sprung und die prozentuale Intensitätszunahme des Sprunges. Das `window size` bezieht sich auf die gemessene Fenstergröße vor und nach dem Sprung für die Intensitäten. Wird die Fenstergröße geändert, so ändern sich ebenfalls die genannten Werte. Mit `Help` wird eine Ansicht aufgerufen, die in einer kleinen Darstellung die einzelnen Parameter graphisch erklärt. Abbildung B.20 zeigt die Bedeutung dieser Werte.

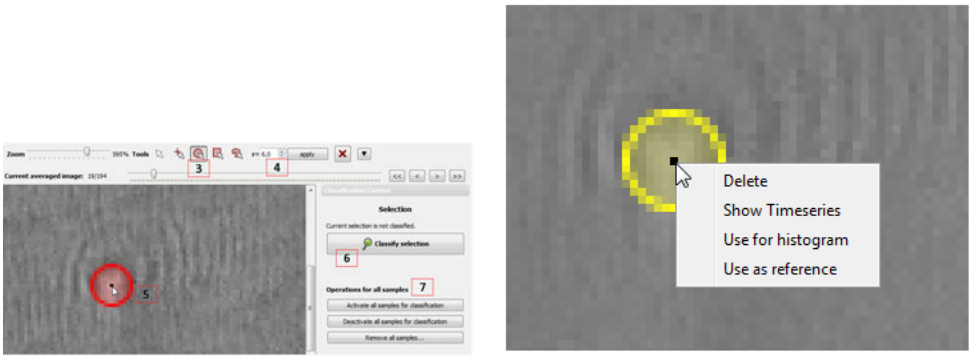
3. Speichern des Samples mit Klick auf `Save Sample` 9. Die gespeicherten Samples stehen später für das automatische Klassifizieren bereit. In Abbildung B.21 wird das Ergebnis des Samples gezeigt.

Für eine erfolgreiche automatische Klassifikation sollten, wenn der Sigma-Template-Matching Algorithmus genutzt werden soll, 5-15 Samples gespeichert werden und für den Bayes-Klassifikator mindestens 10 Samples[5]. Dabei werden alle aktiven Samples mit dem gleichen Klassenlabel aus allen verfügbaren Bilddatenquellen verwendet. In Abbildung B.22 ist beispielhaft eine Menge an Samples zu sehen.

B.4.4. Histogramm

Falls das Intensitäts-Histogramm einer markierten Stelle aufgerufen werden soll, muss die Stelle als Histogramm durch das Auswahlmenü gekennzeichnet, in der unteren Informationsleiste das `Histogram`-Tab aufgerufen und auf `Show histogram` geklickt werden, wie in Abbildung

[5]Die Algorithmen werden im Kapitel zur Analyse näher beschrieben.



(a) (b)

Abbildung B.17.: (a) Zur Auswahl und Untersuchung einer Region wird eine Kreisregion mit passendem Radius auf den hellen Bereich in der Mitte gelegt. Der „Klassifizieren“-Knopf an der rechte Seite erscheint, wenn eine nicht klassifizierte Region gewählt wurde. Unter Punkt 7 finden sich Operationen, die auf alle Samples in diesem Datensatz wirken (aktivieren, deaktivieren, löschen). (b) Per Rechtsklick auf eine Auswahl lässt sich diese löschen, Zeitreihen zu der Region anzeigen, das Histogramm der Intensitätsverteilung innerhalb der Region berechnen und die Region als Referenz-Zeitreihe setzen.

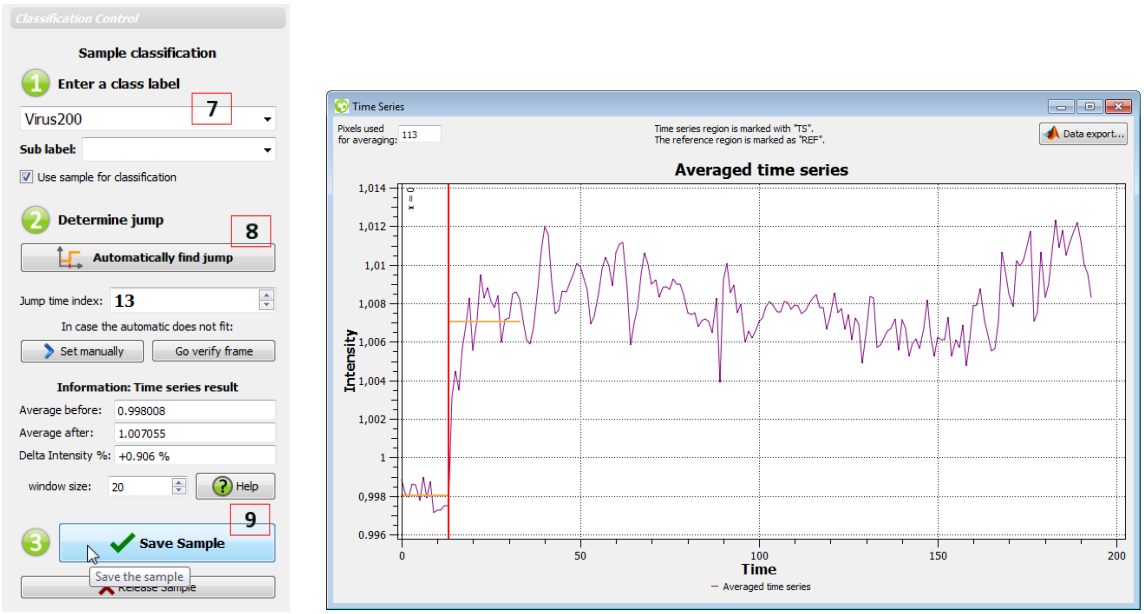
Abbildung B.18.: Eine markierte Stelle und zugehörige Labels: Klassenname, Subklassenname, Zeitreihe (TS), Referenzzeitreihe (REF), Histogramm (HIS).

B.23 verdeutlicht. Die Taste „H“ der Tastatur kann auch eingegeben werden.

Es erscheint ein Fenster, wie in Abbildung B.24, in dem das Histogramm dargestellt wird. a zeigt die Anzahl der Pixel, die zum Erstellen des Histogramms genutzt wurden, an. Unter b werden Informationen über die Intensitätsverteilung angezeigt.
Wird mit der Maus über das Histogramm gefahren, erscheinen am Cursor die aktuellen Werte des Histogramms.
Falls das aktuell angezeigte Bild geändert wird, aktualisiert sich das Histogramm automatisch mit.

B.4.5. Zeitreihe

Die Ansicht der Zeitreihe ist der Ansicht des Histogramms ähnlich. Falls die Zeitreihe einer markierten Stelle aufgerufen werden soll, muss die Stelle als Zeitreihe durch das Auswahlmenü gekennzeichnet, in der unteren Informationsleiste der `Time series`-Reiter aufgerufen und auf `Time series for selection` geklickt werden, wie in Abbildung B.25 verdeutlicht. Alternativ



(a) (b)

Abbildung B.19.: (a) Die Oberfläche zur Klassifizierung (Klassenlabel und Zeitreihenanalyse) (b) Zeitreihe einer Region mit eingepasstem Sprung (rote vertikale Linie).

kann auch die Taste „T“ auf der Tastatur verwendet werden. Zusätzlich werden in der Leiste noch Informationen über die Markierung geliefert, z.B. welches Markierungswerkzeug gewählt wurde und die zugehörigen Koordinaten der Markierung.

In Abbildung B.26 sind zwei Graphen zu erkennen. Der Graph der Referenzzeitreihe ist für eine bessere Übersicht in grüner Farbe und der Graph der Zeitreihe der aktuellen Markierung lilafarben gehalten.
Wird mit der Maus über die Zeitreihe gefahren, erscheinen am Cursor die aktuellen Werte der Zeitreihe.
Durch den Datenexport lassen sich die Werte der Zeitreihen auf die Festplatte exportieren.

B.5. Analyse

Der PamonoExplorer ermöglicht eine automatische Erkennung von antrainierten Strukturen mit Hilfe von verschiedenen Verfahren. Dabei kann keine Garantie auf korrekte Ergebnisse gegeben werden, es muss eventuell durch manuelle Verifikation eine optimale Parameterkombination für das verwendete Analyseszenario ermittelt werden.

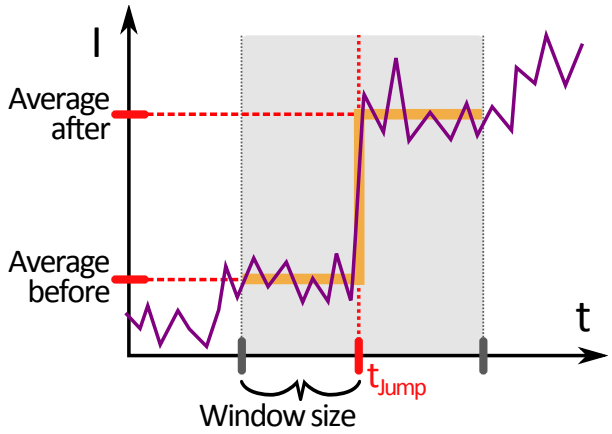


Abbildung B.20.: Erklärung der Ergebnisse der Zeitreihenanalyse. Es wird nach dem größten Anstieg im Signal gesucht - innerhalb eines Fensters werden Mittelwerte vor und nach diesem Sprung berechnet und angezeigt.

B.5.1. Start einer Analyse

Bevor eine automatische Klassifikation starten kann, können auch hier mit dem „Analyse-Wizard“, wie beim „Trainings-Wizard“, einige Einstellungen vorgenommen werden. Der Analysewizard ist in Abbildung B.27 abgebildet.

Zuerst muss der Pfad der Bilder, die analysiert werden sollen, gewählt werden. Die Liste 1 enthält alle Pfade, die vorher importiert worden sind. Das Importieren der Pfade wurde in Kapitel B.2.2 erläutert.

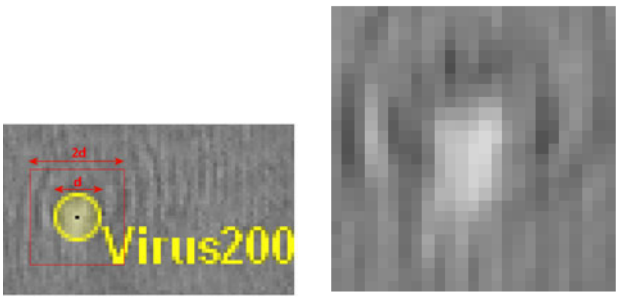
Unter 2 werden Einleseoptionen eingestellt. Die Option `Number of Images to read` bietet die Möglichkeit, alle Bilder in dem gewählten Ordner zu importieren (`-1`) oder eine Grenze zu setzen. Zudem kann hier der Index des Bildes eingegeben werden, mit dem der Import der Bilder beginnen soll.

Unter 3 wird der Analyse-Alogrithmus über eine Auswahl-Liste ausgesucht. Es stehen dafür drei Algorithmen bereit, die genutzt werden können - siehe dazu Abschnitt B.6.

Mit Klick auf `Expert Parameters` 4 öffnet sich ein Dialog mit Experten-Einstellungen bezüglich der Analyse (siehe dazu Abschnitt B.6.1).

Mit `Merge sub classes` werden die Subklassen mit den zugehörigen Klassen zusammengeführt. Die Klassen werden in der kleinen Box dargestellt und können auch abgewählt werden, wenn sie nicht in der Analyse eingesetzt werden sollen.

Sollen Parametereinstellungen gesichert oder alte Einstellungen geladen werden, geschieht das



(a) (b)

Abbildung B.21.: (a) Nach Anlegen eines Samples erscheint der gewählte Klassenname neben der Auswahl, welche gelb dargestellt wird. Wird eine Auswahl der Größe d als Sample gewählt, so geht automatisch eine doppelt so große, rechteckige Umgebung in den Trainingsvorgang ein. (b) Visualisierung der gespeicherten differenziellen Sprungmatrix, welche zu dem Sample gespeichert wird.

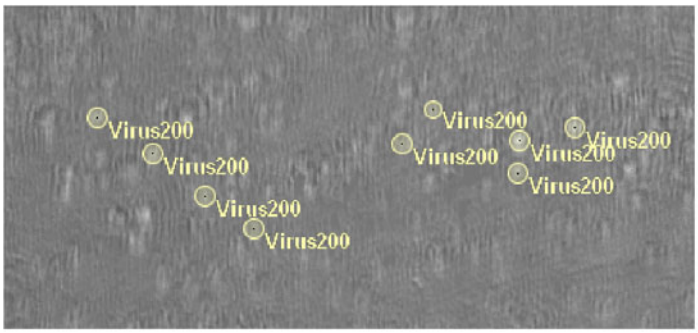


Abbildung B.22.: Nach Wahl von 5-15 charakteristischen Anhaftungen kann klassifiziert werden.

durch 6 `save settings...` oder `load settings...`. Die Schaltfläche `Default settings` lädt voreingestellte Werte.

Schließlich wird mit Klicken auf `Start analysis` die Analyse gestartet.

B.5.2. Analysefenster

In Abbildung B.28 ist eine Ansicht des Analysefensters während einer Analyse zu sehen. Da schon in den vorherigen Kapiteln die obere Toolbar und die untere Informationsleiste erläutert wurden, sind sie hier nicht vollständig abgebildet und sollten dem Nutzer mittlerweile vertraut sein. Das Augenmerk liegt nun auf der rechten Leiste des Programms.

Sollen Anmerkungen oder Notizen zur aktuellen Analyse gemacht werden, ist dies unter 1 möglich. Ein kleiner Text kann dort eingetragen und abgespeichert werden.

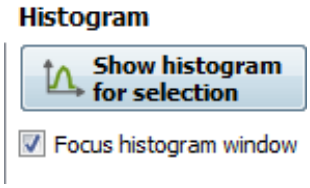


Abbildung B.23.: Der Knopf zum Anzeigen des Histogramms befindet sich in der unteren Informationsleiste im Histogram-Reiter

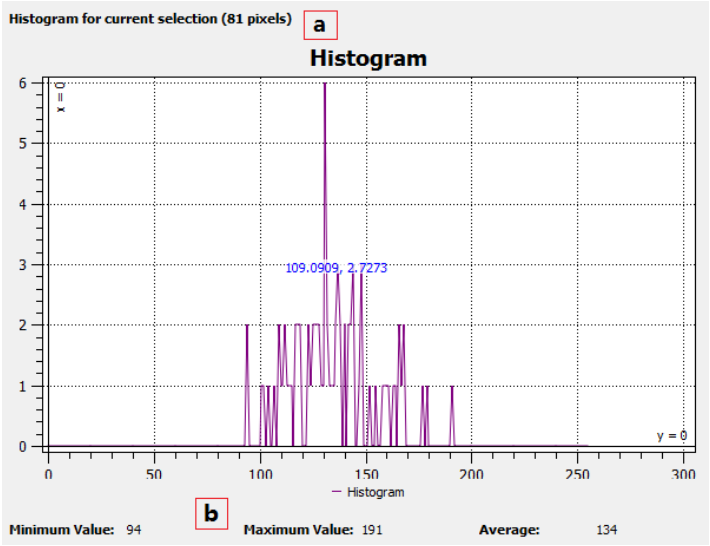


Abbildung B.24.: Das Histogramm einer markierten Auswahl.

Die Analyse lässt sich jederzeit pausieren, fortsetzen, stoppen und neu starten 2.

Die Fundergebnisse werden in der Box 3 nach Klassen aufgelistet.

Unter 4 können, wie beim „Analyse-Wizard“, mit Klick auf `Open Analysis parameters...` fortgeschrittenen Einstellungen vorgenommen werden (siehe dazu Abschnitt B.6.1). Einstellungen können hier ebenfalls gespeichert werden. Mit `Start new analysis` wird die Analyse unterbrochen und komplett neu gestartet.

Soll der Hintergrund neu abgezogen werden, lässt sich das unter 5 mit `Remove background now` realisieren. Der Drehknopf links davon verzögert (in *ms*) die Aktualisierung der Bilder in der Ansicht während der Analyse, falls diese zwecks manueller Verifikation zu schnell ablaufen sollte.

Ist die Analyse beendet, erscheint ein Knopf, der die Seite mit den Statistikausgaben öffnet, siehe Abbildung B.29.

Abbildung B.25.: Der Button zum Anzeigen der Zeitreihe befindet sich in der unteren Informationsleiste im Zeitreihen-Tab.

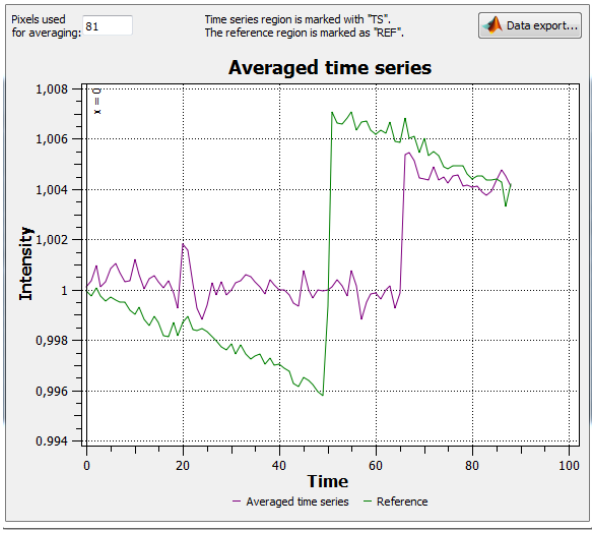


Abbildung B.26.: Die Zeitreihe einer markierten Auswahl: Der Graph der Referenzzeitreihe in grün und der Graph der Zeitreihe der aktuellen Markierung in lila

B.6. Analyseverfahren

Hier soll erläutert werden, wie die einzelnen Analyse-Algorithmen funktionieren und welche für Experimente gewählt werden sollten:

- Sigma-Template-Matching: Aus den gewählten Samples wird ein Mittelwert-Sample berechnet, sowie die pixelweise Standardabweichung der einzelnen Samples zu diesem Mittelwert-Bild. Als Bewertung wird ein statistischer Konfidenz-Test pro Pixel im Template-Bildbereich durchgeführt. Es werden mindestens fünf Samples pro Klasse empfohlen. Die Stabilität des Verfahrens ist gut.

- Bayes-Klassifikator: Aus den Samples werden Features gewonnen, welche in einem Bayes-Klassifikator verwendet werden. Es werden mindestens zehn Samples pro Klasse empfohlen. Die Stabilität des Verfahrens ist gut.

- GPU-Methode: Hier handelt es sich um eine experimentelle Version des Sigma-Template-Matching Algorithmus auf der Grafikkarte. Es wird eine höhere Performance als auf der CPU angestrebt. In Version 1.0 ist das Verfahren aber noch nicht ausreichend getestet.

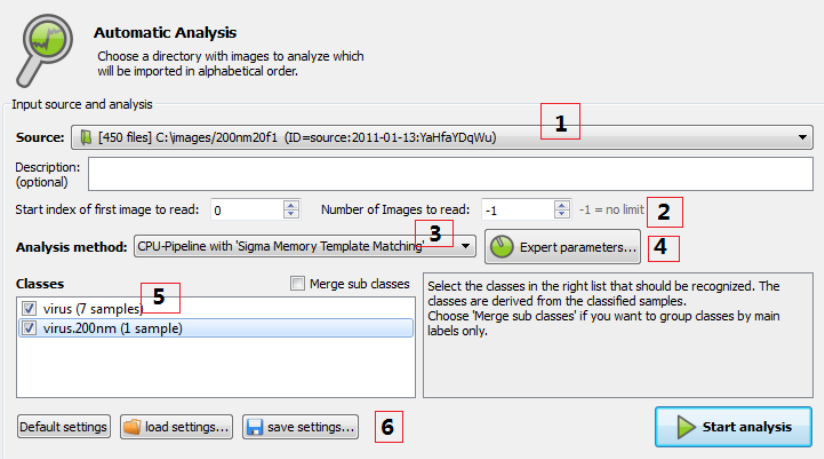


Abbildung B.27.: Mit dem „Analyse-Wizard“ können vor der Analyse Einstellungen vorgenommen werden.

B.6.1. Parameterfenster

Das Parameterfenster in Abbildung B.30 ermöglicht es, weitere Einstellungen vorzunehmen. Die wichtigsten Parameter sollen im Folgenden kurz erwähnt werden.

- **Image averaging**: Anzahl der zu mittelnden Bilder.
- **Background Removing**: Intervall, nach dem der Hintergrund neu abgezogen werden soll. Ist der Wert „0“ eingestellt, so wird der Hintergrund nur einmal am Anfang abgezogen.
- **Timeseries Analysis**: Es wird eine Vorauswahl an Kandidaten bestimmt. Diese wird mit dem Sprungindikator bestimmt, welcher hier eingestellt werden kann. Die minimale und maximale prozentuale Sprunghöhe wird als Step criterion trigger eingetragen. Hinweise auf brauchbare untere Schranken lassen sich im Trainingsmodus bei der Zeitreihen-Analyse gewinnen.
- **Sigma-Template-Matching**: Wenn der Sigma-Template-Matching-Klassifikator gewählt ist, werden diese Einstellungen aktiv. Die Einstellung Confidence Interval gibt die Breite des akzeptierenden Konfidenz-Intervalls in Prozent an. Je näher der Wert an 100% liegt, desto mehr Funde werden markiert. Der Wert Outlier Punishment gibt den Bestrafungsfaktor für einzelne Ausreißer-Pixel beim Template-Matching an. Je größer dieser Wert ist, desto genauer muss die Region mit dem angelernten Template übereinstimmen, um als Fund erkannt zu werden.
- **Bayes-Parameter**: Wenn der Bayes-Klassifikator gewählt ist, werden diese Einstellungen aktiv. Der Wert Rejection Probability gibt die Schwelle an, ab der Kandidaten abgelehnt werden. Ein kleinerer Wert sorgt hier für mehr positive Funde.
- **Segmentizer**: Hier wird die Methode gewählt, mit der positiv klassifizierte Pixel grup-

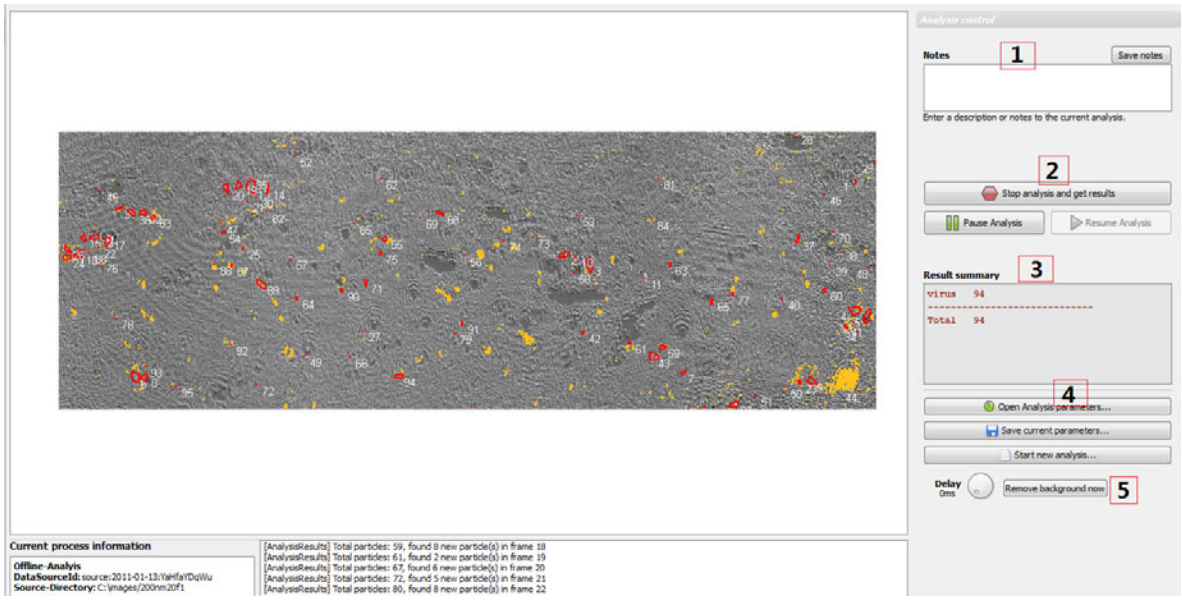


Abbildung B.28.: Die Ansicht des Analysefensters während einer Analyse. Kandidaten werden mit gelb hervorgehoben. Fundstellen erscheinen mit roter Umrandung.

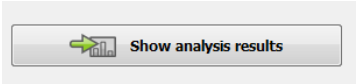


Abbildung B.29.: Der Button öffnet die Seite mit den Statistikausgaben.

piert werden. Hier stehen *Clustering* und *Region Growing*-Verfahren zur Verfügung.

B.7. Ergebnisse

Der PamonoExplorer exportiert vielfältige Statistiken nach jeder Analyse, welche mit allen gängigen Programmen, die das CSV-Format (Comma-Separated-Values) importieren können, weiterverwendet werden können.

B.7.1. Auswertungen

Nach jeder Analyse werden Ergebnisse der Analyse erstellt. In Abbildung B.31 finden sich alle Auswertungen des Projekts in der linken Liste 1. Dort sind alle Unterordner mit ihren sortierten Zeitstempeln aufgelistet. Die Ergebnis-Ordner lassen sich auch löschen, wenn diese nicht mehr gebraucht werden. Wird nun auf einen Ergebnis-Ordner geklickt, erscheint in 2 die Liste der zur Auswertung gehörenden Dateien. Welche Datei dabei welche Daten liefert ist in der folgenden Tabelle aufgelistet:

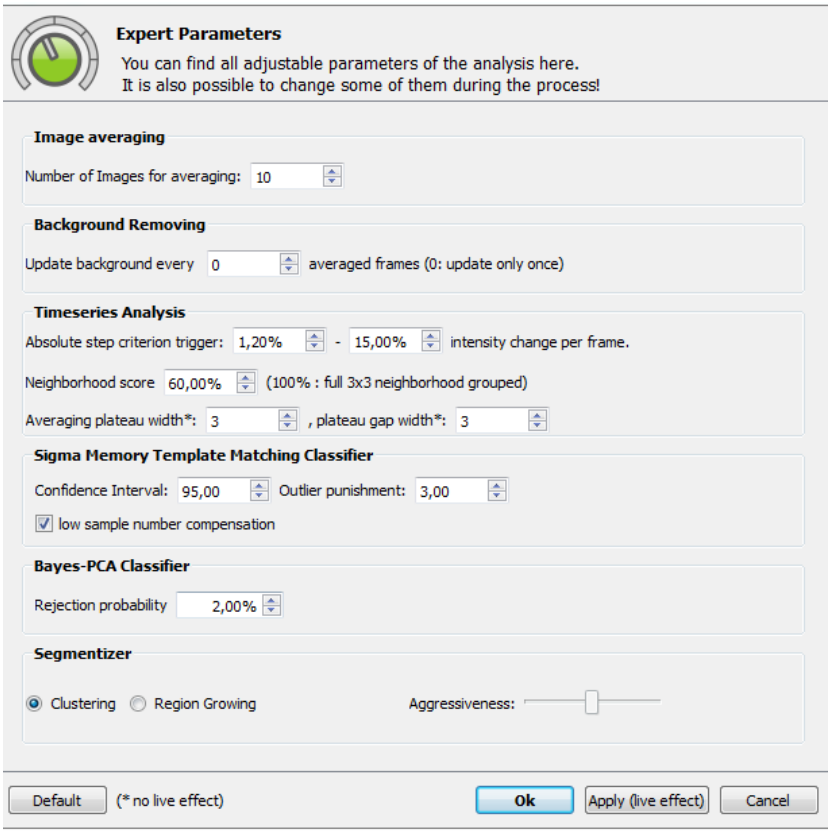


Abbildung B.30.: Die einstellbaren Experten-Parameter.

Dateiname	**Inhalt**	**gnuplot-kompatibel**
analysisInfos.txt	Allgemeine Infos zu der aktuellen Analyse (Datum, Datenquelle mit Pfad, verwendete Klassen sowie Übersicht über die Ergebnisse)	-
analysisParameters.pas	Verwendete Analyseparameter als XML-Datei, welche auch wieder geladen werden können.	-
analysisSnapshot.png	Letztes Bild der Analyse zur schnellen Übersicht über die Fundstellen	-
notes.txt	Datei mit den Anmerkungen und Notizen zu der Analyse	-
particlesOverIntensity.csv	Liste Sprungintensität zu Partikelzahl	ja
particlesOverSize.csv	Liste Bereichsgröße zu Partikelzahl	ja
particlesOverTime.csv	Liste Zeit (Index) zu Partikelzahl (zu Zeitpunkt) und kumulierte positive Funde	ja

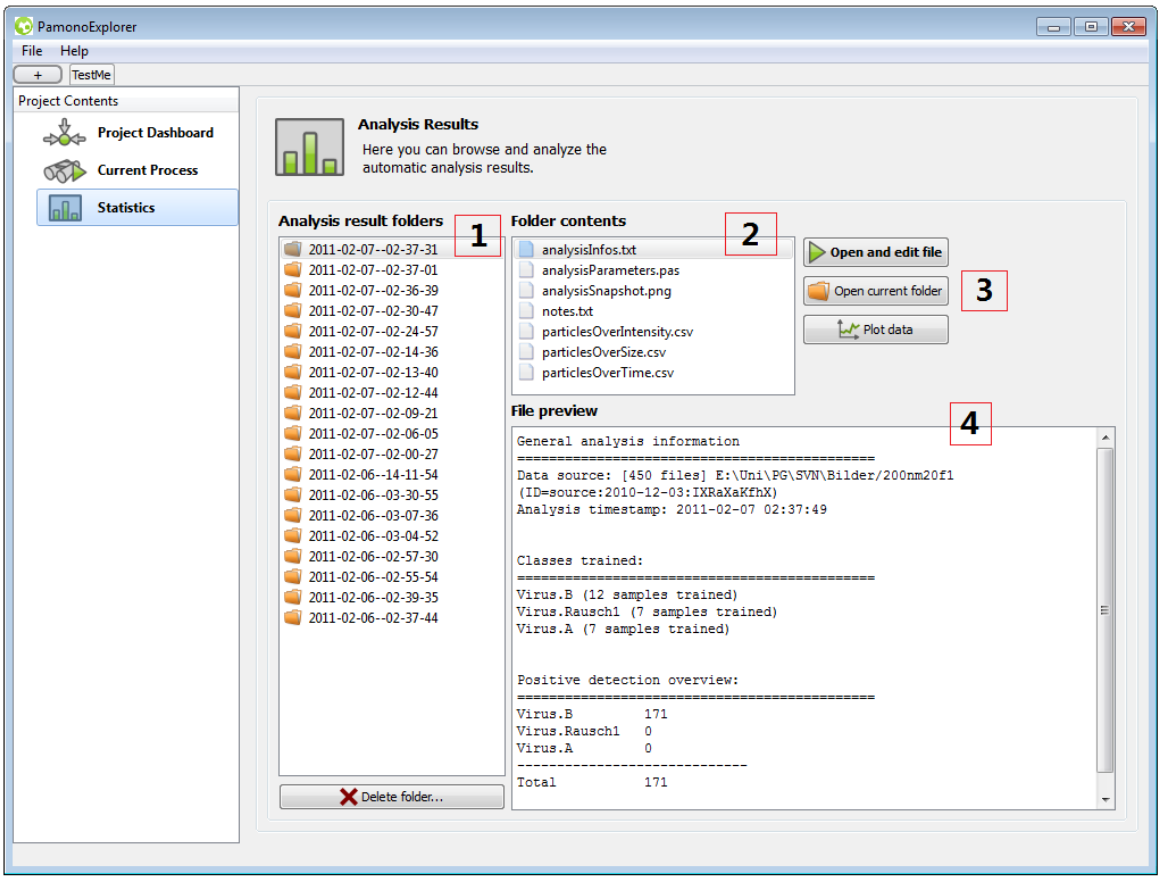


Abbildung B.31.: Das Fenster für Ansicht und Ausgabe von Statistiken.

B.7.2. Export

Des weiteren ist es möglich, einzelne Ergebnisse unter 3 per `Open and edit file` mit externen Programmen zur Weiterverarbeitung zu öffnen, sofern entsprechende Programme installiert sind.
Es besteht eine Verbindung zu *gnuplot*, einem Graph-Zeichen-Programm, in dem die Ergebnisse in CSV-Dateien graphisch dargestellt werden können.
Das Öffnen des übergeordneten Verzeichnisses in einem lokalen Browser ist ebenfalls integriert, wenn auf `Open current folder` geklickt wird.

Die Ergebnisse sind in weit verbreiteten Formaten (`.csv`, `.txt` und `.png`) abgelegt, um eine schnelle Weiterverarbeitung der Daten (mit anderen Programmen) zu ermöglichen.

Für eine schnelle Ansicht der Ergebnisse ist eine kleine Anzeige-Box 4 integriert. Wenn auf ein bestimmtes Ergebnis geklickt wird, erscheint eine Vorschau des Datei-Inhalts.

C. Pflichtenheft

C.1. Zielbestimmungen

C.1.1. Motivation

Die Cool-IP-Applikation dient dem Zweck einer automatischen Erkennung von Viren in Flüssigkeiten, wie zum Beispiel Blut. Dazu wird die Flüssigkeit mit Viren durch eine Flusszelle geleitet, wobei es zu einer Bindung der Viren an einer zu beobachtenden biologisch präparierten Sensoroberfläche, einem Biosensor, kommt. Während der Durchleitung wird mit einer CCD-Kamera die Oberfläche des Biosensors - diese reflektiert das Licht eines Lasers – beobachtet und Folgen von Bildern werden aufgenommen. Ein an die Sensorfläche anhaftender Virus hebt sich mit einer prägnanten Intensität vom Hintergrund ab.

Die Aufgabe der Projektgruppe 548 ist die Entwicklung einer automatisierten Bildverarbeitungs-Kette zur schnellen Detektion von Partikeln, in Vertretung von Viren, und die Angabe der örtlichen Partikelverteilung auf dem Sensorsystem. Im Folgenden wird immer die Rede von Partikeln sein. Da es zu gefährlich wäre, mit echten Viren zu arbeiten, wird mit Partikeln von der Größe typischer Viren gearbeitet.

Der Ablauf einer Probenanalyse kann aus Sicht des Benutzers und der Benutzerin[1] in folgende Ablaufschritte unterteilt werden:

1. Projekt
    a) Anlegen / Laden
    b) Neues Datum anlegen
2. Referenzbild generieren
3. Probe in das Durchflusssystem injizieren
4. Bildreihenerstellung
    a) Festlegen der Dauer

[1] Im Weiteren wird durchgängig die männliche Formulierung verwendet. Dies bezieht das weibliche Geschlechte im vollen Umfang mit ein.

   b) Starten

   c) Beenden nach Dauer / Abbrechen

5. Bildreihe speichern
6. Analyse
   a) On-Line / Off-Line

   b) Optional: Training

   c) Festlegen der Dauer

   d) Starten

   e) Beenden nach Dauer / Abbrechen
7. Ausgabe des Ergebnisses
   a) Detektionsangaben (Anzahl, Liste von Polygonen / Verteilung erkannter Partikel)

   b) Darstellung des relevanten Bildmaterials mit Polygonen (markierte Partikel)

Der Vorgang der Bildverarbeitung wird durch folgendes Prozessmodell veranschaulicht:

1. Bildakquise mittels CCD-Kamera
2. Vorverarbeitung mit dem Kamera-FPGA
   - Verlustfreie Kompression
   - Wenn möglich interessanten Intensitätswertebereich separieren
   - Gegebenenfalls Bildvorverarbeitung
3. Datenübermittlung
4. Laptop: Bildverarbeitung mittels OpenCL-fähiger Grafikkarte
   - Bildreihenanalyse (1D+t)
   - Zeitreihenanalyse (2D+t)
   - Überwacht oder nicht überwacht
5. Klassifikation
   - Gesuchter Partikel: Ja oder Nein

6. Ausgabe
    - Visualisierung (in Echtzeit oder nicht)
    - Statistik

C.1.2. Muss- und Wunschkriterien nach Themen

Musskriterien enthalten Leistungen, die für das Produkt für den vorgesehenen Einsatzzweck unabdingbar sind. Wunschkriterien sind zwar nicht unabdingbar, ihre Erfüllung wird allerdings angestrebt. Im folgenden Abschnitt werden für verschiedene Bereiche zuerst die Musskriterien und anschließend die Wunschkriterien aufgeführt.

C.1.2.1. Geräteüberprüfung

Musskriterien

- Konnektivität zwischen Kamera und Laptop
    - Elphel-Kamera erreichbar
    - Übermittlung von Bildern testen

Wunschkriterien

- Funktionsüberprüfung einiger Komponenten direkt nach dem Start der Software, d.h.
    - Laptopbetrieb: Netzversorgung gegeben beziehungsweise genügend Akku-Energie vorhanden?
    - Lüfterkühlung funktionstüchtig?

C.1.2.2. Projekt-Vorbereitungen

Musskriterien

- Anlegen eines Ordners mit Projektdatum
- Anlegen eines Unterordners mit Bildreihen und Analysedaten
- Projektdatei (XML) enthält Informationen zur Bildreihenerstellung, den Bildreihen und den Analysen

- Nach jeder durchgeführten Analyse wird ein Statistik-Report abgelegt

Wunschkriterien

- Projekte auflisten
- Projekte löschen können

C.1.2.3. Generierung eines Referenzbildes

Musskriterien

- Wird vom Benutzer initiiert
- Benutzerinteraktion: Selektion von ROIs, *Default Setting*: Maximale Sensorfläche
- Mittelung über eine Bilderserie der Lösungsflüssigkeit ohne Probe (z.B. 20 Bilder)
- Anzahl der Bilderserien durch Benutzer justierbar

Wunschkriterien

- Unterstützung des Benutzers bei der Kalibrierung des Messaufbaus

C.1.2.4. Bildreihenerstellung-Vorbereitungen

Musskriterien

- Hinweis anzeigen: „Probe in das Durchflusssystem injizieren“
- Bildreihenerstellungsdauer vom Benutzer festlegbar (Default: 5 Minuten)
- Überprüfung Festplattenspeicher (abhängig von Dauer)

C.1.2.5. Analyse-Durchlauf

Musskriterien

- Analyse vom Benutzer startbar

- Analyse kann vom Benutzer jederzeit abgebrochen werden; Auswertung der Ergebnisse erfolgt bis dahin

C.1.2.6. Visualisierung

Musskriterien

- Liste erstellter Bilder, auswählbar zur Darstellung
- Verarbeitete Bilder werden an eine Zeitleiste gefügt; Slider zur Bildauswahl erscheint nach Diagnose horizontal unter Zeitleiste (nicht in Echtzeit)
- Anzeige der Koordinaten des Mauszeigers im Bild
- Originalframe zum Zeitpunkt t darstellbar
- Verarbeitetes Frame (nach Bildverarbeitung) zum Zeitpunkt t darstellbar
- Darstellung eines Metrikbalkens unterhalb der Bildausgabe
- Markierung von vermeintlich gefundenen Partikeln, um nachvollziehen zu können, wie die Software zu ihrem Ergebnis kommt
- Statistik: Anzahl von Partikeln, Konzentration Partikel/ml von eingegebener Probe
- Liste gefundener Partikel mit:
  - Polygon des Partikels
  - Konfidenz gefundener Partikel

Wunschkriterien

- GUI-Design in Rücksprache mit ISAS
- Verarbeitete Bilder werden in Echtzeit in eine Zeitleiste angefügt (Slider horizontal unter Zeitleiste)
- Aktualisierung der Statistik in Echtzeit
- Verschieben des Sliders unter Zeitleiste wählt das korrespondierende Frame aus
- Konfidenzlevel im Nachhinein vom Benutzer dynamisch wählbar:
  - Änderung der Darstellung von „Markierungsmengen“

  - Änderung der Statistikausgaben

- Liste gefundener Partikel mit jeweiliger Position; bei Auswahl des Partikels in der Tabelle wird die entsprechende Markierung hervorgehoben

C.1.2.7. Datenspeicherung

Musskriterien

- Speicherung im JPG/PNG/TIF-Format (verlustfreie Kompression), vom Benutzer auswählbar
- Bilder zur Einsicht auf Festplatte belassen: RAW und Processed mit Markierungen als Liste von Polygonen im Projektfile
- Platz zur Datenspeicherung vom Benutzer selbstverwaltet

C.1.2.8. Dokumentation

Wunschkriterien

- Kontextsensitive Hilfe

C.1.3. Abgrenzungskriterien

Die Abgrenzungskriterien beschreiben, welche Ziele mit dem Produkt nicht verfolgt werden.

C.1.3.1. Keine Datenbank

Die Software beinhaltet keine Datenbank. Lediglich Bildmaterial sowie Analyseergebnisse werden gespeichert.

C.1.3.2. Keine zusätzliche Verwaltung der Projekte

Die Software bietet keinen Schutz gegen äußere Veränderung der Daten, d.h. der Benutzer ist für die abgelegten Daten selbst verantwortlich. Erweiterte Funktionalitäten sind nicht vorgesehen. Insbesondere gibt es keine Such- oder Sortierfunktionen.

C.1.3.3. Keine Anbindung an andere Softwaresysteme oder Datenbanken

Das Produkt arbeitet als eigenständige Software und sieht keine Anbindung an bereits bestehende Softwaresysteme oder Datenbanken vor.

C.1.3.4. Keine beliebigen Formate

Die gewonnenen Bilddaten werden nicht in beliebigen (Datei-)Formaten gespeichert. Es werden ausschließlich JPG, PNG und TIFF als Dateiformate unterstützt.

C.2. Produkteinsatz

C.2.1. Anwendungsbereiche

Die Cool-IP-Applikation führt eine Analyse der Bildfolgen der PAMONO-Technik zur schnellen Detektion von Partikeln durch und macht Angaben über die örtliche Verteilung von Partikeln auf dem Biosensor.

C.2.2. Zielgruppen

Zielgruppe der Cool-IP-Applikation sind Mitarbeiter des ISAS, dem Leibniz-Institut für Analytische Wissenschaften.

C.2.3. Betriebsbedingungen

Die Betriebsbedingungen für die Cool-IP-Applikation sind durch den physikalischen Versuchsaufbau gegeben. Das System muß ausreichend mit Strom versorgt sein und die Komponenten auf Funktion überprüft werden. Nach dem Generieren eines Referenzbildes muss eine zu analysierende Probe in die Durchflusszelle injiziert werden. Störungen des Versuchsaufbaus jeder Art (zum Beispiel Erschütterungen oder Tageslicht) müssen vermieden werden. Für die Verwaltung des Speicherplatzes ist der Benutzer verantwortlich.

C.3. Produktumgebung

C.3.1. Software

Als Betriebssysteme werden Windows 7 und Linux unterstützt. Voraussetzung bei Linux ist die C-Bibliothek glibc ab Version 2. Als Programmiersprache kommt C++ zum Einsatz. Für die graphische Oberfläche wird die Qt-Bibliothek in der Version v4.6.2 verwendet und für die Bildverarbeitung OpenCL in der Version v1.0.

C.3.2. Hardware

Die Software läuft auf einem Laptop mit folgenden Hardware-Mindestanforderungen:

- Mehrkern-Prozessor ab 2.0 GHz
- 4 GB RAM
- dedizierte Grafikkarte mit mindestens 512 MB und OpenCL-Unterstützung
- 100 GB freier Festplattenspeicher
- 100 MBit Ethernet für die Anbindung an die Kamera
- Netzteil mit vorhandener Netzversorgung

Alternativ laufen Teile der Software auch auf dem Cell-Board MVX8i. Die Bildaufnahme erfolgt mittels einer Elphel 353 mit den folgenden Anforderungen:

- Farbtiefe: mindestens 12 Bit
- 16 FPS bei 1000x1000 Pixeln, max: 50 FPS
- Pixelgröße $2\mu\text{m} \times 2\mu\text{m}$

Als Schnittstellen zwischen Laptop und Kamera dient eine Ethernetverbindung.

C.4. Produktfunktionen

In diesem Abschnitt findet sich eine Übersicht über die bereit gestellten Funktionen des Produkts. Funktionen sind mit */F Funktionsnummer/* eindeutig gekennzeichnet. Trägt die Funktion zusätzlich noch ein *W*, steht dies dafür, dass es sich um ein Wunschkriterium handelt.

C.4.1. Projektverwaltung

- **/F10/ Projekt anlegen**
  - Der Projekt-Name wird vom Nutzer eingegeben. Weitere Daten, die zum Projekt gehören (Proben-Zusammensetzung, zeitlicher Ablauf während der Bildreihenerstellung, Zeitraum der Durchführungen, etc.) werden ebenso eingegeben. Es wird eine XML-Datei angelegt, die diese Daten enthält und in einem Verzeichnis mit dem Projektnamen liegt.
- **/F20 W/ Projekt löschen**
  - Nach einer Rückfrage zum Nutzer mit dem Hinweis, was ggf. gelöscht wird, werden alle zu einem Projekt gehörenden Daten gelöscht.
- **/F30/ Projekt schließen**
  - Das aktuell geöffnete Projekt und alle damit verbundenen geöffneten Bildreihen und Analysen werden geschlossen. Das Programm kehrt in seinen Ausgangszustand zurück.
- **/F40/ Neue Bildreihenerstellung beginnen**
  - Das Datum der Bildreihenaufnahme und ein Freitextfeld werden abgefragt. Die Bildreihe wird in einem Unterordner des aktiven Projekts abgelegt.
  - Der Nutzer hat die Wahl zwischen einer Online-Analyse (mit vorausgewählten Parametern) oder der reinen Bildreihenerstellung zur späteren Offline-Analyse.
  - Bei der reinen Bildreihenerstellung hat der Nutzer die Möglichkeit, eine Zeit einzugeben, nach der die Erstellung automatisch beendet wird; zusätzlich dazu besteht die Möglichkeit, die laufende Bildreihenerstellung zu beenden.
  - Bevor die Bildreihenerstellung beginnt, muss der Nutzer bestätigen, dass der Versuchsaufbau vorbereitet ist.
  - Während die Bildreihenerstellung läuft, wird dem Nutzer ein Schätzwert angezeigt, für welche Aufnahmezeit der freie Festplattenspeicher noch in etwa ausreicht.

- **/F50/ Referenzbild generieren**
  - Zur Unterstützung der Analysealgorithmen stellt der Nutzer den Versuchsaufbau auf und bestätigt dem System, dass es sich bei dem Kamerabild um den Ausgangszustand (keine Probe injiziert) handelt.
- **/F60/ Dauer der Probenanalyse festlegen**
  - Die Dauer, nach der die Analyse automatisch beendet werden soll, wird eingegeben.
- **/F70/ Bildreihe öffnen**
  - Es wird eine Liste aller zu einem Projekt gespeicherten Bildreihen angezeigt. Nach Auswahl wird die gewählte Bildreihe geladen.
  - Die Bildreihe kann in verschiedenen Modi abgespielt werden (Einzelbild, vor, zurück, schnell vor, schnell zurück, „Endlos“-Wiederholung einer Auswahl).
- **/F80 W/ Bildreihe löschen**
  - Der Nutzer wird auf das Löschen der Daten hingewiesen; wenn der Nutzer das Löschen bestätigt, wird die aktuell gewählte Bildreihe gelöscht.
- **/F90/ Bildreihe schließen**
  - Die aktuell geöffnete Bildreihe wird geschlossen. Es wird eine Liste mit den zum aktuell geöffneten Projekt gespeicherten Bildreihen angezeigt.
- **/F100/ Analyse durchführen**
  - Zu einer gewählten Bildreihe wird eine Analyse mit ggf. vom Nutzer wählbaren Parametern durchgeführt.
- **/F110/ Analyse zur aktuellen Bildreihe öffnen**
  - Die zu einer Bildreihe bereits durchgeführte, gespeicherte Analyse wird geladen.
- **/F120/ Analyse abbrechen**
  - Die laufende Analyse wird unverzüglich abgebrochen.
- **/F 130/ Partikel-Markierung aus-/ einblenden**
  - Nach Nutzer-Auswahl werden die vom Progamm erkannten Partikel im Bild markiert oder nicht markiert.
- **/F 140/ Kontexthilfe**

  - Für alle grundlegenden Funktionen wird eine Kontexthilfe angeboten.

- **/F 150 W/ Deutsch als zusätzliche UI-Sprache**

  - Das UI ist auf Deutsch lokalisiert.

C.4.2. Globale Einstellungen

- **/F 160/ Konfigurationsdialog**

  - Der Konfigurationsdialog ermöglicht die Angabe eines Basisverzeichnisses zur Datenspeicherung. Zusätzlich kann die Sprache des UIs voreingestellt werden.

  - Desweiteren kann der Nutzer Trainingsdaten für die Analysealgorithmen in das System importieren.

C.5. Produktdaten

C.5.1. Projektdaten

Die Projektdaten werden in einer XML-Datei abgelegt. Der Name der Datei ist das jeweilige Datum.
Die Datei enthält einen Verweis auf die Bildreihen und die Analysedaten.

C.5.2. Bildreihe

Das Produkt speichert die aufgenommenen Bilder auf der Festplatte. Für je eine Bildreihenerstellung gibt es einen Ordner mit Datum im Projektordner, unter dem die aufgenommene Bildreihe gespeichert wird. Zusätzlich dazu wird mit jeder Bildreihe eine XML-Datei erstellt, in der das aktuelle Datum, die Anzahl der Bilder und deren Auflösung gespeichert ist.
Die Bilddaten liegen wahlweise im TIFF/JPG/PNG - Format vor. Sie werden mittels dem Lempel-Ziv-Welch (LZW) Verfahren verlustfrei komprimiert.

C.5.3. Analysedaten

Die Analysedaten sind in dem Ordner der zugehörigen Bildreihe zu finden. Für je eine Bildreihe gibt es eine Analyse. Auf die gleiche Bildreihe können mehrere Analysedurchläufe angewendet werden. Die Analysen enthalten Statistiken zur zugehörigen Bildreihe über das Auftreten von Partikeln.

C.6. Produktleistungen

- **/L10/ Aussagegenauigkeit**
  - Während der Analyse werden die Partikel in einer Größe und Signalstärke, die für Viren, die in der Natur existieren, typisch ist, gemessen.
  - Als Fehlermaß werden Abweichungen in der Größe eines detektierten Partikels, sowie dessen Form und Signalstärke im Vergleich zu einem idealen Partikel mit einbezogen.
  - Eine absolute Aussage kann dabei nicht getroffen werden, da das Wissen über das tatsächliche Vorhandensein beziehungsweise das Nichtvorhandensein von virusähnlichen Partikeln in der Probe nicht vorliegt.
- **/L20/ Datensatz**
  - Es entstehen in jeder Sekunde zwischen 16 und 50 Bilder.
- **/L30/ Bildgröße**
  - Die maximale Größe eines Bildes beträgt horizontal und vertikal jeweils 1000 Pixel. Die maximale Tiefe eines Bildes beträgt 12 Bit beziehungsweise 16 Bit pro Pixel.
- **/L40/ Bearbeitungszeit**
  - Bei einer Online-Analyse geschieht die Klassifikation ohne Verzögerung.
  - Die Bearbeitungszeit bei einer Offline-Analyse wird so gering wie möglich sein, maximal aber 15 Minuten pro Minute Bilddaten betragen.

C.7. Benutzeroberfläche

- **/B10/ Benutzerschnittstelle**
  - Die Bedienung des Produkts erfolgt graphisch und menüorientiert.
- **/B20/ Eingabegeräte**
  - Das Produkt wird mit einer Tastatur und einer Maus bedient.

C.7.1. Visualisierbare Informationseinheiten

- **/B30/ Analysereport**
  - Die Graphen und Tabellen zu Versuchsauswertungen werden als Grafiken angezeigt.

C.7.2. Datenvisualisierung

- **/B40/ Referenzbilder**
  - Generierte Referenzbilder werden angezeigt.

C.7.3. Datenvisualisierung

- **/B50/ Analyseergebnisse**
  - Je nach Auswahl des Nutzenden werden die gefundenen Partikel im Bild markiert oder nicht.

C.7.4. Datenvisualisierung

- **/B60/ Bilderserie**
  - Die gespeicherte Bilderserie zum aktuell geöffneten Versuch wird angezeigt und mit verschiedenen Modi abgespielt (Einzelbild vor oder zurück, schnell vor oder schnell zurück, Endloswiederholung einer Auswahl von Bildern).

C.8. Qualitätszielbestimmung

Anforderung	normal	gut	sehr gut
/Q10/ Präzision	x		
/Q20/ Portierbarkeit	x		
/Q30/ Benutzerergonomie		x	
/Q40/ Bedienbarkeit	x		
/Q50/ Erweiterbarkeit		x	
/Q60/ Projekmanagement		x	
/Q70/ Autarke Modultests			x

C.8.1. Erläuterungen

/Q10/ Präzision:

Alle Elemente in einem Bild, die Partikel sind, werden erkannt und korrekt klassifiziert.

/Q20/ Portierbarkeit:

Das Programm kann auf jeden Laptop übertragen werden, der die entsprechenden Anforderungen erfüllt, die der in Abschnitt 3 aufgeführten Produktumgebung zu entnehmen sind.

/Q30/ Benutzerergonomie:

Die Benutzeroberfläche des Programms ist verständlich aufgebaut und somit leicht zugänglich. Die Elemente der grafischen Benutzeroberfläche sind so angeordnet, dass die Bedienung des Programm leicht zu erlernen ist.

/Q40/ Bedienbarkeit:

Alle Einstellungen, die für die Bildverarbeitung und die Verbindung mit der Kamera notwendig sind, können über die grafische Benutzeroberfläche vorgenommen werden.

/Q50/ Erweiterbarkeit:

Dem Programm können weitere Funktionalitäten hinzugefügt werden.

/Q60/ Projekmanagement:

Ein Qualitätsmanagementhandbuch einschließlich Dokumentation, Testfälle und Modellierungsgrundsätze wird geführt und gepflegt.

/Q70/ Autarke Modultests:

Die korrekte Funktionalität der einzelnen Komponenten wird sichergestellt.

C.9. Globale Testszenarien und Testfälle

C.9.1. Teststrategie

Um die Leistung des Systems zu testen, wird ein dreiphasiges Testsystem verwendet. Das Gesamtsystem besteht aus den Komponenten Software, Kamera und Durchflusseinheit (PAMONO-Zelle). Da sich die Komponenten teilweise an unterschiedlichen Orten befinden, ist die Untergliederung in 3 Phasen sinnvoll; erst die abschließende Testphase 3 umfasst das Gesamtsystem mit allen Komponenten. Durch die Komponententests kann die Testeffizienz optimiert werden. Die Abbildung C.1 zeigt einen Überblick über das System und hebt die jeweils zu testenden Komponenten hervor.

C.9.2. Testphasen

- **Phase 1:** Testen der Software mit bereits vorhandenen Testbildreihen als Bilderfolge. Abgleich mit manuell verifizierten Partikelfunden (Partikelreferenzkarte).
- **Phase 2:** Zusammenschluss mit der Kamera und Test der Bildübertragung auf Fehler und Bildübertragungsrate. Test der Softwarekomponenten, die auf der Kamera selbst implementiert sind (Kompression und Bildvorverarbeitung).
- **Phase 3:** Zusammenschluss der PAMONO-Durchflusseinheit samt realen Proben - Flüssigkeiten mit virusähnlichen Nanopartikeln. Die Bilddaten werden jeweils manuell ausgewertet und die Ergebnisse mit denen der Software verglichen. Usability-Test der Software auch mit Benutzern außerhalb der PG.

C.9.3. Ziel der Tests

Ziel der Tests ist sicherzustellen, dass die Software alle Testfälle korrekt behandelt und die Kommunikation zwischen Kamera und Software reibungslos funktioniert. Des Weiteren ist auch eine Übereinstimmung der Ergebnisse der Analysen mit denen einer manuellen Bewertung der Bilddaten angestrebt. Da die Projektgruppe nur die Auswertung des Versuchs vornimmt, sind keine Aussagen der Qualität oder Erkennungsrate des PAMONO-Verfahrens selbst vorgesehen.

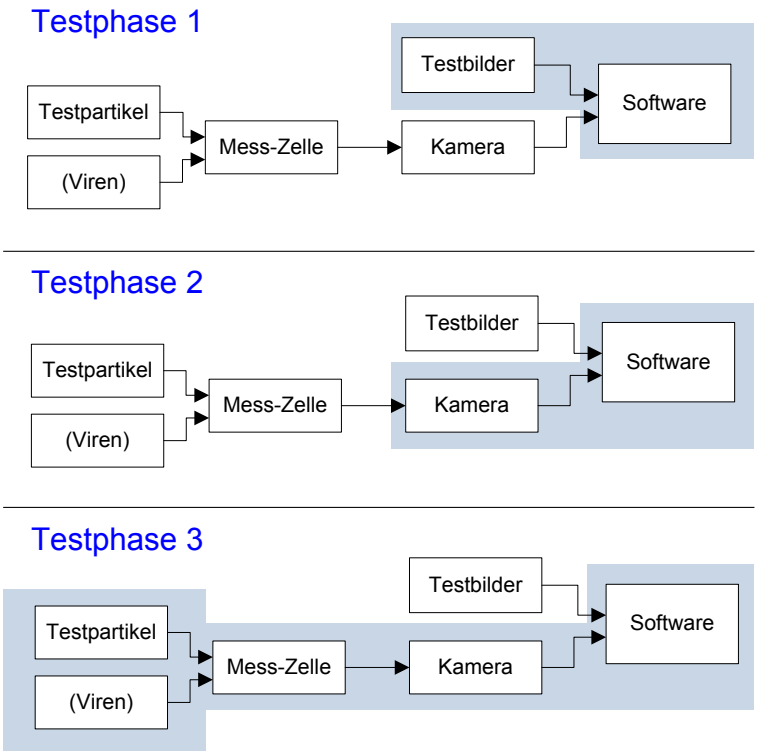


Abbildung C.1.: 3 Testphasen: Aufteilung und jeweils betroffene Komponenten (blau hervorgehoben).

C.10. Entwicklungsumgebung

C.10.1. Software

- Entwicklungssystem: Windows (ab XP) / Linux (wahlweise)
- Diagramme: Microsoft Visio / Inkscape (wahlweise)
- Versionskontrolle: SVN
- Programmiersprachen: C++ / C / OpenCL C / VHDL
- Compiler: gcc (für Portierbarkeit auf Windows-OS)
- Verwendete Bibliotheken: Qt ab Version 4.6.2, OpenCL 1.0
- IDE: Eclipse / Qt-Creator, Xilinx ISE 10.1

C.10.2. Hardware

- Handelsüblicher PC mit 1 – 4 GB Arbeitsspeicher, 10 GB freiem HDD-Speicher
- OpenCL-fähige Grafikkarte, bzw. Zugang zum Cell-Board bei Entwicklung auf diesen Komponenten

C.10.3. Orgware

- Internetanschluss
- Projekt-TRAC mit Wiki

C.11. Lizenzen und Zertifizierungen

Da das System nicht als Medizinprodukt benutzt werden soll, sondern als Hilfestellung in der Forschung dient, werden keine besonderen Zertifizierungen angestrebt.

C.12. Glossar

- **Analyse**: Eine Bildanalyse wird wird eine ausgewählte Bildreihe angewendet. Sie ist dabei keine medizinische Analyse im Sinne einer Versuchsdurchführung.
- **Bildreihenerstellung**: Die Bildreihenerstellung ist die Versuchsdurchführung mit Bildaufnahme.
- **Biosensor**: Sensor, der in der biotechnologischen Messtechnik angewendet wird. Biosensoren bestehen aus einer biologischen Komponente (biologischer Rezeptor) und einem Signalwandler (Transducer). Der Transducer setzt die biologische Erkennungsreaktion des Rezeptors in ein elektronisch verwertbares Signal um. Beispiele für Rezeptoren sind Enzyme oder Antikörper. Biosensoren werden häufig in der Medizin, Umweltanalytik, Lebensmittelindustrie sowie in der pharmazeutischen und biotechnologischen Industrie verwendet.
- **CCD**: Charge-coupled Device; ein lichtempfindlicher Sensor zur Bildaufnahme. Ein CCD-Sensor enthält lichtempfindliche Fotodioden, die in einer Matrix angeordnet sind und Pixel genannt werden.
- **Cell-Board**: Das Cell-Board ist ein Entwicklungsboard für die Cell-Prozessorarchitektur. Das Cell-Board zeichnet sich durch eine sehr hohe Rechenleistung aus, vor allem im Multimediabereich, beispielsweise bei Videokodierung und Videospielen, sowie wissenschaftlichen Anwendungen. Die Kosten und der Energiebedarf sind verhältnismäßig gering.
- **Intensität**: Grau- oder Farbwert, welcher in jedem Pixel enthalten ist. Jedem Wert wird eine Zahl zugeordnet, die einem bestimmten Grau- oder Farbton entspricht.
- **JPG**: Ein Grafikformat, welches zur Speicherung von Bildern dient, die nach der JPEG-Kompression komprimiert wurden. Die Kompressionsmethode kann dabei verlustfrei oder verlustbehaftet sein.
- **Farbtiefe**: Farb- und Helligkeitswerte von digitalen Bildern.
- **FPS**: Frames per second bezeichnet die Anzahl der Einzelbilder, die innerhalb einer Sekunde aufgenommen werden.
- **Frame**: Einzelbild aus einer Sequenz von Bildern.
- **gcc**: `gcc` ist ein Compiler für die Programmiersprache C/C++.
- **glibc**: glibc ist eine C-Bibliothek auf einem Linux-Betriebssystem.
- **GUI**: Graphical User Interface. Die GUI ist eine Software-Komponente, die dem Benutzer eines Computers die Interaktion mit der Maschine über grafische Symbole erlaubt. Die Darstellungen und Elemente können meist unter Verwendung eines Zeigegerätes wie

einer Maus gesteuert werden.

- **IDE**: Eine IDE ist eine Entwicklungsumgebung.
- **ISAS**: Leibniz-Institut für Analytische Wissenschaften
- **LZW**: Das Lempel-Ziv-Welch Verfahren ist eine verlustfreie Kompression. Die Komprimierung verläuft mit Hilfe eines Wörterbuches, in der die häufigsten Zeichenketten gespeichert und nur die Abkürzungen der Wörter benutzt werden. Das Wörterbuch selbst wird innerhalb einer Datei gespeichert. Der entsprechende Decoder kann das Wörterbuch aus der Datei wiederherstellen.
- **OpenCL**: Offener Standard für paralleles Rechnen auf eingebetteten Systemen (wie Smartphones), Grafikkarten und speziellen Beschleuniger-Karten (wie das Cell-Board).
- **Qt**: Qt ist eine C++-Klassenbibliothek für die plattformübergreifende Programmierung grafischer Benutzeroberflächen.
- **ROI**: Region of Interest. Ein Bereich der für weitere Analysen von besonderem Interesse ist. Dieser Bereich kann entweder vom Benutzer selbst oder vollautomatisch gewählt werden.
- **PAMONO**: Die PAMONO-Messtechnik ist ein Verfahren zum Nachweis von einzelnen Partikeln. Partikel haften an der Sensoroberfläche an und durch Messung des Intensitätsanstiegs können diese nachgewiesen werden.
- **Pixel**: Pixel (picture element) sind kleinste Bildpunkte eines Digitalbildes. Sie enthalten Informationen über Farbe (Farbtiefe) oder Intensität. Ein Pixel hat eine bestimmte Größe.
- **PNG**: Portable Network Graphic ist ein Dateiformat für Rastergrafiken zur verlustfreien Speicherung von Bilddaten. Es ist möglich, Bilder mit einer Farbtiefe von bis zu 16 Bit pro Farbkomponente zu speichern. Die Struktur ist weniger komplex als das TIFF-Dateiformat.
- **Polygon**: Zusammenhängende Fläche oder Figur, die aus mindestens drei, durch Strecken miteinander verbundenen, verschiedenen Punkten, besteht.
- **Preprocessing**: Vorverarbeitung der Bilder, wie zum Beispiel Entfernung bzw. Verringerung unerwünschter oder irrelevanter Signalbestandteile.
- **Projekt**: Im Rahmen der PG wird ein Projekt als mindestens eine Bildreihenerstellung und Bildreihe mit einer, keiner oder mehreren Analysen verstanden.
- **RAW**: Rohdatenformat der Bilder.
- **Superluminzens Diode**: Die Superlumineszenzdiode (SLD) erzeugt durch spontane Emission Licht, das in einem Laser optisch verstärkt wird.

- **TIFF**: Tagged Image File Format. Ein Dateiformat zur verlustfreien Speicherung von Bilddaten.
- **TRAC-Wiki**: Trac ist ein freies, webbasiertes Projektmanagement-Werkzeug zur Softwareentwicklung.
- **UI**: Das User-Interface bezeichnet eine Interaktionsstelle zwischen Mensch und System. Sie wird auch Benutzerschnittstelle gennant.
- **XML**: Auszeichnungssprache zur textbasierten Darstellung von Daten.