

Bachelorarbeit

Florian Wysk

Implementierung von Bildverarbeitungsfunktionen und Analyse der
Nutzerfreundlichkeit für ein Python-basiertes Framework zur
vereinfachten Beschreibung von FPGAs in bildgebenden Systemen

Florian Wysk

Implementierung von Bildverarbeitungsfunktionen und
Analyse der Nutzerfreundlichkeit für ein Python-basiertes
Framework zur vereinfachten Beschreibung von FPGAs in
bildgebenden Systemen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Tim Tiedemann
Zweitgutachter: Prof. Dr. Marc Hensel

Eingereicht am: 08.06.2023

Florian Wysk

Thema der Arbeit

Implementierung von Bildverarbeitungsfunktionen und Analyse der Nutzerfreundlichkeit für ein Python-basiertes Framework zur vereinfachten Beschreibung von FPGAs in bildgebenden Systemen

Stichworte

Bildverarbeitung, FPGA, Python, Nutzerfreundlichkeit, Bildeinzugskarten, Kameras

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Implementierung der gängigsten Bildverarbeitungsfunktionen in Python in Anlehnung an OpenCV und mit der Analyse der Nutzerfreundlichkeit der implementierten Funktionen. Es werden sowohl Software- als auch Hardwaretests für die Verifikation benutzt. Für die Analyse der Nutzerfreundlichkeit wird ein Testscenario entworfen, welches nach dem System Usability Score ausgewertet wird.

Florian Wysk

Title of Thesis

Implementation of image processing functions and analysis of their usability in a Python-based framework for the simplified description of FPGAs in imaging systems.

Keywords

Image processing, FPGA, Python, Usability, Framegrabber, Cameras

Abstract

This thesis deals with the implementation of the most common image processing functions in Python following OpenCV and analysis of the usability of the implemented functions. Both software and hardware tests are used for verification. For the usability analysis a test scenario is designed, which is evaluated according to the System Usability Score.

Inhaltsverzeichnis

Abbildungsverzeichnis	vii
Tabellenverzeichnis	ix
Abkürzungen	x
Listings	xi
1 Einleitung	1
1.1 Motivation	1
1.2 Ziel der Arbeit	2
1.3 Aufbau der Arbeit	2
2 Grundlagen	3
2.1 Digitale Bildverarbeitung	3
2.2 Visual Applets	8
2.3 PyWizion Framework	9
2.4 Nutzerfreundlichkeit	10
3 Anforderungen	12
3.1 Funktionale Anforderungen an die Bildverarbeitungsfunktionen	12
3.1.1 Visual Applets Operatoren	12
3.1.2 Parameter der Filterfunktionen	14
3.2 Nichtfunktionale Anforderungen an die Bildverarbeitungsfunktionen	14
3.2.1 Auswahl der Bildverarbeitungsfunktionen	15
3.2.2 FPGA Ressourcen	15
3.2.3 Nutzerfreundlichkeit	16
3.2.4 Einstieg in das PyWizion Framework	17
3.3 Anforderungskatalog	18

4	Bildverarbeitungsfunktionen	19
4.1	Auswahl der Funktionen	19
4.2	Konzept	23
4.2.1	Bildverarbeitungsfunktionen in PyWizion	23
4.2.2	Filterfunktionen in Visual Applets	26
4.2.3	Sicherstellung der weiteren Funktionalität	31
4.2.4	Funktionstest	32
4.3	Implementierung	34
4.3.1	Aufbau der Filterklasse	34
4.3.2	Filter in PyWizion	37
4.3.3	Weitere Funktionalitäten	40
4.4	Verifikation	41
4.4.1	Visual Applets	42
4.4.2	PyWizion vs. OpenCV	43
4.4.3	Hardware	50
5	Nutzerfreundlichkeit	54
5.1	Auswahl der Nutzer:innen	55
5.2	Testdesign	56
5.3	Probedurchlauf	57
5.4	Hauptstudie	58
5.5	Auswertung der Tests	59
5.6	Auswertung der Anforderungen der Nutzerfreundlichkeit	66
6	Schlussbetrachtung	67
6.1	Fazit	67
6.2	Diskussion	68
6.3	Ausblick	69
	Literaturverzeichnis	70
A	Anhang	74
A.1	Bildverarbeitungsfunktionen	74
A.1.1	Fragebogen firmeninterne Umfrage	74
A.1.2	Aktivitätsdiagramme	75
A.1.3	Ergebnisse der Differenzbilder	76

A.2	Nutzerfreundlichkeit	78
A.2.1	Testaufbau in Visual Applets	78
A.2.2	Testaufbau in PyWizion	79
A.2.3	Anlagenverzeichnis	82
	Selbstständigkeitserklärung	85

Abbildungsverzeichnis

2.1	Anwendung einer Filtermaske [35]	4
2.2	Glättung eines Bildes mit einem Gaussfilter. Links: Original, Mitte: Ergebnis 3x3 Filterkernel, Rechts: Ergebnis 7x7 Filterkernel	5
2.3	Kantendetektion Sobel 3x3 mit/ohne Glättung. Links: Original, Mitte: Ergebnis ohne Glättung, Rechts: Ergebnis mit Glättung	6
2.4	Strukturelement für binäre morphologische Anwendungen [35]	7
2.5	Beispiel Dilatation[35]	7
2.6	Beispiel Erosion[35]	7
2.7	Benutzeroberfläche Visual Applets [13]	8
2.8	Arbeitsablauf PyWizion [23]	9
2.9	Kontextbezug von Nutzerfreundlichkeit[14]	10
3.1	Synchronisierungsprobleme Operatoren	13
3.2	Synchronisierungsoperatoren	13
3.3	Beispielhaftes Design eines Gauss Filter 3x3 in Visual Applets	15
3.4	Ermittelte Ressourcen des Gauss Filter Designs auf der Bildeinzugskarte microEnable 5 Marathon VCX-QP	16
4.1	Ergebnisse aus der Literatur	20
4.2	Ergebnisse der firmeninternen Umfrage	21
4.3	Aktivitätsdiagramm des generellen Filterfunktionsablaufs	25
4.4	Ressourcenverbrauch: Boxfilter 3x3 mit FIRkernelNxM	26
4.5	Ressourcenverbrauch: Boxfilter 3x3 mit PixelNeighbours1xM und Line-NeighboursNx1	27
4.6	Ressourcenverbrauch: Boxfilter 3x3 mit PixelNeighbours1xM und Line-NeighboursNx1 verschoben	27
4.7	Ressourcenverbrauch: Division mit 16	29
4.8	Ressourcenverbrauch: Division mit 42	29
4.9	Klassendiagramm Filter mit den wichtigsten Methoden	34

4.10	Ressourcenverbrauch: FIRkernelNxM vor SplitComponents	38
4.11	Ressourcenverbrauch: FIRkernelNxM nach SplitComponents	38
4.12	Klassendiagramm ImageIO	40
4.13	Differenz der Ergebnisse aus der adaptiven Faltung und des FIRoperatorNxM	42
4.14	Exemplarisches Simulationsbild des RAISE Datensatzes [19]	45
4.15	Differenzbild Bilateralfilter Durchmesser 5, Sigma Color 50, Sigma Space 50, maximale Differenz 3, Abweichende Pixel 1.65 %	46
4.16	Differenzbild Bilateralfilter Durchmesser 7, Sigma Color 50, Sigma Space 50, maximale Differenz 12, Abweichende Pixel 46.18 %	47
5.1	Bewertungsskala SUS Auswertung[1]	55
5.2	Kastengrafik: Vorkenntnisse der Proband:innen	59
5.3	Kastengrafik System Usability Score mit der jeweiligen Software	60
5.4	Kastengrafik: Benötigte Zeit für die erfolgreich abgeschlossene Aufgabe mit der jeweiligen Software	63
5.5	Zeit der erfolgreichen Tests in Abhängigkeit der Selbsteinschätzung in OpenCV	63
5.6	Kastengrafik: Benötigte Zeit in Abhängigkeit der Reihenfolge der Software	64
A.1	Fragebogen Umfrage zu den gängigsten Bildverarbeitungsfunktionen	74
A.2	Aktivitätsdiagramm Debayering	75
A.3	Aktivitätsdiagramm Imread	75
A.4	Ausschnitt der Ergebnistabelle des Gaussfilters für natürliche Graustufenbilder	77
A.5	Testaufbau in Visual Applets komplettes Design	78
A.6	Testaufbau in Visual Applets Gauss Filter	78
A.7	Testaufbau in Visual Applets Closing	78
A.8	SUS Fragebogen PyWizion Teil 1	80
A.9	SUS Fragebogen PyWizion Teil 2	81

Tabellenverzeichnis

3.1	Funktionale Anforderungen an die Bildverarbeitungsfunktionen	18
3.2	Nichtfunktionale Anforderungen an die Bildverarbeitungsfunktionen	18
4.1	Ausgewählte Bildverarbeitungsfunktionen	22
4.2	Übersicht Ressourcenverbrauch Box Filter 3x3	27
4.3	Implementierte Funktionen und zulässige Eingangsbilder	39
4.4	Zulässige Fehlertoleranzen der Filter	44
4.5	Übereinstimmung der Filter mit deren Äquivalent aus OpenCV anhand der maximalen Abweichung	48
4.6	Übereinstimmung der Filter mit deren Äquivalent aus OpenCV anhand des Medians der Abweichung	48
4.7	Parametereinstellung Testdesigns	50
4.8	Median der gemessenen FPS der Designs in microDisplay X	52
5.1	Berufe der Proband:innen	58
5.2	SUS Ergebnisse nach Software	60
5.3	Zusammengefasste Kernaussagen aus der Methode “Lautes Denken“	62
5.4	Benötigte Zeit für die erfolgreich abgeschlossene Aufgabe	62
5.5	Auftretende Fehler während des Tests	65
A.1	Bilateral Filter Abweichungen natürliche Graustufenbilder	76

Abkürzungen

ALU Arithmetic Logic Unit

BRAM Block Random Access Memory

CXP CoaXPress

DMA Direct Memory Access

DRC2 Design Rule Check 2

FPGA Field Programmable Gate Array

FPS Frames Per Second

HAP Hardware Applet Datei

IP Image Processing

LUT Lookup Table

LUTRAM Lookup Table Random Access Memory

RAM Random-Access Memory

RGB Red Green Blue

SUS System Usability Score

VA Visual Applets

VHDL Very High Speed Integrated Circuit Hardware Description Language

Listings

4.1	Exemplarischer Aufruf einer Filtermethode anhand des Bilateralfilters . . .	40
A.1	Testaufbau in PyWizion	79

1 Einleitung

1.1 Motivation

Die Verwendung von digitalen Kameras ist allgegenwärtig in verschiedensten Einsatzgebieten, wie beispielsweise der Medizintechnik, Automatisierung in der Industrie oder autonomen Fahren. [21] Für die Weiterverarbeitung der aufgenommenen Daten werden effiziente und zuverlässige Bildverarbeitungsalgorithmen benötigt. Besonders bei Anwendungen, die Echtzeitanforderungen verlangen, ist es essentiell, niedrige Latenzen bei der Bildverarbeitung zu gewährleisten. Ein möglicher Baustein, welcher die Umsetzung dieser strikten Anforderungen ermöglicht, ist ein Field Programmable Gate Array (FPGA), da dieser eine hohe Flexibilität und Leistungsfähigkeit bietet und es ermöglicht digitale Schaltungen für spezifische Probleme zu entwickeln. [28] Durch diese einfach veränderbaren Schaltungen ist ein FPGA eine sehr attraktive Lösung für Anwendungen, die hohe Leistung und Anpassungsfähigkeit erfordern, wie unter anderem die Bildverarbeitung. Bei der Basler AG werden FPGAs mit einer firmeninternen Software namens Visual Applets beschrieben. Die effiziente Anwendung dieses Programms setzt von den Entwickler:innen sowohl Kenntnisse in digitaler Schaltungstechnik als auch Einarbeitungszeit in Visual Applets voraus. Die möglichen Entwickler:innen, die für diesen Anwendungsbereich potentiell in Frage kommen, sind aus den zuvor genannten Gründen selten vorhanden. Deswegen wird ein neues Python-basiertes Framework entwickelt, mit dem Entwickler:innen ohne Vorkenntnisse in digitaler Schaltungstechnik und Visual Applets FPGAs für Bildverarbeitungsanwendungen beschreiben können.

Mit dem heutigen Stand des Frameworks ist es noch nicht möglich das Spektrum an Anwender:innen zu vergrößern, da für die Verwendung grundlegende Kenntnisse in der Software Visual Applets benötigt werden. Um dieses Problem zu lösen sollen Funktionen, welche die Entwickler:innen leicht bedienen können, in dem Framework angeboten werden.

1.2 Ziel der Arbeit

Um die einfache Bedienbarkeit in dem Python-basierten Framework zu schaffen, werden die gängigsten Bildverarbeitungsfunktionen in diesem implementiert und auf Nutzerfreundlichkeit hin analysiert. Dabei soll untersucht werden, ob durch die neue Funktionalität des Frameworks die Bedienbarkeit im Vergleich zu Visual Applets steigt. Durch das funktionale Framework, welches von Nutzer:innen ohne Kenntnisse von digitaler Schaltungstechnik und grundlegender Visual Applets Programmierung benutzt werden soll, soll es möglich sein das Spektrum an Entwickler:innen, die Bildverarbeitungsanwendungen auf FPGAs lösen können, zu vergrößern. Um den Einstieg in das Framework zu erleichtern und Entwickler:innen, die Kenntnisse in Bildverarbeitung mit Python haben, den Zugang zu ermöglichen, wird bei der Implementierung auf gute Nutzerfreundlichkeit geachtet. Außerdem ist es das Ziel, die Bildverarbeitungsfunktionen ressourcenarm auf dem FPGA zu implementieren

1.3 Aufbau der Arbeit

In dem Kapitel Grundlagen werden die allgemeinen Funktionsweisen von Filtern in der digitalen Bildverarbeitung, Visual Applets und dem Python-basierten Framework erläutert. Außerdem wird definiert, wie Nutzerfreundlichkeit in dieser Arbeit zu verstehen ist. Im Kapitel Anforderungen sind die Kriterien, die die Bildverarbeitungsfunktionen zu erfüllen haben, beschrieben. In dem Kapitel Bildverarbeitungsfunktionen wird aus den Anforderungen ein Konzept für die Implementierung der Funktionen in dem Framework entwickelt. Ebenfalls werden in dem Kapitel Bildverarbeitungsfunktionen die Funktionen durch Soft- und Hardwaretests verifiziert. Anschließend wird die Nutzerfreundlichkeit der implementierten Funktionen in dem gleichnamigen Kapitel verifiziert und analysiert. Außerdem werden der Entwurf und die Durchführung sowie die Ergebnisse der Tests erläutert. Abschließend wird in dem Kapitel Schlussbetrachtung überprüft, ob die anfänglich gestellten Anforderungen erfüllt sind. Außerdem wird die Arbeit im Gesamten im Hinblick das anfänglichen Ziels bewertet.

2 Grundlagen

In dem Kapitel der digitalen Bildverarbeitung werden die grundlegenden theoretischen Eigenschaften der Bildverarbeitungsfunktionen beschrieben. Dafür wird insbesondere die allgemeine Funktion von Filtern und die Funktionsweise der Filterarten, die später bei der Implementierung relevant sind, erläutert. Außerdem werden sowohl die kommerzielle Software Visual Applets als auch Python-basierte Framework grundlegend erläutert. Des Weiteren wird auf Kriterien der Nutzerfreundlichkeit, die in dieser Arbeit relevant sind, eingegangen.

2.1 Digitale Bildverarbeitung

Hinter dem Begriff der digitalen Bildverarbeitung steht die Erfassung, Verarbeitung, Analyse und Interpretation digitaler Bilder. Dabei steht die Verarbeitung der Bilder in dieser Arbeit im Vordergrund. Insbesondere die Bildvorverarbeitung ist sehr wichtig, da diese nichtlineare Charakteristiken wie z.B. Rauschen von Sensoren kompensieren kann. Des Weiteren ist es möglich, dass die Helligkeit oder der Kontrast angepasst wird. [24]. Für die Analyse und Interpretation ist die Identifikation von Objekten notwendig, dafür sind Vorverarbeitungsoperationen wie die Anwendung von Glättungs- und Kantenfiltern essentiell. [24]

Filter allgemein

Grundsätzlich werden Filter in der digitalen Bildverarbeitung in lineare und nichtlineare Filter unterteilt. Beide Filterarten gehören zu den gängigsten lokalen Bildoperationen, welche dabei die Grauwerte einer definierten Gruppe von Pixeln innerhalb des Ursprungsbildes miteinbezieht. Die Gruppe von Pixel wird als Nachbarschaft, Filtermaske oder Filterkernel bezeichnet. [18]. Der einzige Unterschied zwischen linearen und nichtlinearen Filtern ist die Verknüpfung der Pixelwerte innerhalb der Region und der

resultierende mathematische Ausdruck. [35] Die Anwendung eines linearen Filters kann wie folgt dargestellt werden:

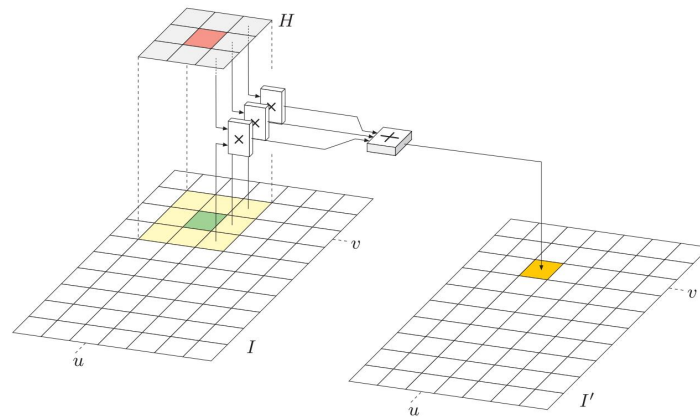


Abbildung 2.1: Anwendung einer Filtermaske [35]

Für jedes Pixel im Ursprungsbild I wird die Filtermaske H so platziert, sodass der Hot Spot $H(0,0)$ (der rot markierte Punkt innerhalb der Maske) auf dem zu bearbeitenden Pixel $I(u,v)$ liegt. Alle Bildelemente werden mit dem dazugehörigen Filterkoeffizienten $H(i,j)$ multipliziert und aufsummiert. Häufig wird der neue Pixelwert mit dem Skalierungsfaktor s , welcher meist der Summe der Filterkoeffizienten entspricht, normiert. Abschließend wird der berechnete Wert dem Bildpunkt im Ergebnisbild $I'(u,v)$ zugeordnet. [18] Mathematisch lässt sich der Prozess, beispielhaft für eine Filtermaske mit der Größe 3x3, wie folgt beschreiben:

$$I'(u,v) = \frac{1}{s} \cdot \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u+i, v+j) \cdot H(i,j) \quad (2.1)$$

Der Prozess der digitalen Filterung entspricht der Operation der Faltung in der Signalverarbeitung. [28] Sowohl die Größe der Filtermaske als auch die Gewichtung der einzelnen Pixel in der Nachbarschaft ist für das Ergebnisbild von entscheidender Bedeutung, je größer die Maske bzw. je stärker die Gewichtung, desto stärker ist der Effekt der Maske auf das Ursprungsbild. [35] Ein nichtlineares Filter unterscheidet sich lediglich

durch die nichtlineare Verknüpfung der Pixelwerte innerhalb der Filterregion. [35] Es besteht bei nichtlinearen Filtern keine lineare Abhängigkeit von Ein- und Ausgang. An den Rändern des Bildes führt die Anwendung der Filtermatrix zu Artefakten, weil ein Teil der Nachbarschaft unbekannt ist. Dieses Problem der fehlerbehafteten Pixel lässt sich durch die Methode des Paddings lösen. Dafür werden zusätzliche Pixel an dem Bildrand hinzugefügt. Die Anzahl der neuen Pixel ist davon abhängig, wie groß die Filtermatrix ist, d.h. wie weit der Abstand vom Mittelpunkt zu den äußeren Pixeln ist. Die neuen Werte können entweder konstant sein oder durch eine Spiegelung an dem Randpixel des ursprünglichen Bildes festgelegt werden. [35]

Glättungsfilter

Typische Arten von Filtern sind Glättungsfilter, die eine Tiefpasscharakteristik aufweisen und unerwünschtes hochfrequentes Rauschen aus dem Bild entfernen. Rauschen äußert sich in zufälliger Veränderung einzelner Pixelwerte und kann z.B. durch die einfache Mittelung der Nachbarschaft verringert werden. Das gilt nur bei Regionen innerhalb des Bildes, die durch konstante Grauwerte charakterisiert werden können. In der realen Welt kommt es selten vor, dass diese Bedingung erfüllt ist, sodass die einfache Mittelung nicht ausreicht. Durch die Anwendung einer gewichteten Mittelung lässt sich dies beheben. Somit können die Störungen durch die Gewichtung innerhalb der Nachbarschaft unterdrückt werden. [18] Diese Operation wird auch als Glättung oder Blurring bezeichnet. In Abbildung 2.2 ist der Effekt eines Glättungsfilters, in diesem Fall ein Gaussfilter, dargestellt.



Abbildung 2.2: Glättung eines Bildes mit einem Gaussfilter. Links: Original, Mitte: Ergebnis 3x3 Filterkernel, Rechts: Ergebnis 7x7 Filterkernel

Kantenfilter

Kanten- oder Differenzfilter haben die Eigenschaften eines Hochpassfilters und heben die Kanten in einem Bild hervor. Mathematisch wird für den einfachsten Fall die Summe aus den Pixelwerten innerhalb der Filtermaske bestimmt. [18] Die Koeffizienten haben im Gegensatz zu den Glättungsfiltern auch negative Vorzeichen. Vor der Anwendung eines Kantenfilters ist es hilfreich einen Glättungsfilter zu verwenden, um das Rauschen zu unterdrücken und somit auch nur die Kanten von größeren Konturen zu detektieren. In Abbildung 2.3 sind die Ergebnisse eines Sobelfilters, der häufig für die Kantendetektion eingesetzt wird [28], mit und ohne Glättung zu sehen.

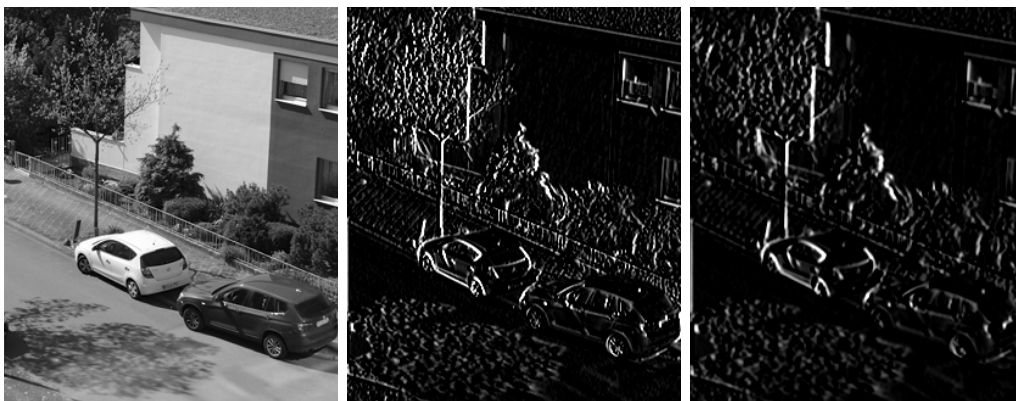


Abbildung 2.3: Kantendetektion Sobel 3x3 mit/ohne Glättung. Links: Original, Mitte: Ergebnis ohne Glättung, Rechts: Ergebnis mit Glättung

Morphologische Transformation

Die mathematische Morphologie (abgeleitet von dem griechischen Wort "morphe", welches Gestalt bedeutet) umfasst die Theorie der Analyse räumlicher Strukturen. Die angewandten Methoden beschäftigen sich vor allem mit dem Erkennen und der Veränderungen der Konturen von Objekten. Dies basiert auf der Verknüpfung von Mengen.[16] Für die elementaren Operationen der morphologischen Transformation ist es von Vorteil, ein Binärbild, in dem die Pixel nur 1 oder 0 annehmen, als Arbeitsgrundlage zu verwenden, weil dieses weniger rechenaufwändig und weniger fehleranfällig ist. In der Morphologie sind die grundlegenden Operationen Erosion und Dilatation. Bei der Anwendung der Erosion werden Objekte durch das Entfernen von Pixeln im Randbereich des Objektes verkleinert. Die Dilatation bewirkt die Vergrößerung der Objekte durch Hinzufügen von Pixeln an den Randbereichen der Objekte. Häufig werden diese beiden

Operationen zusammen verwendet, um Rauschen im Bild zu reduzieren, Konturen zu glätten oder die Form von Objekten zu verändern. Wird die Erosion mit anschließender Dilatation angewendet, beschreibt der Vorgang die morphologische Operation des Öffnens (eng. Opening) der Objekte im Bild. In umgekehrter Reihenfolge wird der Vorgang Schließen (eng. Closing) genannt.

Für die Anwendung der morphologischen Operationen wird ein Strukturelement (Abbildung 2.4), das ähnlich aufgebaut ist wie eine Koeffizientenmatrix aus Abbildung 2.1, verwendet. Diese enthält nur die Werte 0 (Matrixkoeffizient ist leer) und 1 (Matrixkoeffizient entspricht \bullet).

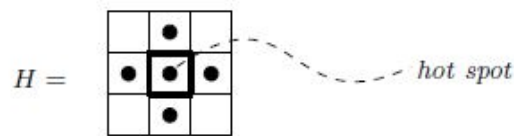


Abbildung 2.4: Strukturelement für binäre morphologische Anwendungen [35]

Auf diesem Strukturelement basieren die morphologischen Operationen der Dilatation und Erosion. In Abbildung 2.5 ist beispielhaft eine Anwendung der Dilatation dargestellt. Auf jedes weiße Pixel des Originalbildes I wird der Hot Spot des Strukturelements H gelegt und in einem neuen Bild gespeichert. Das Prinzip basiert auf dem logischen Oder, welches auf die Bildpunkte des ursprünglichen Bildes und das Strukturelement, angewendet wird.

Die Erosion (Abbildung 2.6) funktioniert nach dem gleichen Prinzip, jedoch wird der Pixelwert im Ergebnisbild nur mit 1 gewertet, wenn das Strukturelement vollständig in das Objekt des ursprünglichen Bildes passt.[35]

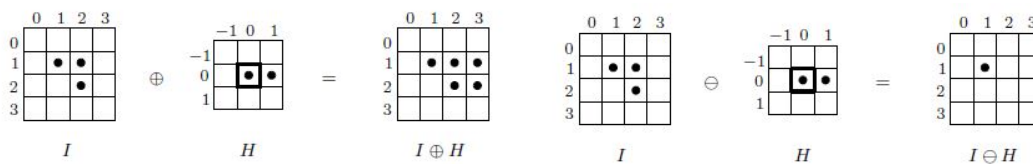


Abbildung 2.5: Beispiel Dilatation[35]

Abbildung 2.6: Beispiel Erosion[35]

2.2 Visual Applets

Visual Applets ist eine kommerzielle Software, die für die Konfiguration von FPGAs auf den Bilderfassungskarten von Basler verwendet wird. Die Entwicklungsumgebung bietet eine grafische Benutzeroberfläche, in der die Nutzer:innen ein Datenflussmodell der Bildverarbeitungsanwendung erstellen können. Es ist auch möglich FPGAs, die sich nicht auf den Einzugskarten von Basler befinden, zu konfigurieren. Dafür wird der embedded Visual Applets Image Processing (IP) Core, welcher Xilinx FPGAs der Serie 6 oder höher unterstützt, genutzt. Damit die Logikblöcke auf der Hardware korrekt verbunden werden, wird eine Netzliste von Visual Applets erstellt, welche die Grundlage für z.B. Vivado ist, um den FPGA zu beschreiben. Für die Nutzung der Software sind weder Kenntnisse in Very High Speed Integrated Circuit Hardware Description Language (VHDL) noch Verilog notwendig, da die Abstraktionsebene sehr viel höher ist als die der Hardwarebeschreibungssprachen. Die Programmierung basiert auf der Modellierung des Signalfusses und der Manipulation von Bits in Form von Programmierbausteine, die Visual Applets bereitstellt. [6] Die verfügbaren Bausteine in der Entwicklungsumgebung werden als Operatoren bezeichnet. Diese umfassen u.A. arithmetische Operationen, klassische Filter, Farbraumtransformationen und Bildkompressionstechniken. Zusätzlich gibt es eine Simulation, die es ermöglicht visuelle Ergebnisse mit Bitgenauigkeit des erstellten Designs zu liefern.

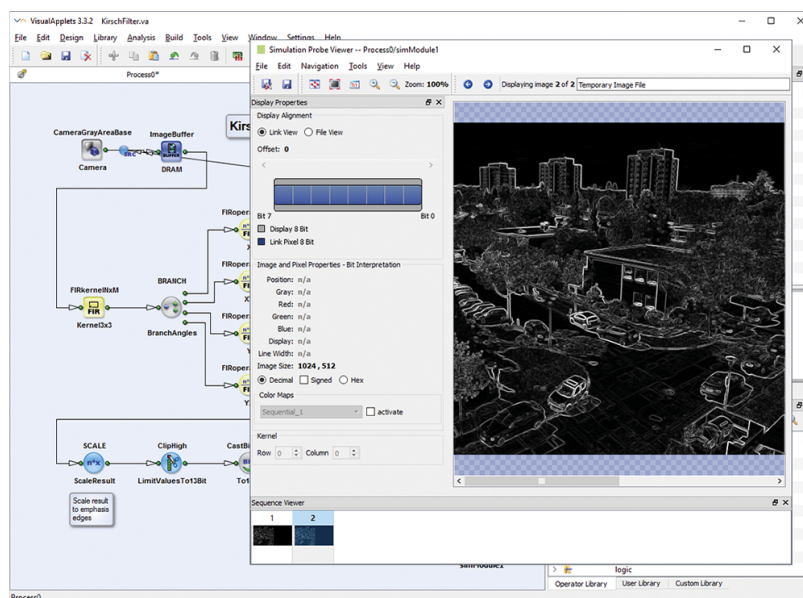


Abbildung 2.7: Benutzeroberfläche Visual Applets [13]

Durch den Einsatz von Visual Applets müssen die Nutzer:innen sich nicht mit Timings der Signale bzw. logischen Bausteine auf dem FPGA auseinandersetzen. In Abbildung 2.7 ist die Benutzeroberfläche mit Operatoren und einem Simulationsfenster, die zwischen beliebigen Operatoren eingefügt werden können, dargestellt.

Im Folgenden wird auf weitere Funktionalitäten von Visual Applets an den jeweils relevanten Stellen eingegangen.

2.3 PyWizion Framework

PyWizion ist ein neues Python-basiertes Framework, das für die vereinfachte Konfiguration von FPGAs entwickelt wird. Dabei basiert die Funktionalität auf der Software Visual Applets. In dem Framework wird das Design, wie in Kapitel 2.2 beschrieben, entwickelt, jedoch ist der Vorgang der Bildverarbeitungsanwendungen nicht mehr grafisch, sondern textbasiert realisiert.

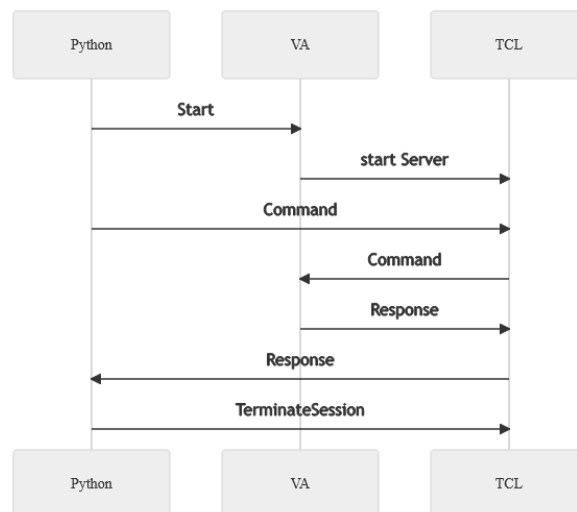


Abbildung 2.8: Arbeitsablauf PyWizion [23]

Der Ablauf für das Erstellen eines Designs mit PyWizion ist in Abbildung 2.8 dargestellt. Über das Framework wird Visual Applets (Visual Applets (VA) in der Abbildung 2.8) in einem separaten Prozess gestartet und kommuniziert dann über einen Websocket mit dem PyWizion-Prozess über den Einstellungen wie z.B. Konfigurationen der Ports vorgenommen werden. Die Anweisungen beinhalten u.A. welche Operatoren zu verwenden

sind, wie diese untereinander zu verknüpfen sind oder wie nach der Simulation verfahren werden soll. Anschließend werden die Befehle direkt aus dem Framework zum Server geschickt. Dieser leitet die Befehle zum Erstellen des Designs an Visual Applets weiter und wartet auf die Antwort, dass die Befehle korrekt interpretiert worden sind. Abschließend sendet der Server die Antwort von Visual Applets an das Framework und erhält danach den Befehl, die Verbindung zu beenden.

2.4 Nutzerfreundlichkeit

Nutzerfreundlichkeit, oder auch Usability im Englischen, wird in der DIN EN ISO 9241-11 als ein Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem Kontext angewandt werden, definiert, um die Ziele effektiv, effizient und zufriedenstellend zu lösen. [20] Die Definition verdeutlicht, dass Nutzerfreundlichkeit nicht als eindimensionale objektive Eigenschaft bewertet werden darf. Laut Bevan wird diese durch die Faktoren Systeme, Ziele/Aufgabe und Nutzer:innen beeinflusst (Abbildung 2.9), wenn diese in einem technischen, physischen, sozialen oder organisatorischen Kontext steht. [14]

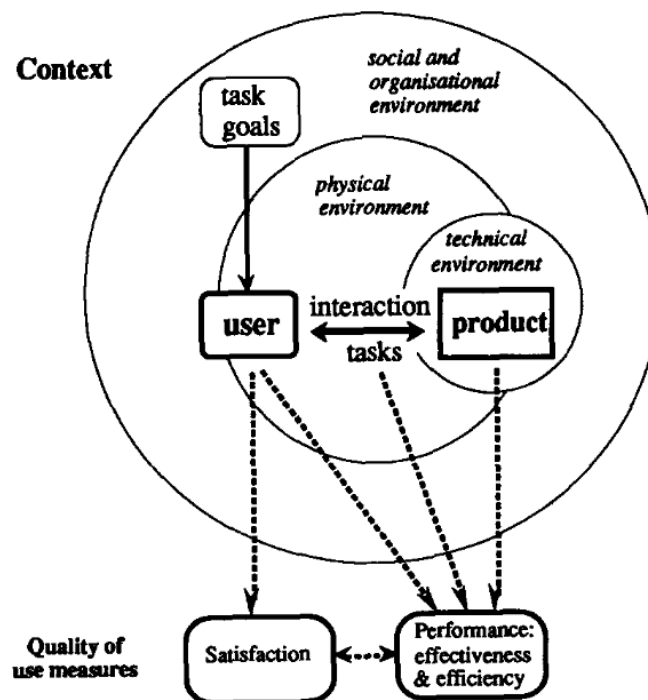


Abbildung 2.9: Kontextbezug von Nutzerfreundlichkeit[14]

In dem System, welches sich durch eine Interaktion mit Nutzer:innen auszeichnet, wird die Nutzerfreundlichkeit häufig durch eine gute Bedienbarkeit der grafischen Benutzeroberfläche gewährleistet. In dieser Arbeit ist die Benutzerschnittstelle die Handhabung der Bildverarbeitungsfunktionen. Im Folgenden wird von Software anstatt System gesprochen, wenn es sich um Nutzerfreundlichkeit handelt, da die Analyse auf Funktionen des Frameworks bezogen ist.

Die Qualität der Nutzerfreundlichkeit ist sehr stark davon abhängig, über welche Kenntnisse die Nutzer:innen verfügt. Wird für die Benutzung der Software spezielles Wissen vorausgesetzt, bewerten die Tester, welche dieses nicht haben, generell schlechter.

Gute Nutzerfreundlichkeit der Software wird durch das Ziel bzw. die Aufgabe, welches erreicht werden soll, definiert. Dabei ist es relevant, ob die Software für den ursprünglich entwickelten Zweck eingesetzt oder anderweitig benutzt wird. [14] Die Qualität der Nutzerfreundlichkeit hängt demnach nicht nur von der Software, sondern auch von dem Kontext, in der diese eingesetzt wird, ab.

3 Anforderungen

Um die Aufgabenstellung zu konkretisieren, werden in diesem Kapitel die Anforderungen, welche an die zu implementierenden Bildverarbeitungsfunktionen gestellt werden, beschrieben. Dafür werden diese in funktionale und nichtfunktionale Anforderungen unterteilt.

3.1 Funktionale Anforderungen an die Bildverarbeitungsfunktionen

Die funktionalen Anforderungen charakterisieren die erforderlichen Eigenschaften, welche die Software erfüllen müssen. [34] In diesem Fall beziehen sich diese Anforderungen auf die Kriterien, welche die Bildverarbeitungsfunktionen erfüllen sollen. Dazu werden ebenfalls die Anforderungen, die an ein Design in Visual Applets gestellt werden, gezählt. In den folgenden Unterkapiteln wird auf die funktionale Anforderung näher eingegangen.

3.1.1 Visual Applets Operatoren

In Visual Applets werden Bildverarbeitungs- und Signalverarbeitungsfunktionen durch abstrakte Operatoren dargestellt. Die Operatoren sind in O-, M- und P-Typen unterteilt und werden in dieser Arbeit fett geschrieben.[10] Relevant für die Implementierung ist lediglich die Eigenschaft, dass O-Typen den Datenfluss nicht blockieren, wohingegen M- und P-Typen diesen verändern. Eine mögliche Ursache ist dabei die Berechnung von neuen Pixelwerten, die mit der Einbeziehung von benachbarten Bildpunkten realisierbar wird. Dafür werden Daten zwischengespeichert und die Latenz erhöht.

In Abbildung 3.1 ist ein eigenes simples Design, in dem ein beliebiges Filter auf das Eingangsbild angewendet und zusätzlich eine Konstante auf das Ergebnis der Filterung

addiert wird, dargestellt. Die Faltung wird mit den beiden Operatoren **FIRkernelNxM** und **FIRoperatorNxM** umgesetzt. Die Addition der Konstante erfolgt über den **CONST** Operator und die Verknüpfung mit dem **ADD** Operator. Die O-Typen sind an der runden Darstellungsform sehr gut von den eckigen M-Typen zu unterscheiden. In diesem Beispiel werden keine P-Typen miteinbezogen, da diese sich in den grundlegenden Eigenschaften nicht sonderlich von den M-Typen abheben.

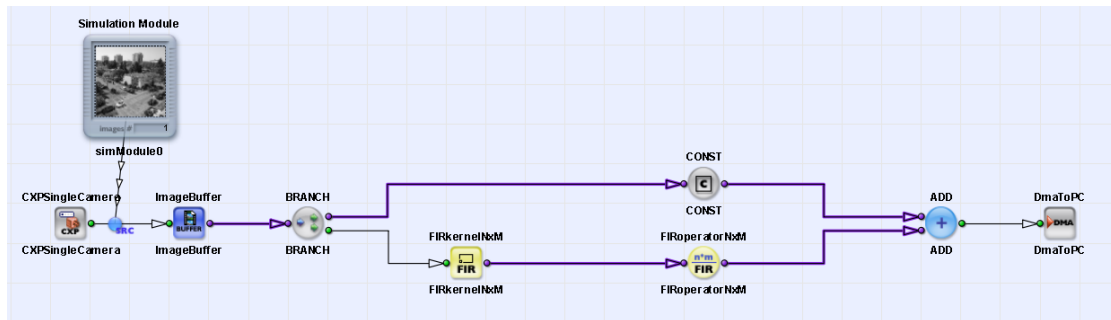


Abbildung 3.1: Synchronisierungsprobleme Operatoren

In der Simulation wird den Nutzer:innen durch die lilafarbene Hervorhebung der Verbindungen zwischen den Operatoren darauf hingewiesen, dass es auf der Hardware zu Synchronisierungsproblemen kommen wird. Der Grund für diese Probleme ist die Verwendung des **FIRkernelNxM** in dem unteren Zweig. Durch den Einsatz des Synchronisierungsoperators **SYNC**, der in das Design eingebaut ist (Abbildung 3.2), werden beide Eingangsströme aufeinander abgestimmt, sodass die richtigen Pixelwerte korrekt verrechnet werden. [10]

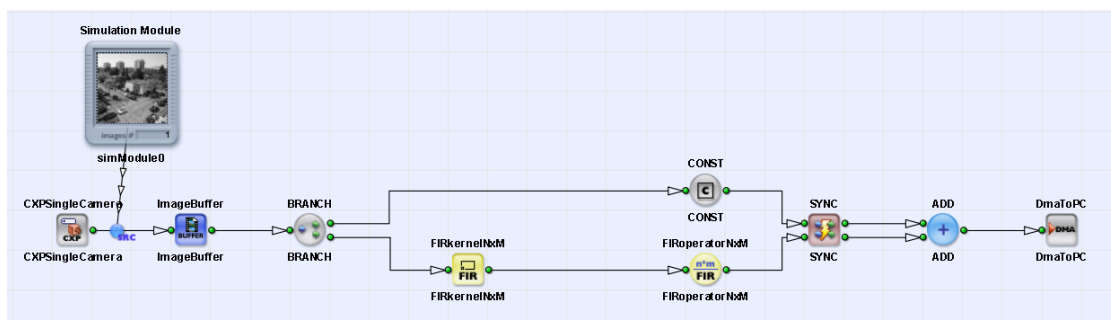


Abbildung 3.2: Synchronisierungsoperatoren

Da bei falscher Synchronisierung die potentielle Gefahr von Deadlocks auftreten kann, ist bei der Implementierung darauf zu achten, dass keine Synchronisierung von mehreren parallelen Datenströmen erforderlich ist. Wenn es nicht vermeidbar ist, muss der **SYNC** Operator verwendet.

3.1.2 Parameter der Filterfunktionen

Die implementierten Filterfunktionen müssen sowohl für Graustufen- als auch Farbbilder, die in Red Green Blue (RGB)-codiert sind, anwendbar sein. Im Folgenden werden RGB-codierte Bilder als Farbbilder bezeichnet. So ist es möglich Kameras, die Mono- oder Farbsensoren haben, als Bildquelle zu benutzen. Außerdem ist die Anwendung eines Bayer-Algorithmus, der z.B. durch mit Interpolation einzelner Pixelwerte ein Farbbild aus einer Graustufenaufnahme erzeugt, realisierbar.[28]

In die Berechnung der Filterkoeffizienten fließen unterschiedlichste Parameter, wie z.B. die Richtung der Ableitung (Sobel), die Breite der Normalverteilung (Gauss) oder ein Skalierungsfaktor (Laplace). Damit sichergestellt ist, dass die Filter mit unterschiedlichen Parametern und Kernelgrößen in PyWizion korrekt implementiert sind, müssen die Ergebnisse verifiziert werden. In dem Framework ist es möglich, das gefilterte Bild als numpy-array in der Entwicklungsumgebung ausgeben zu lassen. Für einen Vergleich sollen die Ergebnisse aus PyWizion mit denen aus Standardbibliotheken übereinstimmen. Dafür bietet es sich an, in Python eine Bibliothek, die ein sehr großes Spektrum von Filterfunktionen bietet, zu nutzen und dasselbe Ursprungsbild mit dem Filter zu falten. Es ist sehr unwahrscheinlich, dass beide Ergebnisbilder für alle Filter identisch sind, da die Filterfunktionen nicht deckungsgleiche Algorithmen benutzen und die Filterkoeffizienten in Visual Applets nur ganzzahlige Werte annehmen können. Daraus ergeben sich Fehlertoleranzen, die gering gehalten werden müssen.

3.2 Nichtfunktionale Anforderungen an die Bildverarbeitungsfunktionen

Zu den nichtfunktionalen Anforderungen werden alle Eigenschaften, die nicht zu der Funktionalität der Bildverarbeitungsfunktionen zusammenhängen, gezählt. Dazu gehören Erwartungen und Wünsche der Entwickler:innen. Diese werden in mittelbare und

unmittelbare nichtfunktionale Anforderungen eingeteilt. Ersteres beschreibt die Vorgaben und Gegebenheiten, die eingehalten werden müssen. Zweiteres wird verwendet, um Qualitätsattribute, wie Leistungsverhalten und Erweiterbarkeit, zu beschreiben. [34]

3.2.1 Auswahl der Bildverarbeitungsfunktionen

Damit PyWizion unterschiedliche Bildverarbeitungsalgorithmen abdeckt und der initiale Implementierungsaufwand absehbar ist, dürfen nur die gängigsten Funktionen bei der Implementierung einbezogen werden. Dabei ist unabdingbar, dass durch die Funktionalität ein breites Spektrum an Anwendungen in der Bildverarbeitung abgedeckt wird, um sehr flexibel auf Probleme von Entwickler:innen eingehen zu können.

3.2.2 FPGA Ressourcen

Die verschiedenen Operatoren in Visual Applets benötigen unterschiedlich viele Ressourcen bei der Beschreibung der Hardware. Ressourcen sind Komponenten wie z.B. FlipFlops, Lookup Tables (LUTs) oder Random-Access Memory (RAM), diese sind in endlicher Anzahl vorhanden und es muss bei dem Erstellen des Designs der Bildverarbeitungs-funktionen darauf geachtet werden, möglichst sparsam vorzugehen. Außerdem bietet die Basler AG eine Vielzahl von FPGAs, die unterschiedliche Spezifikationen aufweisen, auf den Einzugskarten an und es muss sichergestellt werden, dass die Implementierung auf diesen ausführbar ist.

In Abbildung 3.3 ist das typische Visual Applets Design eines Gaussfilters mit einer Kernelgröße von 3x3 dargestellt. Im Folgenden wird kurz erklärt wie die Prüfung der benötigten Ressourcen funktioniert.

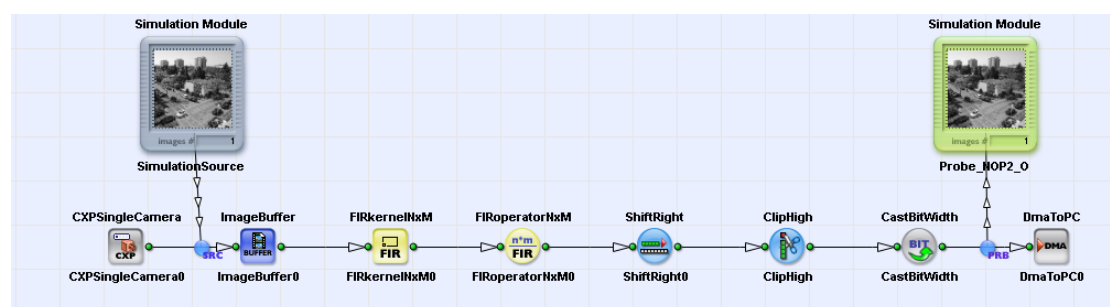


Abbildung 3.3: Beispielhaftes Design eines Gauss Filter 3x3 in Visual Applets

Über die eingebaute Funktion Design Rule Check 2 (DRC2) in Visual Applets wird ermittelt, wie viele Ressourcen im aktuellen Design benötigt werden. Für die Implementierung aus Abbildung 3.3 sind die über den DRC2 ermittelten Komponenten in Abbildung 3.4 dargestellt.

FPGA resource estimation:

Resource	Count	Fill Level
LookupTables	36034	~ 35%
Flip Flops	38031	~ 18%
Block RAM	70	~ 10%
Embedded ALU	59	~ 9%

Abbildung 3.4: Ermittelte Ressourcen des Gauss Filter Designs auf der Bildeinzugskarte microEnable 5 Marathon VCX-QP

Die Ressourcen für den Block RAM und die Embedded Arithmetic Logic Unit (ALU) werden jeweils exakt berechnet, im Gegensatz zu einer sehr genauen Schätzung von LUT und FlipFlops. Der genaue Verbrauch ist erst nach dem Hardwarebau verfügbar. [5]

3.2.3 Nutzerfreundlichkeit

Die Anforderungen an die Nutzerfreundlichkeit der Funktionen in PyWizion legen fest, wie gut diese zu bedienen sein sollen. Dabei gibt es laut Nielsen verschiedene Aspekte, welche die Qualität sicherstellen: [25]

- *Erlernbarkeit:* Die Software sollte einfach zu erlernen sein, damit die Benutzer:in schnell mit der Arbeit mit der Software beginnen kann.
- *Effizienz:* Die Software sollte effizient zu nutzen sein, damit, nachdem die Benutzer:in die Software erlernt hat, ein hohes Produktivitätslevel möglich ist.
- *Behaltbarkeit:* Die Software soll einfach zu behalten sein, damit die gelegentliche Benutzer:in in der Lage ist, nach einer bestimmten Zeit der Nichtnutzung der Software zurückzukehren, ohne alles neu erlernen zu müssen.
- *Fehler:* Die Software soll eine geringe Fehlerquote aufweisen, damit Benutzer:innen während der Verwendung der Software wenig Fehler verursachen. Treten Fehler sollen diese leicht zu beheben sein. Zudem dürfen keine kritischen Fehler, die zu fehlerhaften Ergebnissen führen, auftreten.

- *Zufriedenheit*: Die Software soll angenehm zu verwenden sein, damit die Benutzer:in subjektiv zufrieden sind, wenn sie es nutzen.

Weil diese Kriterien nicht alle direkt messbar sind, müssen die Anforderungen teilweise anders greifbar gemacht werden. Ein Beispiel für die Messung der Effizienz ist es entweder eine maximale Zeit bei einem Test Nutzerfreundlichkeit vorzugeben oder die Zeit direkt zu messen, um einen Indikator für das Kriterium zu liefern. [25] Die Anforderungen an die Bildverarbeitungsfunktionen ist es, bei der Implementierung die Nutzerfreundlichkeit zu maximieren.

3.2.4 Einstieg in das PyWizion Framework

Um Entwickler:innen den Einstieg in das Framework zu erleichtern, soll bei der Implementierung von Bildverarbeitungsfunktionen auf eine leicht verständliche Syntax geachtet werden. Im Moment ist unabdingbar grundlegende Visual Applets Kenntnisse zu besitzen, da die Namen der Operatoren aus Visual Applets übernommen sind. Außerdem müssen teilweise diverse Einstellungen in Bezug auf die Bitbreite oder Parallelität vorgenommen werden. Das Ziel der Arbeit ist es, Entwickler:innen, die bereits mit Python Bildverarbeitungsanwendungen umsetzen können, den Zugang auf die Hardware zu verschaffen ohne grundlegende Einstellungen an den Operatoren vornehmen zu müssen.

3.3 Anforderungskatalog

Die wichtigsten Anforderungen aus diesem Kapitel werden in den Tabellen 3.1 und 3.2 aufgelistet.

Nummer	Anforderung
1.	Funktionen aus geeigneten VA Operatoren
2.	Vermeidung von Synchronisierungsproblemen
3.	Graustufen-/Farbbilder verarbeiten
4.	Korrekte Filterung mit verschiedenen Filtergrößen und unterschiedlichen Parametern
5.	Vergleichbar mit anderen Frameworks

Tabelle 3.1: Funktionale Anforderungen an die Bildverarbeitungsfunktionen

Nummer	Anforderung
1.	Auswahl der gängigen Bildverarbeitungsfunktionen
2.	Leicht verständliche Syntax
3.	Ressourcenarme Implementierung auf dem FPGA
4.	Hohe Nutzerfreundlichkeit

Tabelle 3.2: Nichtfunktionale Anforderungen an die Bildverarbeitungsfunktionen

Aus diesen Anforderungen wird die Aufgabenstellung konkretisiert. Die Funktionalität der Bildverarbeitungsfunktionen muss aus geeigneten Operatoren aus Visual Applets bestehen. Bei der Implementierung ist darauf zu achten, dass durch falsche Handhabung der Synchronisierung, durch die Verwendung von M/P-Typ Operatoren, keine Probleme in Form von Deadlocks auftreten. Außerdem sollen die Funktionen wenig Ressourcen auf der Hardware verbrauchen und sowohl Graustufen- als auch Farbbilder mit diversen Parametern korrekt verarbeiten können. Bei der Auswahl von Funktionen sollen die gängigsten verwendet und in leicht verständlicher Syntax implementiert werden, um eine möglichst hohe Nutzerfreundlichkeit zu bieten.

4 Bildverarbeitungsfunktionen

4.1 Auswahl der Funktionen

Bevor das Konzept für die Funktionen entwickelt wird, ist es erforderlich, die gängigsten Bildverarbeitungsfunktionen zu bestimmen. Bei der Bestimmung werden nur Bildvorverarbeitungsfunktionen betrachtet, da die Ressourcen auf der Hardware begrenzt sind und somit speicherintensive Algorithmen nicht implementiert werden können. Elementarer Bestandteil der Vorverarbeitung sind vor allem die Filterfunktionen, die das Bild für die Weiterverarbeitung manipulieren. [28]

Um herauszufinden, welche Funktionen in der Bildverarbeitung häufig verwendet werden, wird eine Literaturanalyse von drei Werke über digitale Bildverarbeitung herangezogen. Parallel dazu wird eine firmeninterne Umfrage durchgeführt, um herauszufinden was die häufig verwendeten Bildverarbeitungsfunktionen sind. Die Teilnehmer:innen wurden u.A. gefragt, welche Algorithmen/Funktionen in der Bildverarbeitung unerlässlich seien und ohne welche die Arbeit nicht machbar sei. Der komplette Fragebogen ist im Anhang A.1 dargestellt.

Literatur

Für die Analyse der Funktionen in der Bildverarbeitung werden drei Bücher herangezogen. Diese behandeln verschiedene Aspekte der Bildverarbeitung und sind die Grundlage der Literaturrecherche. Bei den ausgewählten Büchern handelt es sich um folgende Werke:

- “Industrielle Bildverarbeitung: Wie optische Qualitätskontrolle wirklich funktioniert“ von Christian Demant, Bernd Streicher-Abel, Axel Springhoff
- “Digitale Bildverarbeitung: Eine algorithmische Einführung mit Java“ von Wilhelm Burger, Mark James Burge
- “Digitale Bildverarbeitung und Bildgewinnung“ von Bernd Jähne

Das Buch über industrielle Bildverarbeitung wird ausgewählt, um einen Bezug zu der Industrie, in der die Einzugskarten mit den FPGAs eingesetzt werden, herzustellen. Zusätzlich biete dieses Werk ein ausführliches Kapitel zur Vorverarbeitung. Zweiteres ist eine Mischung aus dem theoretischen und praktischen Ansatz und bietet eine Übersicht über Grundlagen und Techniken der digitalen Bildverarbeitung. Das Buch von Bernd Jähne wird wegen des großen Stellenwertes der Person und dem Beitrag zu der Bildverarbeitung im Allgemeinen ausgewählt.

Es werden die Kapitel, welche thematisch von der Vorverarbeitung und im Speziellen von Filter handeln, gesichtet und die unterschiedlichen Funktionen festgehalten. Für die Auswertung werden alle Funktionen thematisch kategorisiert und der Häufigkeit nach aufgelistet (Abbildung 4.1). Die Kategorien sind Oberbegriffe für die Funktionen in der Bildvorverarbeitung. [31]

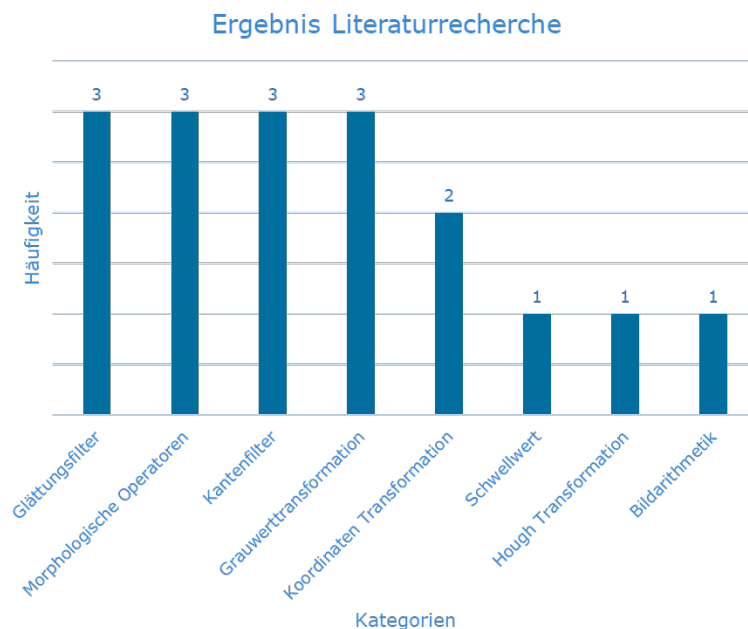


Abbildung 4.1: Ergebnisse aus der Literatur

Umfrage

Als zweite Informationsquelle wird eine firmeninterne Umfrage herangezogen. Inhaltlich geht es um die praktische Anwendung von Bildverarbeitungsfunktionen im Arbeitsalltag und welche Funktionen für die tägliche Arbeit unabdingbar sind. Diese Umfrage richtet

sich an Entwickler:innen, die bereits Erfahrungen in Themengebiet Bildverarbeitung haben und zusätzlich mit Kund:innen interagieren. Die Gesamtzahl der Teilnehmer:innen beläuft sich auf neun Mitarbeiter:innen der Basler AG.

Das Vorgehen bei der Auswertung ist sehr ähnlich, wie bei der Literatur. Die Antworten werden gesammelt und thematisch in Kategorien eingeteilt. Genannte Funktionen, die nicht zur Bildvorverarbeitung gehören, werden bei der Aufteilung der Ergebnisse nicht berücksichtigt. Für die Auswertung, dargestellt in Abbildung 4.2, werden die kategorisierten Antworten der Häufigkeit nach aufgelistet. Die Antworten der einzelnen Teilnehmer:innen befinden sich auf dem beigelegten Datenträger in dem Ordner Umfrageergebnisse mit dem Titel Bildverarbeitungsfunktionen.

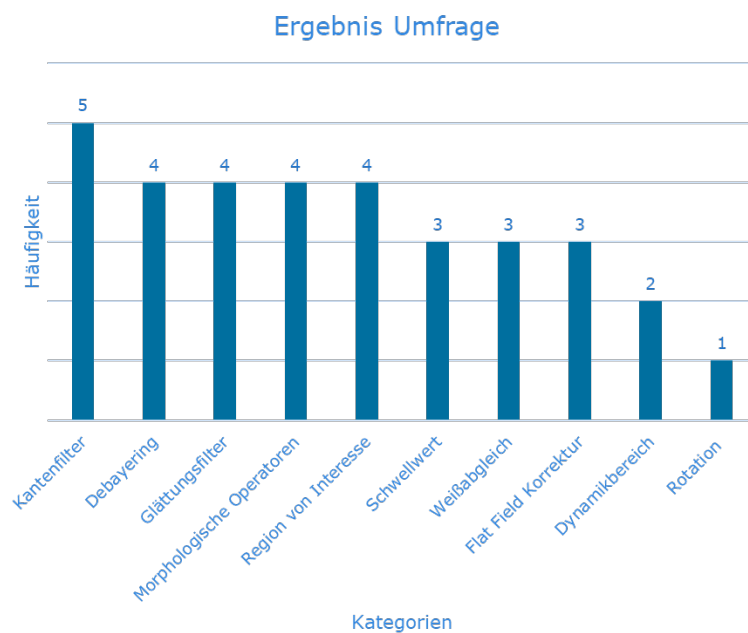


Abbildung 4.2: Ergebnisse der firmeninternen Umfrage

Ausgewählte Funktionen

Die Auswahl der Bildverarbeitungsfunktionen, die in PyWizion implementiert werden, wird aus Resultaten, welche in den Abbildungen 4.1 und 4.2 ist, abgeleitet. In dem Literaturergebnis ist zu erkennen, dass in allen Werken Filterfunktionen, die zu den Glättungs- und Kantenfilitern gehören, vorkommen. Außerdem werden die morphologischen Operatoren, wie Erode oder Dilate und die Verknüpfung beider erwähnt. Die Funktionsgruppe der Grauwerttransformationen wird in der Literatur in jedem Werk beschrieben, bleibt

in der Umfrage jedoch unerwähnt. Besonders das Debayering wird neben den Filterfunktionen sehr häufig von den Entwickler:innen in der Umfrage erwähnt und weist auf die große Bedeutung dieser Bildverarbeitungsfunktion in der Anwendung hin.

Nach der Sichtung der Ergebnisse werden diese auf Machbarkeit überprüft und entschieden, welche Funktionen implementiert werden. In der folgenden Tabelle sind die Resultate aufgelistet:

Kategorie	Ausgewählte Funktionen	Kategorie	Ausgewählte Funktionen
Morphologische Operationen	Dilate (nicht linear) Erode (nicht linear) Closing (nicht linear) Opening (nicht linear)	Glättungsfilter	Box (linear) Blur (linear) Bilateral (nicht linear) Gauss (linear) Median (nicht linear)
Kantenfilter	Laplace (linear) Sobel (linear) Scharr (linear)	Sonstige Funktionen	Debayering Adaptive Faltung (linear) Imread

Tabelle 4.1: Ausgewählte Bildverarbeitungsfunktionen

Außerdem sollen zusätzlich zu den Filtern noch die Funktionen Debayering, adaptive Filterung und Imread implementiert werden. Das Debayering beschreibt den Prozess, aus einer Bayer-Muster-basierten Graustufenaufnahme ein Farbbild zu erzeugen. Die adaptive Faltung beschreibt den Vorgang eines linearen Filters, mit dem Unterschied, dass die Koeffizienten der Filtermatrix nach dem Hardwarebau geändert werden können. Die Funktion Imread soll es ermöglichen ein Bild in den RAM des FPGA zu speichern und nach dem Hardwarebau auszulesen, um es z.B. für einen Vergleich mit einem neu aufgenommenen Bild zu verwenden.

4.2 Konzept

In diesem Kapitel wird mit Hilfe des Anforderungskataloges (Tabelle 3.1 und 3.2) ein Konzept für die Implementierung der Bildverarbeitungsfunktionen entworfen, welches die Anforderungen erfüllen soll. Dazu wird zunächst auf den Aufbau der Bildverarbeitungsfunktionen eingegangen. Danach wird die Auswahl der Visual Applets Operatoren für die Filter detailliert beschrieben, die Sicherstellung der weiteren Funktionalität und abschließend der Aufbau der Funktionstests für die Verifikation der Filter.

4.2.1 Bildverarbeitungsfunktionen in PyWizion

Für die Filterfunktionen in PyWizion wird eine neue Klasse erstellt in der alle Filter als Methoden implementiert werden, ebenfalls enthält diese Klasse alle Hilfsmethoden, die für Implementierung der einzelnen Filter notwendig sind. Für einen leichten Einstieg in das Framework wird die Syntax der Methoden an die Notation von OpenCV angelehnt. Es gilt sowohl für die Namen der Methoden als auch für die Übergabeparameter. Es kann angenommen werden, dass die Entwickler Kenntnisse in OpenCV besitzen, da diese Bibliothek sehr weit verbreitet ist.[33]

So bietet diese Bibliothek sehr viele Algorithmen, die sowohl in der Lehre als auch der Forschung verbreitet sind und dem aktuellen Stand der Technik entsprechen, an. Weltweit hat OpenCV mehr als 14 Millionen Downloads und wird von Unternehmen wie Google oder Microsoft verwendet.[33]

Bei dem generellen Aufbau der Filter muss nicht zwischen der Eigenschaft der Linearität und Nichtlinearität unterschieden werden, da nur die Verknüpfung der benachbarten Pixel unterschiedlich ist und dies durch die Anwendung von verschiedenen Visual Applets Operatoren realisiert wird.

In den Methoden muss sichergestellt sein, dass diesen nur zulässige Parameter bei Aufruf übergeben werden, sodass keine Fehler auf Grund von ungültigen Übergabeparametern auftreten. Eine valide Ausnahmebehandlung wird mit der Verwendung des Assert-Mechanismus erzielt.[15] Gleichzeitig kann durch eine Textausgabe auf der Konsole den Nutzer:innen die genaue Fehlerbeschreibung mitgeteilt werden.

Nach erfolgreicher Prüfung der Übergabeparameter findet die Auswertung der Eigenschaften des Eingangsdatenstroms statt. Relevant ist vor allem die Anzahl der Kanäle des Bildes, um zu verifizieren, ob es ein Graustufen- oder Farbbild ist. Für zweiteres werden die Kanäle einzeln gefiltert und später zusammengefügt. Des Weiteren wird die

Bittiefe des Bildes ermittelt, um den Datenstrom am Ausgang auf diese zu begrenzen, insofern es notwendig ist. Konzipiert werden die Methoden für 8-Bit Graustufen und 24-Bit RGB-Bilder.

Anschließend findet die Berechnung der Filterkoeffizienten statt. Für effiziente Berechnung können teilweise die Funktionen aus OpenCV verwendet werden. Dabei ist bei den Elementen der Matrix darauf zu achten, dass es sich zum einen um den List-Datentyp in Python handelt und zum andern nur ganzzahlige Werte annehmen darf. Grund dafür ist der **FIRoperatorNxM**, welcher in PyWizion bei Verwendung diese Anforderungen an die Parameter stellt. Mit geeigneter Skalierung der Filtermatrix kann das Problem der ganzzahligen Koeffizienten auf dem FPGA gelöst werden. Genauer beschrieben wird dies im folgenden Unterkapitel 4.2.2, welches die Filterfunktionen in VA behandelt.

Vor dem Hardwarebau ist es notwendig bei den Operatoren, die Datenströme aufteilen und zusammenführen, zu berechnen, ob diese fehlerfrei im Design implementiert werden können, damit der spätere Prozess ohne Störungen durchläuft. Die maximal zulässige Anzahl an Ausgängen ist aus der Visual Applets Dokumentation für die spezifischen Operatoren ersichtlich. [9]

Wenn keine Fehler bei der Prüfung der Eingabeparameter und der Machbarkeit auf der Hardware auftreten, wird das Design erfolgreich in Visual Applets gebaut. In Abbildung 4.3 ist das konzipierte Aktivitätsdiagramm exemplarisch für die Filter dargestellt.

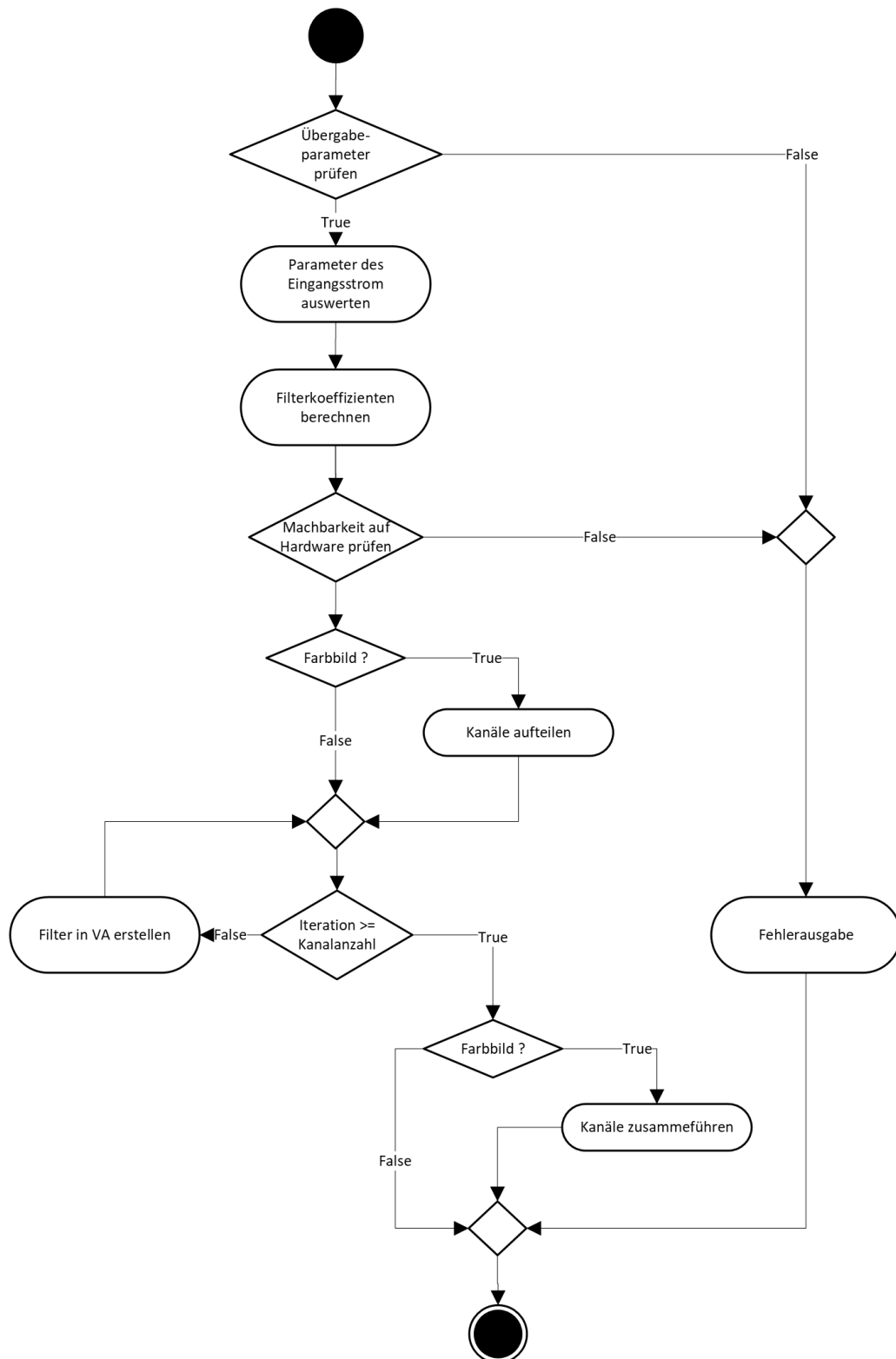


Abbildung 4.3: Aktivitätsdiagramm des generellen Filterfunktionsablaufs

4.2.2 Filterfunktionen in Visual Applets

Die Funktionalität von PyWizion basiert auf der Software Visual Applets. Um die Fehlerquote bei der Implementierung gering zu halten, werden alle Filter zunächst exemplarisch in Visual Applets erstellt, bevor diese in PyWizion implementiert werden. Deswegen ist ein grundlegendes Konzept für die Visual Applets Operatoren und deren Verwendung in den Filtern unerlässlich.

In dem Konzept für die Filterfunktionen in Visual Applets muss zunächst zwischen linearen und nichtlinearen Filtern unterschieden werden.

Für die Realisierung von linearen Filtern stellt die Bibliothek von Visual Applets bereits einige Operatoren zur Verfügung und es ist möglich, diese auf zwei verschiedene Arten zu verwenden. Ist die Filtermatrix separierbar, d.h. die Matrix kann durch eine Multiplikation von einem Spalten- und einem Zeilenvektor [32] abgebildet werden oder besteht aus nur aus einem Spalten- bzw. Zeilenvektor, ist diese durch die Operatoren **PixelNeighbours1xM** und **LineNeighboursNx1** realisierbar. Durch die Anwendung dieser Operatoren wird das Ausgangsbild, um dem Abstand vom Mittelpunkt des Vektors zum ersten Element, sowohl in X- als auch in Y-Richtung verschoben. [7][11] Die andere Möglichkeit ist es mit der Verwendung eines **FIRkernelNxM** den Filter zu implementieren. Dieser speichert die benötigten Pixelwerte für anschließende Operatoren zwischen. Beide Varianten bieten die Möglichkeit die Filtergröße anzupassen, indem die Parametereinstellungen der Verbindung zu dem **FIRoperatorNxM** geändert werden. Durch die Verwendung des **FIRkernelNxM** kann es zu Synchronisierungsproblemen auf dem FPGA kommen, weil es sich um einen M-Typ Operator handelt. Außerdem bietet dieser Operator zusätzlich zu Bildrandhandhabung mit einem konstanten Wert, den neuen Pixelwert durch eine Spiegelung an den Randpixeln zu bestimmen. [4]. Für beide Varianten wird der Operator **FIRoperatorNxM**, in dem die Koeffizienten eingetragen werden, benötigt.

In den Abbildungen 4.4 und 4.5 ist der Ressourcenverbrauch, welcher durch Visual Applets berechnet wird dargestellt. Durch die Implementierung der separierten Vektoren werden deutlich weniger Ressourcen benötigt als durch die Realisierung der vollständigen Filtermatrix.

Module	Type	LUT	RAM LUT	Flip Flops	Block RAM	Embedded ALU
 FIRkernelNxM	FIRkernelNxM	231	0	905	4	0
 FIRoperatorNxM	FIRoperatorNxM	568	0	1448	0	0

Abbildung 4.4: Ressourcenverbrauch: Boxfilter 3x3 mit FIRkernelNxM

4 Bildverarbeitungsfunktionen





Module	Type	LUT	RAM LUT	Flip Flops	Block RAM	Embedded ALU
 FIROperatorNxM0	FIROperatorNxM	136	0	416	0	0
 FIROperatorNxM1	FIROperatorNxM	168	0	512	0	0
 LineNeighboursNx10	LineNeighboursNx1	39	0	82	4	0
 PixelNeighbours1xM0	PixelNeighbours1xM	1	0	263	0	0

Abbildung 4.5: Ressourcenverbrauch: Boxfilter 3x3 mit PixelNeighbours1xM und LineNeighboursNx1

Es ist zu beachten, dass die entstandene Bildverschiebung durch die Variante der separierbaren Matrix, korrigiert werden muss, um die Ressourcen bei identischem Ergebnis zu vergleichen. Dafür werden zusätzlich ein **FIRkernelNxM** und der Operator **SelectSubKernel** benötigt und der Verbrauch ist in Abbildung 4.6 dargestellt. In der Tabelle 4.2 sind alle Ressourcen addiert aufgelistet. Die Anzahl von FlipFlops nimmt deutlich zu und übertrifft die der Matrix Implementierung. Der Grund dafür ist das Speicherverhalten durch den **FIRkernelNxM**.







Module	Type	LUT	RAM LUT	Flip Flops	Block RAM	Embedded ALU
 FIRkernelNxM	FIRkernelNxM	231	0	1321	6	0
 FIROperatorNxM0	FIROperatorNxM	136	0	416	0	0
 FIROperatorNxM1	FIROperatorNxM	168	0	512	0	0
 LineNeighboursNx10	LineNeighboursNx1	39	0	82	4	0
 PixelNeighbours1xM0	PixelNeighbours1xM	1	0	263	0	0
 SelectSubKernel	SelectSubKernel	0	0	0	0	0

Abbildung 4.6: Ressourcenverbrauch: Boxfilter 3x3 mit PixelNeighbours1xM und LineNeighboursNx1 verschoben

Ressourcen/ Implementierung	LUT	RAM LUT	FlipFlops	BlockRAM	Embedded ALU
Matrix	799	0	2353	4	0
Vektoren	344	0	1273	4	0
Vektoren + Bildverschiebung	575	0	2594	10	0

Tabelle 4.2: Übersicht Ressourcenverbrauch Box Filter 3x3

Beide Verfahren eignen sich für die Implementierung von linearen Filtern in Visual Applets, jedoch wird die Vorgehensweise mit der vollständigen Matrix verwendet, weil durch

die Verschiebung des Bildes der Vorteil, welcher durch die O-Typ Operatoren gewonnen wird, nichtig ist, da für die Verschiebung ein M-Typ Operator verwendet werden muss. Außerdem ist bei der Anwendung von größeren Filterkernel die Handhabung der Ränder relevanter und um einen großen Informationsverlust zu vermeiden, bietet sich das Verfahren der Kantenspiegelung, welches nur mit den **FIRkernelNxM** möglich ist, an. Bei größeren Filter steigen die Ressourcen an und der Vorteil des geringeren Verbrauchs überwiegt dem Nachteil der Verschiebung. Für den wissenschaftlichen Vergleich mit anderen Frameworks wird die Matriximplementierung verwendet. Der Vollständigkeit halber, für spätere Erweiterungen innerhalb des Frameworks und für die Hardwaretests wird das Verfahren mit den separierten Vektoren auch für einen Filter angewendet.

Nichtlineare Filter unterscheiden sich wie in Kapitel 2.1 beschrieben durch die nichtlineare Verknüpfung der benachbarten Pixel. Analog zu den linearen Filtern wird durch die Verwendung der **FIRkernelNxM** eine Filtermatrix mit einer definierten Größe aufgespannt. Da der **FIRoperatorNxM** nur für lineare Filter geeignet ist, kann dieser nicht verwendet werden.

In Visual Applets sind bereits einige nichtlineare Filter wie die Erosion, Dilatation und Medianfilter vorhanden. Der Medianfilter ist für die Kernelgrößen 3x3 und 5x5 implementiert. **Erosion** und **Dilatation** werden als Grundlage für die morphologischen Operationen verwendet und der **Median** Operator kann ebenfalls für den gleichnamigen Glättungsfilter benutzt werden, muss jedoch auch für größere Filterkernel verwendbar sein. Für die Filter, welche noch nicht in der Visual Applets Bibliothek vorhanden sind, muss anhand der mathematischen Beschreibung zunächst ein Design erstellt werden. Anschließend wird das entworfene Design aus Visual Applets in PyWizion implementiert und auf Übereinstimmung mit dem gleichen Filter aus der OpenCV Bibliothek verglichen. Dieses Vorgehen wird sowohl für die linearen- als auch für die nichtlinearen Filter angewendet.

Division

Für den Vergleich mit OpenCV müssen die Koeffizienten der Matrix von den Glättungsfilter in Visual Applets angepasst werden. In OpenCV werden die Elemente des Filterkernels als Gleitkommazahl dargestellt und auch verrechnet. Da es in Visual Applets nur möglich ist ganzzahlige Koeffizienten für den **FIRoperatorNxM** zu verwenden, müssen diese durch eine nachfolgende Division approximiert werden. Das Ergebnis der Näherung ist weiterhin ganzzahlig. Visual Applets bietet zwei Varianten wie die Division realisierbar ist. Zum einen gibt es den Divisionsoperator **DIV**, welcher bei Nutzung sehr

viele Ressourcen benötigt, zum anderen ist es möglich durch die Operatoren **SCALE** und **ShiftRight** eine Division durchzuführen. Zweierpotenzen lassen sich sehr einfach über eine Bitverschiebung realisieren und benötigen wenige Ressourcen im Gegensatz zu der äquivalenten Rechnung mit dem **DIV** Operator. In Abbildung 4.7 ist eine Division mit 16 mit beiden Varianten durchgeführt und es ist deutlich zu sehen, dass durch Bit Verschiebung weniger Ressourcen verwendet werden.

Module	Type	LUT	RAM LUT	Flip Flops	Block RAM	Embedded ALU
CONST	CONST	0	0	0	0	0
DIV	DIV	1528	0	3520	0	0
ShiftRig...	ShiftRight	0	0	0	0	0

Abbildung 4.7: Ressourcenverbrauch: Division mit 16

Ist der Divisor keine Zweierpotenz, muss die Division angenähert werden. Durch Bitverschiebung und geeignete Skalierung ist dies möglich. Für Bittiefen von acht Bit ist empirisch ermittelt worden, dass es ausreichend ist den Skalierungsfaktor wie folgt zu nähern:

$$Scale = \frac{Bit\ Verschiebung}{Divisor} = \frac{2^{10}}{42} \approx 24.38 \quad (4.1)$$

Das Ergebnis muss gerundet werden, da die der **SCALE** Operator ebenfalls nur ganzzahlige Parameter zulässt. Diese Variante hat den Nachteil, dass wegen des Rundens Abweichungen von einem Bit möglich sind. Durch den zusätzlichen Operator werden mehr Ressourcen benötigt, jedoch ist der Verbrauch deutlich geringen als mit dem **DIV** Operator (Abbildung 4.8).

Module	Type	LUT	RAM LUT	Flip Flops	Block RAM	Embedded ALU
Hierarchical...	Hierarchical...	1360	0	2496	0	0
CONST	CONST	0	0	0	0	0
DIV	DIV	864	0	1920	0	0
SCALE	SCALE	496	0	576	0	0
ShiftRight0	ShiftRight	0	0	0	0	0

Abbildung 4.8: Ressourcenverbrauch: Division mit 42

Wegen des deutlich geringeren Ressourcenverbrauchs wird für die Division auf der Hardware die Variante der Bit Verschiebung verwendet und Abweichungen von einem Bit pro Rundung als Toleranz erlaubt.

Vermeidung von Synchronisierungsproblemen

Durch Verwendung von M-Typ Operatoren ist es möglich, dass Synchronisierungsprobleme auftreten können. Das kann entweder durch den **SYNC** Operator oder durch geeignete Positionierung des M-Typ Operators vermieden werden. Wenn alle Datenströme, die für die Verarbeitung notwendig sind, aus dem gleichen M-Typ Operator entstehen, gibt es keine Synchronisierungsprobleme. [10] Für die Filter, insbesondere bei Farbbildern, kann der **FIRkernelNxM** vor die Aufteilung der Datenströme platziert werden. Wenn es aufgrund von Berechnung neuer Pixelwerte nicht möglich ist, kann auch der **SYNC** Operator eingesetzt werden, um Synchronisierungsprobleme zu vermeiden.

Anpassung der Bitbreiten

Für den Vergleich der Ergebnisbilder aus PyWizion/Visual Applets und OpenCV ist es bei einigen Filterfunktionen notwendig die Bitbreiten des Ausgangsdatenstroms auf die Eingangsbitbreite zu bringen. Bei Filtern, welche eine normalisierte Filtermatrix besitzen, wird das Vorgehen, wie zuvor bei der Division beschrieben, das Konzept der Bitverschiebung angewendet. Eine andere Möglichkeit ist es den Operator **CastBitWidth** zu verwenden. Mit diesem wird die Bitbreite des Datenstromes am Ausgang einstellbar. Weil durch den Einsatz alle Bits, welche höherwertig als die eingestellte Bitbreite sind, abgeschnitten werden, ist dieser nicht für alle Anwendungen geeignet. Um den Wertebereich zu limitieren, werden **ClipHigh** und **ClipLow** verwendet. Mit ersterem werden alle Pixel, welche über dem einstellbaren Schwellwert liegen, auf diesen begrenzt, zweiteres setzt alle Pixel, die kleiner als der Schwellwert sind, auf das Minimum.

Auswahl der Simulationsplattform in Visual Applets

Visual Applets bietet an für das Design unterschiedliche Bildeinzugskarten mit den jeweiligen Spezifikationen zu wählen. Um sicherzustellen, dass bei Änderung der Hardware die Ressourcen noch in ausreichend vorhanden sind, muss der FPGA mit der geringsten Anzahl von logischen Bausteinen und den niedrigsten Hardwarespezifikationen ausgewählt werden.

4.2.3 Sicherstellung der weiteren Funktionalität

Die zu implementierenden Bildverarbeitungsfunktionen werden um die Methoden Debayering, Filter2D und Imread erweitert, da diese elementare Bestandteile der Bildverarbeitungskette sind.

Debayering

Wie in Kapitel 3.1.2 erläutert, wird der Debayering Algorithmus dafür verwendet, um aus einem Bayer-Bild ein Farbbild zu generieren. In Visual Applets sind bereits die Operatoren für 3x3 und 5x5 **Bayer** vorhanden. Diese können auch für die Implementierung verwendet werden. Es soll möglich sein, bei dem Funktionsaufruf zwischen den beiden Operatoren unterscheiden zu können. Außerdem muss das Muster des Bayer Operators mit angegeben werden, damit die Berechnung auf dem FPGA äquivalent zu dem Bayer Muster auf dem Sensor abläuft. Des Weiteren müssen die Spezifikationen der Operatoren eingehalten werden.

Filter2D

Bei der Methode Filter2D handelt es sich um die adaptive Faltung, die einen linearen Vorgang beschreibt. Wie bereits erwähnt, soll es möglich sein nach dem Hardwarebau die Koeffizienten der Filtermatrix zu ändern. Mit der Implementierung aus dem bisherigen Konzept ist es nicht möglich, die Koeffizienten zu ändern, weil der **FIRoperatorNxM** nur die Möglichkeit bietet, die Koeffizienten als statische Parameter zu implementieren. Statisch bedeutet, dass dem Operator der Wert zum Zeitpunkt des Hardwarebaus zugewiesen wird. Für die Änderung nach dem Hardwarebau muss der Operator die Parametereinstellung dynamisch bereitstellen.

Die Faltung besteht, wie in den Grundlagen 2.1 beschrieben, aus mehreren Additionen und Multiplikationen. Für diese beiden arithmetischen Anweisungen gibt es die Operatoren **ADD** und **MULT** in Visual Applets. Die Gewichtung der einzelnen Koeffizienten kann durch den **CONST** Operator realisiert werden. Dieser besitzt auch die Eigenschaft die Parameter dynamisch anzupassen. Um alle Pixel innerhalb der Nachbarschaft mit einbeziehen zu können, wird mit dem **FIRkernelNxM** die benötigte Anzahl zwischengespeichert und mit der Verschiebung (**SelectSubKernel**) werden die einzelnen Elemente der Matrix ausgewählt.

Imread

Für die Methode `Imread` wird ein Operator, der es ermöglicht ein Bild in Form einer Textdatei zu laden und die einzelnen Pixelwerte in dem RAM der Bildeinzugskarte zu speichern, in Visual Applets benötigt. Für das Ablegen der Daten im Speicher bietet sich das Konzept einer LUT an, da auf die Speicheradresse sehr schnell zugegriffen werden kann und diese auf der Hardware leicht zu implementieren sind. Visual Applets bietet für die Erstellung von LUTs die Operatoren **LUT**, **RAMLUT** und **KNEELUT** an. Letzterer wird bei der Auswahl nicht berücksichtigt, weil dieser Baustein nicht auf allen FPGAs implementierbar ist. Bei dem **LUT** Operator wird der Adressraum durch die Bitbreite des Eingangsdatenstroms definiert, im Gegensatz zu dem einstellbaren Adressraum, der durch den RAM der Bildeinzugskarte limitiert wird, des **RAMLUT** Operators [12] [8]. Wegen des limitierten Adressraumes des **LUT** Operators können nur kleine Bilder, die maximal 64 KByte groß sind, gespeichert werden. [8] Der **RAMLUT** Operator kann durch die Auslagerung des Speichers auf dem RAM der Bildeinzugskarte mehr Pixelwerte speichern. Auf den Bildeinzugskarten der Generation `microEnable 4` können bereits 16 MByte gespeichert werden. [12]

Die Breite des Ausgangsstroms ist ebenfalls durch den Adressraum bei dem **LUT** Operator definiert, im Unterschied zu einer variablen anpassbaren Bitbreite, welche durch die Hardware begrenzt wird, bei dem **RAMLUT** Operator. Letzteres bietet auch die Möglichkeit einzustellen, wie viele Pixel parallel übertragen werden sollen. Damit kann die Bandbreite angepasst und erhöht werden. Vor allem wegen der anpassbaren Bandbreite wird für die Implementierung der `Imread` Methode der **RAMLUT** Operator verwendet.

4.2.4 Funktionstest

Damit die Implementierung der Filter als korrekt verifiziert werden kann, muss das Ergebnisbild, wie in Unterkapitel 4.2.2 erwähnt, mit dem Resultat des identischen OpenCV-Filters verglichen werden. In diesem Vergleich wird geprüft, ob die Pixelwerte an gleichen Koordinaten identisch sind. Dafür wird das Ergebnisbild aus PyWizion von dem aus OpenCV subtrahiert, sodass ein Differenzbild entsteht, und anschließend der Betrag gebildet. (Gleichung 4.2). Dieser Ablauf wird bei Graustufenbildern angewendet, bei Farbbildern muss jeder einzelne Kanal die eben beschriebene Prozedur durchlaufen.

$$Diff(m, n) = |I_{PW}(m, n) - I_{CV}(m, n)| \quad (4.2)$$

Für einen effizienten Testablauf werden die Tests mit einer Software durchgeführt. Die Wahl fällt auf Pytest, weil bisher implementierte Funktionen in PyWizion mit diesem Werkzeug durchgeführt werden. Für Konsistenz innerhalb der Tests wird dies fortgeführt.

Da in einer Vielzahl der Fachliteratur die Filter mit Kernelgrößen von 3x3 und 5x5 getestet werden [22][28][18][35][24], werden diese Größen auch für PyWizion verwendet. Zusätzlich wird die Größe 7x7 getestet. Die Beschränkung auf ungerade Filtergrößen resultiert aus dem fehlenden Hot Spot für die Abbildung des neuen Pixelwertes bei geraden Kernelgrößen, des Weiteren lassen die meisten Filterfunktionen nur ungerade Kernelgrößen zu. Die verschiedenen Filter werden jeweils mit unterschiedlichen Parametern aufgerufen und für alle Kombinationen getestet. Bei den Parametern wird darauf geachtet, dass diese geringe und hohe Wertigkeiten abdecken.

Es werden sowohl Graustufen- als auch Farbbilder natürlichen und synthetischen Ursprungs für die Tests herangezogen. Weil das spätere Anwendungsgebiet auf reelle Bilder basiert, werden diese vor allem getestet. Dafür werden 50 zufällig gewählte Bilder des RAISE Datensatz [19], sowohl in Graustufen als auch in Farbe, herangezogen. Für die Verifikation synthetischer Bilder wird exemplarisch mit einem Graustufen- und einem Farbbild getestet. Die synthetischen Bilder werden auf der Hardware erzeugt und den Bildern wird Rauschen in Form von zufälligen Pixelwerten mit einer Bittiefe von vier addiert. Sowohl die natürlichen als auch die synthetischen Bilder sind auf dem beigelegten Datenträger in dem Unterordner Rohbilder vorhanden.

Die Methoden Debayering, Filter2D und Imread werden nicht explizit mit ihrem Äquivalent aus OpenCV getestet. Sowohl für die Debayering als auch für die Imread Methode wird die Funktionsweise manuell auf der Hardware getestet. Die Methode Filter2D wird gegen den Faltungsoperator in Visual Applets getestet.

4.3 Implementierung

In diesem Unterkapitel wird zunächst der Aufbau der Filterklasse und die Sicherstellung der Bildverarbeitungskette beschrieben. Anschließend wird auf Besonderheiten der Filter in PyWizion und den Aufbau der Funktionstests eingegangen.

4.3.1 Aufbau der Filterklasse

In Abbildung 4.9 sind alle implementierten Methoden der Filterklasse, ohne die Übergabeparameter für die Übersichtlichkeit, dargestellt. Diese erhält als einziges Attribut den PyWizion Wizard, der notwendig ist, um neue Operatoren in Visual Applets zu erstellen und in dem Design zu platzieren.

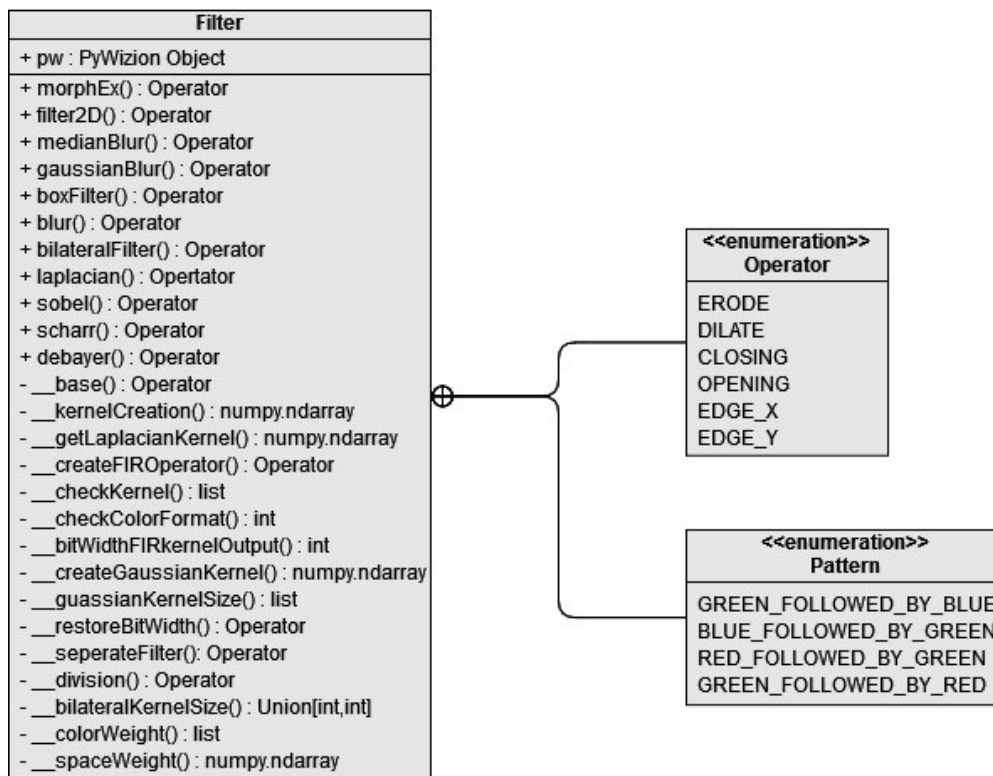


Abbildung 4.9: Klassendiagramm Filter mit den wichtigsten Methoden

Die Methoden in der Filterklasse werden nach dem Modularitätsprinzip implementiert und sind in der Funktionalität abgegrenzt. Diese Art der Implementierung bietet die

Möglichkeit, das Abstraktionsniveau zu erhöhen. [34] In der Filterklasse enthalten die öffentlichen Methoden die Filter und die privaten Methoden werden verwendet, um die Funktionalität für die unterschiedlichen Filter zu erzeugen. Die Namensgebung der öffentlichen Methoden ähnelt der Syntax der Filterfunktionen aus OpenCV und werden nach dem jeweiligen Filter bzw. in dem Fall der morphologischen Operation nach dieser benannt.

Des Weiteren sind in der Klasse zwei Aufzählungen, die für das Farbmuster (Pattern) der `debayer()` Methode und die Operation (Operator) der `__base()` Methode, enthalten. Der gesamte Quellcode der FilterHelper Klasse ist auf dem beigelegten Datenträger in dem Unterordner PyWizion des Verzeichnisses Quellcode zu finden.

Private generelle Methoden

Die privaten Methoden sind nach dem Zweck, der durch den Aufruf erfüllt werden soll, benannt. Durch den modularen Aufbau ist es möglich, die Funktionalitäten, welche für mehrere Filter benötigt werden, in einer allgemein gehaltenen Methode, wie die Folgenden, unterzubringen. Auf der Methode `__kernelCreation()` basiert das Zwischenspeichern der Pixelwerte für die der Koeffizientenmatrix, die für alle Filter verwendet wird. `__checkColorFormat()` prüft anhand der Parameter des eingehenden Datenstroms die Anzahl der Farbkanäle und wird bei Filtern, die Farbbilder verarbeiten können, verwendet. In `__restoreBitWidth()` wird durch Bitverschiebung und Skalierung, wenn notwendig, der Ausgangsdatenstrom des Filters normalisiert. Entspricht die Summe der Filterkoeffizienten keiner Zweierpotenz, wird mit `__division()` der geeigneter Skalierungsfaktor ermittelt, um die Bitbreite dem Eingangsdatenstrom anzupassen. Die Aufteilung in X- und Y-Richtung der Filtermatrix für separierbare lineare Filter wird in `__seperateFilter()` angewendet.

In `__base()` werden die Operation der morphologischen Filter durch die Basisoperatoren **Dilate** und **Erode** aus Visual Applets realisiert. Die Methode besitzt noch weitere Funktionalitäten, aber werden nicht weiter erwähnt, da diese nicht im Rahmen der Arbeit entstanden sind.

Private explizite Methoden

Die anderen privaten Methoden sind explizit für einen Filter, der aus dem Namen abgeleitet werden kann, implementiert. In `__getLaplacianKernel()` findet die Berechnung der Filterkoeffizienten für den Laplace Filter statt.

Für den Filter mit adaptiven Koeffizienten werden die Methoden `__createFIROperator()`, `__checkKernel()` und `__bitWidthFIRkernelOutput()` verwendet. In ersterer Me-

thode wird die Faltung durch die Visual Applets Operatoren, wie in dem Konzept für Filter2D (4.2.2) beschrieben, realisiert. Zweitere prüft explizit, ob der Datentyp des übergebenen Kerns korrekt ist. Letztere passt die Bitbreite des Datenstroms am Ausgang des adaptiven Filters an, sodass diese explizit auf die Koeffizienten und möglichen neuen Pixelwerte angepasst ist.

In der Methode `__createGaussianKernel()` werden die Koeffizienten, die abhängig von der Abweichung der Normalverteilung in X/Y-Richtung und der Kernelgröße sind, für den Gaussfilter berechnet. Sollte die Größe von den Nutzer:innen bei dem Aufruf des Filters nicht übergeben werden, wird die Größe des Filters mit `__gaussianKernelSize()` anhand der Abweichungen der Normalverteilung ermittelt.

In `__bilateralKernelSize()` wird der Durchmesser, der ebenfalls durch die Nutzer:innen bestimmt werden kann, und der Radius der Filtermatrix für den Bilateralfilter berechnet. Die Berechnung der Koeffizienten findet in `__colorWeight()` statt und in `__spaceWeight()` wird ermittelt, welche Koeffizienten in die Kalkulation des neuen Pixelwertes miteinbezogen werden sollen.

Öffentliche Methoden

Bei den öffentlichen Methoden der Filterklasse handelt es sich bis auf `debayer()` um die Filterfunktionen, welche in PyWizion verwendet werden können.

Mit der Methode `morphEx()` werden die morphologischen Filter durch den Aufruf von `__base()` in das Visual Applets Design implementiert. Für das Closing und Opening können unterschiedliche Strukturelemente für die aufeinanderfolgenden Basisoperationen verwendet werden.

Die Kantenfilter werden als die Methoden `laplace()`, `sobel()` und dem Spezialfall des Sobelfilters `scharr()` in PyWizion implementiert. Der Aufbau dieser Methoden ist bis auf die Berechnung der Koeffizienten identisch und ist wie in dem Konzept für lineare Filter beschrieben realisiert. Des Weiteren wird die Bitbreite des Datenstroms am Ausgang des Filters auf die Eingangsbitbreite begrenzt. Da bei den Kantenfiltern die Differenz innerhalb der Nachbarschaft gebildet wird, müssen die Pixelwerte als vorzeichenbehaftet interpretiert werden. Die Änderung der Vorzeicheninterpretation wird mit dem Operator `CastType` umgesetzt.

Die linearen Glättungsfiler werden in den gleichnamigen Methoden `blur()`, `boxFilter()` und `gaussianBlur()` in der Filterklasse implementiert. Der Aufbau der Methoden ist bis auf die Berechnung der Koeffizienten identisch. Für die Kalkulation der Matrix des Gaussfilters wird die Funktion `cv2.getGaussianKernel()` aus der OpenCV Bibliothek verwendet. Für die Filtervektoren des Boxfilters werden der Kernelgröße entsprechend jeweils

ein Spalten- und ein Zeilenvektor, die jeweils aus Einsen bestehen, erstellt. Beide Filter verwenden für die Normalisierung die Methode `__restoreBitWidth()`.

Für das nichtlineare Medianfilter wird die Methode `medianBlur()` erstellt. Diese verwendet bei den Kernelgrößen 3x3 und 5x5 die bereits vorhandenen Implementierung aus der Visual Applets Bibliothek. Bei größeren Kernel wird durch den **SORT** Operator die gespeicherten Pixelwerte der Größe nach sortiert und durch **SelectSubKernel** der Median ausgewählt.

In `bilateralFilter()` wird das nichtlineare Glättungsfilter implementiert. Dieses hat keine festen Filterkoeffizienten, sondern diese sind abhängig von der miteinzubeziehenden Nachbarschaft. Deswegen müssen die Pixel in der Nachbarschaft separat behandelt werden. Um alle möglichen Koeffizienten abdecken zu können, wird eine **LUT**, die nach der Bitbreite des Eingangstroms ausgelegt wird, implementiert. Um Ressourcen zu sparen wird vor dem Designbau ermittelt, welche Nachbarn mit in die Berechnung des neuen Pixelwertes miteinzubeziehen sind. Durch die Separierung müssen die einzelnen Berechnungen addiert und zum Schluss normalisiert werden. In diesem Fall ist es nicht möglich die Division durch Bitverschiebung und einer Konstanten zu realisieren, da die Wertigkeit der Parameter zu Beginn der Designerstellung nicht feststehen und für jedes Eingangsbild unterschiedlich sind. Aus diesem Grund wird der **DIV** Operator für die Normalisierung verwendet.

4.3.2 Filter in PyWizion

Verzögerung des Datenstroms

Im Rahmen der Berechnung der Filterkoeffizienten und Parameter der Visual Applets Operatoren in PyWizion sind Informationen des Eingangsdatenstroms, wie die Bitbreite und Anzahl der Kanäle, erforderlich. Während des Implementierungsprozesses wurde festgestellt, dass die Informationen des Datenstroms an den Eingängen der Operatoren nicht sicher anliegen und somit in PyWizion nicht abrufbar sind. Vermehrt tritt dies auf, wenn in der Erstellung des Designs die Informationen des Datenstroms abgerufen werden. Ein Grund für diese Verhalten kann die Interprozesskommunikation zwischen PyWizion und Visual Applets sein. Damit die korrekten Informationen in PyWizion verarbeitet werden können, werden zusätzliche Operatoren in das Design eingebaut. Dabei handelt es sich um den **NOP** Operator. Dieser verbraucht minimale Ressourcen in Form eines Registers und fungieren als Speicherelement. Durch das Erstellen von

zusätzlichen Operatoren liegen die abzufragenden Informationen sicher am Operator und somit ebenfalls in PyWizion an.

Behandlung von Farbkanälen

Um zu ermitteln, wie viele Kanäle das Eingangsbild besitzt, wird die Methode `__checkColorFormat()` verwendet. Der Rückgabewert dieser Methode entspricht der Anzahl der vorhandenen Kanäle. Wie im Konzept beschrieben, wird jeder Farbkanal einzeln gefiltert. Damit in Visual Applets auf die jeweiligen Kanäle zugegriffen werden kann, muss der Eingangsdatenstrom mit dem Operator **SplitComponents** aufgeteilt werden. Um auf dem FPGA Ressourcen zu sparen und Synchronisierungen zu vermeiden, wird der **FIRkernelNxM**, mit dem die benötigten Daten zwischengespeichert werden, vor der Aufteilung der Farbkanäle verwendet. In Abbildung 4.10 sind die durch Visual Applets veranschlagten Ressourcen für die Implementierung der Kernels vor der Aufteilung der Kanäle dargestellt.

Module	Type	LUT	RAM LUT	Flip Flops	Block RAM	Embedded ALU
▼ Hierarchical...	Hierarchical...	231	0	2569	11	0
FIRkernel...	FIRkernelNxM	231	0	2569	11	0
SplitCom...	SplitCompo...	0	0	0	0	0

Abbildung 4.10: Ressourcenverbrauch: FIRkernelNxM vor SplitComponents

Der Ressourcenverbrauch für die Implementierung für jeden einzelnen Farbkanal ist in Abbildung 4.11 dargestellt. Zusätzlich muss der **SYNC** Operator verwendet werden, da die Farbkanäle mit dem Operator **MergeComponents** zusammengeführt werden müssen. Durch die Verwendung des M-Typ Operators kann auf dem FPGA nicht sichergestellt werden, dass die Kanäle ohne Synchronisierungsprobleme korrekt zusammengeführt werden können.

Module	Type	LUT	RAM LUT	Flip Flops	Block RAM	Embedded ALU
▼ HierarchicalB...	Hierarchical...	6765	3474	9749	12	0
FIRkernel...	FIRkernelNxM	231	0	905	4	0
FIRkernel...	FIRkernelNxM	231	0	905	4	0
FIRkernel...	FIRkernelNxM	231	0	905	4	0
SplitCom...	SplitCompo...	0	0	0	0	0
SYNC	SYNC	6072	3474	7034	0	0

Abbildung 4.11: Ressourcenverbrauch: FIRkernelNxM nach SplitComponents

Wegen des geringeren Ressourcenverbrauchs wird bei der Implementierung die Variante mit dem Kernel vor der Aufteilung verwendet. Außerdem besteht mit dieser Variante keine Möglichkeit Deadlocks innerhalb des Filters zu erzeugen, da in den Datenströmen der einzelnen Farbkanäle keine M-Typ Operatoren enthalten sind.

Die Filter, welche Farbbilder verarbeiten können, sind in Tabelle 4.3 aufgelistet. Die morphologischen Operationen arbeiten in Visual Applets nur mit Binärbildern [3], die aus zwei unterschiedlichen Werten bestehen, und werden aus diesem Grund nicht für Farbbilder implementiert. Die anderen Filter werden sowohl für Farb- als auch für Graustufenbilder implementiert.

Funktionen	Grau	RGB	Funktionen	Grau	RGB
Dilate	✓	-	Box	✓	✓
Erode	✓	-	Blur	✓	✓
Closing	✓	-	Bilateral	✓	-
Opening	✓	-	Gauss	✓	✓
Laplace	✓	✓	Median	✓	✓
Sobel	✓	✓	Adaptive Faltung	✓	✓
Scharr	✓	✓			

Tabelle 4.3: Implementierte Funktionen und zulässige Eingangsbilder

Aufruf der Funktionen

Um die Filter in PyWizion zu verwenden, muss zunächst ein neues Objekt der Filterklasse erstellt werden. Mit diesem kann ein beliebiges Filter aus der Klasse ausgewählt werden. Exemplarisch ist in Quellcode 4.1 der Methodenaufruf des Bilateralfilters in PyWizion dargestellt. Einige der Übergabeparameter sind für alle Filter identisch. Dazu gehören `x` (Eingangsdatenstrom), `edgehandling` (Handhabung der Ränder des Bildes), `constant`, (Wert für die Kantenspiegelung mit Konstanten Werten) und `Hbox_Name` (Name der hierarchischen Box in Visual Applets). Die hierarchische Box in Visual Applet dient der Übersichtlichkeit, da Operatoren in dieser zusammengefasst werden können. In Abbildung 4.13 ist eine hierarchische Box mit dem Namen adaptive dargestellt. Standardmäßig werden bei dem Funktionsaufruf die Parameter `edgehandling`, `constant` und `Hbox_Name` mit den Werten, wie in dem Quellcode 4.1 dargestellt, aufgerufen. Die Parameter `d`, `sigma_color` und `sigma_space` sind für die Koeffizientenvergleich erforderlich.

```
46     # Using the bilateral filter of the filter class
47     filters = Filter(pw)
48     filter = filters.bilateralFilter(x=input,
49                                     d=diameter,
50                                     sigma_color=sigma_Color,
51                                     sigma_space=sigma_Space,
52                                     edgehandling="EdgeMirrored",
53                                     constant=0,
54                                     HBox_Name="None")
```

Listing 4.1: Exemplarischer Aufruf einer Filtermethode anhand des Bilateralfilters

4.3.3 Weitere Funktionalitäten

Imread

In Abbildung 4.12 ist die ImageIO Klasse mit den enthaltenen Attributen und Methoden dargestellt. Die Klasse enthält als einziges Attribut den PyWizion Wizard, welcher benötigt wird, um Operatoren in dem Design zu erstellen.

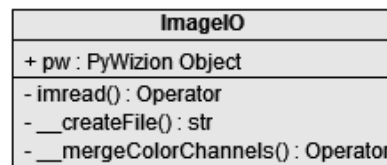


Abbildung 4.12: Klassendiagramm ImageIO

In der Methode `__createFile()` wird die Textdatei, in der die Pixelwerte als Dezimalzahlen enthalten sind, erstellt. Es können sowohl Farb- als auch Graustufenbilder gespeichert werden. Für den Fall, dass ein Farbbild aus dem RAM des FPGA geladen werden soll, wird die Methode `__mergeColorChannels()` verwendet. Diese teilt den Eingangsstrom in drei äquidistante Datenströme für die Farbkanäle auf. Um die Bandbreite des Designs zu maximieren, können die Nutzer:innen die Parallelität nach dem **RAMLUT** Operator anpassen. Ein vereinfachtes Aktivitätsdiagramm der Methode `imread()` ist im Anhang A.3 dargestellt. Der Quellcode dieser Klasse ist auf dem beigelegten Datenträger in dem Unterordner PyWizion des Verzeichnisses Quellcode zu finden.

Debayering

Der Ablauf der `debayering()` Methode aus der Filterklasse (Abbildung 4.9) ist im Anhang A.2 vereinfacht dargestellt. Die Besonderheit bei beiden Bayer Operatoren in Visual Applets ist die Anforderung an die Parallelität des Datenstroms am Eingang. Die Parallelität muss eine Zweierpotenz sein. Für den Fall, dass dies nicht zutrifft, wird die Parallelität auf die nächstgelegene Zweierpotenz gebracht. Nach dem debayering ist es den Nutzer:innen durch einen Übergabeparameter der Methode möglich, die Parallelität auf den ursprünglichen Wert zu setzen oder die aktuelle beizubehalten. Außerdem wird die Bitbreite des Datenstromes am Ausgang auf das dreifache des Eingangsdatenstroms angepasst. Damit wird sichergestellt, dass die Pixelwerte jedes Farbkanals korrekt übertragen werden können.

Filter2D

Zusätzlich wie in dem Konzept beschrieben, ist es möglich zwischen dem Filter mit adaptiven Koeffizienten und einer normalen Faltung durch einen Übergabeparameter zu wechseln. Außerdem kann die Implementierungsweise der **ADD** Operatoren auf dem FPGA gewählt werden. Standardmäßig ist der Parameter auf Auto eingestellt, kann aber durch die Nutzer:innen auf Lookup Table Random Access Memory (LUTRAM) oder Block Random Access Memory (BRAM) geändert werden. Damit kann die Ressourcenverteilung aktiv beeinflusst werden für den Fall, dass der FPGA an die Ressourcenbegrenzung kommt.

4.4 Verifikation

In diesem Unterkapitel werden die Filterfunktionen aus PyWizion mit den Äquivalenten aus OpenCV und Visual Applets getestet. Das Filter mit adaptiven Koeffizienten wird mit dem Operator **FIRoperatorNxM** aus Visual Applets getestet. Anschließend werden die anderen Filter gegen die gleichnamigen Funktionen aus OpenCV getestet. Für den Test wird, wie in dem Konzept 4.2.4 beschrieben, die Differenz der gefilterten Bilder aus beiden Frameworks gebildet. Abschließend werden Hardwaretests mit den implementierten Funktionen auf der Bildeinzugskarte CXP-12 C1 [2] durchgeführt.

4.4.1 Visual Applets

Für den Test des Filters mit adaptiven Koeffizienten wird dieser mit dem **FIRoperatorNxM** aus Visual Applets getestet. Das Eingangsbild wird parallel mit beiden Filtern, welche identische Koeffizienten und Kernelgröße besitzen, gefaltet und anschließend wird die Differenz gebildet. In Abbildung 4.13 ist das Design und das Ergebnis der Subtraktion dargestellt. Das Histogramm in der rechten Seite des Fensters zeigt die Verteilung der Pixelwerte über das gesamte Bild. Es ist zu erkennen, dass alle Pixel in dem Bild die Wertigkeit null haben. Das kann an den Ergebnissen unterhalb des Histogramms entnommen werden. Der Mittelwert und sowohl der maximale und minimale Pixelwert ist null. Somit sind die Ergebnisse beider Faltungen identisch und es ist sichergestellt, dass der adaptive Filter auf der Hardware eingesetzt werden kann.

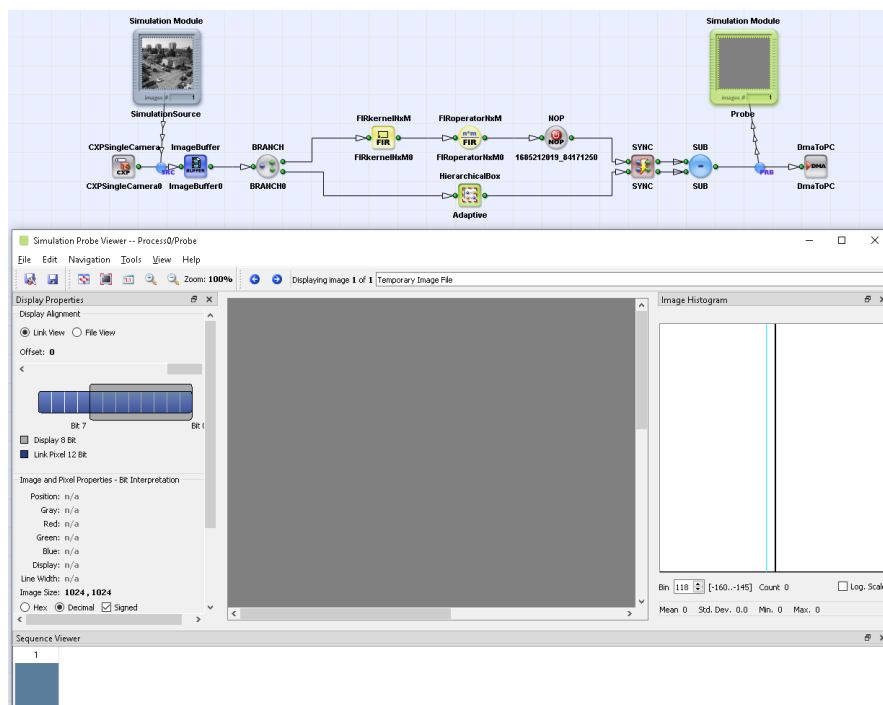


Abbildung 4.13: Differenz der Ergebnisse aus der adaptiven Faltung und des FIRoperatorNxM

Dieses Design wurde mit PyWizion erstellt und in Visual Applets ausgewertet. Durch die Erhöhung des Abstraktionsgrades in PyWizion ist es nicht möglich den **FIRkernelNxM** vor der Aufteilung des Datenstroms zu implementieren. Aus diesem Grund muss bei der Zusammenführung der Ströme eine Synchronisierung stattfinden.

4.4.2 PyWizion vs. OpenCV

Für die Verifikation der Filter werden die Ergebnisse der Filterung aus PyWizion mit dem äquivalenten Filtern aus OpenCV verglichen. Dabei ist wichtig, dass die Parameter, die relevant für die Berechnung der Koeffizienten sind, in beiden Frameworks identisch sind. Deswegen wird bei der Auswahl der Parameter darauf geachtet, dass diese in beiden Frameworks valide sind. Für den Vergleich zwischen den Filterfunktionen aus OpenCV und PyWizion wird, wie in Unterkapitel 4.2.4 beschrieben, der Betrag der einzelnen Pixel aus dem Differenzbild herangezogen.

Auswahl der Parameter

Die morphologischen Filter werden mit den Strukturelementen, die ebenfalls in OpenCV implementiert sind, getestet. Die Filter haben die Struktur einer Ellipse, eines Kreuzes und eines Rechteckes. [29]

Die Kantentfilter Sobel und Scharr werden sowohl für die erste Ableitung in X- als auch in Y Richtung und verschiedenen Skalierungsfaktoren getestet. Die Faktoren befinden sich innerhalb und außerhalb des Wertebereichs, der mit der Bittiefe des Bildes darstellbar ist. Die Skalierungsfaktoren haben die Wertigkeit von 1, 10, 100 und 500. In den getesteten Fällen wird jeweils mit der Bittiefe acht gearbeitet. Der Laplacefilter wird ebenfalls mit dem gleichen Skalierungsfaktoren getestet.

Der Boxfilter wird sowohl mit normalisiertem Kernel als auch mit nicht normalisiertem Kernel (blur) getestet. Der Medianfilter besitzt außer der Kernelgröße keine zusätzlichen Parameter, die variiert werden können. Für den Bilateralfilter wird mit unterschiedlichen Einstellungen für die räumliche und farbliche Gewichtung getestet. Es werden Parameter, die geringe (0.5), mittlere (50) und große (150) Auswirkungen auf das Ergebnisbild haben, ausgewählt. [29] Bei dem Gaussfilter wird die Standardabweichung von null bis zwei Sigma in X- und Y Richtung in 0.5er Schritten getestet.

Alle Filter werden mit den eben erwähnten Parametern in den vorher definierten Größen 3x3, 5x5 und 7x7 getestet.

Festlegen der Fehlertoleranzen

In dem Konzept für die Division in Visual Applets (Kapitel 4.2.2) wird erläutert, dass pro Rundung in Visual Applets die Toleranz in der Differenz um eins erhöht wird. Bei der Berechnung der Koeffizienten in PyWizion wird ebenfalls pro durchgeführter Rundung die Toleranz um eins inkrementiert. Während des Tests für Farbbilder stellte sich

heraus, dass durch die Verwendung der Operatoren **SplitComponents** und **MergeComponents** Abweichungen von einem Bit entstehen. Die Toleranzen beziehen sich auf alle Pixel des Bildes und gewähren jedem Pixel die folgenden Abweichungen:

Filter	Grau	RGB
Bilateral	3	-
Box	1	2
Blur	0	1
Gauss	2	3
Laplace	0	1

Filter	Grau	RGB
Median	0	1
Morphologisch	0	-
Scharr	0	1
Sobel	0	1

Tabelle 4.4: Zulässige Fehlertoleranzen der Filter

Vorgehensweise für die Auswertung der Differenzbilder

Auf jedes Bild aus dem Datensatz werden alle Filter mit den vorher festgelegten Parametern angewendet und die Daten in einer Textdatei gespeichert. Mit einem Python-Skript werden die Messwerte aufbereitet und automatisiert in Tabellen festgehalten. Das Python-Skript, die Textdateien und die Skripte für die Aufnahme der Differenzen sind auf dem beigelegten Datenträger zu finden. Die Aufbereitung der Daten umfasst die Berechnung des Medians und des Durchschnittes der abweichenden Pixel und der Differenz. Außerdem wird noch die maximale Differenz, die pro Filter in dem Datensatz vorkommt aufgenommen. Exemplarisch sind die Resultate des Bilateralfilters für natürlichen Graustufenbildern (Tabelle A.1) im Anhang dargestellt. Die anderen Ergebnisse der Filter sind in Tabellen auf dem beigelegten Datenträger zu finden. Die Differenzbilder, welche für die Daten der Tabelle herangezogen werden, sind exemplarisch in der Abbildung 4.15 und 4.16 dargestellt. Es handelt sich um die Differenzbilder des Bilateralfilters mit identischer Gewichtung von Raum und Farbe, jedoch mit unterschiedlichen Filtergrößen. Das Ursprungsbild für beide Filter ist in Abbildung 4.14 dargestellt.



Abbildung 4.14: Exemplarisches Simulationsbild des RAISE Datensatzes [19]

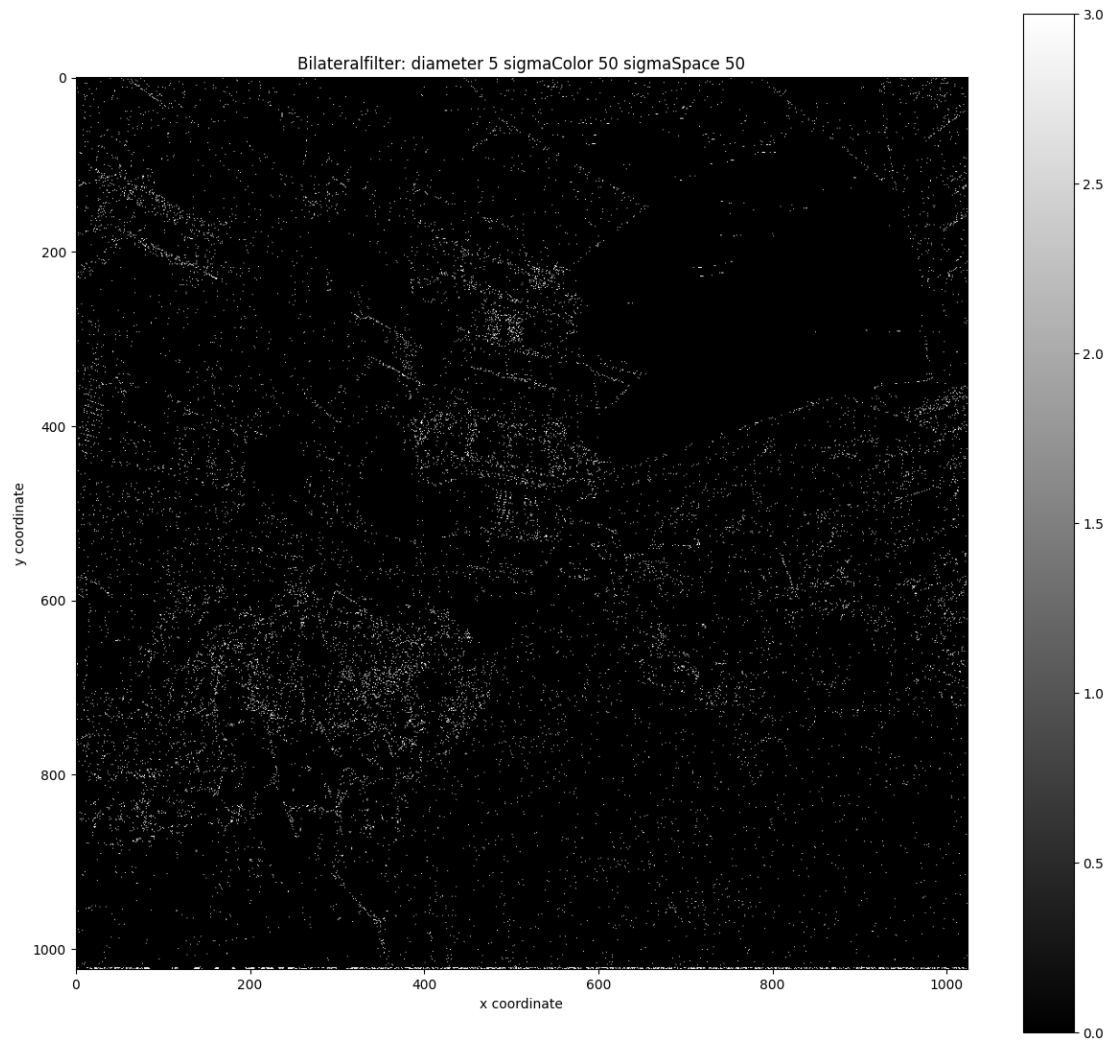


Abbildung 4.15: Differenzbild Bilateralfilter Durchmesser 5, Sigma Color 50, Sigma Space 50, maximale Differenz 3, Abweichende Pixel 1.65 %

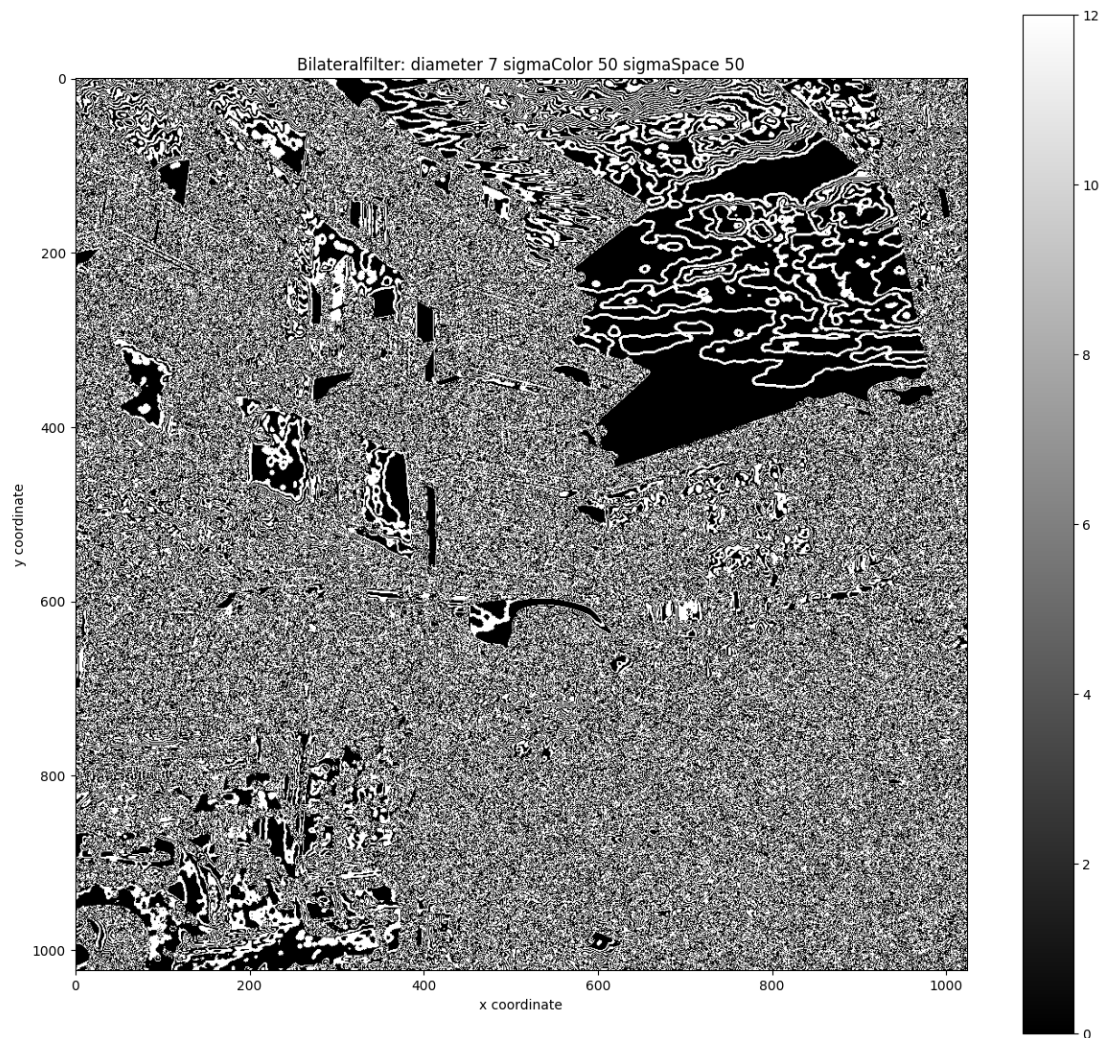


Abbildung 4.16: Differenzbild Bilateralfilter Durchmesser 7, Sigma Color 50, Sigma Space 50, maximale Differenz 12, Abweichende Pixel 46.18 %

Um die Abweichungen deutlich erkennbar werden zu lassen, werden mit dem Schwellwertverfahren aus der OpenCV Bibliothek alle Pixelwerte, die größer als null sind auf 255 gesetzt und anschließend invertiert. Die originalen Differenzbilder sind auf dem Datenträger einsehbar. Die unter den Bildern dargestellt Abweichung über alle Pixel beträgt in dem Differenzbild des Bilateralfilters mit dem Durchmesser 5 (Abbildung 4.15) 1.65 % und in dem Differenzbild des Bilateralfilters mit dem Durchmesser 7 (Abbildung 4.16) 46.18 %. Die maximale Differenz beträgt bei dem erst genannten drei und dem anderen zwölf im. Es ist gut zu erkennen, dass die Abweichungen im Bild vermehrt in Regionen

auftreten, in denen viele Kanten vorhanden sind. Durch das Differenzbild Abbildung 4.16 wird verdeutlicht, dass die Abweichungen bei größerer miteinzubeziehender Nachbarschaft stärker ausfallen. Dadurch wirkt das Bild in stark verrauscht außerhalb von homogenen Flächen. Innerhalb von homogenen Flächen gibt es in beiden Bildern weniger Abweichungen als bei nicht homogenen Regionen. Je größer die Filtermatrix, desto mehr Abweichungen sind in den homogenen Flächen vorhanden. Diese wirken nicht zufällig verteilt, sondern erstrecken sich entlang von Kanten.

Auswertung

Mit den Messdaten für die jeweiligen Filter und die Fehlertoleranzen wird bestimmt, zu welchem Verhältnis die Ergebnisse der Filter aus PyWizion mit denen aus OpenCV übereinstimmen. Um das Verhältnis zu bestimmen, werden die Anzahl der erfolgreichen Tests durch die Gesamtanzahl der Tests für den jeweiligen Filter dividiert. Erfolgreiche Tests sind alle Testergebnisse, die innerhalb der definierten Toleranzen liegen. Dafür werden die maximale Differenz und der Median verwendet. Der Median wird verwendet, wegen des unempfindlichen Verhaltens gegenüber Ausreißern in der Messreihe. Die Ergebnisse mit natürlichen Bildern werden sowohl für Graustufen- als auch für Farbbilder in den Tabellen 4.5 und 4.6 dargestellt.

Filter	Ergebnisse innerhalb der Toleranz [%]			
	Grau	RGB		
		Rot	Grün	Blau
Bilateral	81.48	-	-	-
Box	100.00	100.00	100.00	100.00
Blur	100.00	-	-	-
Gauss	76.00	64.00	76.00	76.00
Laplace	100.00	100.00	100.00	100.00
Median	100.00	100.00	100.00	100.00
Erode	100.00	-	-	-
Dilate	100.00	-	-	-
Opening	100.00	-	-	-
Closing	100.00	-	-	-
Sobel	100.00	100.00	100.00	100.00
Scharr	100.00	100.00	100.00	100.00

Tabelle 4.5: Übereinstimmung der Filter mit deren Äquivalent aus OpenCV anhand der maximalen Abweichung

Filter	Ergebnisse innerhalb der Toleranz [%]			
	Grau	RGB		
		Rot	Grün	Blau
Bilateral	96.30	-	-	-
Box	100.00	100.00	100.00	100.00
Blur	100.00	-	-	-
Gauss	76.00	76.00	76.00	76.00
Laplace	100.00	100.00	100.00	100.00
Median	100.00	100.00	100.00	100.00
Erode	100.00	-	-	-
Dilate	100.00	-	-	-
Opening	100.00	-	-	-
Closing	100.00	-	-	-
Sobel	100.00	100.00	100.00	100.00
Scharr	100.00	100.00	100.00	100.00

Tabelle 4.6: Übereinstimmung der Filter mit deren Äquivalent aus OpenCV anhand des Medians der Abweichung

Maximale Differenz Tabelle

Der Bilateralfilter liegt für 81.84 % der Testfälle bei Graustufenbildern innerhalb der Toleranz. Sowohl für Graustufen- als auch für Farbbilder weisen der Box-Filter, der Laplacefilter, der Medianfilter, alle morphologischen Operationen und der Sobel/Scharrfilter eine hundertprozentige Einhaltung der Toleranzen auf. Der Blurfilter liegt ebenfalls für Graustufenbilder innerhalb der Grenzen. Für Farbbilder ist es nicht möglich valide Messwerte aufzunehmen, da PyWizion zum jetzigen Zeitpunkt Farbbilder mit einer Breite von acht Bit pro Kanal in der Simulation ausgibt. Aus diesem Grund werden diese nicht in die Auswertung mit aufgenommen. Der Laplacefilter und der Sobel Filter erreicht bei Graustufenbildern 100 % , jedoch bricht der Test bei dem Skalierungsfaktor von 500 und der Kernelgröße von 7x7 ab. Der Grund dafür ist die Begrenzung des Wertebereiches des Visual Applets Operators **FIRoperatorNxM**. Deswegen ist es nicht möglich, den Filter für diese Parametereinstellungen zu testen. Der Gaussfilter erfüllt nur in 76 % bei Graustufenbildern und zu 64 % bei Farbbildern die Toleranz. Aus dem Ausschnitt der Tabellen für die Ergebnisse des Gaussfilters (A.4) ist zu erkennen, dass die Abweichungen erst auftreten, wenn die Standardabweichung in X Richtung (σ_X) die Wertigkeit eins erreicht und solange anhalten, bis σ_Y denselben Wert wie σ_X hat. Die Berechnung der Koeffizienten kann diese Abweichungen nicht verursachen, weil diese mit der Funktion aus OpenCV durchgeführt wird. Da die Koeffizienten in OpenCV als Fließkommazahlen dargestellt werden, müssen diese für die Visual Applets Operatoren auf ganzzahlige Werte gebracht werden. Ein möglicher Grund in der großen Abweichung kann in dieser Anpassung liegen.

Die Ergebnisse für synthetische Bilder sind nahezu identisch mit den für natürliche Bilder und werden deswegen nicht dargestellt, sondern sind auf dem Datenträger einzusehen. Lediglich der Bilateralfilter weist geringe Abweichungen zu diesem Ergebnis auf. Für das synthetische Bild liegt die maximale Differenz über alle Parametervarianten zu 100 % innerhalb der Toleranzen.

Median Differenz Tabelle

Da ein Großteil der Resultate identisch zu denen aus der Tabelle mit den maximalen Differenzen ist, wird nur noch auf die Unterschiede eingegangen. Die Ergebnisse des Medians der Differenzen sind sehr ähnlich zu denen der maximalen Differenz (4.5). Lediglich die Resultate der Bilateralfilter und des Gaussfilters sind unterschiedlich. Im Median des Bilateralfilter liegt dieser zu 96.3 % innerhalb der Toleranzen. Das deutet bei der maximalen Differenz (Tabelle 4.5), die in dem Datensatz ermittelt wird, auf einen Ausreißer

hin. Bei dem Gaussfilter ändert sich nur das Ergebnis des roten Kanals, das ebenfalls auf einen Ausreißer bei der maximalen Differenz hindeutet. Im Median liegen die Ergebnisse zu 76 % für alle Kanäle innerhalb der Toleranzen.

4.4.3 Hardware

Für die Hardwaretests werden jeweils Designs mit allen Filterfunktionen, soweit es wegen der endlichen Ressourcen auf dem FPGA möglich ist, erstellt. Für Graustufenbilder stellt es bei Filtergrößen von 3x3 und 5x5 kein Problem dar, alle Filter in einem Design unterzubringen. Für 7x7 Kernelgrößen werden, bis auf den Bilateralfilter und die adaptive Faltung, alle weiteren Filter verwendet. Der Bilateralfilter mit der Größe 7x7 wird in einem separaten Design erstellt. Die adaptive Faltung (Filter2D) ist wegen der Einschränkung von Operatoren nicht in der Größe abbildbar. Für Filter, die Farbbilder verarbeiten können, wird ebenfalls für die Kernelgrößen 3x3, 5x5 Kernel und 7x7 jeweils ein Design erstellt. Außerdem wird noch ein Debayering mit in die Verarbeitungskette eingebaut.

Als Bildquelle für alle Designs kann entweder die Imread Funktion oder ein auf der Hardware erzeugtes synthetisches Bild verwendet werden. Sowohl die Quelle als auch die Filter können über den Operator **SourceSelector** ausgewählt werden. Durch die Verwendung des Operators **DmaToPC** werden die Bilddaten direkt in den RAM des PCs transferiert. Durch den Direct Memory Access (DMA) ist es möglich, 128 Bit gleichzeitig zu übertragen. Da die Bits, die ein Pixel repräsentieren, zusammenhängend übertragen werden müssen, muss die Parallelität auf fünf verringert werden. Diese Veränderung verursacht das Hinzufügen von Dummypixeln am Rand des Bildes, da die Parallelität kein Vielfaches der Bildbreite ist. Somit wird das Bild um eine Spalte vergrößert. Für die Hardwaretests werden folgenden Parameter gewählt:

Parameter/ Farbe	Parallelität DMA Eingang	Bitbreite	Breite des Bildes	Höhe des Bildes	Pixel gesamt	Systemtakt
RGB	5	24 Bit	1025 Pixel	1024 Pixel	3148800	312.5 MHz
Grau	8	8 Bit	1024 Pixel	1024 Pixel	1048576	312.5 MHz

Tabelle 4.7: Parametereinstellung Testdesigns

Um das Design aus Visual Applets auf dem FPGA verwenden zu können, muss dieses in einen Bitstrom übersetzt werden. Dafür wird über Visual Applets das Xilinx Werkzeug

aufgerufen. Die dadurch erzeugte Hardware Applet Datei (HAP) wird mit der Software microDisplay X auf die Hardware geladen und der Datenstrom am DMA kann visualisiert werden.

Test auf der CXP-12 Interface Card 1C

Das Design wird für die Interface Karte CoaXPress (CXP)-12 1C mit einem CXP Eingang erstellt. Auf der Karte befindet sich der Xilinx Ultrascale+ KU3P FPGA, welcher mit einem Systemtakt zwischen 312.5 MHz und 400 MHz betrieben wird. Getestet wird mit einem Systemtakt von 312.5 MHz. Die zur Verfügung gestellten Bandbreite, an der Schnittstelle zum Computer, liegt in der Theorie bei 3.938 MB/s und in der Praxis bei maximal 3260 MB/s. [2] Die praktische Bandbreite ist durch den Hersteller definiert als Maximum, das sicher erreicht werden kann. Die maximale Bandbreite in den Designs für die Hardwaretests kann wie folgt berechnet werden:

$$\text{Bandbreite} = \text{Parallelität} \cdot \text{Systemtakt} \quad (4.3)$$

Die Parallelität gibt an, wie viele Pixel parallel innerhalb eines Taktes verarbeitet werden können.[10] Für die Tests wird eine Parallelität von acht gewählt (Tabelle 4.7). Aus der Bandbreite und Pixelanzahl des Bildes kann die Wiederholungsrate pro Sekunde (Frames Per Second (FPS)) ermittelt werden, wie in Gleichung 4.4 dargestellt.

$$FPS = \frac{\text{Bandbreite}}{\text{Pixelanzahl Bild}} \quad (4.4)$$

Bei der Berechnung der maximalen FPS müssen zusätzlich die Parameter des DMAs berücksichtigt werden. Dieser ermöglicht es maximal 128 Bit gleichzeitig zu übertragen. Die Anzahl, der zu übertragenden Bits, wird aus dem Produkt der Parallelität und der Bittiefe berechnet und darf das Limit des DMAs nicht überschreiten. Mit den Parametern aus der Tabelle 4.7 ergeben sich in der Theorie für Graustufenbilder 2384 Bilder pro Sekunde und für Farbbilder sind 1488 Bilder pro Sekunde möglich.

Mit der Software microDisplay X wird die Anzahl der Bilder pro Sekunde direkt ermitteln. Es werden die FPS-Kennwerte für Filter jeder Kernelgröße, die sowohl auf synthetischen als auch reelle Graustufen- und Farbbildern angewendet werden, ermittelt.

Kernelgröße	Grau		RGB	
	natürlich [FPS]	synthetisch [FPS]	natürlich [FPS]	synthetisch [FPS]
3x3	2291	2282	1101	1100
5x5	2289	2280	1100	1100
7x7	2286	2279	1100	1101

Tabelle 4.8: Median der gemessenen FPS der Designs in microDisplay X

Die Unterschiede zwischen den einzelnen Filtern ist minimal und deswegen werden diese nicht separat aufgeführt, sondern zusammengefasst in den Kernelgrößen dargestellt (Tabelle 4.8).

Die minimalen Unterschiede zwischen den natürlichen und synthetischen Farbbildern lässt sich auf Messungenauigkeiten zurückführen. Bei den synthetischen Graustufenbildern ist die Anzahl der Bilder pro Sekunde minimal geringer als bei den natürlichen. Der geringe Unterschied lässt sich durch zwei verschiedene Ursachen erklären. Zum einen kann es an dem Betriebssystem Windows des Testrechners liegen. Die Messungen können durch Prozesse mit höhere Priorität kurzzeitig ausgesetzt haben, sodass die Ergebnisse minimal voneinander abweichen. Zum anderen kann es durch Wärmeentwicklung auf der Bildeinzugkarte zu Einbußen in der Leistung kommen. Die Abweichungen zwischen den theoretischen Bildern pro Sekunde und den gemessenen bei Graustufenbildern sind ebenfalls auf diesen Grund zurückzuführen. Bei Farbbildern limitiert der Durchsatz am DMA die maximal erreichbaren FPS im Gegensatz zu der Limitierung innerhalb des Designs bei Graustufenbildern. Dementsprechend wird für die Berechnung der maximalen Bilder pro Sekunde die praktische Bandbreite verwendet. Das Ergebnis der Division beträgt 1035 Bilder pro Sekunde. Die Messungen weichen um ca. sechs Prozent ab und sind etwas über den erwarteten Ergebnissen.

Nicht abbildbare Filter

Auf dem FPGA der vorliegenden Bildeinzugkarte ist es nicht möglich den Median Filter mit der Größe 7x7 für Farbbilder auf Grund des zu hohen Ressourcenverbrauchs zu implementierten. Der Operator **SORT**, mit dem die Pixelwerte innerhalb des Kerns der Reihenfolge nach sortiert werden, beansprucht zu viele Ressourcen. Vor allem die Anzahl der LUTs und FlipFlops überschreitet die Ressourcen des FPGAs.

Der adaptive Filter ist ebenfalls nicht für die Kernelgröße 7x7 realisierbar, weil die Anzahl der benötigten Verzweigungen, die mit dem **BRANCH** Operator erstellt werden, in der

aktuellen Implementierung die maximale Anzahl der zur Verfügung stehenden Zweige überschreitet.

5 Nutzerfreundlichkeit

Die in dieser Arbeit implementierten Bildverarbeitungsfunktionen sollen in Hinsicht auf Nutzerfreundlichkeit evaluiert und analysiert werden. Als Vergleich für PyWizion wird Visual Applets, in dem bisher Bildverarbeitungsanwendungen implementiert werden, verwendet. Die Funktionalität beider Umgebungen ist nahezu identisch und somit liegt der Unterschied in der Bedienung der Software.

In den Anforderungen 3.2.3 werden Aspekte, die gute Nutzerfreundlichkeit sicherstellen, aufgelistet. Für die Messbarkeit dieser Kriterien wird eine Kombination aus qualitativen Daten in Form von Bemerkungen, die während der Testdurchführung von den Nutzer:innen geäußert werden ("Lautes Denken" [30]), und quantitativen Daten, die nach dem Test durch den System Usability Score (SUS) erhoben werden, verwendet. Mit dem quantitativen Test werden die Kriterien für Erlernbarkeit und Zufriedenheit messbar gemacht. Für die Bestimmung des Testkriteriums der Effizienz wird die benötigte Zeit der Testdurchführung erhoben. Ob die Funktionen mit einer niedrigen Fehlerrate implementiert werden können, wird durch die Zählung und Kategorisierung der Fehler festgestellt.

System Usability Scale

Bei der System Usability Scale Methode handelt es sich um ein robustes und zuverlässiges Verfahren, um die Nutzerfreundlichkeit von Systemen oder Software zu evaluieren. Die Methode basiert auf einem Fragebogen, der zehn standardisierte Fragen, die alternierend positiv und negativ formuliert sind, aufweist. [27] Durch diesen Wechsel sollen die Proband:innen beim Ausfüllen des Fragebogens nicht in ein Muster verfallen. Durch den Fragebogen werden die Erlernbarkeit (Frage 4,10) und die Zufriedenheit (Frage 1-3, 5-9) der Nutzerfreundlichkeitsmerkmale abgedeckt. [27] Für die Zwecke dieser Arbeit wird der Begriff System durch die jeweilige Testumgebung ersetzt (Exemplarisch der Fragebogen für PyWizion A.8, A.9). Für die Auswertung des Fragebogens werden die Ergebnisse der Fragen mit ungeraden Nummern mit eins subtrahiert und anschließend zusammen addiert. Bei den Fragen mit geraden Nummern muss das Ergebnis von 5 subtrahiert

werden, bevor die Ergebnisse aufsummiert werden dürfen. Die Wertigkeit aller Fragen ist identisch und liegt zwischen Null und Vier. Die Endergebnisse aus den gerade und ungerade Fragen müssen addiert und anschließend mit 2.5 multipliziert werden, um ein aussagekräftiges Endergebnis für die SUS Methode zu erhalten. [17] In Abbildung 5.1 ist die Skala, welche für die Bewertung des Ergebnisses herangezogen wird, dargestellt. [1]

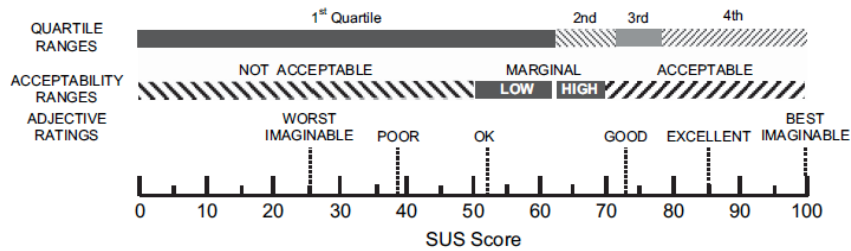


Abbildung 5.1: Bewertungsskala SUS Auswertung[1]

“Lautes Denken“

Die Methode “Lautes Denken“ oder im Englischen “Thinking Aloud“ ist laut Nielsen eine der am weit verbreitetsten und effektivsten Werkzeuge, um die Nutzerfreundlichkeit zu messen. [25] Dabei verbalisieren die Proband:innen die Gedanken, welche während des Tests aufkommen, und zeigen potentielle Schwächen der zu testenden Software auf. Durch diese ungewohnte Art der Kommunikation kann die Bearbeitungsdauer beeinflusst werden.

5.1 Auswahl der Nutzer:innen

Die Zielgruppe für den Test sind Entwickler:innen, die keinerlei Kenntnisse in digitaler Schaltungstechnik und der Software Visual Applets haben. Diese sollen mit den bereitgestellten Funktionen innerhalb von PyWizion in der Lage sein, Bildverarbeitungsfunktionen auf einem FPGA zu implementieren. Um diese Funktionen anwenden zu können, müssen die Personen der Zielgruppe grundlegende Kenntnisse in der Programmiersprache Python besitzen. Zusätzlich müssen die Testpersonen Bildverarbeitungsanwendungen mit OpenCV umsetzen können, da die Syntax der implementierten Funktionen an die von den Algorithmen aus OpenCV angelehnt ist.

Typischerweise liegt die Anzahl der Teilnehmer:in bei klassischen Tests, in denen nur der Testleiter und eine Teilnehmer:in anwesend sind, bei vier bis zehn Proband:innen.[26]

5.2 Testdesign

Ziel des Tests ist es verschiedene Filter sowohl in PyWizion als auch Visual Applets zu implementieren. Als TestszENARIO werden drei Aufgabestellungen konzipiert. Den Teilnehmer:innen werden in die Bedienung der jeweiligen Software kurz eingewiesen. Bei der Schwierigkeit des Tests muss berücksichtigt werden, dass sehr wenige Entwickler:innen am Basler Standort Ahrensburg Visual Applets bedienen können und somit komplexere Aufgaben wegfallen, um eine Tendenz zugunsten PyWizion zu vermeiden. Jede Aufgabestellung beinhaltet die Implementierung von zwei Filtern, dabei handelt es sich jeweils um einen linearen und einen nichtlinearen Filter. Das Gerüst der Aufgabestellung (A.2.2 A.5 A.6 A.7) ist so konzipiert, dass nur die Filter implementiert werden müssen. Um alles Weitere, wie z.B. Ein- und Ausgänge des Bildverarbeitungsprozesses müssen sich die Proband:innen nicht kümmern.

Es wird bewusst auf die Festlegung der Reihenfolge, mit welcher Software gestartet werden soll, verzichtet, um sicherzustellen, dass die Teilnehmer:innen keinen Lerneffekt haben. Stattdessen wird den Proband:innen die Entscheidung überlassen, mit welcher Software gestartet wird. Dadurch soll vermieden werden, dass alle Versuchsteilnehmer:innen mit derselben Software starten und somit einen Lerneffekt gegenüber der nachfolgenden Software haben. Sollte sich herausstellen, dass die Teilnehmer regelmäßig mit derselben Software starte, wird die Reihenfolge durch den Versuchsleiter vorgegeben.

Vor dem Start der Tests werden die Vorkenntnisse in Python, OpenCV, VHDL, Visual Applets und PyWizion jedes Einzelnen aufgenommen. Für die Bearbeitung der Aufgaben bekommen die Proband:innen eine maximale Zeit, die nicht überschritten werden darf. Die maximale Zeit wird mithilfe eines Probelaufs ermittelt.

Die Teilnehmer:innen werden vor dem Start der Studie von dem Testleiter in die jeweilige Software eingeführt, indem anhand eines ausgewählten Beispiels die Syntax erläutert wird. Neben der Aufgabestellung erhalten die Proband:innen die Dokumentation zu Visual Applets und zu den Filterfunktionen von OpenCV. Da die Dokumentation von PyWizion nicht vollständig, bzw. nur im Quellcode vorhanden ist, wird die Dokumentation von OpenCV zur Verfügung gestellt. Des Weiteren erhalten die Proband:innen den Hinweis, wie die Berechnung der Koeffizienten des Gauss Filter durchzuführen ist.

Anschließend haben die Proband:innen die Möglichkeit Fragen zu der jeweiligen Software zu stellen. Danach beginnt der Test und die Bearbeitungszeit mit der jeweiligen Software wird mit der Stoppuhr eines Mobiltelefons gemessen und festgehalten. Ebenso werden grobe Fehler, die zu einer Verhinderung der Funktionalität führen, gezählt und vermerkt. Ist die Bearbeitung in der jeweiligen Software abgeschlossen, die maximale Bearbeitungsdauer erreicht oder die Proband:innen bricht die Aufgabe ab, wird der Fragebogen der System Usability Scale ausgefüllt. Der gleiche Ablauf gilt auch für die zweite Software.

5.3 Probedurchlauf

Um den Testablauf zu validieren, wurde ein Probedurchlauf mit einer Probandin durchgeführt. Dabei wurde vor allem auf die Verständlichkeit der Aufgabenstellung und die Erklärung der verschiedenen Umgebungen durch die Testleitung geachtet. Des Weiteren wurde die maximale Bearbeitungszeit für die Aufgabenstellungen nach dem Probelauf festgelegt. Die Probandin für den Probelauf hat gute Kenntnisse in Python und OpenCV und keine Erfahrung mit VHDL, Visual Applets und PyWizion.

Durch den Probelauf wurde festgestellt, dass die ursprünglich geplanten drei Aufgaben zu umfangreich sind und angepasst werden müssen. Es stellte sich heraus, dass der Probelauf mit der Bearbeitung einer Aufgabe 45 Minuten dauert. Dabei wurde ein großer Teil der Zeit für die Erklärung von Visual Applets benötigt. Da die potentiellen Proband:innen keine Erfahrung mit dieser Software haben und eine ausführliche Erklärung erfolgen muss, wird die Anzahl der finalen Aufgaben reduziert. Die finale Aufgabe basiert auf der Implementierung von zwei Filtern mit den folgenden Spezifikationen:

- *Gauss: 3x3 Filtergröße, $\text{SigmaX} = 0$, $\text{SigmaY} = 0$, Handhabung der Kanten = Kantenspiegelung*
- *Closing: 3x3 Filtergröße, Kernel 1 = kreuzförmiges Strukturelement, Kernel 2 = rechteckiges Strukturelement, Handhabung der Kanten = Kantenspiegelung*

Für die Bearbeitung dieser Aufgabe wurden insgesamt 17:45 Minuten benötigt. Diese Zeit setzt sich aus den Zeiten der Teilaufgaben zusammen. Um Teilnehmer:innen mit weniger Vorkenntnissen die Möglichkeit zu gewähren die Aufgaben zu lösen, wurde die Zeit auf 25 Minuten für beide Aufgabenteile angesetzt. Bis auf den Umfang der Aufgaben, gab es keine Probleme, die den Durchlauf beeinträchtigt haben. Die Implementierungsaufgabe

aus dem Probedurchlauf ist als die finale Aufgabenstellung für die Nutzerfreundlichkeitstests ausgewählt worden. Deswegen werden die Ergebnisse aus der Vorstudie mit in die Auswertung aufgenommen.

5.4 Hauptstudie

Die Studie ist mit sieben Proband:innen (sechs männlich und eine weiblich) mit den gleichen Voraussetzungen der Technik und Software durchgeführt worden. Der Test fand in Präsenz im Raum 2.20 der Basler AG im Standort Ahrensburg auf dem Computer des Testleiters statt, damit keine Probleme durch unterschiedliche Versionen der Software entstehen können. Während der gesamten Testreihe musste den Proband:innen die Auswahl der Software, mit welcher der Test gestartet wird, nicht vorgeschrieben werden, da drei Proband:innen mit Visual Applets und vier Proband:innen mit PyWizion anfangen. In der Tabelle 5.1 sind alle Proband:innen mit dem jeweiligen Beruf aufgelistet. Bei den Studierenden wird zusätzlich nach Programmiererfahrung unterteilt. Für die restlichen Proband:innen kann die Annahme getroffen werden, dass diese über ausreichend Programmiererfahrung im Allgemeinen verfügen.

Beruf	Anzahl
Studierende	2 (viel Programmiererfahrung)
	1 (wenig Programmiererfahrung)
Software Entwickler	2
Elektronik Entwickler	1
System Entwicklerin	1

Tabelle 5.1: Berufe der Proband:innen

In Abbildung 5.2 sind die Selbsteinschätzungen der Vorkenntnisse der Teilnehmer:innen in einer Kastengrafik dargestellt. Bis auf einen Teilnehmer haben alle gute Kenntnisse in Python und bis auf zwei ebenso grundlegende Erfahrungen mit OpenCV. Die Teilnehmer:innen haben bis auf eine Ausnahme sehr wenig bis keine Erfahrungen mit VHDL und ebenso bis auf eine Person keine Kenntnisse von PyWizion und Visual Applets.

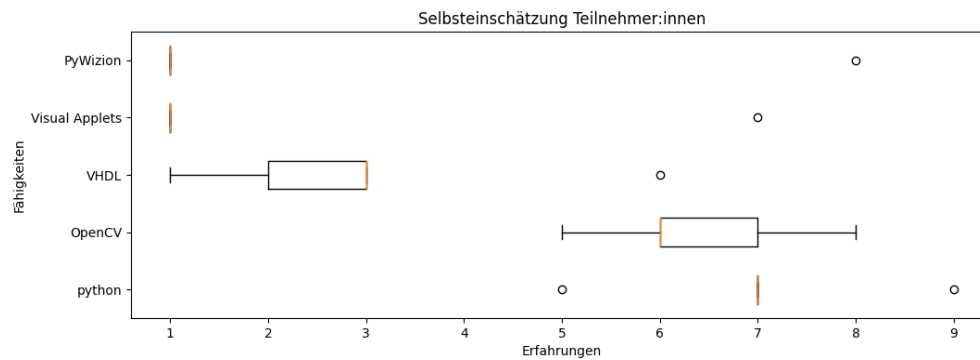


Abbildung 5.2: Kastengrafik: Vorkenntnisse der Proband:innen

5.5 Auswertung der Tests

Obwohl die Anzahl der Proband:innen mit der typischen Beteiligung an Einzeltests übereinstimmt [26], ist es dadurch nicht möglich allgemeingültige statistische Ergebnisse aus den Tests abzuleiten, weil die Stichprobe der Teilnehmer:innen zu gering ist. Dennoch kann aus den Resultaten eine Tendenz der Nutzerfreundlichkeit abgeleitet werden.

Auswertung der SUS Methode

Die Ergebnisse der Auswertung für die implementierten Bildverarbeitungsfunktionen in PyWizion ergeben eine gut Nutzerfreundlichkeit im Median von 77.5 und im Durchschnitt bei 74.6. Der Ausreißer von 47.5 in Abbildung 5.3 ist dem Studierenden mit wenig Programmiererfahrung zuzuordnen. Die restlichen Werte sind zwischen 70 und 85 einzuordnen und somit im akzeptablen Bereich auf der SUS Skala (5.1). Für Visual Applets ergibt die Auswertung einen Median von 40 und einen Durchschnitt von 44.3. Der höchste Wert liegt bei 60 und der niedrigste bei 20. Dieses Ergebnis liegt im nicht akzeptablen Bereich der Skala und ist ein Indikator für Probleme bei der Interaktion der Nutzer:innen mit der Software. [1] Alle relevanten Werte der Auswertung sind in Tabelle 5.2 aufgeführt.

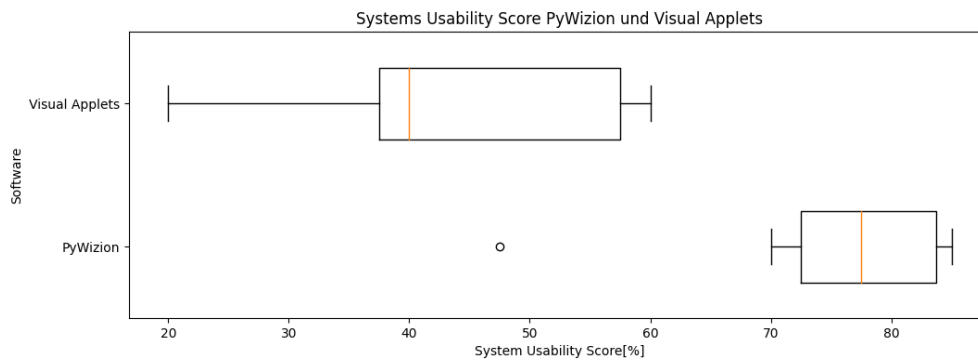


Abbildung 5.3: Kastengrafik System Usability Score mit der jeweiligen Software

Software	Minimum	Median	Durchschnitt	Maximum
Visual Applets	20	40	44.3	60
PyWizion	47.5	77.5	74.6	85

Tabelle 5.2: SUS Ergebnisse nach Software

Auswertung der Methode “Lautes Denken“

Die qualitativen Aussagen, die während der Tests von den Proband:innen unter der Anwendung der Methode “Lautes Denken“ getätigt wurden, werden gesammelt und separat nach PyWizion und Visual Applets aufgeführt. Zusätzlich werden generelle Probleme, die auftraten und für beides Umgebungen relevant sind, zusammengefasst. (Tabelle 5.3)

PyWizion

Während der Interaktion mit PyWizion haben drei Teilnehmer:innen gesagt, dass es durch die Entwicklung in der Programmiersprache Python der Einstieg sehr angenehm sei. Eine weitere Anmerkung, die von mehreren Personen positiv erwähnt wurde, ist die Ähnlichkeit zu der OpenCV-Bibliothek. Vereinzelt haben die Teilnehmer:innen die Dokumentation für PyWizion verwendet und äußerten unterschiedlicher Ansichten in Bezug auf den Umfang. Die sporadisch vorhandene Dokumentation, die bisher in dem Quellcode vorhanden ist, wurde von drei Teilnehmer:innen negativ und von zwei positiv erwähnt.

Bei der Implementierung des Gauss Filters haben zwei Proband:innen gesagt, dass sehr angenehm sei, keine Koeffizienten zu berechnen. Dazu muss erwähnt werden, dass diese Teilnehmer:innen den Test mit Visual Applets gestartet haben. Die Fehlerhandhabung

bzw. Fehlermeldung ist als aussagekräftig und äußerst positiv aufgefasst worden. Es wurde angemerkt, dass durch diese der Fehler eindeutig zu identifizieren sei und behoben werden konnte. Das gilt nur für die Teilnehmer:innen, die bei dem Ausführen des Python-Skriptes Fehlermeldungen erhielten.

Visual Applets

Bei der Implementierung in Visual Applets wurde von fünf Proband:innen erwähnt, dass die Aufgabe ohne die vorhandenen Beispiele nicht zu lösen sei. Das liege unter anderem an den Einstellungen der Operatoren und deren Verbindungen, die von vier Personen als sehr umständlich und unübersichtlich in Bezug auf das Erstellen einer Filtermatrix, beschrieben wurde. Dabei ist den Teilnehmer:innen unklar gewesen mit welchen Operatoren, die geforderten Parameter aus der Aufgabenstellung umzusetzen sind. Zwei weitere Probanden haben die Benennung der Operatoren als irreführend empfunden und benötigten die Dokumentation und ein Beispiel für das Verständnis. Drei Teilnehmer empfanden die grafische Programmierung als angenehm und übersichtlich.

Allgemeine Äußerungen

Drei Versuchsteilnehmer, in diesem Fall nur die Studierenden, äußerten die Bemerkung "Was ist eine morphologische Operation?" . Das ist auf die fehlende Erfahrung in der Bildverarbeitung zurückzuführen. Denn für die restlichen Proband:innen ist der Filter geläufig, lediglich bei der Reihenfolge der Basisoperatoren wurden Unsicherheiten geäußert.

Ein Teilnehmer, der sowohl Kenntnisse in PyWizion als auch in Visual Applets besitzt, empfand die Aufgabe als sehr einfach, merkte jedoch noch an, dass für Neueinsteiger das Schwierigkeitslevel in Ordnung sei.

Sehr auffällig war es, dass alle Teilnehmer:innen anfänglich sehr ausführlich beschrieben haben, was gerade gemacht wird. Je länger der Test dauerte und je unsicherer das Vorgehen wurde, desto weniger wurde mit dem Versuchsleiter kommuniziert. Dieser hat Fragen wie "Was gehst du gerade vor?" oder "Was denkst du im Moment?" gestellt, wenn bei der Bearbeitung zu lange nicht kommuniziert wurde. Nach dem Test hat es ebenfalls einen Informationsaustausch gegeben, in dem die Teilnehmer:innen weniger angespannt wirkten als während der Durchführung.

Software	Software-spezifische Anmerkungen	Allgemeine Anmerkungen
PyWizion	<ul style="list-style-type: none"> - einfacher Einstieg wegen der Syntax - ausreichende Dokumentation - mehr Dokumentation gewünscht - keine Berechnung der Koeffizienten - gute Fehlerbeschreibung - Morphologische Operationen sind identisch mit OpenCV 	<ul style="list-style-type: none"> - morphologische Operatoren nicht bekannt - Reihenfolge Closing - Aufgabe ist zu einfach
Visual Applets	<ul style="list-style-type: none"> - ohne Beispiele nicht lösbar - Einstellungen zu umständlich - Operatoren nicht intuitiv - grafische Programmierung angenehm 	

Tabelle 5.3: Zusammengefasste Kernaussagen aus der Methode “Lautes Denken“

Auswertung der Zeitmessungen

Wie es bei Nutzerfreundlichkeitstests gängig ist, werden für die Auswertung nur die Zeiten, welche bei erfolgreichem Abschluss einer Aufgabe, verwendet. Wurde die Bearbeitungszeit überschritten oder der Versuchsleiter intervenierte, so gilt die Aufgabe als abgebrochen. Zwei Teilnehmer:innen haben nicht aus der eigenen Entscheidung abgebrochen, sondern wurden von dem Testleiter hingewiesen, dass die Zeit in Kürze ablaufe und die aktuelle Vorgehensweise nicht korrekt sei und damit zu keinem auswertbaren Ergebnis führen würde. Aus diesen Gründen musste der Test bei diesen Teilnehmer:innen abgebrochen werden (Tabelle 5.4). Von den beiden Teilnehmer:innen hatte einer wenig Programmiererfahrungen und der andere besitzt geringerer Kenntnisse in Bildverarbeitung mit OpenCV als der Durchschnitt der restlichen Teilnehmer:innen.

Software	Minimum	Median	Durchschnitt	Maximum	Abgebrochen
Visual Applets	5:30 min	9:50 min	8:56 min	11:41 min	2
PyWizion	5:15 min	7:50 min	7:36 min	11:24 min	2

Tabelle 5.4: Benötigte Zeit für die erfolgreich abgeschlossene Aufgabe

In Abbildung 5.4 ist zu erkennen, dass bis auf einen Ausreißer, bei dem es sich um den Studierenden mit wenig Programmiererfahrung handelt, die benötigte Zeit für die

Bearbeitung in PyWizion kürzer ist als die in Visual Applets. Ein Grund dafür ist die Ähnlichkeit der Bildverarbeitungsfunktionen zu OpenCV. Je fundierter die Kenntnisse in OpenCV der Probanden:innen, desto leichter ist der Einstieg in PyWizion und kürzer die Zeit der Bearbeitung (Abbildung 5.5). Ein Grund für den größeren Zeitaufwand in Visual Applets sind die Parametereinstellungen der Filter, insbesondere die Berechnung der Filterkoeffizienten für die Implementierung des Gaussfilters.

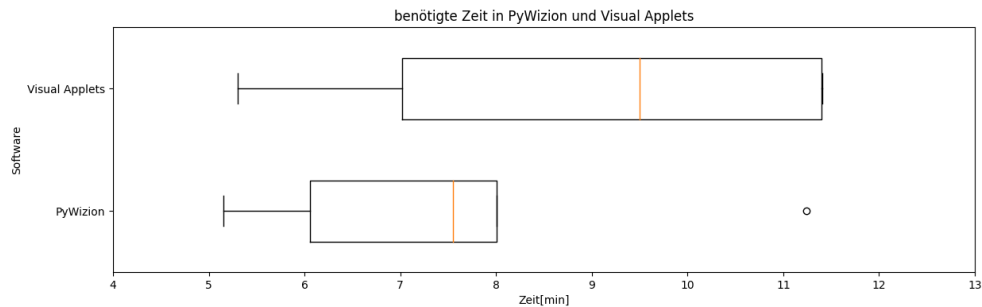


Abbildung 5.4: Kastengrafik: Benötigte Zeit für die erfolgreich abgeschlossene Aufgabe mit der jeweiligen Software

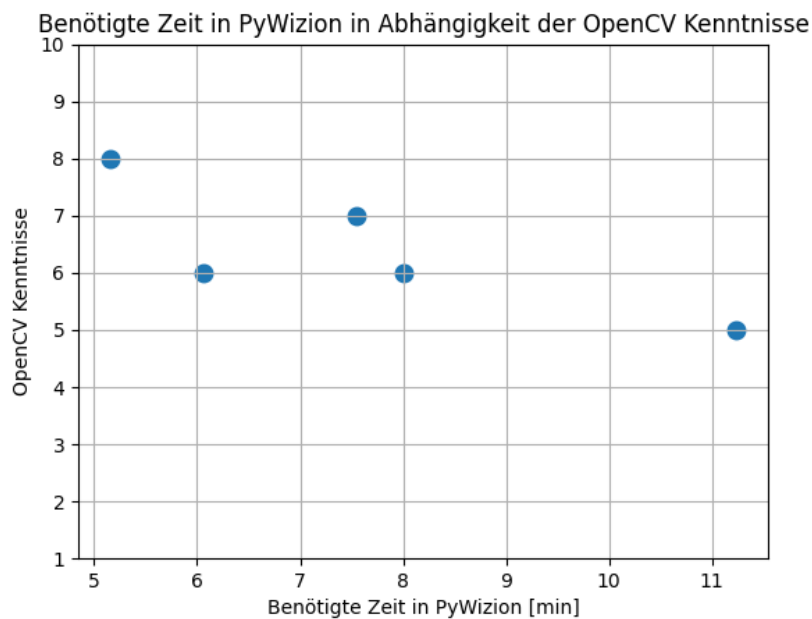


Abbildung 5.5: Zeit der erfolgreichen Tests in Abhängigkeit der Selbsteinschätzung in OpenCV

Von den fünf Teilnehmer:innen, welche die Aufgaben erfolgreich abgeschlossen haben, starteten vier mit PyWizion und einer mit Visual Applets. In Abbildung 5.6 sind die Zeiten in Abhängigkeit der Startreihenfolge der jeweiligen Software dargestellt. Es ist deutlich erkennbar, dass die Aufgabe in der zweite Software im Mittel nicht in einer kürzeren Zeit abgeschlossen wurde. Trotz bekannter Aufgabenstellung und eines potentiellen Lerneffektes haben die Teilnehmer:innen länger gebraucht.

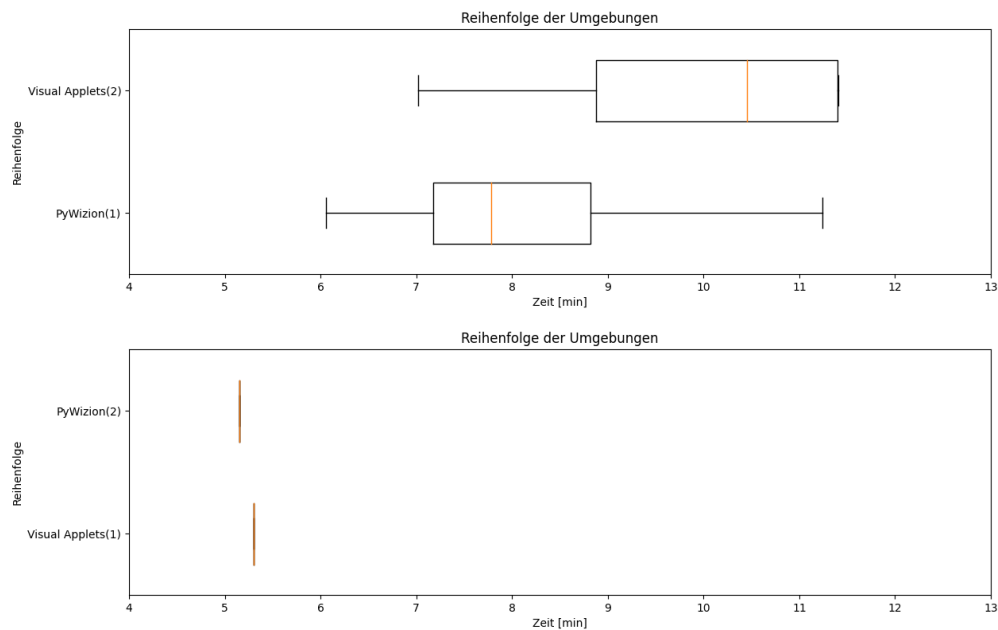


Abbildung 5.6: Kastengrafik: Benötigte Zeit in Abhängigkeit der Reihenfolge der Software

Bei der Proband:in, die mit Visual Applets angefangen hat, ist die benötigte Zeit in PyWizion geringer. Dieser Versuchsteilnehmer besitzt bereits Kenntnisse in Visual Applets und auch in PyWizion und absolvierte die Aufgaben schneller als die restlichen unerfahrenere Teilnehmer:innen. Es muss noch erwähnt werden, dass die Teilnehmer, bei denen der Versuch abgebrochen werden musste, beide den Test mit Visual Applets gestartet haben und aus diesem Grund nicht mit in die Statistik aufgenommen werden können. Es ist sehr gut zu erkennen, dass unerfahrenere Teilnehmer:innen deutlich schneller die Aufgabe in PyWizion lösen konnten als in Visual Applets. Das tendiert zu einer effizienteren Benutzung sowohl bei Entwickler:innen, die bereits Erfahrungen in dem Framework haben, als auch die, welche keine Kenntnisse besitzen.

Auswertung der Fehler

Für die Auswertung der aufgetretenen Fehler, während der Durchführung der Tests, werden diese in der Tabelle 5.5 dargestellt. Dabei wird zwischen kritischen und nicht-kritischen Fehlern unterschieden. Für zweiteres gibt es die Möglichkeit, diese mit Fehlermeldungen zu konkretisieren, wie z.B. bei einem Syntaxfehler. Kritische Fehler werden von dem System nicht erkannt, da die Syntax korrekt ist, jedoch ist die Funktionalität fehlerhaft. Sowohl in PyWizion als auch in Visual Applets sind bei der Implementierung des Gauss Filters keine Probleme aufgetreten. Bei dem nichtlinearen morphologischen Filter sind in beiden sowohl in PyWizion als auch Visual Applets ähnliche Fehler aufgetreten.

Nummer	Fehlerbeschreibung	Vorkommen	Software	Kritisch ?
1.	Datentypen Filterkernel Closings falsch	3	PW	-
2.	Schließen als linearen Filter	2	PW/VA	+
3.	Reihenfolge des Schließens	2	unabhängig	+

Tabelle 5.5: Auftretende Fehler während des Tests

In Visual Applets war die Auswahl der Operatoren von zwei Personen falsch. Diese implementierten das nichtlineare Closing, mit den Operatoren für lineare Filter. In PyWizion wurde ebenfalls die Funktion Filter2D, die für die adaptive Faltung konzipiert ist und damit lineare Eigenschaften aufweist, verwendet. Beide Fehler führen zu unerwünschten Ergebnissen und werden nicht mit Fehlererkennung der Software detektiert und sind somit als kritisch einzustufen.

Ebenfalls ist die Reihenfolge des Closings von zwei Teilnehmern vertauscht worden, somit wurde anstatt des Schließens ein Öffnen des Bildes implementiert. Für diesen Fall kann auch keine Fehlermeldung durch die Software ausgegeben werden, weil die Implementierung korrekt ist. Jedoch unterscheidet sich das Ergebnis und deswegen wird der Fehler ebenso als kritisch eingeordnet. Dieser Fehler ist nicht spezifisch einer der beiden Softwares zuzuordnen, sondern resultiert aus nicht vorhandenen Kenntnissen in der Bildverarbeitung.

Ein Fehler, der nur in PyWizion aufgetreten ist, ist die Wahl des Datentyps des Filterkerns für den Funktionsaufruf des Schließens. Da für diesen Fall eine Fehlermeldung beim Kompilieren ausgegeben wird und der Bau des Designs in Visual Applets nicht beginnt, ist der Fehler als nicht kritisch einzuordnen. Durch die Fehlermeldung konnte dieser von allen Teilnehmer:innen behoben werden.

5.6 Auswertung der Anforderungen der Nutzerfreundlichkeit

Abschließend werden die Auswertungen aus den Nutzerfreundlichkeitstests den Anforderungen an gute Nutzerfreundlichkeit gegenübergestellt und überprüft, ob diese erfüllt sind.

- *Erlernbarkeit:* Die Teilnehmer:innen, bis auf eine Teilnehmer:in, haben den Test ohne Kenntnisse in PyWizion erfolgreich abgeschlossen. Wegen der Ähnlichkeit der Syntax zu OpenCV konnten die Funktionen ohne große Schwierigkeiten genutzt werden (siehe Kapitel 5.5). Außerdem weisen die Beantwortungen der Fragen vier und zehn des SUS Fragebogens, im Durchschnitt 3.89 (Frage 4) und 2.00 (Frage zehn), auf gute Erlernbarkeit hin.

- *Effizienz:* Alle Zeiten in PyWizion sind im Vergleich zu denen in Visual Applets deutlich niedriger und weisen somit auf eine effizientere Arbeitsweise in dem Framework hin. Sowohl erfahrene als auch unerfahrene Entwickler:innen ist es möglich effizienter Aufgaben zu lösen

- *Behaltbarkeit:* Die Behaltbarkeit kann nicht ermittelt werden, weil es dafür einen erneuten Test in einem längeren zeitlichen Abstand geben muss. Das ist nicht möglich, sodass die Behaltbarkeit nicht bewertet werden kann.

- *Fehler:* Die Funktionen während der Tests fehlerfrei zu implementieren, ist nahezu unmöglich. Bei den Fehlern ist die Handhabung und die Schwere der Fehler entscheidend. Die kritischen Fehler wurden vor allem in Visual Applets gemacht, was darauf hindeutet, dass die potentielle Fehlerrate in PyWizion geringer ist.

- *Zufriedenheit:* Das Ergebnis der SUS Fragebögen zeigt eindeutig, dass PyWizion als nutzerfreundlicher und angenehmer beschrieben wird als Visual Applets. Der deutliche Unterschied im Median von PyWizion (77.5) und Visual Applets (40) gibt diesen wieder.

Bis auf die Behaltbarkeit sind alle Kriterien für Nutzerfreundlichkeit erfüllt. Daraus kann abgeleitet werden, dass durch die Implementierung der Bildverarbeitungsfunktionen die Nutzerfreundlichkeit in PyWizion im Vergleich zu in Visual Applets signifikant höher ist.

6 Schlussbetrachtung

In diesem Kapitel wird überprüft, ob die festgelegten Anforderungen aus dem Kapitel 3 an die Bildverarbeitungsfunktionen, erfüllt sind. Außerdem werden die Vorgehensweise und Ergebnisse der Arbeit kritisch hinterfragt. Abschließend wird es einen Ausblick geben wie die diese Arbeit fortgeführt werden kann.

6.1 Fazit

Die implementierten Bildverarbeitungsfunktionen sind umfassend auf die gestellten Anforderungen geprüft und bewertet worden. In Bezug auf Auswahl der geeigneten Visual Applets Operatoren in den implementierten Bildverarbeitungsfunktionen wird unter anderem darauf geachtet, dass die gefilterten Bilder vergleichbar mit anderen Frameworks bleiben. Außerdem werden für die Filterfunktionen die geeigneten Operatoren aus der Visual Applets Bibliothek verwendet. Damit gilt diese Anforderung als erfüllt.

Des Weiteren wird durch geeignete Platzierung von M-Typ Operatoren dafür gesorgt, dass keine Synchronisierungsprobleme innerhalb der Filtervorgänge entstehen können. Die funktionale Anforderung ist somit erfüllt.

Das Framework ermöglicht die Verarbeitung sowohl von Graustufen- als auch Farbbildern. Dabei wird automatisch anhand des Datenstroms am Eingang des Filters erkannt um welchen Typen es sich handelt woraufhin entsprechend weiter verfahren wird. Aufgrund dieser Funktionalität können beide Bildtypen ohne umfangreiche Einstellungen verarbeitet werden, was den Anforderungen entspricht.

Die funktionale Anforderung der korrekten Filterung mit verschiedenen Parametern und unterschiedlichen Kernelgrößen werden bei den meisten Filtern erfüllt. Dafür werden die Ergebnisse aus den Tabellen 4.5 und 4.6 verwendet. Nur für den Gaussfilter und für eine spezifische Parameterkonstellation des Bilateralfilters sind die Anforderungen nicht hundertprozentig erfüllt. Bei dem Gaussfilter sind weitere Verbesserungen erforderlich, im Gegensatz zu dem Bilateralfilter, bei dem das Ergebnis innerhalb des Toleranzbereiches.

Die nicht funktionale Anforderung die gängigsten Bildverarbeitungsfunktionen auszuwählen, werden durch die Umfrage und Literaturrecherchen validiert. Damit wird gewährleistet, dass die implementierten Bildverarbeitungsfunktionen zu den Funktionen, die häufig verwendet werden, zählen.

Bei der Implementierung ist sehr darauf geachtet worden die Funktionen ressourcenarm zu implementieren. Dazu zählen die Normalisierung der Filtermatrix und die Positionierung des **FIRkernelNxM** bei Farbbildern. Kompromisse müssen bei der Auswahl der Operatoren für separierbare Filter gemacht werden. Der wissenschaftliche Vergleich zwischen den implementierten Funktionen mit den Funktionen eines anderen Frameworks wurde stärker gewichtet als die ressourcenarme separierbare Implementierung. Deswegen gilt diese Anforderung auch als erfüllt.

Der geforderte einfache Einstieg in PyWizion ist durch die Ähnlichkeit der Syntax zu OpenCV gelungen. Durch die Tests der Nutzerfreundlichkeit wird gezeigt, dass Entwickler:innen, die bereits Erfahrungen mit OpenCV haben, die Bildverarbeitungsfunktionen sehr gut anwenden können.

Die Kriterien für gute Nutzerfreundlichkeit (Kapitel 5.6) sind bis auf die Behaltbarkeit, welche durch den Test nicht ermittelt werden konnte, erfüllt worden und weisen auf gute Handhabung der Software hin. Außerdem zeigen die Tests der Nutzerfreundlichkeit, dass durch den erhöhten Abstraktionsgrad grundlegende Bildverarbeitungsfunktionen ohne Erfahrungen in digitaler Schaltungstechnik und grundlegender Visual Applets Kenntnisse erfolgreich verwendet werden können. Das Ziel der Arbeit das Spektrum an potentiellen Entwickler:innen, durch Erhöhung der Abstraktionsebene, zu vergrößern und die Funktionen ressourcenarm zu implementieren, kann damit erreicht werden.

6.2 Diskussion

Im folgenden Unterkapitel wird die Vorgehensweise und die Ergebnisse dieser Arbeit kurz diskutiert. Dabei wird sowohl auf Aspekte, die positiv als auch negativ zu bewerten sind eingegangen.

Die Durchführung der Nutzerfreundlichkeitstests wäre mit einer größeren Anzahl von Teilnehmer:innen von Vorteil gewesen, weil eine umfangreichere Gruppe von Teilnehmer:innen umfassendere und aussagekräftigere Ergebnisse geliefert hätte. Das wäre ein Aspekt, der bei erneuter Durchführung dieser Arbeit beachtet werden müsste. Trotzdem sind die aktuellen Ergebnisse positiv zu bewerten, da die messbaren Kriterien für Nutzfreundlichkeit erfüllt werden und dies auf eine gute Handhabung der Funktionen in

PyWizion deutet.

Um noch weniger Ressourcen auf dem FPGA zu verwenden, müssten die Separierbarkeit der Filter ausgenutzt werden. In dieser Arbeit wurde ein Filter, aus genannten Gründen, mit diesem Vorgehen implementiert. Dieses Konzept wäre bei erneuter Bearbeitung auf die separierbaren Filter anzuwenden, um noch effizienter zu werden.

Ein wichtiger Erfolgsfaktor für die Ergebnisse der Arbeit ist die Verifikation der Filterfunktionen. Diese Resultate entsprechen bis auf eine Ausnahme den Erwartungen. Es verdeutlicht, dass die Vorgehensweise bei der Implementierung durch den modularen Ansatz der Filterklasse und die mehrfache Verwendung von einzelnen Methoden eine geringe Fehlerrate der Filterfunktionen aufweist. Des Weiteren wird durch die ausgiebigen Tests sichergestellt, dass die Funktionen mit einer Vielzahl von Eingabeparametern korrekt funktionieren und verwendet werden können. Ebenfalls positiv zu erwähnen ist die große Anzahl der implementierten Funktionen. Mit dieser Grundlage ist es möglich Anwendungen, die für die Bildvorverarbeitung relevant sind, in PyWizion umzusetzen.

6.3 Ausblick

Für zukünftige Arbeiten bzw. um diese Arbeit fortzuführen kann sich mit der weiteren Optimierung der Ergebnisse dieser Arbeit, vor allem bei dem Gaussfilter, beschäftigen werden. Dabei könnten weiter Untersuchungen und Verbesserungen erforderlich sein, um die Ergebnisse, die mit der jetzigen Implementierung des Gaussfilters erreicht wurden, zu verbessern. Des Weiteren könnte das Konzept der Separierbarkeit von Filtern aufgegriffen werden, um noch ressourcenschonender zu implementieren. Es ist außerdem möglich weitere Bildvorverarbeitungsfunktionen, die ebenfalls bei der firmeninternen Umfrage mehrfach genannt wurden wie z.B. Grauwerttransformationen oder verschiedene Schwellwertverfahren, in das Framework einzupflegen, um das Spektrum an umsetzbaren Anwendungen zu erweitern. Zusätzlich könnte das bisher nicht testbare Kriterium der Nutzerfreundlichkeit an den gleichen Nutzer:innen getestet werden.

Literaturverzeichnis

- [1] BANGOR, Aaron ; KORTUM, Philip T. ; MILLER, James T.: An Empirical Evaluation of the System Usability Scale. In: International Journal of Human-Computer Interaction 24 (2008), Nr. 6, S. 574–594
- [2] BASLER AG: CXP 12 1C Spezifikationen. – URL <https://www.baslerweb.com/de/produkte/bildeinzugskarten/cxp-12-interface-cards/basler-cxp-12-interface-card-1c/>. – Zugriffsdatum: 25.05.2023
- [3] BASLER AG: Visual Applets Erode. – URL <https://docs.baslerweb.com/visualapplets/files/manuals/content/Filter.ERODE.html>. – Zugriffsdatum: 22.05.2023
- [4] BASLER AG: Visual Applets FIRkernelNxM. – URL <https://docs.baslerweb.com/visualapplets/files/manuals/content/Filter.FIRkernelNxM.html>. – Zugriffsdatum: 01.05.2023
- [5] BASLER AG: Visual Applets FPGA Ressourcen Schätzung. – URL https://docs.baslerweb.com/visualapplets/files/manuals/content/fpga_resource_estimation.html. – Zugriffsdatum: 25.04.2023
- [6] BASLER AG: Visual Applets Introduction. – URL <https://docs.baslerweb.com/visualapplets/files/manuals/content/introduction.html>. – Zugriffsdatum: 22.03.2023
- [7] BASLER AG: Visual Applets LineNeighboursNx1. – URL <https://docs.baslerweb.com/visualapplets/files/manuals/content/Filter.LineNeighboursNx1.html>. – Zugriffsdatum: 01.05.2023
- [8] BASLER AG: Visual Applets LUT. – URL <https://docs.baslerweb.com/visualapplets/files/manuals/content/Memory.LUT.html>. – Zugriffsdatum: 14.05.2023

- [9] BASLER AG: Visual Applets Operator Referenz. – URL https://docs.baslerweb.com/visualapplets/files/manuals/content/operator_documentations.html. – Zugriffsdatum: 03.05.2023
- [10] BASLER AG: Visual Applets Operator Types. – URL https://docs.baslerweb.com/visualapplets/files/manuals/content/fundamental_functionalities.html. – Zugriffsdatum: 21.04.2023
- [11] BASLER AG: Visual Applets PixelNeighbours1xM. – URL <https://docs.baslerweb.com/visualapplets/files/manuals/content/Filter.PixelNeighbours1xM.html>. – Zugriffsdatum: 01.05.2023
- [12] BASLER AG: Visual Applets RAMLUT. – URL <https://docs.baslerweb.com/visualapplets/files/manuals/content/Memory.mE6RamLUT.html>. – Zugriffsdatum: 14.05.2023
- [13] BASLER AG: Visual Applets Version 3. – URL https://www.baslerweb.com/de/produkte/visualapplets/#tabs__visualapplets#tab=version3. – Zugriffsdatum: 17.04.2023
- [14] BEVAN, Nigel: Measuring usability as quality of use. In: Software Quality Journal 4 (1995), Nr. 2, S. 115–130. – URL <http://link.springer.com/content/pdf/10.1007%2FBF00402715.pdf>
- [15] BHASIN, Harsh: Python basics: A self-teaching introduction. Dulles Virginia : Mercury Learning and Information, 2019. – 263 S. – ISBN 9781683923534
- [16] BREDIES, Kristian ; LORENZ, Dirk (Hrsg.): Mathematische Bildverarbeitung Einführung in Grundlagen und moderne Theorie. Vieweg+Teubner Verlag / Springer Fachmedien Wiesbaden GmbH, Wiesbaden, 2011. – 82–90 S
- [17] BROOKE, John: SUS: A quick and dirty usability scale. In: Usability Eval. Ind. 189 (1995), 11
- [18] CHRISTIAN DEMANT, BERND STREICHER-ABEL, AXEL SPRINGHOFF: Industrielle Bildverarbeitung Wie optische Qualitätskontrolle wirklich funktioniert, 3. Auflage. (2011), S. 0–56
- [19] DANG-NGUYEN, Duc-Tien ; PASQUINI, Cecilia ; CONOTTER, Valentina ; BOATO, Giulia: RAISE: a raw images dataset for digital image forensics. In: OOI, Wei T. (Hrsg.) ; FENG, Wu chi (Hrsg.) ; LIU, Feng (Hrsg.): Proceedings of the 6th ACM Multimedia Systems Conference, MMSys 2015, Portland, OR, USA, March 18-20,

- 2015, ACM, 2015, S. 219–224. – URL <http://doi.acm.org/10.1145/2713168.2713194>. – ISBN 978-1-4503-3351-1
- [20] DIN EN ISO 9241-11: DIN EN ISO 9241-11:2018-11, Ergonomie der Mensch-System-Interaktion - Teil 11: Gebrauchstauglichkeit: Begriffe und Konzepte (ISO 9241-11:2018); Deutsche Fassung EN ISO 9241-11:2018. 11 2018. – URL <https://doi.org/10.31030%2F2757945>
- [21] ERHARDT, Angelika: Einführung in die digitale Bildverarbeitung : Grundlagen, Systeme und Anwendungen ; mit 35 Beispielen und 44 Aufgaben. Wiesbaden : Vieweg + Teubner, 2008. – 1–3 S. – ISBN 9783519004783 351900478X 9783834895189 3834895180
- [22] GONZALEZ, Rafael C. ; WOODS, Richard E.: Digital image processing. Prentice Hall, 2008. – 172,213–235 S. – ISBN 9780131687288 013168728X 9780135052679 013505267X
- [23] HÖFLE, Kevin: PyWizion Dokumentation. 2023
- [24] JÄHNE, Bernd: Digitale Bildverarbeitung. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012. – 13–14,333–334 S. – ISBN 978-3-642-04951-4
- [25] JAKOB NIELSEN: Usability Engineering. (1993), S. 26–37
- [26] JEFFREY RUBIN, Jared S.: Handbook of Usability Testing: Howto Plan, Design, and Conduct Effective Tests. 2. Wiley, 2008. – 201 S. – ISBN 9780470185483,0470185481
- [27] LEWIS, James R.: Measuring Perceived Usability: The CSUQ, SUS, and UMUX. In: International Journal of Human–Computer Interaction (2018), Nr. 12, S. 1148–1156
- [28] NISCHWITZ, Alfred ; FISCHER, Max ; HABERÄCKER, Peter ; SOCHER, Gudrun: Studium. Bd. Bd. 2: Bildverarbeitung. 3., neu bearb. Aufl. Wiesbaden : Vieweg + Teubner, 2011. – 25,251–255 S. – ISBN 9783834817129
- [29] OPEN SOURCE COMPUTER VISION: Image Filtering. – URL https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html. – Zugriffsdatum: 25.05.2023
- [30] QUIRMBACH, Sonja M.: Evaluationsmethoden im Überblick. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013. – 105–126 S. – URL https://doi.org/10.1007/978-3-642-20778-5_11. – ISBN 978-3-642-20778-5

- [31] SONKA, M. ; HLAVAC, V. ; BOYLE, R.: Image Processing, Analysis, and Machine Vision. Cengage Learning, 2014. – 116–117 S. – ISBN 9781285981444
- [32] SÜSSE, Herbert ; RODNER, Erik: Bildverarbeitung und Objekterkennung. Wiesbaden : Springer Fachmedien Wiesbaden, 2014. – 52 S. – ISBN 978-3-8348-2605-3
- [33] VILLAN, Alberto F.: Mastering OpenCV 4 with Python: A practical guide covering topics from image processing, augmented reality to deep learning with OpenCV 4 and Python 3.7. 1. Packt Publishing, 2019. – 15,19–20 S. – ISBN 1789344913; 9781789344912
- [34] VOGEL, Oliver ; ARNOLD, Ingo ; CHUGHTAI, Arif ; IHLER, Edmund ; KEHRER, Timo ; MEHLIG, Uwe ; ZDUN, Uwe: Software-Architektur: Grundlagen – Konzepte – Praxis. 2. Spektrum, 2009. – 107–109 S. – ISBN 978-3-8274-1933-0
- [35] WILHELM BURGER, MARK JAMES BURGE: Digitale Bildverarbeitung: Eine algorithmische Einführung mit Java. 3. Springer Vieweg, 2015. – 93–119 S. – ISBN 978-3-642-04603-2

A Anhang

A.1 Bildverarbeitungsfunktionen

A.1.1 Fragebogen firmeninterne Umfrage

Umfrage Bachelorarbeit Florian Wysk, gängige Funktionen in der Bildverarbeitung

Das Ausfüllen der Umfrage dauert ungefähr 5 Minuten.

1. Wie alt bist du in Jahren?

2. Wie viele Jahre arbeitest du bereits in dem Themengebiet Bildverarbeitung ?

3. Wie oft kamen Kunden mit dem Wunsch auf dich zu Vorverarbeitung auf der Kamera auszulagern?

Falls du keinen Kundenkontakt hast, wie oft kam dieser Wunsch bei deinen eigenen Projekten vor?

- >10
 5 - 10
 1 - 4
 0

4. Welche Algorithmen/Funktionen sind dabei besonders interessant oder wurden öfters bemerkt?

5. Warum genau diese Algorithmen/Funktionen?

6. Welche Funktion, auch in Hinsicht auf Frameworks wie OpenCV, verwendest du oder Kunden am häufigsten?

7. Gibt es eine Funktion, ohne die du deine Arbeit nicht erledigen könntest?

8. Sonstige Anmerkungen

Abbildung A.1: Fragebogen Umfrage zu den gängigsten Bildverarbeitungsfunktionen

A.1.2 Aktivitätsdiagramme

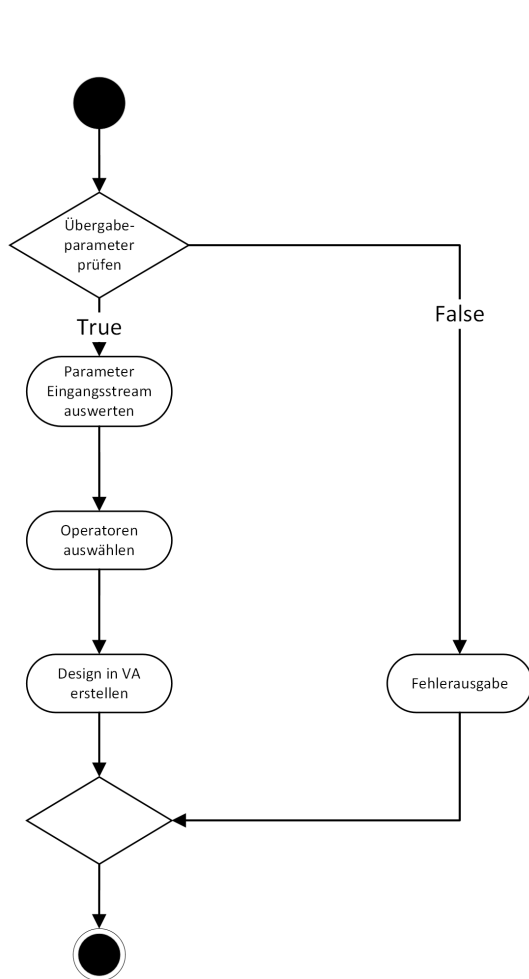


Abbildung A.2: Aktivitätsdiagramm Debayering

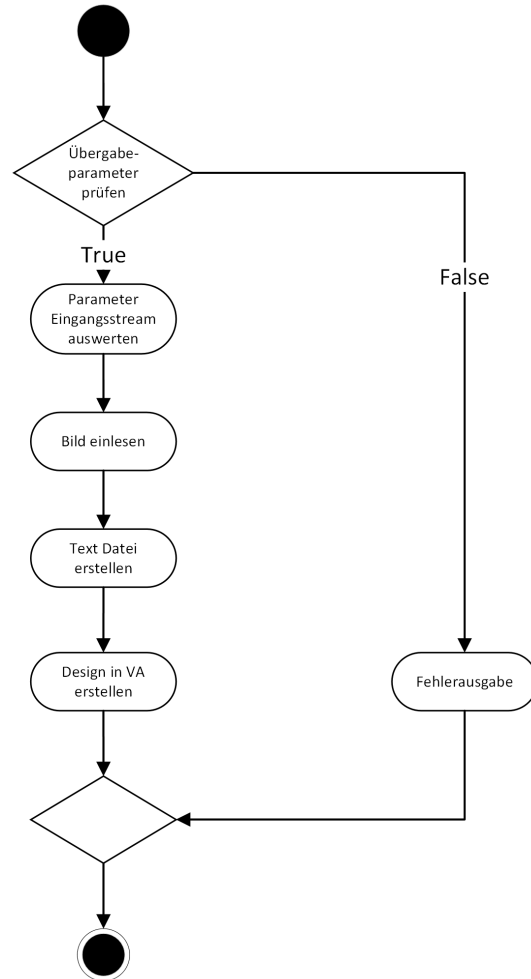


Abbildung A.3: Aktivitätsdiagramm Imread

A.1.3 Ergebnisse der Differenzbilder

Durchmesser	Parameter		Median abweichende Pixel [%]	Median Differenz [D.N.]	Differenz maximal [D.N.]	Durchschnitt Differenz [D.N.]	Durchschnitt abweichende Pixel [%]
	Sigma Color	Sigma Space					
3	0.5	0.5	18.58	1.00	1.00	1.00	19.01
		50	10.82	1.00	1.00	1.00	11.14
		150	14.36	1.00	1.00	1.00	15.62
	50	0.5	6.55	1.50	2.00	1.50	8.42
		50	2.13	1.00	1.00	1.00	2.37
		150	4.14	1.00	1.00	1.00	6.10
	150	0.5	5.19	1.00	3.00	1.28	7.01
		50	4.38	1.00	1.00	1.00	4.51
		150	6.75	1.00	1.00	1.00	8.56
5	0.5	0.5	19.31	1.00	1.00	1.00	19.38
		50	9.44	1.00	1.00	1.00	9.47
		150	10.08	1.00	1.00	1.00	10.07
	50	0.5	5.17	2.00	3.00	1.92	5.64
		50	1.46	1.00	4.00	1.68	1.46
		150	1.01	1.00	2.00	1.18	1.12
	150	0.5	4.85	2.00	4.00	2.18	5.31
		50	1.41	1.00	1.00	1.00	1.50
		150	1.90	1.00	1.00	1.00	1.95
7	0.5	0.5	21.05	1.00	1.00	1.00	21.36
		50	38.83	1.00	1.00	1.00	38.15
		150	35.33	1.00	1.00	1.00	35.28
	50	0.5	49.24	2.00	3.00	2.38	47.93
		50	49.90	4.00	12.00	4.54	48.79
		150	49.54	1.00	4.00	1.58	48.56
	150	0.5	49.27	2.00	4.00	2.46	47.92
		50	49.85	1.00	2.00	1.06	48.73
		150	49.21	1.00	1.00	1.00	48.23

Tabelle A.1: Bilateral Filter Abweichungen natürliche Graustufenbilder

ksize	Parameter		Median	Median	Differenz	Durchschnitt	Durchschnitt
	sigmaX	sigmaY	abweichende Pixel [%]	Differenz [D.N.]	maximal [D.N.]	Differenz [D.N.]	abweichende Pixel [%]
	2.00	0.00	44.67	1.00	106.00	3.10	44.48
		0.50	73.41	93.00	138.00	91.90	73.41
		1.00	65.29	34.00	59.00	34.62	65.56
		1.50	62.48	11.00	17.00	10.98	62.85
		2.00	44.67	1.00	1.00	1.00	44.01
7	0.00	0.00	56.49	1.00	2.00	1.02	57.00
		0.50	44.53	1.00	1.00	1.00	43.56
		1.00	45.40	1.00	2.00	1.02	44.52
		1.50	48.63	1.00	2.00	1.36	47.82
		2.00	56.44	2.00	2.00	1.52	56.94
	0.50	0.00	43.76	1.00	2.00	1.14	42.87
		0.50	44.06	1.00	2.00	1.14	42.92
		1.00	46.45	1.00	2.00	1.06	45.36
		1.50	43.10	1.00	2.00	1.36	42.32
		2.00	46.74	2.00	2.00	1.58	45.95
	1.00	0.00	56.48	1.00	76.00	2.50	57.75
		0.50	69.05	66.00	96.00	64.80	69.48
		1.00	56.43	1.00	2.00	1.02	57.13
		1.50	45.35	1.00	2.00	1.22	44.43
		2.00	43.94	1.00	2.00	1.24	43.18
	1.50	0.00	46.15	2.00	97.00	3.46	45.97
		0.50	72.47	88.50	126.00	85.98	72.94
		1.00	62.99	30.00	45.00	29.78	63.55
		1.50	46.11	2.00	2.00	1.56	45.23
		2.00	43.98	2.00	2.00	1.76	43.39
2.00	0.00	51.62	2.00	108.00	3.82	53.31	
	0.50	76.64	97.00	141.00	96.52	75.61	
	1.00	67.94	47.00	70.00	47.06	68.10	
	1.50	57.31	18.00	25.00	17.36	58.17	
	2.00	51.61	2.00	2.00	1.69	52.69	

Abbildung A.4: Ausschnitt der Ergebnistabelle des Gaussfilters für natürliche Graustufenbilder

A.2 Nutzerfreundlichkeit

A.2.1 Testaufbau in Visual Applets

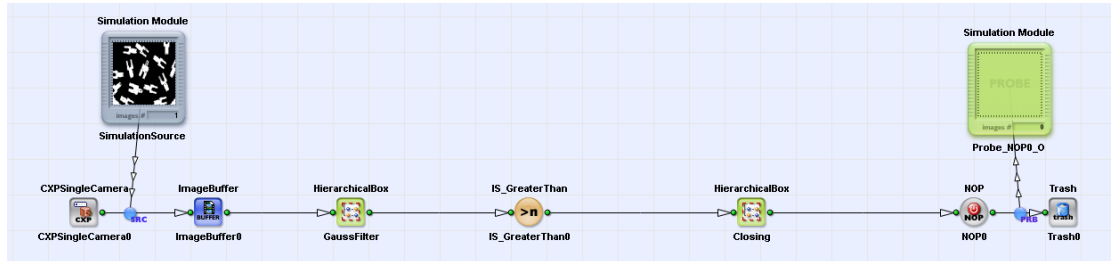


Abbildung A.5: Testaufbau in Visual Applets komplettes Design

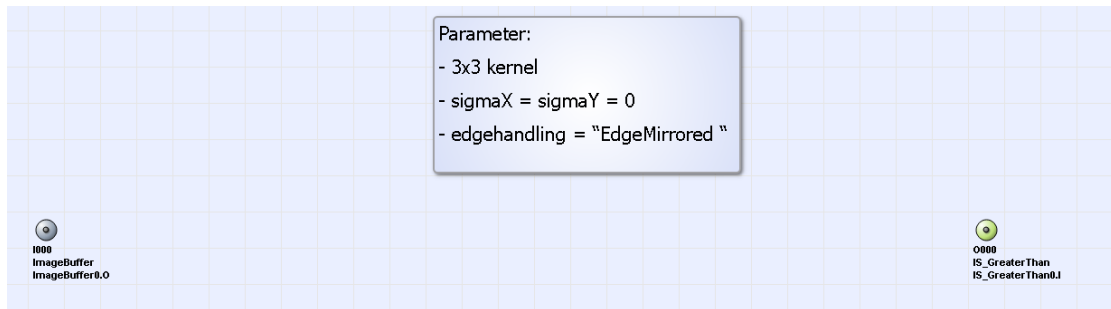


Abbildung A.6: Testaufbau in Visual Applets Gauss Filter

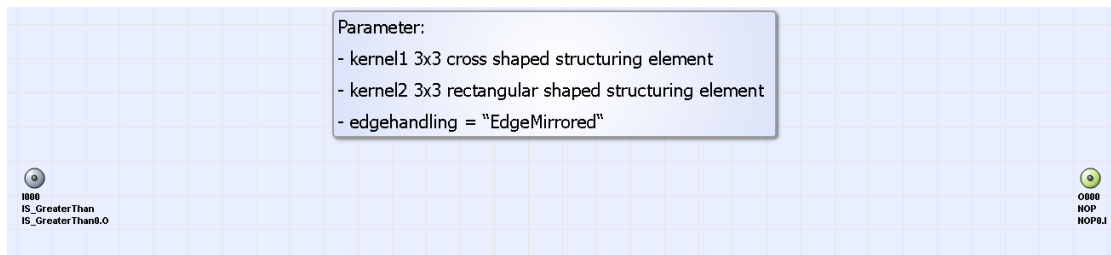


Abbildung A.7: Testaufbau in Visual Applets Closing

A.2.2 Testaufbau in PyWizion

```
1 from pywizion import DesignManager
2 from pywizion.Helpers.FilterHelper import Filter
3 import argparse
4 import os
5 import posixpath
6 import cv2
7 import numpy as np
8
9
10 if __name__ == '__main__':
11
12     # argparse block for calling from commandline
13     parser = argparse.ArgumentParser(description="Creates design of this file in VA")
14     parser.add_argument('--port', default=1234)
15     args = vars(parser.parse_args())
16     preferred_port = int(args['port'])
17
18     # PyWizion block for developing a design
19     with DesignManager.PyWizionWizard(platform="mE5-MA-VCX-QP",
20                                     preferred_port=preferred_port,
21                                     manual_mode=False,
22                                     name="UsabilityTest",
23                                     close_va=False) as pw:
24
25         # object for filters
26         filter = Filter(pw)
27
28         # camera input, 8bit bayer image
29         x = pw.CXPSingleCamera(parallelism=8, bit_width=8)()
30         x = pw.ImageBuffer()(x)
31
32         # ----- DO NOT CHANGE ANYTHING ABOVE! -----
33
34         # reduce noise by using a gaussian filter
35         # parameter: kernel size = 3x3, sigma_x = 0, sigma_y = 0, edgehandling = "EdgeMirrored"
36         # name of the HBox is GaussianBlur3x3
37
38         # binarisation of the image
39         x = x > 128
40
41         # Closing with morphEx
42         # Parameter: op = Closing, kernel_1 = 3x3 cross shape,
43         # kernel_2 = 3x3 rectangular shape, edgehandling = "EdgeMirrored"
44         # name of the HBox is Closing
45
46         # ----- DO NOT CHANGE ANYTHING BELOW! -----
47
48         # no need to save the output image
49         x = pw.NOP()(x)
50         x.enable_debug()
51         x = pw.Trash()(x)
52
53         # path of the simulation image
54         image_abs_path = os.path.abspath(
55             "C:/Users/Florian.Wysk/Bilder/plugs_1024x1024x8.tif")
56         simulation_image = posixpath.join(*image_abs_path.split('\\'))
57
58         # simulate design
59         pw.simulate_design(path_images=simulation_image,
60                           simulation_cycles=1)
61
62         print("success!")
```

Listing A.1: Testaufbau in PyWizion

Nutzerfreundlichkeitstest Test PyWizion

Stimme überhaupt nicht zu ☆
Stimme nicht zu ☆☆
Stimme weder zu noch lehne ich ab ☆☆☆
Stimme zu ☆☆☆☆
Stimme voll und ganz zu ☆☆☆☆☆

Das Ausfüllen der Umfrage dauert ungefähr 3 Minuten.

1. Ich kann mir sehr gut vorstellen, PyWizion regelmäßig zu nutzen.

☆☆☆☆☆

2. Ich empfinde die allgemeine Anwendung von PyWizion als unnötig komplex.

☆☆☆☆☆

3. Ich empfinde die Funktionen von PyWizion als einfach zu nutzen.

☆☆☆☆☆

4. Ich denke, dass ich technischen Support(z.B. Dokumentation) brauchen würde, um die Funktionen von PyWizion zu nutzen.

☆☆☆☆☆

Abbildung A.8: SUS Fragebogen PyWizion Teil 1

5. Ich finde, dass die verschiedenen Funktionen in PyWizion gut integriert sind. (Bsp. Namesgebung der Methoden, Fehlerhandling)

☆ ☆ ☆ ☆ ☆

6. Ich finde, dass es in die Funktionen von PyWizion zu viele Inkonsistenzen gibt.

☆ ☆ ☆ ☆ ☆

7. Ich kann mir vorstellen, dass die meisten Leute PyWizion schnell zu beherrschen lernen.

☆ ☆ ☆ ☆ ☆

8. Ich empfinde die Bedienung als sehr umständlich.

☆ ☆ ☆ ☆ ☆

9. Ich habe mich bei der Nutzung der Funktionen sehr sicher gefühlt.

☆ ☆ ☆ ☆ ☆

10. Ich musste eine Menge Dinge lernen, bevor ich mit den Funktionen von PyWizion arbeiten konnte.

☆ ☆ ☆ ☆ ☆

Abbildung A.9: SUS Fragebogen PyWizion Teil 2

A.2.3 Anlagenverzeichnis

