Hochschule Bochum
Bochum University
of Applied Sciences

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

Entwicklung, Evaluierung und echtzeitfähige Implementierung eines Korrelationsalgorithmus zur optischen Längenmessung mit geringem Messfehler

Autor: Felix Stefan Schneider
felix.schneider@stud.hs-bochum.de
Matrikelnummer: 015214561

Erstprüfer: Prof. Dr.-Ing. Arno Bergmann

Zweitprüfer: Prof. Dr.-Ing. Burkhard Bock

Datum: 04.06.2021

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Masterarbeit:
»Entwicklung, Evaluierung und echtzeitfähige Implementierung eines Korrelationsalgorithmus zur optischen Längenmessung mit geringem Messfehler«

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Unterschrift : *Ort, Datum* :

Inhaltsverzeichnis

Abkürzungsverzeichnis

AXI	Advanced Extensible Interface
CCD	Charge-Coupled Device
CMOS	Complementary Metal Oxide Semiconductor (Transistor)
CPU	Central Processing Unit
DDR	Double Data Rate
DFT	Discrete Fourier Transform
DSP	Digitaler Signalprozessor
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
FPU	Floating Point Unit
GPU	Graphics Processing Unit
IC	Integrated Circuit
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
KKF	Kreuzkorrelationsfunktion
LDV	Laser-Doppler-Velozimetrie
LTI	Linear Time Invariant

LVDS Low Voltage Differential Signaling

MpSoC Multi Processor System-On-Chip

OMP OpenMP, Open Multi-Processing

RAM Random-Access Memory

RIPS Reduced Information Preselection

RMS Root-Mean-Square

RTOS Real Time Operating System

SATA Serial Advanced Technology Attachment

SFTP Secured File Transfer Protocol

SFV Spatial Filter Velocimetry

SIMD Single Instruction Multiple Data

SIPS Scaled Integer Preselection

SoC System-on-Chip

SP Single Precision (Floating Point)

SRIPSL Sign-Reduced Information Preselection

SSD Solid State Drive

SSH Secure Shell

USB Universal Serial Bus

XRIPSL Extremely Reduced Information Preselection

Symbolverzeichnis

Symbol	Bedeutung
C_{cam}	Kamerakonstante
d	Distanz
f	Frequenz
F	Messabweichung
F_{rel}	relative Messabweichung
μ	Ortsfrequenz
N	Periode eines Signals
s	Strecke
τ	Zeitverschiebung
Δt	Zeitdifferenz, Zeitverschiebung
T_{s}	Abtastperiode, Frameperiode
v	(Momentan-)Geschwindigkeit
$\bar{v}$	Durchschnittsgeschwindigkeit
$v_{\parallel}$	Längsgeschwindigkeit
$v_{\perp}$	Quergeschwindigkeit
W_{px}	Abstand zwischen Pixeln
Δx	Ortsverschiebung
x_{r}	richtiger Wert
x_{w}	wahrer Wert
$g(t) \circ\!\!-\!\!\bullet \mathcal{F}\{g(t)\}$	Fouriertransformation
$*$	Faltungsoperator
$\times$	Kreuzkorrelationsoperator
$\otimes$	Zyklischer Kreuzkorrelationsoperator
rd()	Kaufmännische Rundung

1 Einleitung

Am Institut für Systemtechnik wurden und werden Algorithmen und Systeme zur berührungslosen, optischen Geschwindigkeits- und Längenmessung entwickelt und untersucht. Konkret konzentrierten sich die bisherigen Projekte auf sogenannte Ortsfrequenzfiltersysteme. Ortsfrequenzfiltersysteme und die konkurrierenden Laser-Doppler-Velozimeter (LDV) sind am Markt erhältlich und werden industriell eingesetzt, um in Produktionsprozessen Wegfortschritte oder Geschwindigkeiten zu ermitteln. [1] [2] [3] [4] [5]

Die letzten Quantifizierungen von Messunsicherheiten unter realen Bedingungen wurden im Jahr 2016 am COVIDIS-Sensor durchgeführt [4]. Dieses digitale Ortsfrequenzfiltersystem wurde vor über zehn Jahren entwickelt [1] und basiert dementsprechend auf den damals erhältlichen integrierten Schaltungen (ICs).

Mit dem Fortschritt der Technik sind seitdem leistungsfähigere ICs am Markt erhältlich, die im VADER-Projekt im Jahr 2018 zur Entwicklung einer Plattform nach dem Vorbild des COVIDIS verwendet wurden. Das VADER-System setzt zwar leistungsfähigere Komponenten ein, basiert aber weiterhin auf den Algorithmen des COVIDIS [3] [2].

Die vorliegende Arbeit untersucht einen Algorithmus, der sich grundlegend von den Ortsfilter- und Laser-Doppler-Systemen unterscheidet und evaluiert, ob ein solcher Algorithmus auf moderner Hardware in Echtzeit lauffähig ist und das Potential hat, mit den etablierten Systemen hinsichtlich der erreichbaren Messabweichung zu konkurrieren.

1.1 Messsysteme und -Konzepte

Bevor in späteren Abschnitten der Arbeit auf die exakten Funktionsweisen oder einzelne Implementierungsdetails der verschiedenen Verfahren eingegangen wird, wird anhand der Abbildung 1.1 eine Einführung in die bisher am Institut untersuchten Verfahren sowie in das im Hauptteil dieser Arbeit entwickelte, prinzipiell andersartige Verfahren, gegeben.

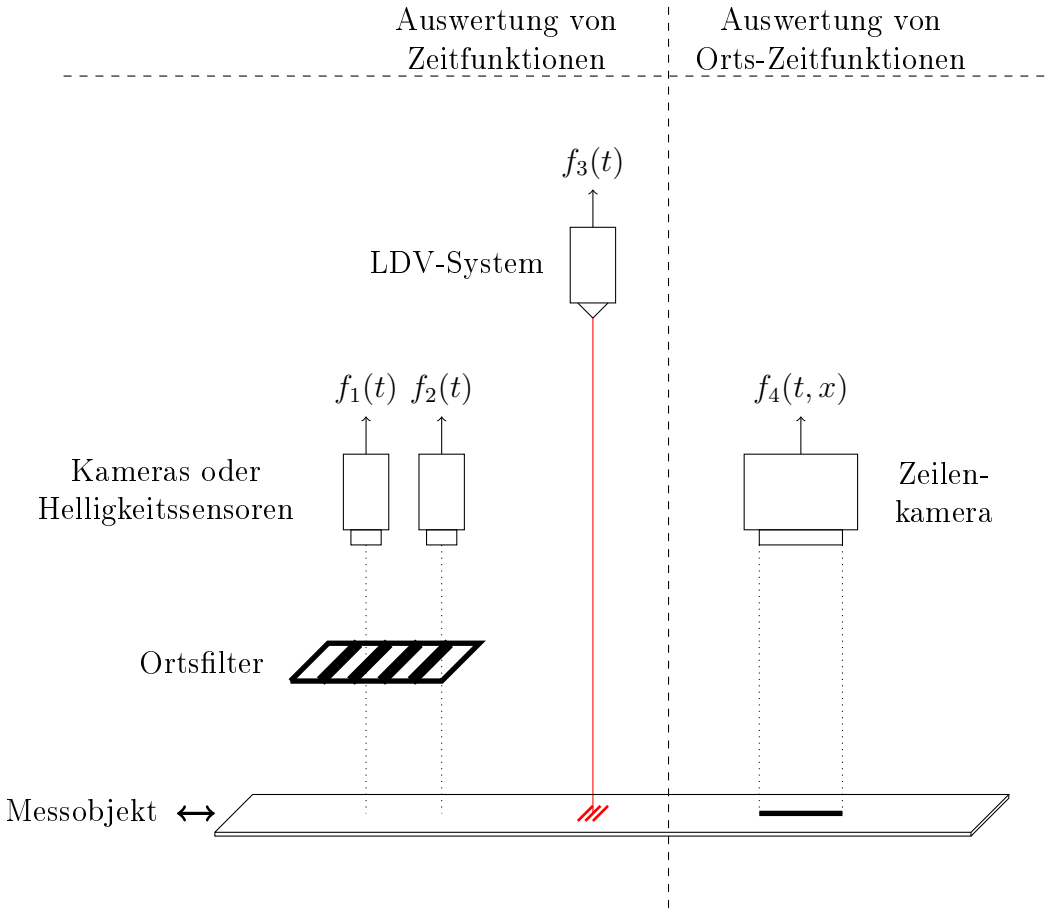

Abbildung 1.1: Verfahren zur Geschwindigkeits- und Längenmessung im Überblick

Die Abbildung teilt die betrachteten Verfahren in solche, die auf der Auswertung von Zeitfunktionen basieren und eines, das auf der Auswertung einer Orts-Zeitfunktion basiert, auf. Grundsätzlich ist festzustellen, dass eine Geschwindigkeit ein Verhältnis von Strecke zu Zeit beschreibt [6], woraus folgt, dass auch die

Zeitfunktionen, die den erstgenannten Verfahren als Grundlage dienen, auf die ein oder andere Weise mit einer Ortsgröße verknüpft sein müssen.

Die Funktionen $f_1(t)$ und $f_2(t)$ beschreiben sich mit der Zeit ändernde Helligkeiten, die durch das vom Messobjekt reflektierte Licht, das durch das gitterförmige Ortsfilter an die Kamera oder den Helligkeitssensor gelangt, entstehen. Es handelt sich bei diesen Funktionen um Ortsfrequenzfilterfunktionen. Das Ortsfilter hat den entstehenden Zeitfunktionen eine Ortsreferenz eingeprägt, sodass die Zeitfrequenz von $f_1(t)$ und $f_2(t)$ proportional zur Geschwindigkeit des Messobjekts ist. [7] [1]

Das Ortsfilter muss nicht wie in der Abbildung symbolisch dargestellt physikalisch zwischen dem Messobjekt und der Kamera oder dem Lichtsensor liegen. Der COVIDIS-Sensor sowie das am Institut entwickelte VADER-Projekt verwenden beispielsweise digitale Zeilenkameras und wenden in der Datenvorverarbeitung virtuelle Ortsfilter auf die Bilder an. Auch in diesen Fällen werden anschließend die digital erzeugten Zeitfunktionen ausgewertet. [2]

Die Funktion $f_3(t)$ gibt ebenfalls einen Helligkeitsverlauf über der Zeit wieder. Zwar werden LDV-Systeme nicht am Institut entwickelt, sie wurden aber als konkurrierende Technologie in vergangenen Projekten zu Vergleichszwecken herangezogen. Bei diesem Verfahren wird eine Ortsreferenz in Form eines Laser-Interferenzmusters auf das Messobjekt projiziert, wodurch das System prinzipiell wie ein Ortsfiltersystem funktioniert, bei dem das Ortsfilter direkt auf dem Messobjekt platziert wird. Auch die Frequenz von $f_3(t)$ ist somit proportional zur Objektgeschwindigkeit. [8]

Die Zeitfunktionen, die die Systeme in der linken Hälfte der Abbildung produzieren, können prinzipiell auf verschiedene Arten ausgewertet werden: Zum Beispiel können sie fouriertransformiert werden oder die Grundfrequenz durch Autokorrelation ermittelt werden; beim Drehzeigerverfahren kann eine Winkelgeschwindigkeit ausgewertet werden. Diese Verfahren sind in der Literatur untersucht worden und nicht Gegenstand der vorliegenden Arbeit. [1] [9]

Zur Ermittelung einer Geschwindigkeit nach dem Ortsfrequenzfilterprinzip reicht eine der Funktionen $f_1(t)$ oder $f_2(t)$ und eines der im letzten Absatz genannten Auswerteprinzipien aus. In der Abbildung sind die zwei Helligkeitssensoren allerdings so entlang der Bewegungsrichtung des Messobjekts platziert, dass während

eines Bewegungsvorganges näherungsweise gelten muss $f_1(t) = f_2(t + \Delta t)$. Ist nun der räumliche Abstand Δx zwischen den Helligkeitssensoren bekannt, können örtliche und zeitliche Verschiebung ins Verhältnis gesetzt werden, um wieder zu einer Geschwindigkeit zu gelangen. Ein Verfahren, das Ortsfilterfunktionspaare nach diesem Schema auswertet, wurde in der Vorarbeit „Simulation einer Geschwindigkeitsmessung durch Kreuzkorrelation zwischen Ortsfilterfunktionen“ modelliert und simuliert. [10]

Abschließend zeigt Abbildung 1.1 auf der rechten Seite eine digitale Zeilenkamera, die die Zeilenbilder $f_4(t, x)$ aufnimmt. Als einziges Verfahren in der Abbildung wird hierbei die Ortsdimension nicht entweder schon beim Aufnahmeprozess oder in der Vorverarbeitung mit der Zeitdimension verwoben. Die Motivation dieser Arbeit ist es, die in $f_4(t, x)$ gegenüber den anderen Funktionen zusätzlich enthaltene Information auszuwerten, um möglicherweise zu besseren Ergebnissen zu gelangen.

1.2 Aufgabenstellung

Gegenstand dieser Arbeit ist im ersten Teil die Anwendung des in der vorhergehenden Arbeit „Simulation einer Geschwindigkeitsmessung durch Kreuzkorrelation zwischen Ortsfilterfunktionen“ modellbasiert entwickelten Algorithmus auf Echtbilder sowie die Bewertung der Ergebnisse und die Entwicklung daraus folgender Arbeitshypothesen für mögliche Folgeprojekte.

Der Hauptteil der vorliegenden Arbeit befasst sich mit der Entwicklung eines Algorithmus zur direkten Messung von Strecken bzw. Wegfortschritten durch Kreuzkorrelation sowie einer entsprechenden Soft- und Hardwareplattform, auf der dieser Algorithmus echtzeitfähig implementiert und an Echtbildern getestet wird.

2 Grundlagen

Dieses Kapitel befasst sich mit grundlegenden Überlegungen zur Bewertung von Messsystemen und -Algorithmen sowie der Kreuzkorrelation. Die theoretischen Grundlagen der Ortsfilter, wie sie im ersten Teil der Arbeit zur Anwendung kommen, werden nicht behandelt, da die entsprechenden Theorien in anderen Arbeiten [10] [1] bereits vollständig erörtert werden.

Weil der Vergleich der Messunsicherheiten zum einen zwischen Modell und Realität und zum anderen zwischen verschiedenen Messsystemen in beiden Teilen der Arbeit von Relevanz ist, wird außerdem grundlegend die Quantifizierung von Messabweichungen und Messunsicherheiten diskutiert.

2.1 Algorithmen und Messsysteme

Für den Begriff Algorithmus finden sich in der Literatur verschiedene Definitionen. Allgemein sind es Handlungsvorschriften zur Bewältigung einer Aufgabe in definierten Rechen- oder Befehlsfolgen, die zum Beispiel nach Kriterien der Korrektheit, der Komplexität, der Laufzeit oder des Speicherbedarfs bewertet werden können. [11] [12]

In dieser Arbeit werden sowohl die Begriffe Algorithmus als auch Messsystem, Sensor oder einfach System verwendet, die im Folgenden abgegrenzt werden.

Die Systembegriffe werden für vollständige, (industriell) einsetzbare und spezifizierte Produkte verwendet (Beispiele s. Abschnitt 2.4), die mehrere Algorithmen einsetzen können um die gewünschte Ausgabe zu erzeugen.

Zum Beispiel besteht der COVIDIS, ein Messsystem, nach der hier verwendeten Definition unter anderem aus dem Algorithmus zur Berechnung der Frequenz

aus einer Ortsfilterfunktion durch die diskrete Fouriertransformation (DFT) (in der vorherigen Arbeit wurde hierfür der Begriff „DFT-Algorithmus für Ortsfilterfunktionen“ eingeführt) sowie z.B. aus dem Blockmatching-Algorithmus zur Berechnung von Wegfortschritten bei Kriechgeschwindigkeiten. Darüber hinaus kommen weitere Algorithmen zur Plausibilitätsprüfung, Interpolation usw. zum Einsatz. [1]

Ziel der Abgrenzung ist es, einzelne Algorithmen, wie z.B. der Algorithmus zur Auswertung von Ortsfilterfunktionen durch Kreuzkorrelation, in der Beurteilung ihrer Eigenschaften von vollständigen Messsystemen zu unterscheiden.

2.2 Geschwindigkeit

Die Geschwindigkeit ist definiert als Ableitung des Ortes nach der Zeit [6]:

$$v = \frac{\mathrm{d}s}{\mathrm{d}t} \tag{2.1}$$

Würde man die Funktion eines „Geschwindigkeitsensors“ strikt aus dieser Definition ableiten, handelte es sich um ein Gerät, das instantan die Geschwindigkeit eines Objekts ermittelt.

Dass es sich z.B. bei Ortsfiltersystemen wie dem VADER oder dem COVIDIS nicht um solche Geräte handeln kann, folgt allein schon aus der Tatsache, dass diese Systeme wie in der Einleitung angedeutet diskrete Zeitfunktionen auswerten und für eine auswertbare Ortsfilterfunktion mindestens zwei (und in der Realität wesentlich mehr) Samples dieser Funktion erforderlich sind, die aus zu zwei durch eine Kamera mit endlicher Framerate bestimmten Zeitpunkten aufgenommenen Bildern erzeugt werden.

Die „Geschwindigkeit“ wird also im Falle der Ortsfiltersysteme aus der Frequenz einer oder mehrerer Ortsfilterfunktionen über einen Zeitraum Δt ermittelt [1]. Es handelt es sich physikalisch um eine Durchschnittsgeschwindigkeit [13, S. 5]:

$$\bar{v} = \frac{\Delta s}{\Delta t} \tag{2.2}$$

Für LDV-Systeme kann auch ohne genaues Wissen über die internen Algorithmen eine ähnliche Überlegung angestellt werden: Auch hier wird die Geschwindigkeit aus der Frequenz eines in der Intensität modulierten Lichtsignals gewonnen [8]. Die Unschärferelation zwischen Zeit und Frequenzbereich besagt dabei, dass die Frequenz einer Schwingung umso undefinierter ist, je kürzer der entsprechende zeitliche Vorgang ist oder betrachtet wurde [14, S. 51].

Unabhängig vom Auswerteverfahren ist es daher für geringe Messabweichungen wünschenswert, das gewonnene Signal über eine gewisse Dauer zu betrachten, um eine möglichst genaue Aussage über dessen Frequenz und damit die Geschwindigkeit treffen zu können.

Auch im Hinblick auf die schlussendliche Anwendung wäre die Ausgabe von Geschwindigkeiten im physikalischen Sinne weniger nützlich als die Ausgabe einer Durchschnittsgeschwindigkeit, denn die als „Geschwindigkeitssensoren“ bezeichneten Geräte werden in der Industrie auch zur Messung von Längen und Wegfortschritten eingesetzt [15].

In einem Industrieprozess mit einer geregelten Geschwindkeit ist wegen den realen Systemen inhärenten Nichtidealitäten statt von einer im physikalischen Sinne konstanten Geschwindigkeit ($\frac{\mathrm{d}v}{\mathrm{d}t} = 0$) eine zumindest minimal schwankende Momentangeschwindigkeit zu erwarten. Dieser Schwankung wird in einer Durchschnittsgeschwindigkeit Rechnung getragen und der gesuchte Wegfortschritt errechnet sich aus $\Delta s = \bar{v} \cdot \Delta t$.

In dieser Arbeit wird die technische Nomenklatur aus dem Bereich der betrachteten Sensorsysteme übernommen und der Begriff Geschwindigkeit für Durchschnittsgeschwindigkeiten, wenn auch über kurze Zeiträume, verwendet während der Begriff Momentangeschwindigkeit für die physikalische Definition der Geschwindigkeit eingesetzt wird.

2.3 Messunsicherheiten und Messabweichungen

Bei einer Messung muss immer mit Abweichungen des Ergebnisses vom wahren Wert x_w der Messgröße gerechnet werden. Die Messabweichung F ist die Differenz zwischen dem Messwert und dem wahren Wert. Da das Ziel der Messung die

Ermittelung einer Größe ist, ist ihr wahrer Wert im Allgemeinen nicht bekannt. In diesen Fällen wird der Begriff richtiger Wert (x_r) eingeführt, der die bestmögliche Annäherung an den wahren Wert sein soll. [16]

In der vorliegenden Arbeit werden teilweise künstlich erzeugte Bewegungsvorgänge vermessen, für die - numerische Ungenauigkeiten in der digitalen Signalverarbeitung vernachlässigend - der wahre Wert der Geschwindigkeit oder Strecke bekannt ist. Andererseits werden insbesondere auch reale Bewegungsvorgänge vermessen, für die der wahre Wert nicht bekannt ist. Je nach der vorliegenden Situation wird die Messabweichung jeweils auf Basis des wahren oder des richtigen Wertes ermittelt.

Um die vergleichende Betrachtung von Algorithmen und Messsystemen zu erleichtern, wird in dieser Arbeit im Allgemeinen eine relative Messabweichung angegeben:

$$F_\mathrm{rel} = \frac{F}{x_\mathrm{r}} \quad \text{oder} \quad F_\mathrm{rel} = \frac{F}{x_\mathrm{w}}$$

Messabweichungen lassen sich in zufällige und systematische Messabweichungen einteilen, wobei letztere reproduzierbar und erstere nicht reproduzierbar sind. Wird das Messergebnis um bekannte systematische Messabweichungen korrigiert, so verbleibt eine Messunsicherheit, die sich aus den restlichen Messabweichungen ergibt. [17, S. 33]

Wie die Messabweichung kann die Messunsicherheit in absoluter oder relativer Form angegeben werden und bildet einen Bereich um den angegebenen Messwert herum, in dem der wahre oder richtige Wert liegt. Möglich ist auch die Angabe einer Messunsicherheit, die mit einer bestimmten Wahrscheinlichkeit eingehalten wird. [17, S. 33f]

2.4 Spezifikationen einzelner Sensorsysteme

Die Spezifikationen am Markt erhältlicher LDV- und Ortsfiltersysteme wurden in der Vorarbeit mit dem Fazit zusammengefasst, dass die Spezifikationen der Messunsicherheiten zwischen den in den Jahren 2010 und 2020 angebotenen Systemen in Größenordnungen von 0,5‰ unverändert geblieben ist und eine erheb-

liche Verbesserung dieses Werts bislang entweder wirtschaftlich oder technisch nicht möglich war. [10]

Im Folgenden werden die Spezifikationen einiger Systeme, bei denen die Messunsicherheit auf verschiedene Weisen angegeben ist, mit dem Ziel interpretiert, eine objektive Basis für Vergleiche zu den in der Arbeit untersuchten Algorithmen herzustellen.

Die Firma Astech gibt für den Ortsfiltersensor „VLM500“ die folgenden Messunsicherheiten an [15]:

- $< 0{,}025\,\%$ bei nominalem Arbeitsabstand
- $< 0{,}05\,\%$ im Arbeitsabstand
- $< 0{,}2\,\%$ im erweiterten Arbeitsabstand

Diese sind unter anderem an folgende Bedingungen geknüpft [15]:

- 10 m Messlänge
- Konstante Temperatur von 20 °C
- Konstante Distanz
- Konstante Geschwindigkeit
- Konstante Beleuchtung

Beim Versuch, aus diesen Angaben die mit dem angebotenen System erreichbare Messunsicherheit für eine gegebene Messaufgabe zu ermitteln, ergeben sich verschiedene Probleme. Einerseits ist zum Beispiel unklar, welche Abweichungen von einer im physikalischen Sinne konstanten Geschwindigkeit (s. Abschnitt 2.2) zulässig sind, um die Bedingung einzuhalten. Sinngemäß gilt die gleiche Überlegung für alle Bedingungen, die die „Konstanz“ einer physikalischen Größe vorschreiben.

Darüber hinaus lässt sich aus der Angabe einer Messunsicherheit, die an eine Messstrecke von 10 m als Bedingung geknüpft ist, keinerlei Aussage über die Messunsicherheit für eine Strecke kleiner 10 m treffen. Selbst hierbei handelt es sich bereits um eine Interpretation der Spezifikation, die darauf basiert, dass andere SFV-Systeme im Allgemeinen bei längeren Messstrecken bessere Ergebnisse erzielen [1, S. 102] [18] [19]. Bei konsequenter Anwendung der Bedingung alleine ist die Messunsicherheit auch für Strecken größer 10 m undefiniert.

Ein weiteres Beispiel ist der Laser-Doppler-Sensor „ProSpeed LSV-2100“ der Firma Polytec. Angegeben werden verschiedene Bedingungen an Arbeitsabstand, Messfeldtiefe und maximale Beschleunigungen sowie Geschwindigkeiten. Weiterhin findet sich im Datenblatt folgende Angabe: [20]

- Genauigkeit: $< 0{,}05\,\%$ vom Messwert mit der Fußnote „unter kontrollierten Bedingungen“

Bei konsequenter Betrachtung wird die Quantifizierung der Genauigkeit durch die Bindung an nicht weiter spezifizierte kontrollierte Bedingungen zu einer vollständig bedeutungslosen Angabe, da keine Möglichkeit besteht, zu überprüfen ob eben diese kontrollierten Bedingungen in einer gegebenen Messsituation eingehalten werden.

Die LDV-Produktserie „Laserspeed“ des Herstellers BETA LaserMike gibt im Datenblatt eine Genauigkeit von $< \pm 0{,}03\,\%$ des gemessenen Wertes an, allerdings ohne dies an Bedingungen zu knüpfen [21]. Damit ließen sich theoretisch Situationen konstruieren, unter denen eine Messung beliebig erschwert wird, bis die angegebene Spezifikation nicht mehr eingehalten wird.

Die bisher angestellten Überlegungen zeigen, dass das Anstreben eines auf absolute Objektivität abzielenden Ansatzes bei der Bewertung der verschiedenen Messsysteme anhand der zur Verfügung stehenden Informationen scheitert, da durch den Versuch des Anlegens eines objektiven Maßstabes alle oben genannten Spezifikationen verworfen werden müssten: Sind Bedingungen spezifiziert, lässt sich deren Einhaltung in der Realität nur in einem gewissen Rahmen überprüfen, der an sich wieder eine Spezifikation erfordern würde. Sind undefinierte Bedingungen angegeben, ist die Spezifikation nicht sinnvoll anwendbar und sind gar keine Bedingungen angegeben, ist die Spezifikation unvollständig.

Dies führt zu der Feststellung, dass bei der abschließenden Bewertung der Messergebnisse in der vorliegenden Arbeit ein gewisses Maß an Subjektivität einerseits wegen der schon inhärent subjektiven Projektanforderung eines „geringen Messfehlers“ und andererseits wegen des Fehlens von objektiven Vergleichsmaßstäben bei den Industriesensoren nicht vermieden werden kann.

2.5 Kreuzkorrelation

Die zeitkontinuierliche Kreuzkorrelationsfunktion ist ein Ähnlichkeitsmaß für zwei Signale unter bestimmten Verschiebungen τ zueinander und für Energiesignale wie folgt definiert [22, S. 206]:

$$\varphi_{a,b}(\tau) = \int_{-\infty}^{\infty} \overline{a(t)} \cdot b(t+\tau)\,\mathrm{d}t \tag{2.3}$$

Das Kreuzkorrelationsintegral basiert auf einer Energieberechnung, wobei die Energie zwischen zwei Signalen das angesprochene Maß für ihre Ähnlichkeit ist [22, S. 204].

In dieser Arbeit werden nur reellwertige Zeit- oder Ortsfunktionen behandelt, weshalb die komplexe Konjugation im Korrelationsintegral im Folgenden nicht mehr geschrieben wird.

Das Argument der Kreuzkorrelationsfunktion hat die Einheit der Argumente der korrelierten Signale. Die Korrelationsfunktionen für Orts- und Zeitverschiebungen lassen sich somit völlig analog ermitteln:

$$\varphi_{c,d}(\Delta t) = \int_{-\infty}^{\infty} c(t) \cdot d(t+\Delta t)\,\mathrm{d}t \tag{2.4}$$

$$\varphi_{e,f}(\Delta x) = \int_{-\infty}^{\infty} e(x) \cdot f(x+\Delta x)\,\mathrm{d}x \tag{2.5}$$

Wegen der Ähnlichkeit des Korrelationsintegrals zum Faltungsintegral und dem Faltungstheorem der Fouriertransformation kann die Kreuzkorrelation im Frequenzbereich berechnet werden [22, S. 212]. Diese Eigenschaft wird in Abschnitt 6.3 auch für den zeitdiskreten Fall näher betrachtet.

$$\varphi_{c,d}(\Delta t) \circ\!\!-\!\!\bullet\, \overline{C(f)} \cdot D(f) \tag{2.6}$$

$$\varphi_{e,f}(\Delta x) \circ\!\!-\!\!\bullet\, \overline{E(\mu)} \cdot F(\mu) \tag{2.7}$$

Für die Kreuzkorrelation wird im Folgenden auch der Operator $\times$ verwendet.

3 Korrelation von Ortsfilterfunktionen

Für die in der Vorarbeit vorgestellte Auswertung von Ortsfilterfunktionen durch Kreuzkorrelation werden aus dort angeführten Gründen (Periodizitäten der Systemantworten) Ortsfilter verwendet, die sich von denen bei der DFT-Auswertung im COVIDIS verwendeten in verschiedenen Hinsichten unterscheiden: [10]

Die Filter

- bestehen jeweils nur aus einer einzelnen Filterperiode.
- bedecken nur einen relativ kleinen Teil des Zeilenbildes.
- stehen still.

Für die Auswertung mittels Korrelation sind mindestens zwei Ortsfilterfunktionen und damit auch mindestens zwei Ortsfilter erforderlich. Aus diesem Grund wird der Begriff Filterpaar eingeführt und um eine Multikalenauswertung zu ermöglichen, werden mehrere Filterpaare verwendet.

In der Simulation werden elf Filterpaare eingesetzt, Abbildung 3.1 zeigt aus Gründen der Übersichtlichkeit ein Beispiel mit zwei Filterpaaren.

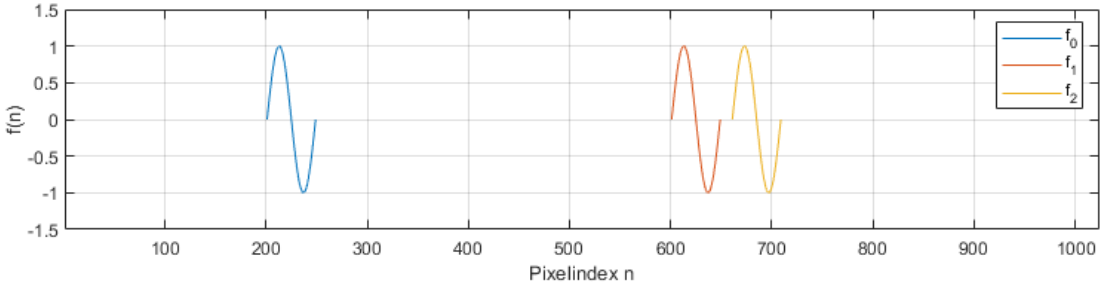

Abbildung 3.1: Zwei Ortsfilterpaare entlang der Kamerazeile bzw. eines Zeilenbildes

Die zwei Filterpaare bestehen in diesem Beispiel aus den Filtern f_0 und f_1 bzw. f_0 und f_2. Das erste Filterpaar hat dabei die Filterdistanz d_1, das zweite d_2.

Bewegt sich ein Oberflächenmerkmal über die Kamerazeile, gilt für die vergangene Zeit zwischen dem Überstreichen der jeweiligen Einzelfilter der Filterpaare:

$$\Delta t_1 = \frac{d_1}{\bar{v}_1} \tag{3.1}$$

$$\Delta t_2 = \frac{d_2}{\bar{v}_2} \tag{3.2}$$

In der an dieser Stelle besprochenen Vorarbeit ist die Betrachtung nicht beschleunigter Bewegungen ausdrücklich vorgegeben [23], weshalb dort $v = \bar{v}$ gilt. Da die vorliegende Arbeit aber explizit reale Bewegungsvorgänge betrachtet, wird der in Abschnitt 2.2 eingeführten Unterscheidung zwischen Momentan- und Durchschnittsgeschwindigkeit Rechnung getragen, indem in die Formeln aus der Vorarbeit entsprechende Durchschnittsgeschwindigkeiten eingesetzt werden.

Dies führt dazu, dass die Gleichungen 3.1 und 3.2 zwei unterschiedliche Durchschnittsgeschwindigkeiten berücksichtigen müssen, da $\bar{v}_1 = \bar{v}_2$ nur gilt, wenn auch die Durchschnittsgeschwindigkeit beim Zurücklegen der Strecke zwischen f_1 und f_2 die gleiche war. Diese Unterscheidung ist allerdings formeller Natur, denn da die Verifikation der Ergebnisse aus der Vorarbeit in der Realität unter Bedingungen erfolgt, die denen der Simulation im Rahmen der Möglichkeiten gleichen, wird $\bar{v} \approx \bar{v}_1 \approx \bar{v}_2 \approx v$ vereinbart.

Mit

$$\bar{v} = \frac{d}{\Delta t} \tag{3.3}$$

ergibt sich unter Berücksichtigung des optischen Abbildungsmaßstabes, des Pixelabstands und der Framerate der Kamera sowie einer Filterdistanz $d = 500 \cdot W_{\mathrm{px}}$ der in Abbildung 3.2 dargestellte Zusammenhang zwischen Δt und $\bar{v}$.

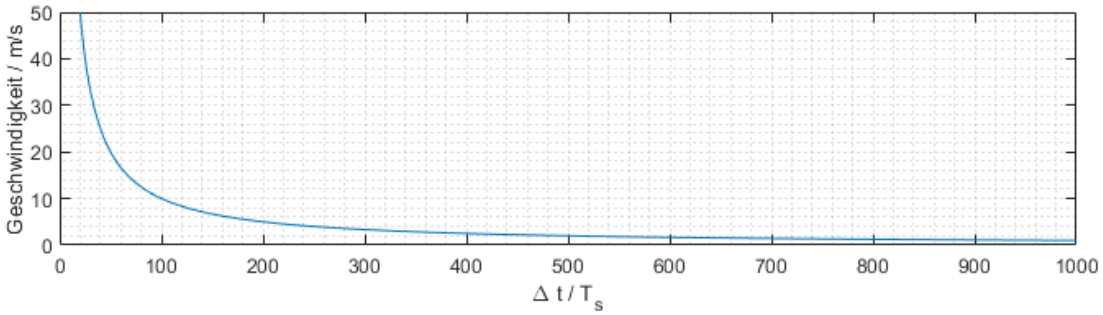

Abbildung 3.2: Zusammenhang zwischen Geschwindigkeit und Zeitverzögerung bei einem Filterpaar mit $d = 500 \cdot W_{\text{px}}$

Der nichtlineare Verlauf der dargestellten Kurve führt dazu, dass es im Bereich großer Steigungen, also kleiner Δt, zu steigenden Rundungsfehlern kommt. Dieser Aspekt wird in Abschnitt 3.1 näher betrachtet.

In der Vorarbeit wird zwischen zwei Ansätzen der Auswertung unterschieden: [10]

1. Die diskrete Kreuzkorrelation, wobei Δt aus dem Maximum der diskreten Kreuzkorrelationsfunktion zwischen den Ortsfilterfunktionen der Einzelfilter eines Filterpaares bestimmt wird, es also gilt $\Delta t = n \cdot T_{\text{s}}, n \in \mathbb{Z}$.
2. Die DFT-interpolierte Kreuzkorrelation, wobei zusätzlich zwischen den Abtastzeitpunkten interpoliert wird, weshalb Δt kein ganzzahliges Vielfaches der Samplezeit bzw. Frameperiode mehr sein muss.

3.1 Diskrete Kreuzkorrelation

Auf den Vergleich der zwei Ansätze im letzten Abschnitt stützt sich die Annahme, dass der Ansatz der diskreten Kreuzkorrelation im Allgemeinen schlechtere Ergebnisse liefern wird als ein Ansatz mit zusätzlicher Interpolation. Diese Annahme bestätigt sich in der Simulation, was bei der Auswertung der Ergebnisse in der Vorarbeit bereits festgestellt wurde [10, S. 37] und erst den zusätzlichen Aufwand für den zweiten Ansatz rechtfertigt.

Dieser Abschnitt behandelt den Rundungsfehler, der sich bei der diskreten Kreuzkorrelationsmethode ergibt theoretisch und ohne den Einbezug von Echtbildern. Auf die Betrachtung von Echtbildern wird verzichtet, da sich zeigen wird, dass

bereits unter optimalen Bedingungen bei relativ niedrigen Geschwindigkeiten Abweichungen oberhalb des angestrebten Ziels von 0,5 ‰ auftreten können.

Da beim Ansatz der diskreten Kreuzkorrelation nur die Einträge der diskreten Kreuzkorrelationsfunktion ohne Zwischenwerte ausgewertet werden können, wird folgende Annahme getroffen:

Arbeitshypothese 1 *Fällt bei der Kreuzkorrelation das Maximum zwischen zwei ganzzahlige Vielfache der Frameperiode, so ist das Maximum der diskreten Kreuzkorrelationsfunktion der Eintrag, dem das tatsächliche Maximum am nächsten liegt.*

Diese Annahme kann sinngemäß so interpretiert werden, dass das gesuchte Δt bei der Auswertung der diskreten Kreuzkorrelation kaufmännisch gerundet vorliegt. So folgt z.B.:

$$\begin{aligned}(f_1 \times f_2)(n_{\max} \cdot T_\mathrm{s}) > (f_1 \times f_2)(n \cdot T_\mathrm{s}) \quad \forall\, n \in \mathbb{Z} \setminus \{n_{\max}\} \\ \Rightarrow (n_{\max} - 0{,}5) \cdot T_\mathrm{s} \leq \Delta t < (n_{\max} + 0{,}5) \cdot T_\mathrm{s}\end{aligned}$$

Die genannte Annahme wurde zunächst aufgestellt, weil sie den Beobachtungen in der Simulation entspricht und darüberhinaus als plausibel betrachtet wird. Bei Einsatz der in Abschnitt 3.2 betrachteten Interpolationsmethode konnte kein Fall festgestellt werden, in dem das interpolierte Maximum gegenüber dem Maximum der diskreten Auswertung um mehr als einen halben Diskretisierungsschritt verschoben ist. Dies wird als weiteres Indiz für die Korrektheit der Annahme interpretiert.

Abbildung 3.2 zeigt, dass die Steigung der dargestellten Kurve mit der Geschwindigkeit wächst. Ein Beispiel für die daraus resultierende Wirkung der Arbeitshypothese 1 ist in Abbildung 3.3 für ein Δt zwischen $2 \cdot T_\mathrm{s}$ und $3 \cdot T_\mathrm{s}$ bei einer Filterdistanz von $500 \cdot W_\mathrm{px}$ gezeigt, was Geschwindigkeiten in der Größenordnung von $500\,\mathrm{m\,s^{-1}}$ entspricht. Das Beispiel wurde aus Gründen der Anschaulichkeit gewählt, obwohl die entsprechenden Geschwindigkeiten nicht im Messbereich liegen, da der Effekt bei niedrigeren Geschwindigkeiten nach dem gleichen Prinzip auftritt.

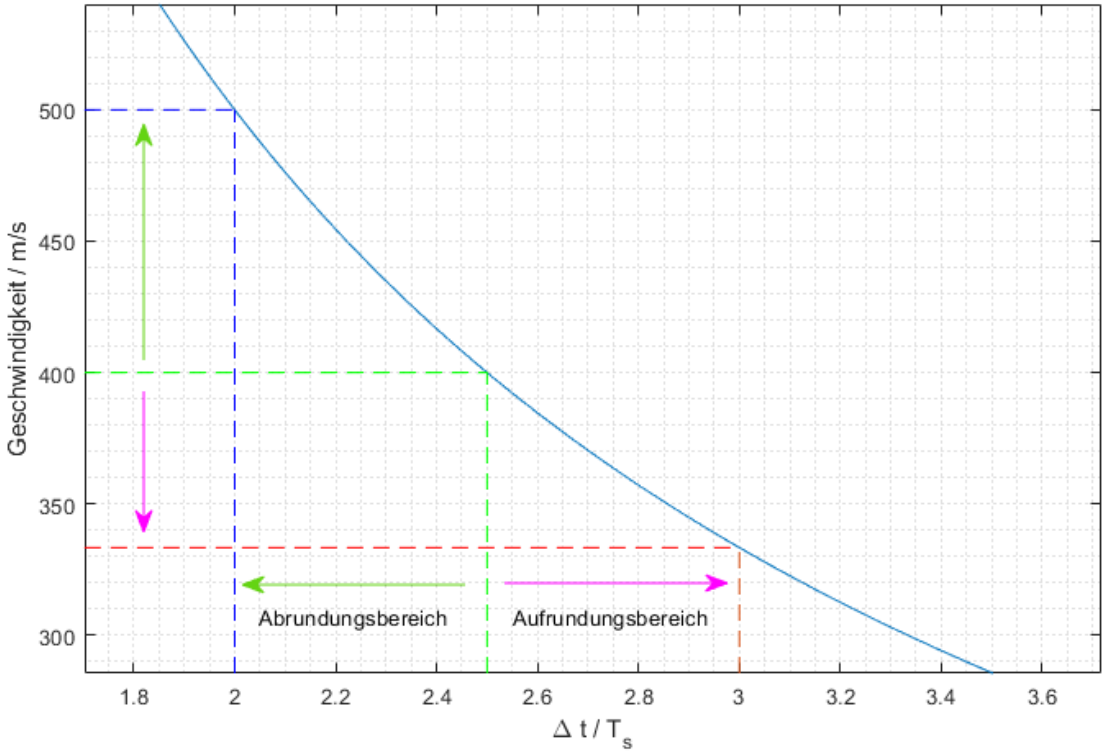

Abbildung 3.3: Rundung auf den Verzögerungs- und Geschwindigkeitsachsen

Werden die über die Kurve auf die Ordinate projizierten Rundungsbereiche betrachtet, so sind die Geschwindigkeitsbereiche, innerhalb derer die Geschwindigkeit aufgerundet wird jeweils größer als die, in denen sie abgerundet wird. Nimmt man daher eine Folge von Messungen vor, bei denen die tatsächliche Geschwindigkeit (im Messbereich) gleichmäßig verteilt ist, so wird die gemessene Geschwindigkeit im Durchschnitt tendenziell zu hoch liegen.

Die Parameter der Simulation in der Vorarbeit (simulierte Geschwindigkeiten, Kameraparameter, Anzahl und Distanz der Filterpaare) werden verwendet, um die aus der Arbeitshypothese 1, sollte sie korrekt sein, resultierenden Messabweichungen vorherzusagen. Dabei wird für die kaufmännische Rundung die Funktion rd() eingeführt:

$$\Delta t_n = \frac{d_n}{\bar{v}} \tag{3.4}$$

$$\bar{v}_n = \frac{d_n}{\mathrm{rd}(\Delta t_n)} \tag{3.5}$$

$$\bar{v}_{\mathrm{mean}} = \frac{1}{N} \cdot \sum_{n=0}^{N-1} \bar{v}_n \tag{3.6}$$

$$F_{\mathrm{rel}} = \frac{\bar{v}_{\mathrm{mean}} - \bar{v}}{\bar{v}} \tag{3.7}$$

Die Gleichung 3.6 repräsentiert dabei die Mittelung der Ergebnisse der einzelnen Filterpaare, es handelt sich also um die Ermittelung der durchschnittlichen Durchschnittsgeschwindigkeit. Tabelle 3.1 vergleicht die nach Gleichung 3.7 vorhergesagten mit den in der Simulation der Vorarbeit ermittelten relativen Messabweichungen bei den simulierten Geschwindigkeiten.

Geschwindigkeit / $\mathrm{m\,s^{-1}}$	F_{rel} / ‰	
	Simulation	Vorhersage
0,923	-0,012	-0,0784
1,17	0,065	-0,0409
1,6734	0,083	-0,0461
2,034	-0,012	-0,012
2,039	0,099	0,0995
2,07	0,227	0,0449
2,41	0,05	0,0505
2,568	0,134	0,0517
3,873	-0,113	-0,113
4,99	1,126	1,126
5,12	0,137	0,137
6,25	0,016	0,016
8,143	-0,108	-0,108
9,263	-0,409	-0,409
15,458	0,297	0,297
19,568	-0,708	-0,708

24,65	-0,133	-0,133
30,103	-0,738	-0,738
34,145	2,173	2,173
39,129	2,868	2,868
50,32	-3,002	-3,002

Tabelle 3.1: Vergleich zwischen Vorhersage und tatsächlich simulierter Messabweichung bei Verwendung der diskreten Kreuzkorrelation

Dass die Gleichung 3.7 die simulierten Messfehler ab etwa $2\,\mathrm{m\,s^{-1}}$ relativ zuverlässig und für die höheren Geschwindigkeiten exakt vorhersagt, wird als Indiz für die Korrektheit der Arbeitshypothese 1, auf der die Gleichung basiert, gewertet. Darüber hinaus wird es als plausibel erachtet, dass der Einfluss des betrachteten Effekts mit der Geschwindigkeit steigt, weil in diesen Bereichen die Steigung der in den Abbildungen 3.2 und 3.3 gezeigten Kurven besonders hoch ist.

Bei der geschilderten Betrachtung ist hervorzuheben, dass in die Vorhersage der Messabweichung keinerlei Bilddaten oder Filterfunktionen sondern lediglich Geschwindigkeiten und Filterdistanzen eingeflossen sind und es sich um die Betrachtung eines sonst fehlerfreien Systems handelt. Dies liefert eine mögliche Erklärung dafür, dass die Vorhersagen für niedrige Geschwindigkeiten nicht zutreffen, ohne dass dies notwendigerweise die Arbeitshypothese 1 widerlegt: Ein anderer Fehler unbekannten Ursprungs, der scheinbar besonders bei niedrigen Geschwindigkeiten hervortritt, könnte sich mit dem aus der Arbeitshypothese 1 hervorgehenden Fehler überlagern.

Abbildung 3.4 zeigt den Verlauf der aus Gleichung 3.7 ermittelten Abweichungen über Messbereichen von $0{,}1\,\mathrm{m\,s^{-1}}$ bis $50\,\mathrm{m\,s^{-1}}$ und $0{,}1\,\mathrm{m\,s^{-1}}$ bis $2\,\mathrm{m\,s^{-1}}$.

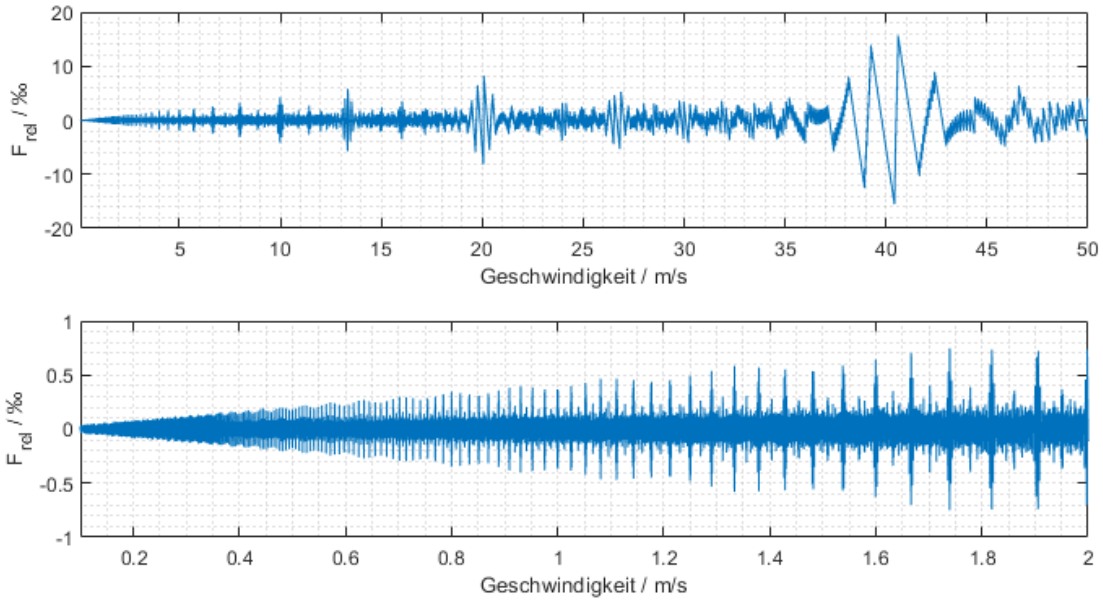

Abbildung 3.4: Relativer Rundungsfehler für die simulierten Parameter über der Geschwindigkeit

Die gezeigte Kurve besitzt einen leicht positiven Mittelwert, wie aus der Asymmetrieüberlegung zu erwarten war. Dieser ist aber so gering, dass er im Vergleich zu den maximal auftretenden Abweichungen in der Größenordnung von 15 ‰ kaum erkennbar ist und im betrachteten Bereich im Vergleich zum sonstigen Rundungsfehler als unerheblich betrachtet wird.

Im Betrachtungsbereich bis $2\,\mathrm{m\,s^{-1}}$ zeigt sich, dass teilweise schon ab Geschwindigkeiten von etwa $1\,\mathrm{m\,s^{-1}}$ Messabweichungen im Bereich 0,5 ‰ auftreten können. Dass z.B. in Tabelle 3.1 für die Geschwindigkeit $6{,}25\,\mathrm{m\,s^{-1}}$ mit 0,05 ‰ ein um eine Größenordnung geringere Messabweichung vorhergesagt und auch simuliert wird, wird daher als Folge dessen interpretiert, dass die gezeigte Kurve in der Nähe von $6{,}25\,\mathrm{m\,s^{-1}}$ einen Nulldurchgang hat, und nicht als Indikation dafür, dass diese Abweichungen generell bis zu dieser Geschwindigkeit oder auch nur im näheren Umfeld dieser Geschwindigkeit immer zu erwarten sind.

Unter den hier betrachteten Gesichtspunkten werden für mögliche Folgeprojekte die nachfolgenden Arbeitshypothesen im Bezug auf die Auswertung von Ortsfilterfunktionspaaren durch diskrete Kreuzkorrelation aufgestellt.

Arbeitshypothese 2 *Solange nicht alle relevanten Fehlerquellen identifiziert sind, muss davon ausgegangen werden, dass es sich bei den in Gleichung 3.7 ermittelten Abweichungen um Mindestwerte handelt, die sich im schlechtesten Fall konstruktiv mit dem oder den unbekannten anderen Fehlerquellen überlagern. Soll die diskrete Kreuzkorrelation daher zur Ermittelung der finalen Geschwindigkeitsabschätzung verwendet werden, muss durch die Wahl der Anzahl der Filterpaare oder deren Distanzen in Kombination mit den Kameraparametern sichergestellt werden, dass der Rundungsfehler innerhalb des spezifizierten Geschwindigkeitsbereichs an keinem Punkt die spezifizierte Messabweichung (z.B.* 0,5 ‰*) überschreitet.*

Arbeitshypothese 3 *Die diskrete Kreuzkorrelation kann trotz eines Rundungsfehlers außerhalb der spezifizierten Messabweichung zur Vorselektion für eine andere Methode der Geschwindigkeitsberechnung verwendet werden, wenn dies insgesamt zu einer kürzeren Laufzeit des Gesamtalgorithmus führt.*

Abschließend wird angemerkt, dass die hier betrachteten Phänomene prinzipiell nicht nur bei der diskreten Kreuzkorrelation auftreten: Auch die eingangs erwähnte und im nächsten Abschnitt betrachtete Methode der DFT-interpolierten Kreuzkorrelation arbeitet mit diskreten Interpolationsstellen, zwischen denen ein Rundungsfehler auftreten kann. Allerdings kann das Interpolationsraster theoretisch beliebig klein gewählt und das Problem damit zu Geschwindigkeiten außerhalb des spezifizierten Messbereichs verschoben werden.

3.2 DFT-interpolierte Kreuzkorrelation

Der Ansatz der DFT-interpolierten Kreuzkorrelation wurde in der Vorarbeit erarbeitet [10, S. 11] und kann verkürzt dargestellt als Kombination der Berechnung einer Kreuzkorrelation im Frequenzbereich und dem Zeitverschiebungssatz der diskreten Fouriertransformation verstanden werden.

Der Algorithmus wird in der vorliegenden Arbeit in Abschnitt 6.5 für die Ermittelung von Subpixel-Verschiebungen zwischen Ortsfunktionen näher behandelt. An dieser Stelle sei auf die Erläuterungen in der Vorarbeit verwiesen oder die

Methode als Erweiterung der diskreten Kreuzkorrelation um Argumente die zwar diskrete, aber nicht ganzzahlige Vielfache von T_s sind, verstanden.

Die Methode kann in der Simulation mit künstlich erzeugten Bildern ohne Querbewegungskomponente bei allen simulierten Geschwindigkeiten bis etwa $35\,\mathrm{m\,s^{-1}}$ die Messabweichung von 0,5 ‰ einhalten und bei den meisten Geschwindigkeiten Abweichungen im Bereich 0,05 ‰ erzielen. Bei einer Querbewegungskomponente von 1 % kann die Messabweichung von 0,5 ‰ nur noch sporadisch eingehalten werden. [10, S. 35ff]

3.2.1 Querbewegung in der Simulation

Aus den Beobachtungen in der Simulation, die im letzten Absatz geschildert wurden, wird der Schluss gezogen, dass die Ausrichtung der Kamera relativ zur Bewegungsrichtung ein entscheidender Faktor für den Erfolg der Methode ist. Wie dort beschrieben, liegt der Unterschied zwischen Messabweichungen, die die Spezifikation von 0,5 ‰ größtenteils um eine Größenordnung unterbieten können und Abweichungen, die die Spezifikation kaum noch einhalten zwischen Querbewegungsanteilen von 0 bis 1 %.

Abbildung 3.5 zeigt die Abweichungen der gemessenen Geschwindigkeit an drei künstlichen Bildserien, die in Beobachtungszeiträume von je $4096 \cdot T_s$ eingeteilt werden. Dabei liegt die Längsgeschwindigkeit aller Bildserien bei $3\,\mathrm{m\,s^{-1}}$ und die Querbewegungskomponenten bei 0 , 0,5 und 1 %.

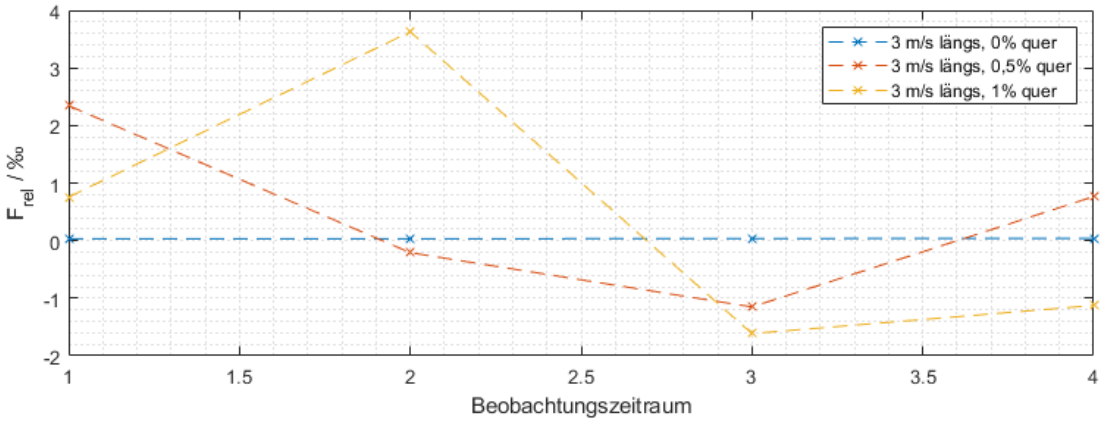

Abbildung 3.5: Einfluss von verschiedenen Querbewegungen auf die Messabweichung in der Simulation

Es zeigt sich, dass die Einzelmessungen mit steigendem Querbewegungsanteil zwischen den Beobachtungszeiträumen stärker schwanken. Während bei der Messung ohne Querbewegung die Abweichungen über alle Beobachtungszeiträume in einer Größenordnung von hundertstel Promille liegen, zeigen die Einzelmessungen mit 0,5 % Querbewegung Abweichungen bis zu etwa ±2 ‰ und die Messungen mit 1 % Querbewegung bis etwa 4 ‰.

Würde eine solche Schwankung zwischen aufeinanderfolgenden Beobachtungszeiträumen in einer Echtbildserie gemessen werden, ließe sich diese Schwankung nicht automatisch auf eine Querbewegung zurückführen: Eine gewisse Schwankung der tatsächlichen Geschwindigkeit ist in der Realität auch bei näherungsweise konstanten Geschwindigkeiten zu erwarten (s. auch Abschnitt 3.2.3). Durch die Verwendung künstlich generierter Bildserien mit konstanter Geschwindigkeit und der Querbewegungskomponente als einzig verändertem Parameter lässt sich der Effekt in diesem Fall aber zuordnen.

3.2.2 Auswertung von Echtbildserien

Erste Serie - Piranha 2

Die erste ausgewertete Serie von Echtbildern stammt aus dem COVIDIS-Projekt und wurde mit einer Kamera vom Typ Piranha 2 bei einer Framerate von 10 kHz aufgezeichnet. Der Simulation in der Vorarbeit wird eine Kamera vom Typ Spyder 3 bei einer Framerate von 60 kHz zugrunde gelegt [10]. Die Kameras haben unterschiedliche Pixelabstände W_{px} von 10 µm (Piranha 2) [24] und 14 µm (Spyder 3) [25]. Hinsichtlich der verwendeten Optik bestehen keine Unterschiede.

Um eine Vergleichbarkeit der gemessenen Geschwindigkeiten im Kontext des verwendeten Verfahrens herzustellen, werden die Frameperioden und Pixelabstände der Kameras ins Verhältnis gesetzt.

Dem Vergleich liegt zu Grunde, dass im zeit- und ortsdiskreten System in Einheiten der jeweiligen Diskretisierungsschritte gerechnet wird, hinsichtlich der Zeit also T_{s} und hinsichtlich des Ortes W_{px}. Bei der Berechnung einer Durchschnittsgeschwindigkeit aus einer Distanz $(n \cdot W_{px})$ und einer Zeitverzögerung $(k \cdot T_s)$ liegt das Ergebnis also zunächst in der Einheit Pixelabstände pro Frameperiode vor.

Der Bereich der akzeptablen Messabweichungen, der in der Simulation der Vorarbeit bis etwa $35\,\mathrm{m\,s^{-1}}$ reicht [10, S. 37], ist bei der betrachteten Echtbildserie mit einem Bereich bis etwa $4\,\mathrm{m\,s^{-1}}$ vergleichbar:

$$35\,\mathrm{m\,s^{-1}} \cdot \frac{10\,\mu\mathrm{m}}{14\,\mu\mathrm{m}} \cdot \frac{10\,\mathrm{kHz}}{68\,\mathrm{kHz}} \approx 3{,}7\,\mathrm{m\,s^{-1}}$$

Die betrachteten Bilder wurden auf der Oberfläche einer Prüfstandrolle aus Aluminium, einerseits auf dem Grundmaterial und andererseits mit verschiedenen Schmirgelpapieren beklebt, jeweils bei Geschwindigkeiten von 0,1 , 0,5 , 1 , 2,5 und $5\,\mathrm{m\,s^{-1}}$ aufgenommen. Diese Geschwindigkeiten entsprechen den Geschwindigkeiten 0,84 , 4,2 , 8,4 , 21 und $42\,\mathrm{m\,s^{-1}}$ in der Simulation.

Abbildung 3.6 zeigt die relativen Messabweichungen der ersten Echtbildserie.

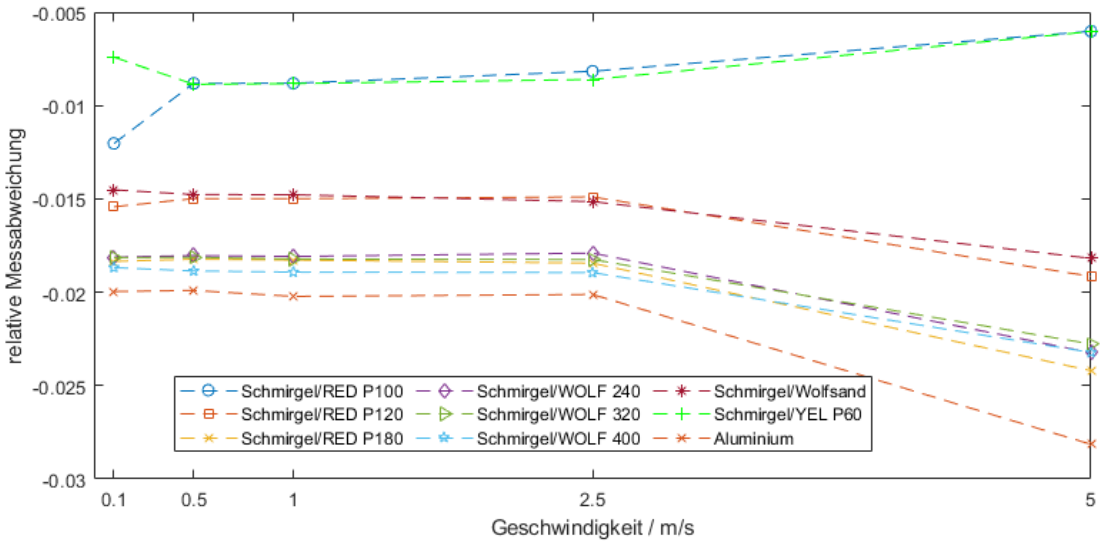

Abbildung 3.6: Messergebnisse für die erste Echtbildserie

In den Ergebnissen sind mehrere Dinge zu beobachten:

- Die Kurven zeigen alle eine Abweichung im Prozentbereich, alle Messungen wurden also mit einem gewissen Offsetfehler gemacht. Dies war zu erwarten, da der angenommene Abbildungsmaßstab der Optik lediglich ein ungefährer Wert ist und im Einzelfall kalibriert werden muss.
- Die Kurven sind untereinander entlang der Ordinate versetzt. Auch dies war zu erwarten, da jeweils Material mit potentiell unterschiedlicher Dicke

auf der Rollenoberfläche befestigt wurde. Dabei können sich von Material zu Material für eine gegebene eingestellte Geschwindigkeit am Teststand unterschiedliche Oberflächengeschwindigkeiten ergeben.

Dass auf der Aluminiumoberfläche die geringste Geschwindigkeit gemessen wurde, erscheint unter diesem Gesichtspunkt plausibel weil es sich dabei um das Grundmaterial des Testrades handelt, das somit den geringsten Umfang hat.

- Die Verläufe der einzelnen Kurven sind zwischen 0,1 und 2,5 $\mathrm{m\,s^{-1}}$ untereinander ähnlich und relativ geradlinig, weichen aber für 5 $\mathrm{m\,s^{-1}}$ vergleichsweise stark vom restlichen Verlauf ab. Dass die Ergebnisse für 5 $\mathrm{m\,s^{-1}}$ abweichen, war aus der Vergleichsgeschwindigkeit von 42 $\mathrm{m\,s^{-1}}$ in der Simulation zu erwarten, da für diese Geschwindigkeit in der Simulation Messabweichungen größer 0,5 ‰ erwartet werden [10, S. 37].

Abbildung 3.7 zeigt die Messreihe mit korrigiertem Offsetfehler.

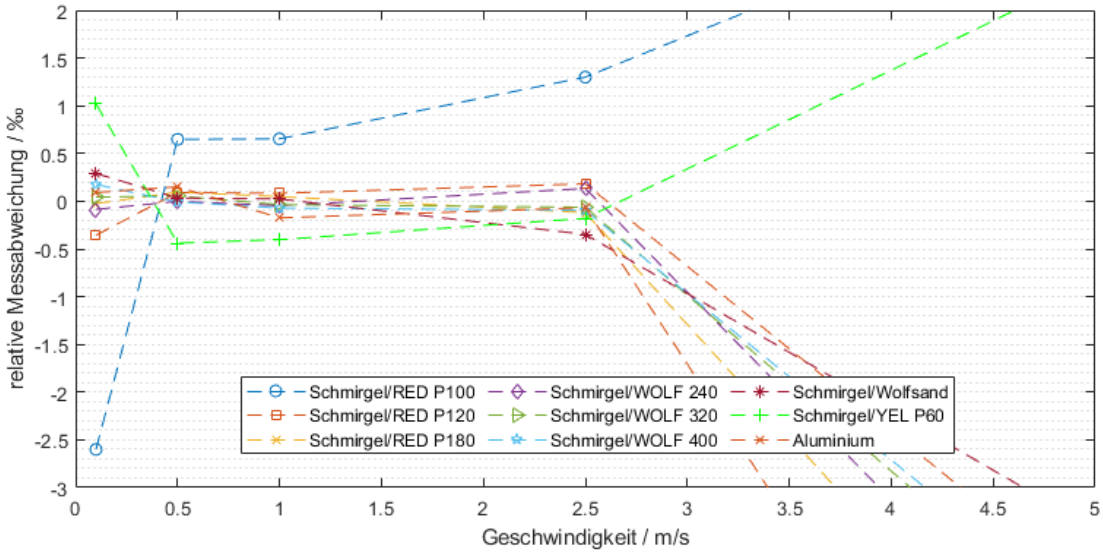

Abbildung 3.7: Offsetkorrigierte Messergebnisse für die erste Bildserie

Ein Großteil der Einzelmessungen im Bereich bis 2,5 $\mathrm{m\,s^{-1}}$ erreicht dabei Abweichungen kleiner 0,5 ‰, was zwar den Spezifikationen des VADER-Systems genügt [26] aber nicht den Simulationsergebnissen aus der Vorarbeit entspricht. Aus dieser Beobachtung wird geschlossen, dass das Simulationsmodell entweder reale

Phänomene nicht berücksichtigt oder die Parameter zwischen Simulation und den Echtbildserien z.B. im Bezug auf eine Querbewegungskomponente voneinander abweichen. Weil die Gründe für die Abweichungen ungeklärt sind, wird eine zweite Messreihe aufgenommen, die im folgenden Abschnitt behandelt wird.

Zweite Serie - Spyder 3

Aufgrund der im letzten Abschnitt beschriebenen Unterschiede zwischen den Parametern der Piranha 2-Kamera und den den Simulationen in der Vorarbeit zugrundeliegenden Modellparameter sowie der daraus resultierenden bedingten Vergleichbarkeit der Ergebnisse wird eine zweite Messreihe mit einer Spyder 3-Kamera aufgezeichnet, die im Folgenden ausgewertet wird.

Aufgrund der Betrachtung in Abschnitt 3.2.1 wird für diese Messreihe der Versuch unternommen, die Kamerazeile im Rahmen der praktischen Möglichkeiten bestmöglich parallel zur Bewegungsrichtung des Messobjekts auszurichten. Dabei wird der in Abbildung 3.8 schematisch gezeigte Ansatz verfolgt.

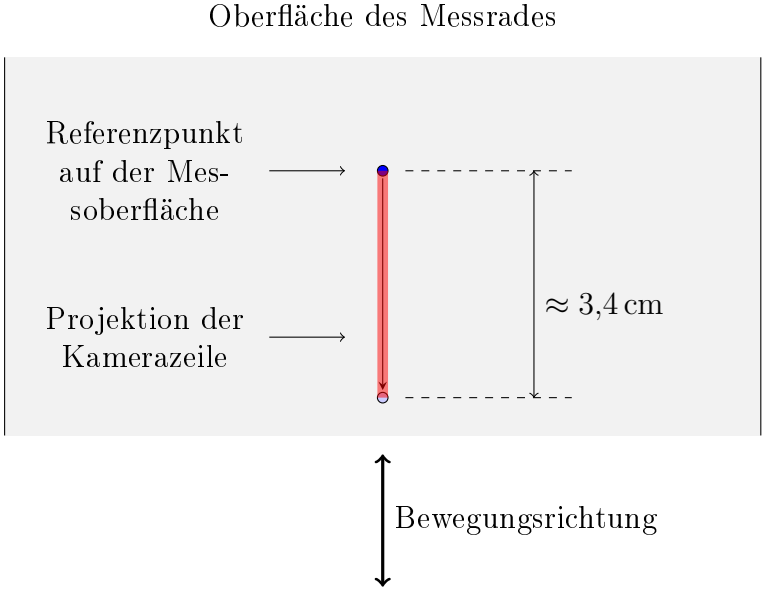

Abbildung 3.8: Vorgehen zur parallelen Ausrichtung der Kamerazeile zur Bewegungsrichtung

Zur Ausrichtung der Kamera wird die CCD-Zeile der Kamera mit einem hellen Licht beschienen, sodass ihre Reflektion durch die Optik auf die Oberfläche des

Messrades projiziert wird. Auf dem Messrad wird ein kleiner Referenzpunkt angebracht, der sich beim Drehen des Messrades über die Projektion der Kamerazeile bewegt. Die Kamera wird so lange gedreht, bis der Referenzpunkt sich beim Drehen des Messrades ohne erkennbare Querverschiebung über die Projektion der Kamerazeile bewegt.

Es wird abgeschätzt, dass die Querbewegung des Referenzpunkts über die gesamte Länge der Projektion der Kamerazeile (etwa 3,4 cm) 0,3 mm nicht überschreitet, was zu einer Abschätzung für die maximale Querbewegung von etwa 1 % führt.

Für die Messreihe werden am Rollenprüfstand konstante Geschwindigkeiten im Bereich zwischen 1 und 6 $\mathrm{m\,s^{-1}}$ eingestellt: Ein Bereich, für den die Simulationen der Vorarbeit Messabweichungen in der Größenordnung von hundertstel Promille vorhersagen [10, S. 37].

Gemessen wird die Durchschnittsgeschwindigkeit über eine Strecke von 1 m, aufgeteilt in Beobachtungszeiträume von $8192 \cdot T_\mathrm{s}$, was den Bedingungen in der Simulation der Vorarbeit entspricht [10, S. 35ff].

Aus technischen Gründen konnte pro Messreihe nur eine begrenzte Anzahl an Bildern aufgezeichnet werden, weshalb die Messreihen für die Geschwindigkeit $1\,\mathrm{m\,s^{-1}}$ nur eine Strecke von etwa 0,7 m auswerten.

Tabelle 3.2 zeigt die Ergebnisse der Messreihe. Die Messungen für $1\,\mathrm{m\,s^{-1}}$ und $2\,\mathrm{m\,s^{-1}}$ liegen relativ weit abseits von denen der restlichen Geschwindigkeiten. Die Messungen wurden mehrmals wiederholt, um die Wahrscheinlichkeit für eine zufällige Abweichung zu senken. Das Muster zeigte sich weiterhin, was für einen systematischen Ursprung der Messabweichung spricht, für den die Untersuchungen in der Vorarbeit keine Erklärung liefern.

Der Korrekturfaktor für den Abbildungsmaßstab wurde aus den Messungen für 3 bis $6\,\mathrm{m\,s^{-1}}$ berechnet.

$\overline{v}$ / $\mathrm{m\,s^{-1}}$	Strecke / m	F_{rel} / ‰ Serie			
		1	2	3	4
1	0,662	6,4440	4,9802	5,6777	5,5943
2	1	1,4531	1,3874		
3	1	0,0196	0,0402		
3,888	1	-0,0065	-0,1927		
4	1	-0,0096	0,0028		
4,337	1	0,1082	0,0800		
5	1	0,0279	0,0044		
6	1	-0,0612	-0,0130		

Tabelle 3.2: Messergebnisse der zweiten Echtbildserie

Abbildung 3.9 stellt die Messergebnisse grafisch dar.

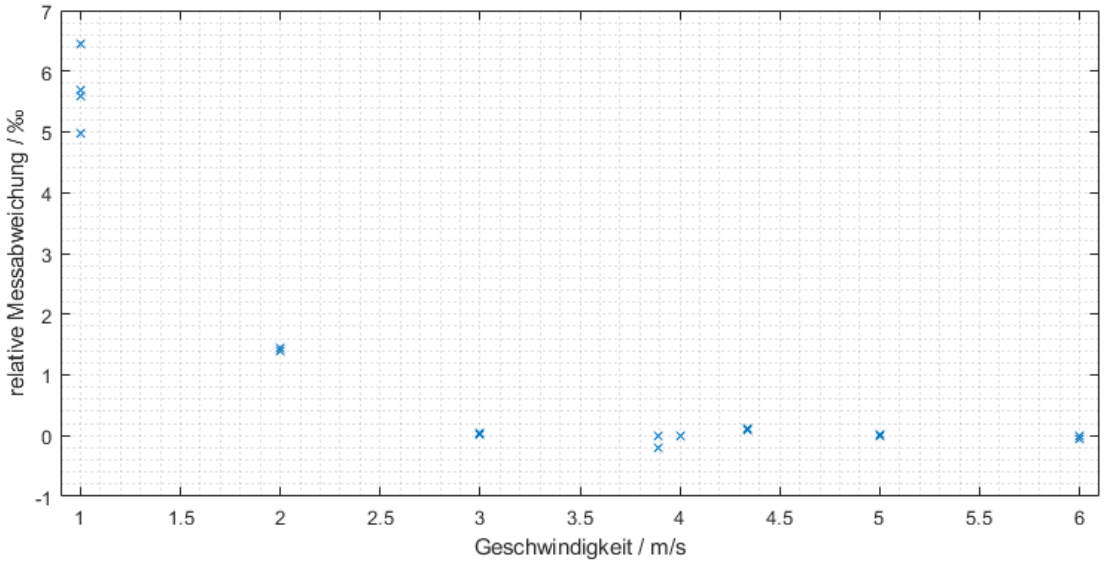

Abbildung 3.9: Relative Messabweichungen der zweiten Echtbildserie

Die erzielten Messabweichungen für die Geschwindigkeiten ab $3\,\mathrm{m\,s^{-1}}$ liegen im Bereich bis etwa 0,2 ‰, was zwar die Spezifikation des VADER-Systems erfüllt, aber nicht durchgängig den Erwartungen aus der Vorarbeit entspricht.

Um eine trotz der oben beschriebenen Ausrichtung der Kamera möglicherweise auftretende Querbewegungskomponente als Ursache für die Abweichungen auszuschließen, wird analog zu Abschnitt 3.2.1 eine Untersuchung aufeinanderfolgender Beobachtungszeiträume von je $4096 \cdot T_\mathrm{s}$ für die Geschwindigkeit $3\,\mathrm{m\,s^{-1}}$ durchgeführt:

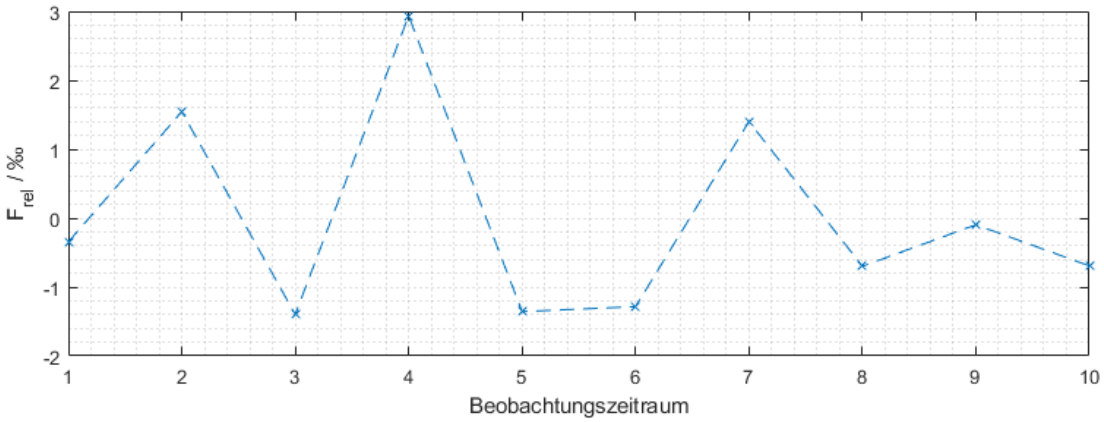

Abbildung 3.10: Geschwindigkeitsmesswerte für aufeinanderfolgende Beobachtungszeiträume

Die Schwankungen zwischen den einzelnen Beobachtungszeiträumen ähneln der Simulation in Abbildung 3.5 für eine Querbewegung zwischen 0,5 % und 1 %. Andererseits könnten die Schwankungen auch tatsächlich auftretende Variationen der Geschwindigkeit am Teststand wiedergeben (s. auch Abschnitt 3.2.3). Weiterhin wurden alle Messreihen mit der gleichen Ausrichtung der Kamera aufgezeichnet und die Ergebnisse für die Geschwindigkeiten zwischen 3 und $6\,\mathrm{m\,s^{-1}}$ sind besser als die in der Simulation in der Vorarbeit für einen Querbewegungsanteil von 1 % vorhergesagten.

Insgesamt kann anhand der vorliegenden Informationen eine Querbewegung in der Messreihe weder ausgeschlossen noch bestätigt werden. Darüber hinaus zeigen sich in den Echtbildern Phänomene, die im Modell der Bilderzeugung und damit auch in der Simulation in der Vorarbeit vollständig unberücksichtigt geblieben sind und die gemeinsam mit einer möglichen Querbewegungskomponente zu weiteren bislang unbekannten Effekten führen könnten.

Die Weiterentwicklung des Bildgenerators ist nicht Bestandteil der vorliegenden Arbeit; stattdessen werden im folgenden Abschnitt einige die Messung beeinflus-

sende aber im Bildgenerator unberücksichtigte Faktoren als Ansatzpunkte für mögliche Folgeprojekte zusammengefasst.

3.2.3 Realismus des zugrundegelegten Modells

Konstanz der Geschwindigkeit

Für das Vorprojekt war die Betrachtung nicht beschleunigter Geschwindigkeiten ausdrücklich vorgegeben [23]. Auch der Bildgenerator aus dem COVIDIS-Projekt generiert Bilder mit konstanten Geschwindigkeiten [27].

Der im zweiten Teil der Arbeit entwickelte Korrelationsalgorithmus wird verwendet, um für eine Bildserie mit $\bar{v} = 1\,\mathrm{m\,s^{-1}}$ aus Abschnitt 3.2.2 eine Abschätzung für den Verlauf der Momentangeschwindigkeit zu ermitteln. Im Gegensatz zu den in diesem Kapitel behandelten zeitfunktionsbasierten Algorithmen, die das Signal über Beobachtungszeiträume zwischen $4096 \cdot T_\mathrm{s}$ und $8192 \cdot T_\mathrm{s}$ betrachten, liefert der Korrelationsalgorithmus in Abbildung 3.11 eine Durchschnittsgeschwindigkeit über Zeiträume von jeweils etwa $75 \cdot T_\mathrm{s}$:

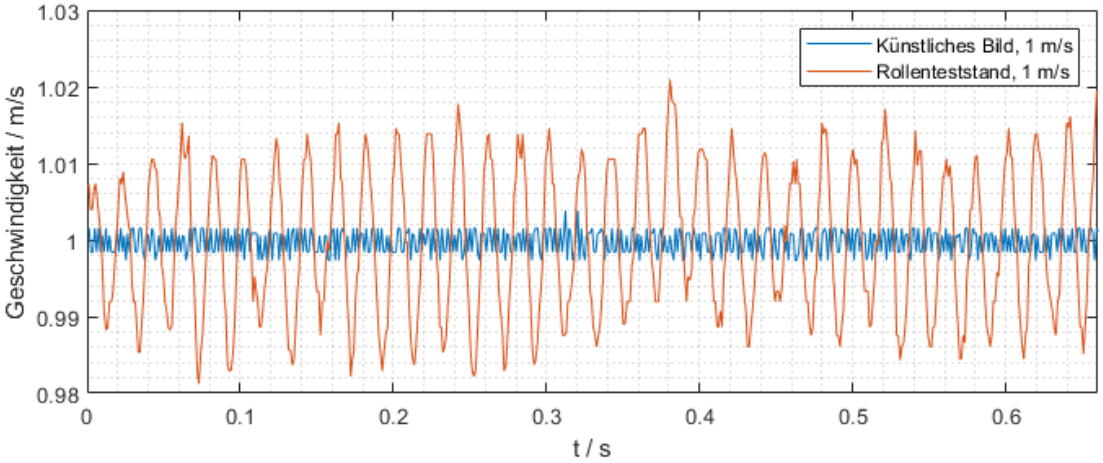

Abbildung 3.11: Kurzzeit-Durchschnittsgeschwindigkeit des Rollenteststands

Es zeigt sich, dass die Momentan-Oberflächengeschwindigkeit des Prüfstandes zeitweise um bis zu 2 % vom Mittelwert abweicht. Ob diese Schwankung einen Einfluss auf die Messergebnisse hat, kann an dieser Stelle mangels eines entsprechenden Modells nicht eruiert werden.

Arbeitshypothese 4 *Um reale Bewegungsvorgänge zu modellieren ist die Annahme einer im physikalischen Sinne konstanten Geschwindigkeit nicht zielführend. Es ist anzunehmen, dass jede in der Realität geregelte Geschwindigkeit eine gewisse Schwankung aufweist, wie das auch bei dem in diesem Abschnitt betrachteten Teststand der Fall ist. Die Ermittelung realistischer Werte für solche Schwankungen und deren Berücksichtigung im Testbildgenerator kann die modellbasierte Untersuchung bislang nicht verstandener Effekte ermöglichen.*

Stillstehende Bildmerkmale

Abbildung 3.12 zeigt die durchschnittlichen Helligkeiten aller Pixel in verschiedenen Bildserien.

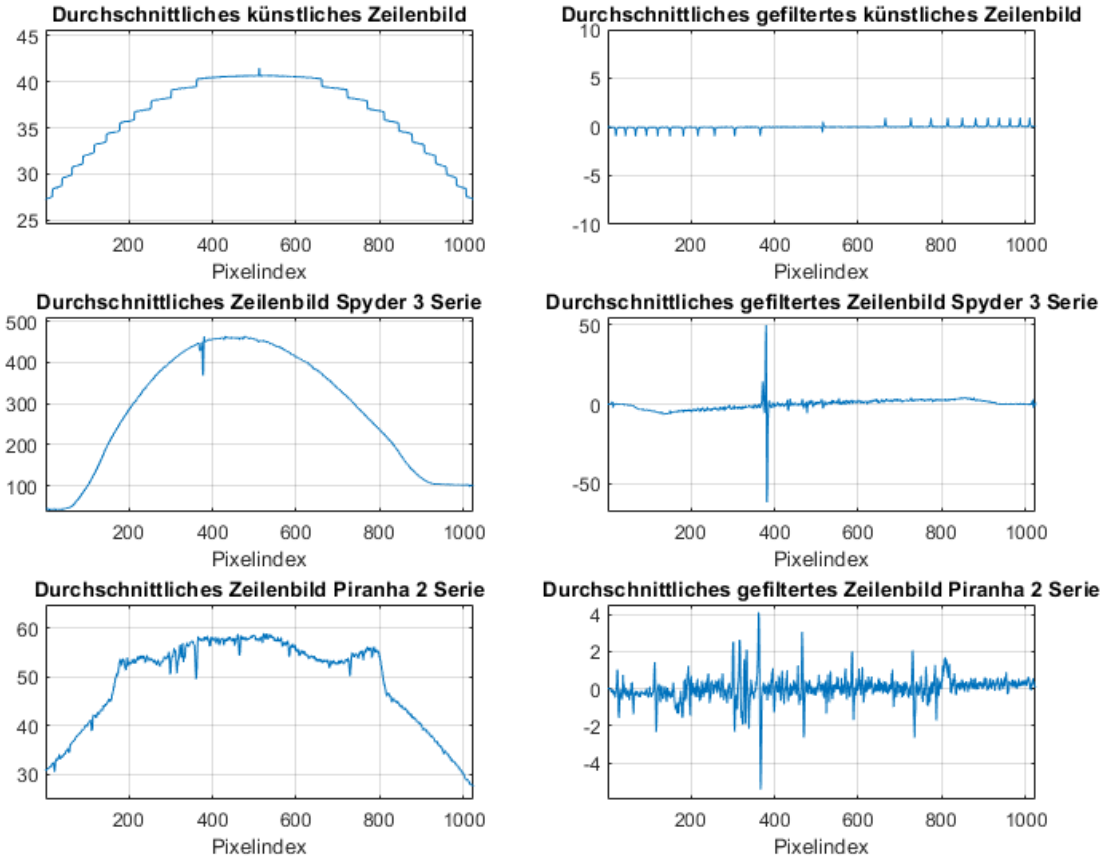

Abbildung 3.12: Vergleich durchschnittlicher Zeilenbilder des Bildgenerators und aus den zwei Echtbildserien

Einerseits ist eine Bildserie eines durch den Bildgenerator künstlich generierten

Bewegungsvorganges und andererseits Bildserien aus den Abschnitten 3.2.2 und 3.2.2, die mit den Piranha 2 bzw. Spyder 3-Kameras aufgezeichnet wurden, gezeigt. In der rechten Spalte dargestellt sind die durchschnittlichen Zeilenbilder nach der Filterung durch das FIR-Rohbildfilter.

Das künstliche Zeilenbild weist abgesehen von durch Quantisierungsfehler in der Bilderzeugung erzeugte Stufen eine Parabelform ohne besondere Merkmale auf. Bei der Spyder 3-Kamera ist eine leicht asymmetrische Parabel mit relativ scharfen Knicken am Anfang und Ende des Bildes zu erkennen. Außerdem zeigt sich ein Bereich von Pixeln, der systematisch geringere Helligkeiten misst. Die Piranha 2-Kamera zeigt mehrere Bereiche mit systematischen Helligkeitsfehlern.

Weil der Ursprung dieser Artefakte unbekannt ist, werden sie unter dem Sammelbegriff stillstehende Bildmerkmale zusammengefasst.

Um den Einfluss dieser Phänomene auf das DFT-interpolierte Kreuzkorrelationsverfahren für Ortsfilterfunktionen abzuschätzen, wird eine künstlich generierte Bildserie als Grundlage verwendet und diese mit den ermittelten stillstehenden Bildmerkmalen überlagert. Es resultieren drei Bildserien, die alle die gleiche künstlich erzeugte Bewegung enthalten und sich nur hinsichtlich der stillstehenden Bildmerkmale unterscheiden.

Szenario	rel. Messabweichung / ‰
Künstliche Bildserie	0,039
Künstliche Bildserie überlagert mit den Bildartefakten aus der Piranha 2-Serie	0,15
Künstliche Bildserie überlagert mit den Bildartefakten aus der Spyder 3-Serie	0,25

Tabelle 3.3: Auswirkung von stillstehenden Bildmerkmalen auf die Messabweichung

Tabelle 3.3 zeigt, dass allein das Hinzufügen von stillstehenden Bildmerkmalen in eine Bildserie die Messabweichung um etwa eine Größenordnung vergrößern kann.

Arbeitshypothese 5 *Bei allen betrachteten Echtbildserien zeigten sich stillstehende Bildmerkmale unterschiedlicher Ausprägung, die wie in diesem Abschnitt gezeigt das Messergebnis beeinflussen können. Die Untersuchung der Ursprünge dieser Artefakte z.B. in der Optik, der Beleuchtung oder der Kamera selbst und eine entsprechende Modellierung dieser Phänomene kann zu realitätsnäheren Simulationen führen.*

3.2.4 Zwischenfazit und Ausblick

Eine weiterführende Untersuchung des Korrelationsverfahrens für Ortsfilterfunktionen ist nicht Teil dieser Arbeit. Zusammenfassend wird festgestellt, dass ein Teil der einzelnen Messungen aus den zwei Echtbildserien in den Abschnitten 3.2.2 und 3.2.2 zwar Messabweichungen von 0,5 ‰ erreichen oder unterschreiten kann, aber auch Phänomene beobachtet werden, die in der Simulation in der Vorarbeit nicht vorhergesehen wurden und sich mit dem bestehenden Modell der Bilderzeugung im Bildgenerator aus dem COVIDIS-Projekt auch nicht reproduzieren lassen.

In den Abschnitten 3.2.3 und 3.2.3 werden zwei real auftretende Phänomene mit der Simulation verglichen und Arbeitshypothesen aufgestellt, die als Grundlage für künftige Projekte zur Verbesserung des Bildgenerators dienen können.

4 Korrelation von Zeilenbildern

Die im Kapitel 3 besprochenen Algorithmen sind nach der in Abbildung 1.1 vorgenommenen Klassifikation den zeitfunktionsbasierten Verfahren zuzuordnen. Zwar werten die anderen Verfahren in dieser Kategorie auf die ein oder andere Weise die Grundfrequenz einer oder mehrerer Zeitfunktionen aus und unterscheiden sich in dieser Hinsicht von den zuletzt vorgestellten Korrelationsverfahren, für die der Zusammenhang zwischen Geschwindigkeit und Frequenz unerheblich ist, das Argument der berechneten Kreuzkorrelationsfunktionen ist aber trotzdem eine Zeitgröße, die wie beschrieben in eine Geschwindigkeit überführt wird.

Weil das Ziel der Arbeit die Messung von Längen und Wegfortschritten ist, wird es als sinnvoll erachtet, direkt Größen in der Dimension einer Strecke zu bestimmen, anstatt zunächst eine Geschwindigkeit zu ermitteln, die anschließend aufintegriert werden muss. Dieser Gedanke führt zur Kreuzkorrelation zwischen Ortsfunktionen.

Ein Vorteil der Auswertung von Ortsfilterfunktionen z.B. im COVIDIS besteht darin, dass jeweils ein gesamtes Zeilenbild mit 1024 oder mehr Pixeln zu einem Eintrag in der Ortsfilterfunktion verrechnet wird, wodurch weniger Daten verarbeitet werden müssen [1, S. 74]. Wird statt eines digitalen Ortsfilters ein physikalisches Ortsfilter oder wie im Falle der LDV-Systeme ein Interferenzmuster verwendet, so ist auch keine Vorverarbeitung der Bilddaten mehr nötig, was zu einer weiteren Vereinfachung des Systems führt.

Das COVIDIS-System hat mit dem Blockmatching-Algorithmus eine Auswertung durch Kreuzkorrelation zur Feststellung von Stillstand bzw. zur Messung von Wegfortschritten bei Kriechgeschwindigkeiten eingeführt. Dies wird dadurch ermöglicht, dass im Falle des Stillstandes oder einer sehr geringen Geschwindigkeit die Auswertung der Zeilenbilder nur bei einem Bruchteil der Framerate erfolgen

muss, konkret ist beispielsweise die Berechnung einer Kreuzkorrelation alle 2 ms ausreichend. [1, S. 70]

Wie in der Einleitung beschrieben, ist es das Ziel dieser Arbeit, einen Algorithmus zu entwickeln, der die Zeilenbilder der Kamera ohne vorherige Reduktion der Framerate in Echtzeit verarbeiten kann. Im Folgenden werden verschiedene Korrelationsalgorithmen in MATLAB implementiert und zunächst an künstlichen Bildserien simuliert, um verschiedene Ansätze zu vergleichen. Die spätere Echtzeitfähigkeit steht dabei noch nicht der Fokus der Modellierung.

Obwohl der zur Verfügung stehende Bildgenerator aus in Abschnitt 3.2.3 genannten Gründen als unzureichende Approximation der Realität betrachtet wird, um aus den generierten Bildern quantitative Aussagen über zu erwartende Messabweichungen abzuleiten, bietet er die Möglichkeit, den Faktor der Querbewegung isoliert zu betrachten und die skizzierten Algorithmen auf ihre grundsätzliche Funktionsfähigkeit zu prüfen.

In Kapitel 6 wird abschließend auf der Hardware geprüft, inwieweit die hier entworfenen Algorithmen in eine echtzeitfähige Implementierung überführt werden können, bevor der finale Algorithmus an realen Messreihen getestet und die Parameter optimiert werden.

4.1 Vorüberlegungen und Anforderungen

SFV- und LDV-Algorithmen funktionieren unterhalb einer minimal darstellbaren Geschwindigkeit nicht mehr [1, S. 69]. Darüber hinaus stellen kurze Strecken für diese Systeme, die auf einer Frequenzauswertung basieren, ein Problem dar, weil wegen der Unschärferelation zwischen Zeit- und Frequenzbereich ein kurzer (scharfer) zeitlicher Vorgang zu einer unscharf definierten Frequenz führt [14, S. 51].

Unter konstanten Bedingungen ist also ein langer Beobachtungszeitraum vorteilhaft für die genaue Ermittelung der Signalfrequenz. Im Falle einer beschleunigten Bewegung kommt es allerdings zu einem sogenannten Schleppfehler, der besonders bei Längenmessungen problematisch ist, weil die Grundfrequenz des Signals über den Beobachtungszeitraum variiert [1, S. 46].

Weil das Ziel dieser Arbeit die Entwicklung eines Einzelalgorithmus und nicht eines vollständigen Sensorsystems als Konkurrenzprodukt zu SFV- oder LDV-Sensoren ist, werden die oben genannten Schwächen der bestehenden Systeme als Ansatzpunkte für den zu entwickelten Algorithmus verwendet. Dadurch könnte das Resultat der Arbeit in einem Messsystem die bestehenden Algorithmen sinnvoll ergänzen.

Die folgenden Punkte dienen daher als Eckpunkte für den Entwurf des Algorithmus:

- Messung von zurückgelegten Strecken im Bereich 1 m und darunter von Stillstand zu Stillstand.
- Daraus folgend: Beschleunigungs- und Bremsphasen innerhalb relativ kurzer Strecken.
- Niedrige Geschwindigkeiten: Da Beschleunigungs- und Bremsvorgänge innerhalb der Messstrecke stattfinden müssen, lassen sich keine beliebig hohen Geschwindigkeiten erreichen.
- Nicht-konstante Geschwindigkeiten sind ein zentraler Bestandteil der Messung. Zum einen während der Beschleunigungs- und Bremsphase als auch in der eigentlichen Bewegungsphase und bei nach dem Abbremsen entstehenden Ruckel- und Schwingeffekten. Dies trägt zum Einen der Beobachtung, dass z.B. die Geschwindigkeit des Rollenprüfstandes im kontinuierlichen Betrieb schwankt, als auch der Überlegung, dass ähnliche Effekte auch in einem Industrieprozess zu erwarten sind, Rechnung.

Weiterhin wird festgelegt, dass der Algorithmus in single precision (SP) Fließkommaarithmetik nach IEEE 754 [28] implementiert wird. Der hauptsächliche Grund hierfür ist praktischer Natur: Im Vorgriff auf Kapitel 6 werden die im Folgenden skizzierten Algorithmen im späteren Verlauf händisch in C implementiert.

Während Software wie MATLAB und Simulink spezielle Tools zur Auslegung von fixed-point Signalverarbeitungsketten anbieten [29], wird die Handhabung von Fließkommazahlen bei der händischen Implementierung als einfacher und wesentlich weniger fehleranfällig betrachtet als die manuelle Behandlung von Zahlen in fixed-point-Darstellung und die Festlegung der entsprechenden Skalierungen.

4.2 Grundlage der Algorithmen

4.2.1 Blockmatching-Prinzip

Bevor die Algorithmen genauer spezifiziert werden, wird festgelegt, dass das grundlegenden Prinzip des Blockmatching-Algorithmus aus dem COVIDIS übernommen wird.

Wird beim Blockmatching zwischen zwei Bildern keine Verschiebung festgestellt, entweder weil es tatsächlich keine Verschiebung gibt oder diese so klein ist, dass sie nicht erkannt wird, wird das chronologisch ältere Bild als Referenz behalten und weiter in der Zukunft nach möglichen Verschiebungen gesucht. Der Hintergrund ist die Überlegung, dass eine kleine, unterschwellige Bewegung zwischen zwei Bildern nicht verloren geht, wenn ein Bild so lange behalten wird, bis zu diesem Bild mindestens eine Verschiebung von einem Pixel erkannt wird [1, S. 70]

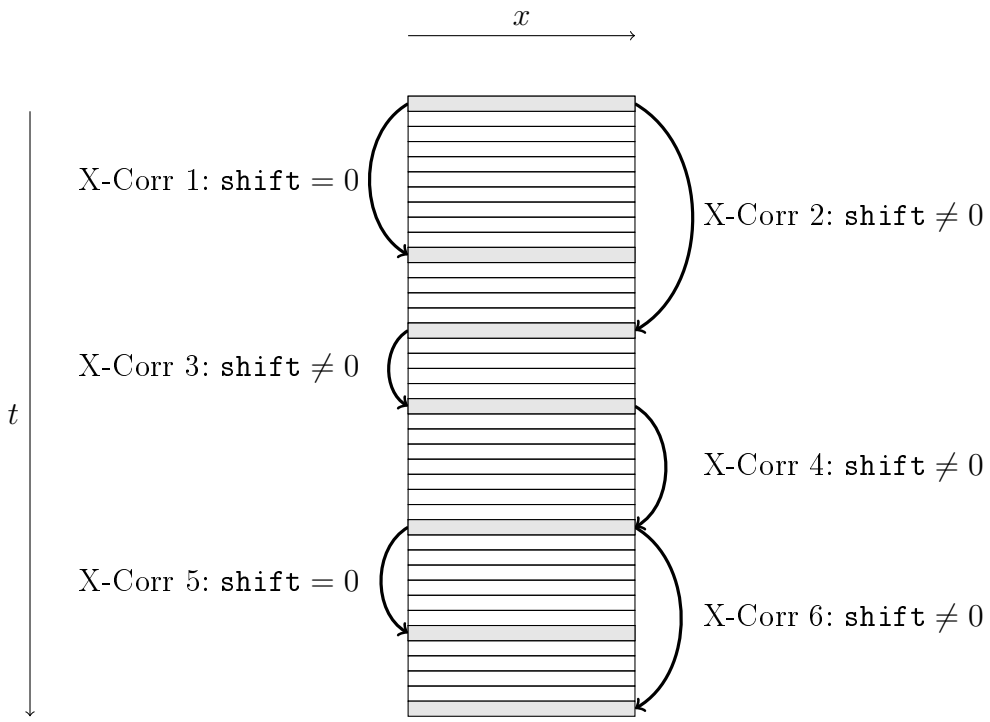

Abbildung 4.1: Blockmatching-Prinzip

Ein Nebeneffekt des Blockmatching-Prinzips ist es, dass das Ende des letzten Korrelationsvorgang immer der Anfang des nächsten Korrelationsvorganges ist,

sodass kein Wegfortschritt verloren geht. Diese Idee ist in Abbildung 4.1 verdeutlicht.

4.2.2 Filterung

Abbildung 4.2 zeigt zwei Zeilenbilder mit einem zeitlichen Abstand von $20 \cdot T_\mathrm{s}$ vom Rollenteststand bei einer Geschwindigkeit von etwa $3\,\mathrm{m\,s^{-1}}$ sowie die diskrete Kreuzkorrelationsfunktion zwischen ihnen.

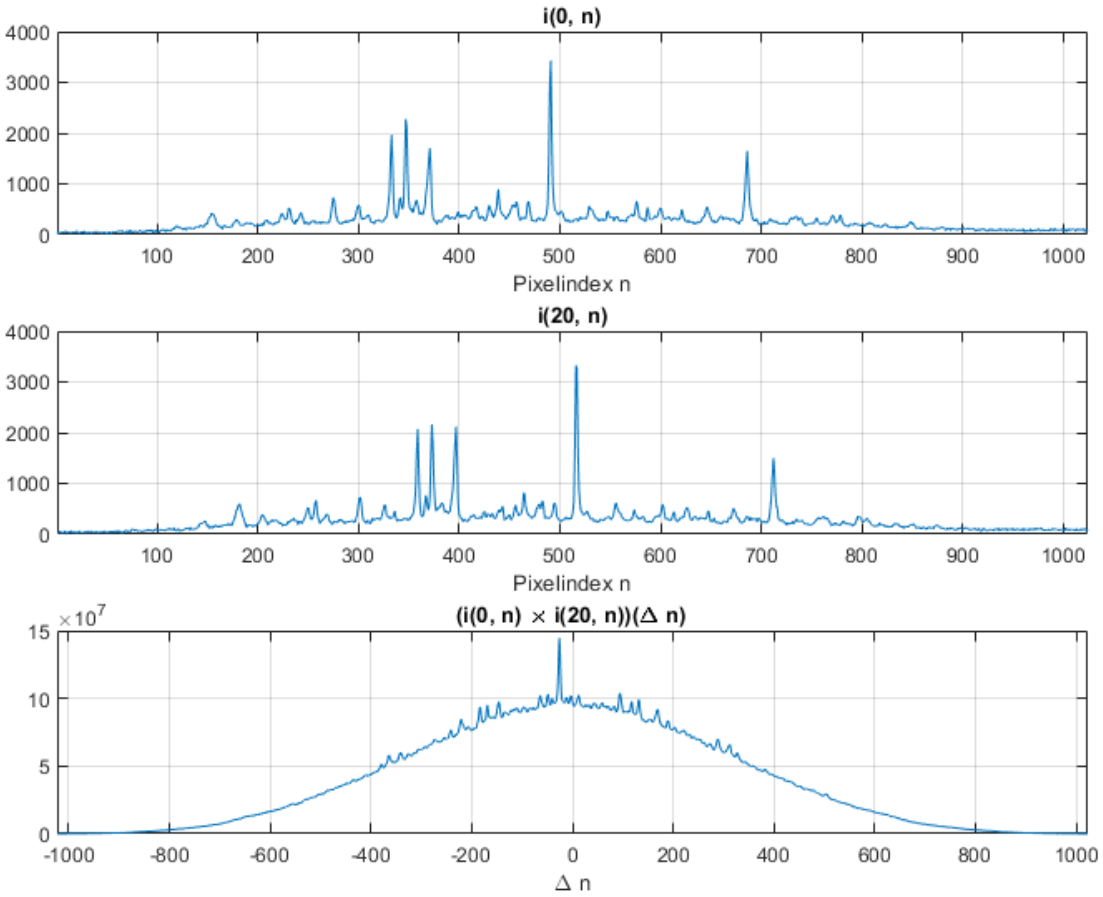

Abbildung 4.2: Zwei Echtbilder und die entsprechende Kreuzkorrelationsfunktion

Mit den Kameraparametern lässt sich die Geschwindigkeit umrechnen: $3\,\mathrm{m\,s^{-1}} \approx 1{,}3\,\mathrm{W_{px}T_s^{-1}}$, womit sich mit dem zeitlichen Abstand von 20 Frameperioden zwischen den Bildern für die erwartete Ortsverschiebung $|\Delta n| = 20 \cdot 1{,}3 = 26$ ergibt. Diese Verschiebung ist zum einen beim Vergleich der

Muster in den zwei Zeilenbildern und zum anderen in der Lage des Maximums der Kreuzkorrelationsfunktion erkennbar.

Die Grundform der Kreuzkorrelationsfunktion verläuft beinahe dreieck- oder parabelförmig mit dem Maximum in der Nähe der Verschiebung $\Delta n = 0$. Dabei handelt es sich nicht um einen Nebeneffekt des gesuchten Korrelationspeaks bei $\Delta n = -26$, wie Abbildung 4.3 an der Kreuzkorrelationsfunktion zwischen zwei nicht in Zusammenhang stehenden Zeilenbildern, die ein ähnliches parabelförmiges Aussehen hat, veranschaulicht.

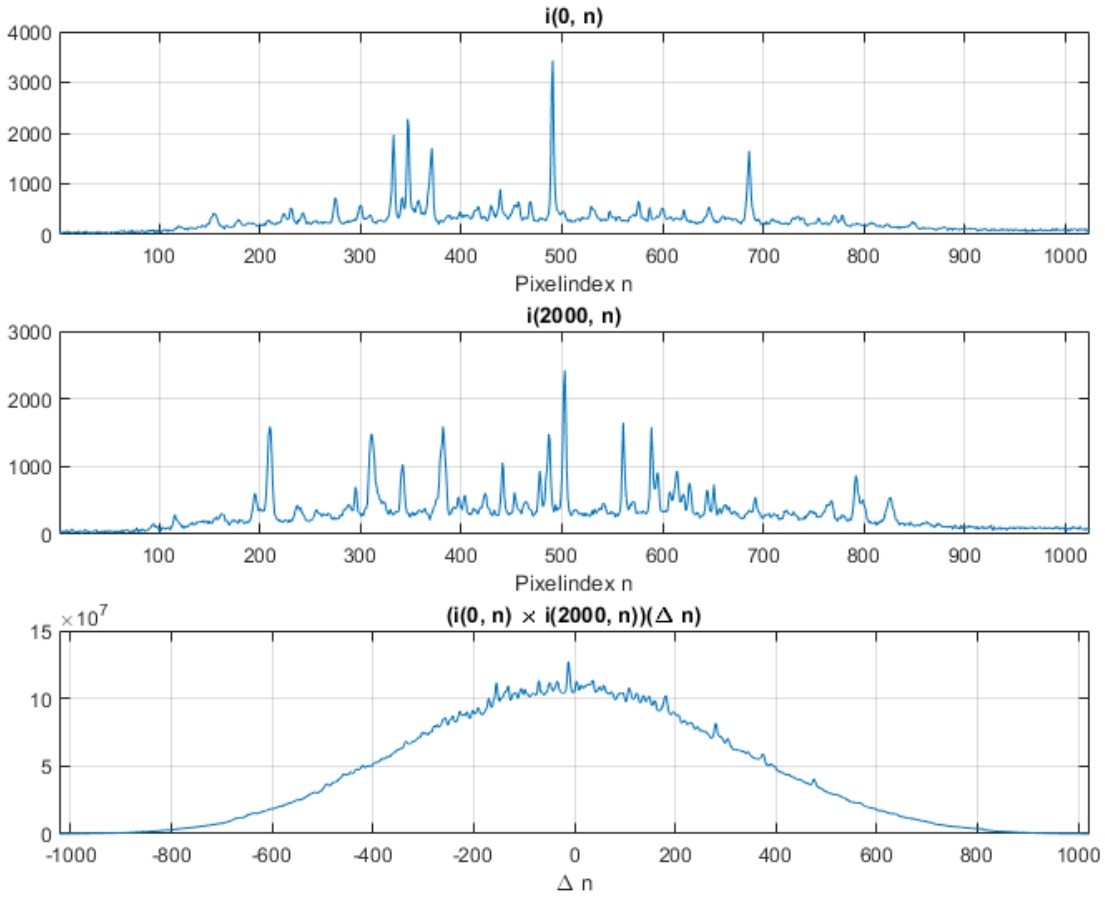

Abbildung 4.3: Zwei Echtbilder ohne korrelierende Oberflächenmerkmale und die entsprechende Kreuzkorrelationsfunktion

Obwohl die Oberflächenmerkmale der in Abbildung 4.3 betrachteten Abschnitte des Messobjekts nicht korrelieren, tun dies die stillstehenden Bildmerkmale, auf die bereits in Abschnitt 3.2.3 eingegangen wurde. Effekte der Beleuchtung und

Pixelfehler stehen still und erzeugen so tendenziell einen Peak bei der Verzögerung $\Delta n = 0$.

Weiterhin haben die Zeilenbilder nicht nur durch die Beleuchtung einen positiven DC-Anteil. Da die Helligkeitswerte der einzelnen Pixel bei der verwendeten Kamera nur zwischen 0 und 4095 liegen können, hätte selbst ein unbelichtetes Bild, das nur das Grundrauschen der Kamera zeigt, einen positiven DC-Anteil.

Unter diesem Umstand ist nämlich jedes Produkt $i(k_1, n_1) \cdot i(k_2, n_2)$ positiv. Das wiederum führt dazu, dass die Summe solcher Produkte mit der Anzahl der Summanden größer wird. Bei der diskreten Kreuzkorrelation werden die meisten Produkte summiert, wenn die einzelnen Funktionen ohne Verschiebung ($\Delta n = 0$) verglichen werden (vgl. Gleichung 6.1).

Die geschilderten Beobachtungen führen zum FIR-Filter, wie es in den COVIDIS- und VADER-Systemen zur Vorverarbeitung der Zeilenbilder verwendet wird, als mögliche Lösung für den DC-Anteil der Kreuzkorrelationsfunktion. Dieses Filter wurde entworfen, um die Einflüsse der Beleuchtung zu unterdrücken und ein möglichst gleichanteilsfreies Bild zu erhalten, das aber noch alle Information über die Bewegung beinhaltet [26, S. 16].

Abbildung 4.4 zeigt die Bilder aus Abbildung 4.2 in gefilterter Form und die Kreuzkorrelationsfunktion der gefilterten Bilder.

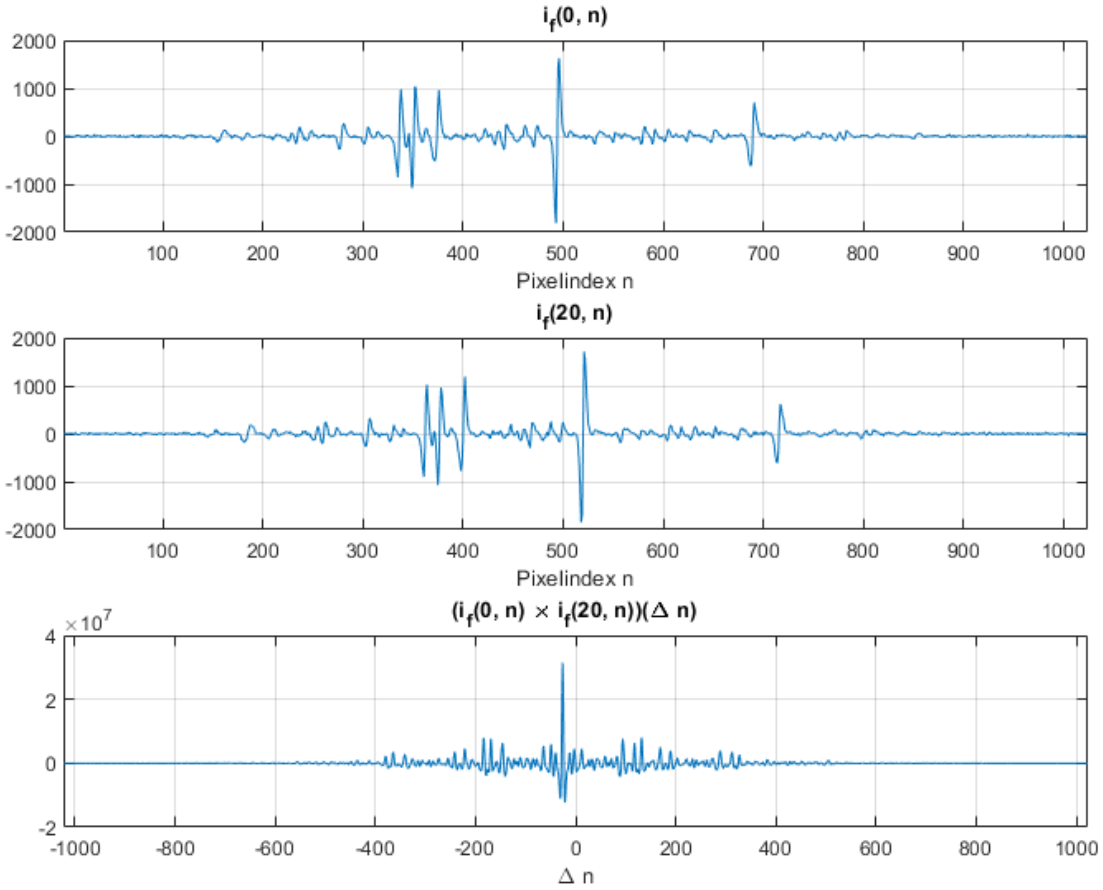

Abbildung 4.4: Zwei gefilterte Echtbilder und die entsprechende Kreuzkorrelationsfunktion

Die gefilterten Bilder enthalten wie hier beispielhaft gezeigt anstatt der ausschließlich positiven Ausschläge der Oberflächenmerkmale im Rohbild symmetrische Ausschläge, wodurch der DC-Anteil verschwindet.

Die Verschiebung von $\Delta n = -26$ ist aber trotzdem weiterhin sowohl in den Mustern der gefilterten Bilder als auch in der Kreuzkorrelationsfunktion erkennbar.

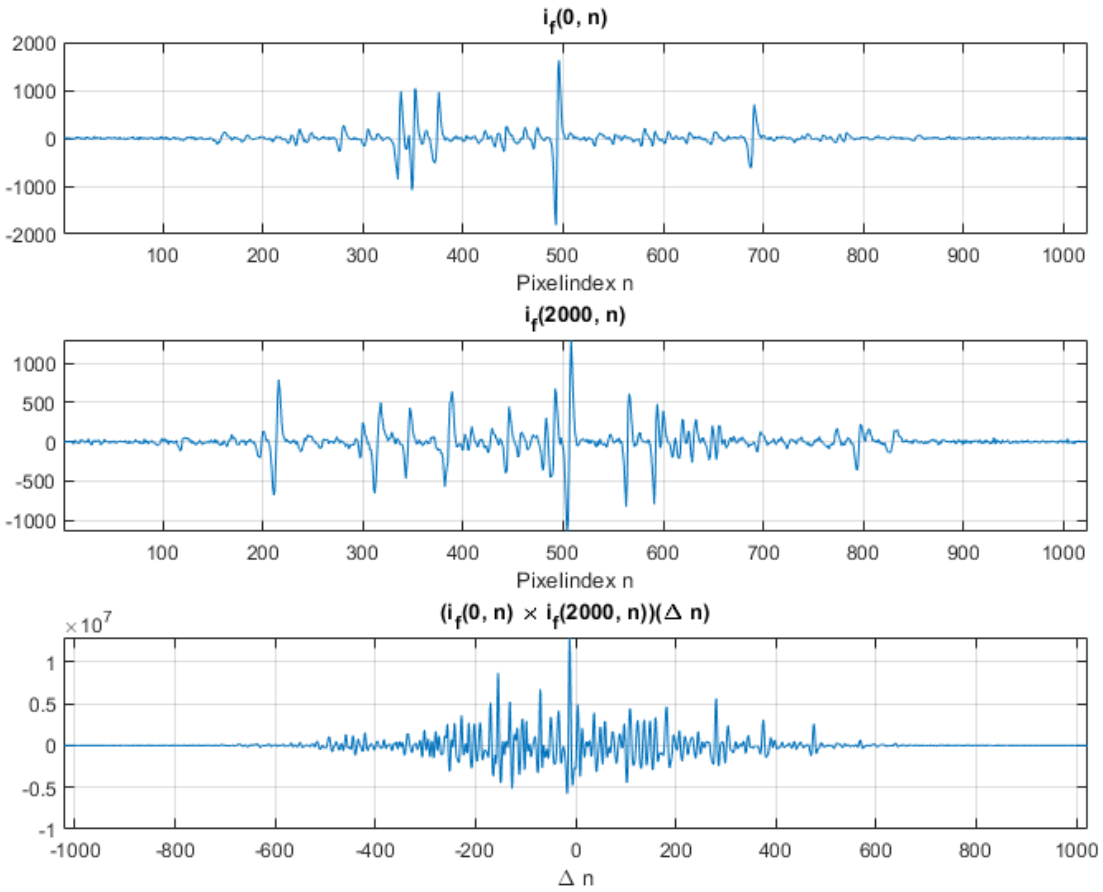

Abbildung 4.5: Zwei gefilterte Echtbilder ohne korrelierende Oberflächenmerkmale und die entsprechende Kreuzkorrelationsfunktion

Abbildung 4.5 zeigt die zwei Zeilenbilder aus Abbildung 4.3 in gefilterter Form und die resultierende Kreuzkorrelationsfunktion.

Sowohl bei den Bildern mit Bezug als auch bei denen ohne Bezug zueinander zeigt sich weiterhin eine höhere Aktivität der Kreuzkorrelationsfunktion im Bereich kleiner Verzögerungen als weiter außerhalb, auch wenn der parabelförmige DC-Anteil entfernt wurde. Hierbei handelt es sich um einen Nebeneffekt der Ausleuchtung der Bilder: Die Pixel 1-200 und 800-1024 enthalten verglichen mit den innen liegenden Pixeln kaum noch Bildinformation.

Sobald eine gewisse Verschiebung entweder in positive oder negative Richtung überschritten wird, werden die aktiven inneren Pixel nicht mehr miteinander sondern nur noch mit den eher inaktiven äußeren Pixeln korreliert, wodurch die Ausschläge der Kreuzkorrelationsfunktion kleiner werden.

4.2.3 Bestimmung der Verschiebung

Als grundsätzliches Mittel zur Bestimmung der gesuchten Verschiebung wird wie in den Algorithmen aus Kapitel 3 die Suche nach einem globalen Maximum der Kreuzkorrelationsfunktion gewählt. Dem liegt zunächst die Annahme zu Grunde, dass korrelierende Abschnitte der zwei Zeilenbilder, wenn sie durch Verschiebung zur Deckung gebracht werden, eine so große Signalenergie miteinander bilden, dass der entstehende Peak alle anderen Peaks, z.B. von stillstehenden Bildmerkmalen oder zufälligen Ähnlichkeiten, übertrifft.

Dass die Suche nach einem globalen Maximum immer erfolgreich ist, sobald die Zeilenbilder korrelierende Abschnitte enthalten, erscheint aus den im letzten Abschnitt genannten Gründen unwahrscheinlich, denn die größeren Verzögerungen sind aus zweierlei Gründen schlechter auswertbar: Einerseits werden schon durch die Verschiebung nur noch kürzere Signalabschnitte miteinander korreliert und zum anderen werden aktive Bereiche mit inaktiven korreliert.

Würde bei der Auswertung der Kreuzkorrelationsfunktion Zusatzinformation zur Position eines gesuchten Peaks vorliegen, würde dies die Anforderungen an den Peak reduzieren: Ein globales Maximum wäre nicht mehr notwendig, sondern lediglich ein Maximum innerhalb des eingeschränkten Suchbereichs.

Dieser Ansatz wird beispielsweise im Korrelationsalgorithmus für Ortsfilter eingesetzt, um mit Hilfe eines sog. Grobfilterpaares eine Abschätzung zur Geschwindigkeit durchzuführen, durch die anschließend die Peaksuche beim Feinfilterpaar auf einen kleinen Bereich eingeschränkt werden kann. Darüber hinaus wurde bereits in der entsprechenden Vorarbeit auf die Möglichkeit hingewiesen, Information zwischen verschiedenen Algorithmen innerhalb eines Messsystems auszutauschen, um auf diese Weise eine Vorselektion durchzuführen. [10]

Im vorliegenden Fall der Korrelation von Zeilenbildern wurden die Bedingungen allerdings so gewählt, dass diese Vorselektion nicht sinnvoll durch Daten aus einem bestehenden Algorithmus durchgeführt werden könnte: Auf Strecken unterhalb eines Meters ist der Messfehler z.B. des COVIDIS nicht spezifiziert und generell können die Verfahren zur Geschwindigkeitsabschätzung erst nach Erfassung eines gewissen Beobachtungszeitraumes eine sinnvolle Ausgabe erzeugen. [1]

Letzteres ist für unbeschleunigte Bewegungsvorgänge, wie sie in Kapitel 3 un-

tersucht wurden, oder Vorgänge mit geringen Änderungen der Geschwindigkeit weniger von Bedeutung, da eine Abschätzung der Geschwindigkeit nur ein Mal erfolgen muss.

Bei den dynamischen Bewegungsvorgängen, die nachfolgend untersucht werden, wäre es notwendig, den Beobachtungszeitraum des vorselektierenden Algorithmus und dessen Berechnung abzuwarten, bevor mit der Auswertung auf Basis der berechneten Durchschnittsgeschwindigkeit begonnen werden könnte.

Weiterhin ist diese Durchschnittsgeschwindigkeit nur von Nutzen, wenn auch eine Kreuzkorrelation über den gesamten Beobachtungsbereich des vorselektierenden Algorithmus durchgeführt werden kann. Hätte sich das Bild während dieses Beobachtungszeitraumes beispielsweise um über 1024 Pixel verschoben, wären mehrere Korrelationen innerhalb des Bereiches notwendig, für den durch die Vorselektion nur eine einzelne Durchschnittsgeschwindigkeit vorliegt. Für diese Korrelationen hätte jene dann keine Aussagekraft mehr.

Zusammenfassend kann festgestellt werden, dass die Wahl der Testbedingungen anhand der Schwächen bestehender Systeme und Algorithmen auch dazu führt, dass diese nicht als Mittel zur Vorselektion eingesetzt werden können. Damit verbleibt zunächst nur die Suche nach dem globalen Maximum als Möglichkeit zur Ermittelung einer gesuchten Verschiebung.

Um zu einer allgemeineren Abschätzung zu gelangen, bis zu welcher Verschiebung ein Peak unter verschiedenen Bedingungen überhaupt noch als globales Maximum detektierbar ist, wäre der modellbasierte Ansatz mit realistisch parametrierbarem Bildgenerator geeignet.

Da ein entsprechendes Modell nicht zur Verfügung steht, wird diese Abschätzung anhand einer statistischen Betrachtung an den Echtbildern aus Serie 2 (Abschnitt 3.2.2) durchgeführt.

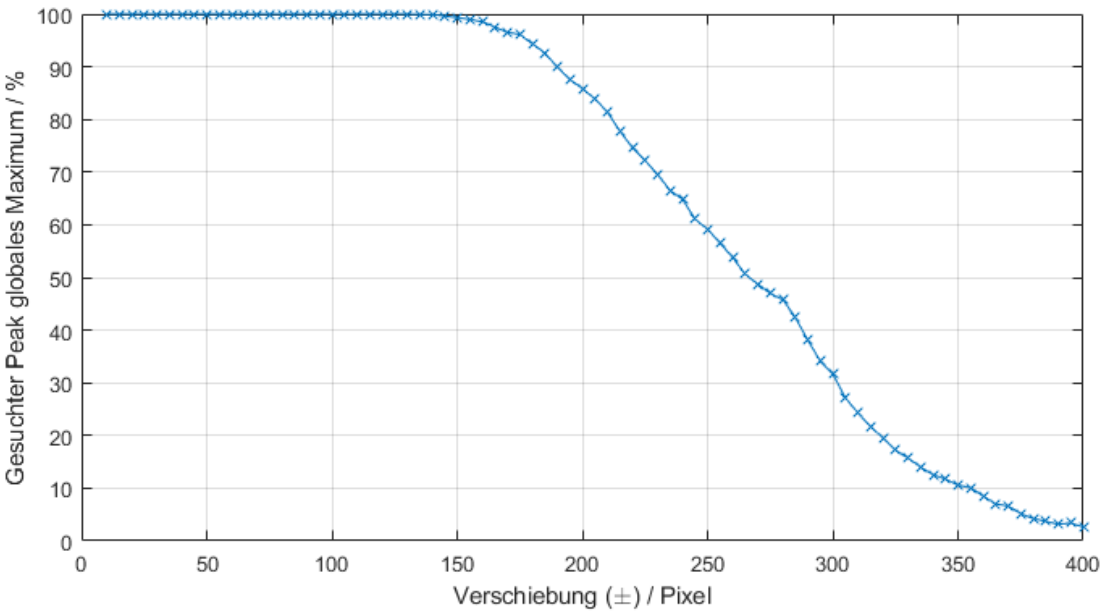

Abbildung 4.6: Abschätzung der Wahrscheinlichkeit, den gesuchten Peak als globales Maximum bei verschiedenen Verschiebungen zu finden

Unter den Bedingungen der Serie 2 (Rollenteststand $1 - 6\,\mathrm{m\,s^{-1}}$) können Verschiebungen um etwa ± 150 Pixel mit beinahe hundertprozentiger Wahrscheinlichkeit als globales Maximum der diskreten Kreuzkorrelationsfunktion gefunden werden. Danach fällt die Rate ab, bis bei einer Verschiebung von ± 400 Pixeln nur noch etwa 2,5 % der gesuchten Peaks das globale Maximum sind. Dies passt zu der Beobachtung aus den Abbildungen 4.4 und 4.5, dass die Aktivität der Kreuzkorrelationsfunktion bei Verschiebungen in dieser Größenordnung bereits stark abgenommen hat.

In den folgenden Beschreibungen der entwickelten Algorithmen wird die Pseudofunktion `max_xcorr` für die Ermittelung der ganzzahligen Verschiebung Δn zwischen zwei Zeilenbildern geschrieben.

4.2.4 Interpolation

Bei der Korrelation von Ortsfilterfunktionen können mittels der Interpolation der Kreuzkorrelationsfunktion im Frequenzbereich wesentlich bessere Ergebnisse erzielt werden, als ohne die Interpolation [10].

Das Verfahren wird daher für die im Folgenden entwickelten Korrelationsalgorithmen übernommen.

In Abschnitt 3.1 wurde die Arbeitshypothese 1 erläutert, die effektiv besagt, dass eine Interpolation nur um ±0,5 Diskretisierungsschritte um das Maximum der diskreten Kreuzkorrelationsfunktion herum notwendig ist.

Die Pseudofunktion `max_fftxcorr` gibt aus diesem Grund in den Algorithmen in den folgenden Abschnitten die Subpixel-Verschiebung zwischen zwei Zeilenbildern zurück, die im Bereich zwischen $-0{,}5$ und $0{,}5$ liegt, und setzt damit implizit voraus, dass die ganzzahlige Verschiebung (`max_xcorr`) bereits bekannt ist, um um diese herum interpolieren zu können.

4.3 Statischer Korrelationsalgorithmus

Der erste vorgestellte Korrelationsalgorithmus für Zeilenbilder orientiert sich in der Herangehensweise an den bestehenden Algorithmen zur Auswertung von Ortsfilterfunktionen, wie sie in Kapitel 3 vorgestellt wurden oder z.B. im COVIDIS zur Anwendung kommen. Dabei werden die Eingangsdaten in Arbeitspakete (Chunks) unterteilt, die separat abgearbeitet werden.

Weil eine konstante Chunk-Größe verwendet wird, wird der Algorithmus als statischer Korrelationsalgorithmus bezeichnet.

Abbildung 4.7 zeigt das grundlegende Vorgehen beim statischen Korrelationsalgorithmus in einem Ablaufdiagramm.

In Worten ausgedrückt iteriert der Algorithmus in Abständen von `chunk_size` Zeilenbildern über die Bildserie und bildet dabei jeweils die Kreuzkorrelationen zwischen Start- und Endpunkt dieser Chunks. Die ausgegebene Strecke berechnet sich anschließend aus der Summe der in den einzelnen Chunks ermittelten Unterstrecken.

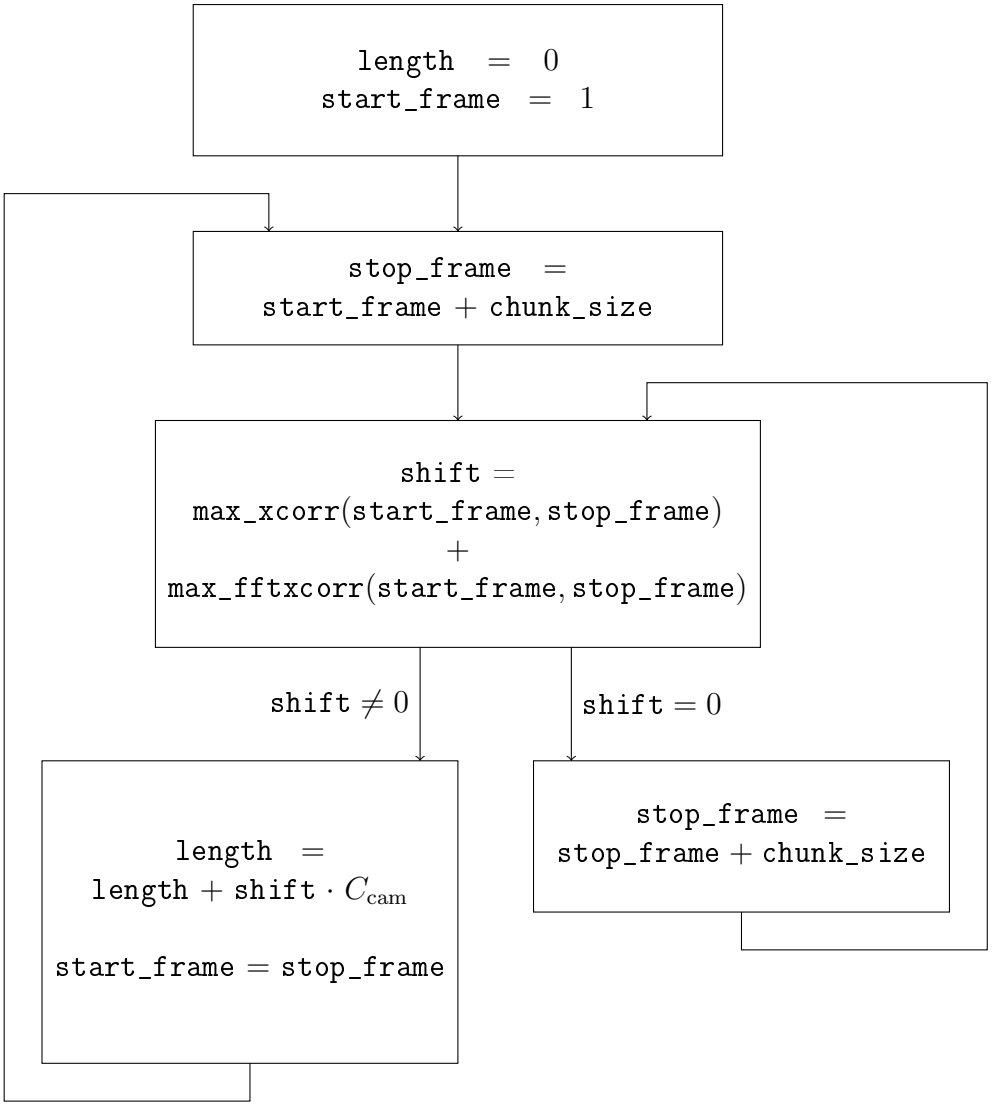

Abbildung 4.7: Ablaufdiagramm des statischen Korrelationsalgorithmus

Zur Plausibilitätsprüfung wird der Algorithmus an künstlich generierten Bildserien mit einer konstanten Geschwindigkeit über eine Strecke von 10 cm erprobt. Ein Ausschnitt der Resultate ist in Tabelle 4.1 dargestellt.

$v_\parallel$ / $\mathrm{m\,s^{-1}}$	$v_\perp$ / $v_\parallel$	Interpolationsschritt / W_{px}	*chunk_size*	F_{rel} / ‰
1	0	-	25	-2,7
			50	-2,7
		0,1	25	-2,7
			50	-0,5
1	2 %	-	25	-2,7
			50	-2,7
		0,1	25	-2,6
			50	0,2
2	0	-	25	-3,0
			50	-3,0
		0,1	25	-0,6
			50	-0,4
2	2 %	-	25	-3,0
			50	-3,0
		0,1	25	-0,3
			50	-0,4

Tabelle 4.1: Simulationsergebnisse des statischen Korrelationsalgorithmus an künstlich generierten Wegstrecken von 10 cm

In den betrachteten Fällen konnte die Messabweichung jeweils durch Verwendung von größeren Chunks in Verbindung mit einer Interpolation gesenkt werden. Dabei ist zu berücksichtigen, dass die Länge der Chunks nicht beliebig hoch gewählt werden kann, da sonst bei höheren Geschwindigkeiten zu große Verschiebungen entstehen können.

Bei den simulierten Messungen ist verglichen mit den Betrachtungen aus Abschnitt 3.2.1 auffällig, dass selbst eine relative Querbewegungsgeschwindigkeit von 2 % keine negativen Auswirkungen auf die Messung zu haben scheint. Eine hohe

Querbewegungsgeschwindigkeit ist dabei zumindest theoretisch ein Argument für kleinere Chunklängen, um die Gefahr zu verringern, dass ein Oberflächenmerkmal während des Chunks aus der Zeile läuft.

4.4 Dynamischer Korrelationsalgorithmus

Der dynamische Korrelationsalgorithmus baut auf der Idee des statischen Korrelationsalgorithmus auf, versucht allerdings die potentiellen Probleme einer statischen Chunklänge zu umgehen. Die Dynamik des Algorithmus besteht darin, angepasst an die jeweilige Situation die möglichst optimale Distanz für eine Korrelation zu finden.

Die zugrundeliegende Annahme ist, dass es grundsätzlich vorteilhaft ist, eine Korrelation über einen möglichst großen zeitlichen und räumlichen Abstand zu bilden, sodass einerseits weniger Korrelationen durchgeführt werden müssen und andererseits mögliche Rundungsfehler (mit oder ohne Interpolation) weniger ins Gewicht fallen. Gleichzeitig ist aus Abschnitt 4.2.3 bekannt, dass die Wahrscheinlichkeit der Erkennung eines Peaks bei Echtbildern ab einer gewissen Verschiebung stark abnimmt. Ziel ist es daher, diese beiden Aspekte im Algorithmus abzuwägen.

Statt die Bilder in Chunks vorgegebener Länge zu unterteilen, wird ein Verschiebungsziel vorgegeben, das der Algorithmus bei jeder Korrelation versucht, zu erreichen. Bei hohen Geschwindigkeiten führt das zu kleineren zeitlichen Abständen, bei niedrigen Geschwindigkeiten zu größeren Intervallen.

Zusammenfassend arbeitet der statische Korrelationsalgorithmus mit konstanten Abständen in der Zeitdimension und der dynamische Korrelationsalgorithmus mit (näherungsweise) konstanten Abständen in der Ortsdimension.

Als Nachteil des dynamischen gegenüber des statischen Korrelationsalgorithmus wird die nicht deterministische Laufzeit betrachtet. Dieser Nachteil wird allerdings in Kauf genommen, da die Abhängigkeit des statischen Korrelationsalgorithmus von der Geschwindigkeit als schwerwiegender betrachtet wird.

Abbildung 4.8 zeigt das Ablaufdiagramm für den dynamischen Korrelationsalgorithmus.

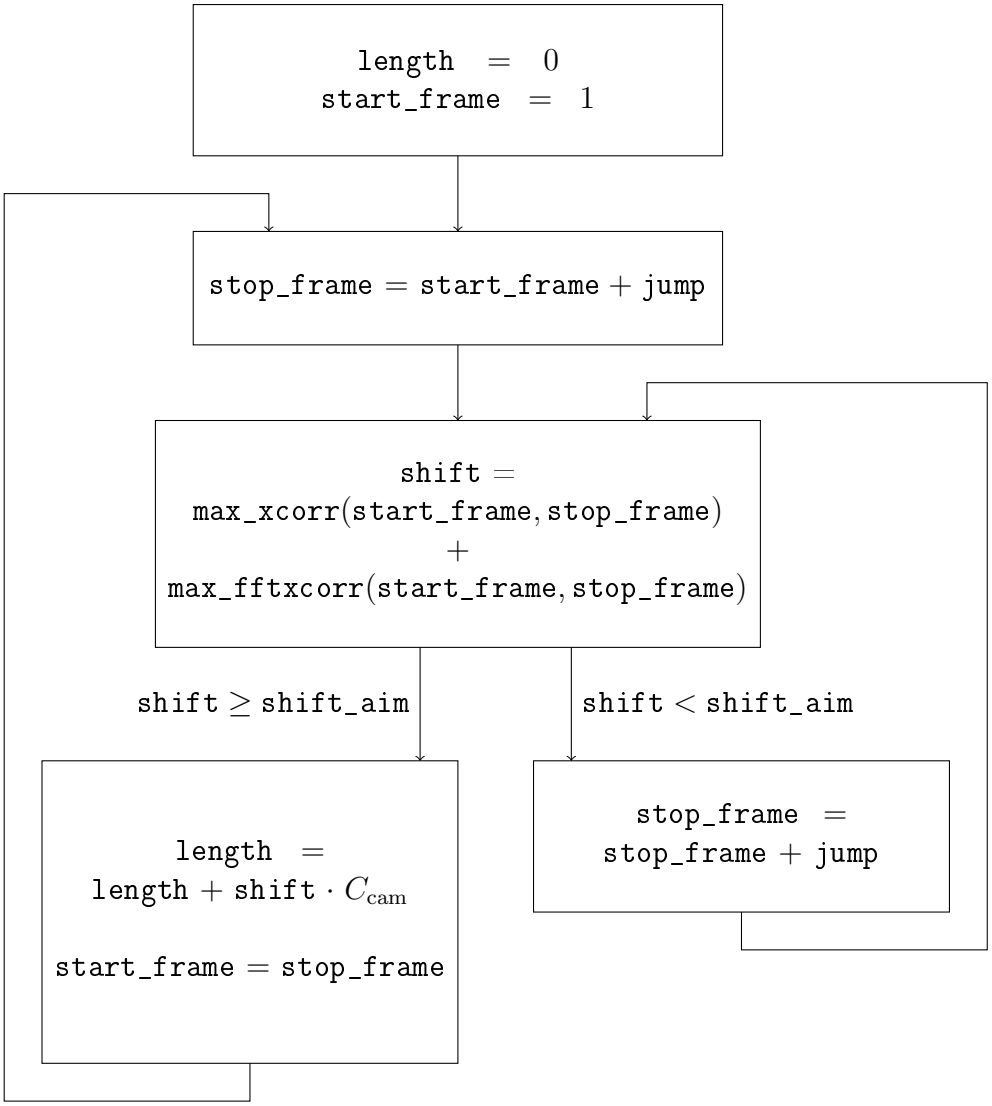

Abbildung 4.8: Ablaufdiagramm des dynamischen Korrelationsalgorithmus

Ausschnitte der Ergebnisse für Messungen an künstlich generierten Bildern sind in Tabelle 4.2 zusammengefasst. Ähnlich wie die Simulationsergebnisse des statischen Korrelationsalgorithmus weisen auch diese Ergebnisse auf eine höhere Toleranz gegenüber Querbewegungen verglichen mit den Korrelationsalgorithmen für Ortsfilterfunktionen hin.

$v_\parallel$ / $\mathrm{m\,s^{-1}}$	$v_\perp$ / $v_\parallel$	Interpolations-schritt / W_{px}	$shift\_aim$	F_{rel} / ‰
1	0	-	15	-2,7
			30	-2,7
			60	-2,7
		0,1	15	-0,5
			30	-0,4
			60	-0,5
1	2 %	-	15	-2,7
			30	-2,7
			60	-1,7
		0,1	15	0,2
			30	0,2
			60	1,1

Tabelle 4.2: Simulationsergebnisse des dynamischen Korrelationsalgorithmus an künstlich generierten Wegstrecken von 10 cm

4.4.1 Qualitätsmerkmal

Der dynamische Korrelationsalgorithmus in der vorgestellten Grundform orientiert sich ausschließlich an der Erreichung des Verschiebungsziels. Als weiterer möglicher Orientierungspunkt wird ein Qualitätsmerkmal mit dem Ziel eingeführt, aus einer Anzahl möglicher Endpunkte einer Korrelation denjenigen auszuwählen, für den die geringste Messabweichung zu erwarten ist.

Motiviert ist der Ansatz durch die bei Echtbildern zu erwartenden Störungen z.B. durch Pixelfehler oder andere stillstehende Bildmerkmale, die sich möglicherweise mit den Abbildungen der Oberflächenmerkmale überlagern oder diese überdecken (s. auch Abschnitt 3.2.3).

Bei den in den letzten Abschnitten betrachteten Simulationen mit künstlichen Bildern ohne Querbewegung erscheint es plausibel, dass sich grundsätzlich je-

des Ziel-Zeilenbild gleich gut für die Korrelation eignet, wenn vom Einfluss der Verschiebung selbst auf die Messabweichung abgesehen wird, da die Bilder durchgehend gleich beleuchtet sind und keine sonstigen Fehler aufweisen.

Für Pixel- oder Optikfehler in Echtbildern lässt sich eine andere Überlegung anstellen: Möglicherweise ist das Festhalten an einem konstanten Verschiebungsziel in einigen Fällen unvorteilhaft, wenn wichtige Merkmale durch Bildfehler verfälscht oder verdeckt sind und ein Ausweichen auf benachbarte Zeilen bei Vernachlässigung des Verschiebungsziels würde bessere Ergebnisse liefern.

Für das Qualitätsmerkmal wird zunächst ein Vorgehen wie in Abbildung 4.9 gezeigt skizziert, das in der finalen Version des Algorithmus an Echtbildern getestet wird.

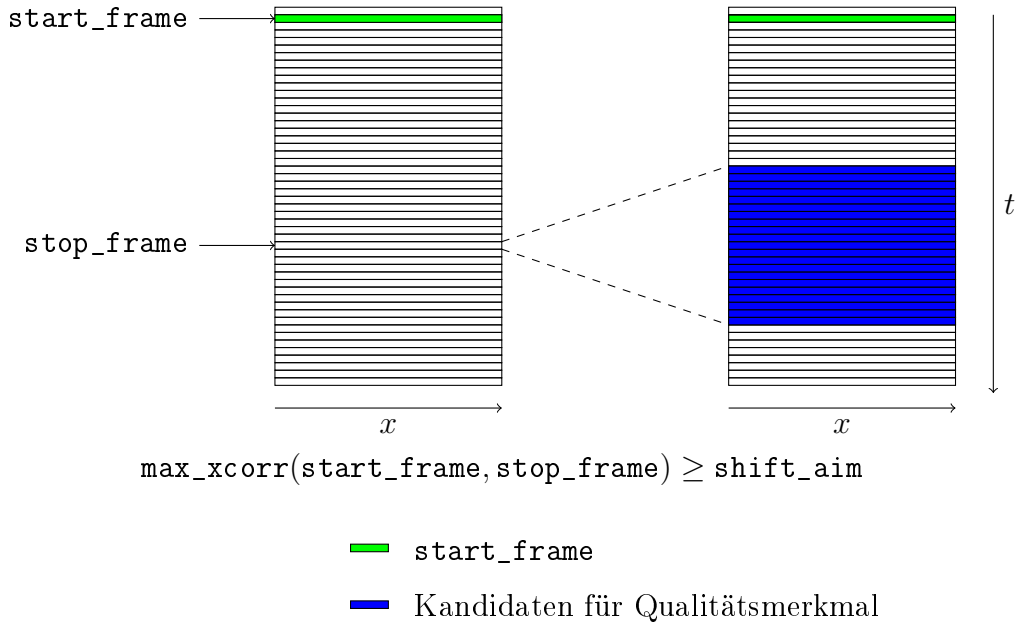

Abbildung 4.9: Suchbereich für das Qualitätsmerkmal

Sobald nach dem Vorgehen des dynamischen Korrelationsalgorithmus in Abschnitt 4.4 ein Zeilenbild gefunden wird, für das das Verschiebungsziel erreicht wird, werden die umliegenden Zeilenbilder als potentielle Ziel-Zeilenbilder für die Kreuzkorrelation markiert, aus denen anhand des noch nicht näher definierten Qualitätsmerkmals das beste ausgewählt werden soll.

Für die Implementierung eines Qualitätsmerkmals kommen beispielsweise folgende Optionen in Betracht:

1. In den Abbildungen 4.4 und 4.5 (korrelierende Bilder und nicht korrelierende Bilder) haben beide Kreuzkorrelationsfunktionen einen klar erkennbaren Peak, deren Energie in der gleichen Größenordnung liegt, wobei der Peak in der Kreuzkorrelationsfunktion zwischen den korrelierenden Bildern sich aber deutlicher vom RMS-Wert der gesamten KKF abhebt.

   Damit käme z.B. ein „Peak to RMS"-Faktor in Betracht, um eine vollständig unverwertbare KKF von einer verwertbaren zu unterscheiden und gleichzeitig ein Maß für die Güte der Korrelation zu liefern.

2. Der COVIDIS setzt für die Korrelationsanalyse die Breite und Höhe des Korrelationspeaks als Maß für die Qualität an [1, S. 70].

Im Vorgriff auf das Kapitel 6 zur Implementierung des Algorithmus wird der Fokus darauf liegen, die hier eingeführte Pseudofunktion `max_xcorr`, also das Finden der Position und mithin der Energie des globalen Maximums der KKF, so weit wie möglich zu beschleunigen. Diese Funktion in Echtzeit ausführen zu können ist Grundvoraussetzung dafür, überhaupt an den Punkt im Algorithmus zu gelangen, an dem ein Bereich für eine potentielle Qualitätsanalyse definiert werden kann.

Im Hinblick auf die echtzeitfähige Implementierbarkeit wird daher aus mehreren Gründen der oben zweitgenannte Ansatz für ein Qualitätsmerkmal gewählt:

- Das Qualitätsmerkmal beeinflusst die Laufzeit des Algorithmus besonders, da es über einen größeren Bereich von Zeilenbildern bestimmt werden muss und damit eine erhöhte Komplexität bei der Bestimmung des Qualitätsmerkmals pro bestimmten Wegfortschritt mehrfach in die Laufzeit eingeht.
- Da das Qualitätsmerkmal über einen zeitlich eng begrenzten Bereich ermittelt wird, wird es als unwahrscheinlich betrachtet, dass zwischen den Kandidaten die Unterscheidung zwischen verwertbarer und komplett unverwertbarer Korrelation notwendig ist.
- Die Ermittelung eines Peak-to-RMS-Faktors ist mit einem vergleichsweise hohen Berechnungsaufwand verknüpft.

- Die Bestimmung der Position und der Höhe des Korrelationspeaks wird stark optimiert und kann für ein Qualitätsmerkmal nach dem Vorbild des COVIDIS eingesetzt werden, bei dem ausschließlich die Höhe des Korrelationspeaks zwischen den Kandidaten verglichen wird.

Das Startbild der Korrelation wird demnach nacheinander mit allen Kandidaten im Umfeld des zunächst ermittelten Zielbildes korreliert. Unter den Maxima der Einzelkorrelationen wird anschließend das mit der größten Energie ausgewählt.

5 Entwicklung der Vader2-Plattform

5.1 Bewertung der ersten Vader-Plattform

Die VADER-Plattform wurde im Jahr 2018 am Institut für Systemtechnik entwickelt, um als Grundlage für ein Ortsfrequenzfiltersystem nach dem Vorbild des COVIDIS-Sensors zu dienen. Wie in Abbildung 5.1 dargestellt, besteht die Plattform aus einem FPGA und einem digitalen Signalprozessor, die über eine serielle Schnittstelle mit mehreren Kanälen Daten austauschen. In der aktuell vorhandenen Version der Hardware befinden sich die zwei Bausteine auf separaten Platinen, die über einen Steckverbinder verbunden sind. [26] [3] [2]

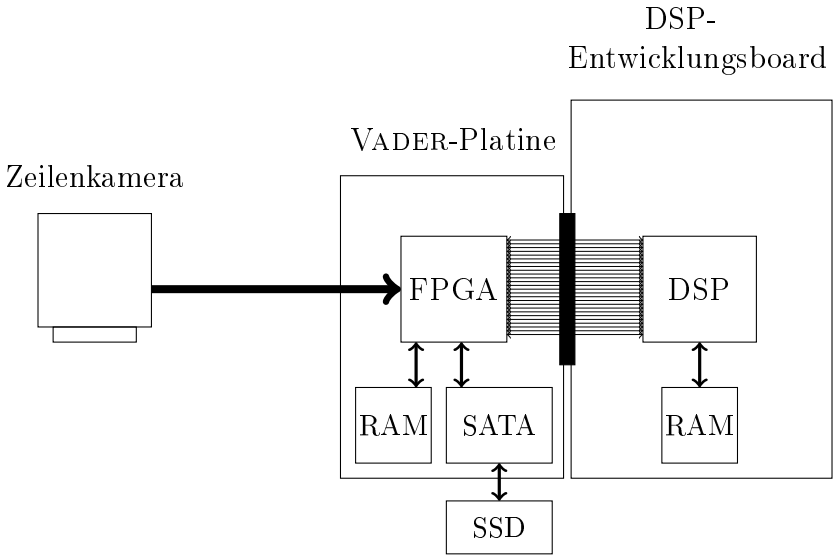

Abbildung 5.1: Die VADER-Plattform

Über den Steckverbinder der VADER-Plattform sind insgesamt 26 parallele Datenleitungen geführt [30] und der DSP unterstützt für diese Ein- bzw. Ausgänge Taktraten bis zu 50 MHz [31].

Ob diese maximale Taktrate in der Praxis über den Steckverbinder ohne Übertragungsfehler erreichbar ist, ist nicht bekannt, da das VADER-Design nur bei einer Taktrate von 10 MHz verifiziert wurde [2, S. 30]. Für die folgende Diskussion wird trotzdem eine Taktrate von 50 MHz angenommen, um das Potential der Schnittstelle optimistisch abzuschätzen. Als maximale Bruttodatenrate ergibt sich $26\,\text{bit} \cdot 50\,\text{MHz} = 1300\,\text{Mbit/s}$.

Die in diesem Projekt verwendete DALSA Spyder 3-Kamera ermöglicht Frameraten bis 68 kHz bei 12 bit pro Pixel und einer Zeilenbreite von 1024 Pixeln [25]. Die Rohbild-Datenrate beträgt damit $12\,\text{bit} \cdot 1024 \cdot 68\,\text{kHz} \approx 836\,\text{Mbit/s}$.

Um die übertragenen Pixel im DSP einzeln adressieren zu können, wäre es notwendig, pro Pixel eine ganzzahlige Anzahl an Bytes zu übertragen [2, S. 32], mindestens also 16 bit, wie es beispielsweise im VADER zur Übertragung von Rohbildern bei reduzierter Framerate implementiert ist. Damit steigt die benötigte Datenrate auf etwa 1100 Mbit/s und liegt nur knapp unter dem theoretischen Maximum der Schnittstelle, sodass es fraglich ist, ob dieses Szenario in der Praxis umsetzbar ist.

Wird für jedes Pixel eine SP-Fließkommazahl vorgesehen, wie in der vorliegenden Arbeit, verdoppelt sich die Datenrate auf etwa 2200 Mbit/s, die sich schon theoretisch unter idealen Bedingungen nicht mehr über die Schnittstelle übertragen lassen.

Das im VADER-Projekt implementierte Protokoll zur Übertragung von Ortsfilterfunktionen vom FPGA zum DSP und Konfigurationsbefehlen vom DSP zum FPGA belegt die zur Verfügung stehende Schnittstelle vollständig [2, S. 35]. Wie Abbildung 5.1 zeigt, verfügen beide Teilplatinen über eigene RAM-Bausteine zur Zwischenspeicherung von Daten, die zum Beispiel die Implementierung von speicherintensiveren Algorithmen ermöglichen würden. Gleichzeitig limitiert die vergleichsweise schmalbandige Schnittstelle zwischen den Platinen die Möglichkeit des Datenaustausches für solche potentielle Algorithmen. Das gleiche gilt für die auf der FPGA-Platine verbaute SATA-Schnittstelle, an die eine Festplatte oder SSD angeschlossen werden kann. Zwar können vom FPGA beispielsweise Bild-

daten über die SATA-Schnittstelle gelesen oder geschrieben werden, der DSP ist aber vom Zugriff auf die Schnittstelle de facto abgeschnitten.

5.2 Die Vader2-Plattform

Bei der Auswahl der Hardware für eine mögliche Folgeplattform des VADER wird angenommen, dass die Aufteilung von Algorithmen auf einen programmierbaren Logikbaustein und einen Signalprozessor prinzipiell nicht nachteilig ist, sofern zwischen den Elementen kein Flaschenhals durch eine Übertragungsschnittstelle entsteht.

Am Institut steht ein Ultra96-Entwicklungsboard zur Verfügung, für das in Abbildung 5.2 ein vereinfachtes Systemdiagramm dargestellt ist.

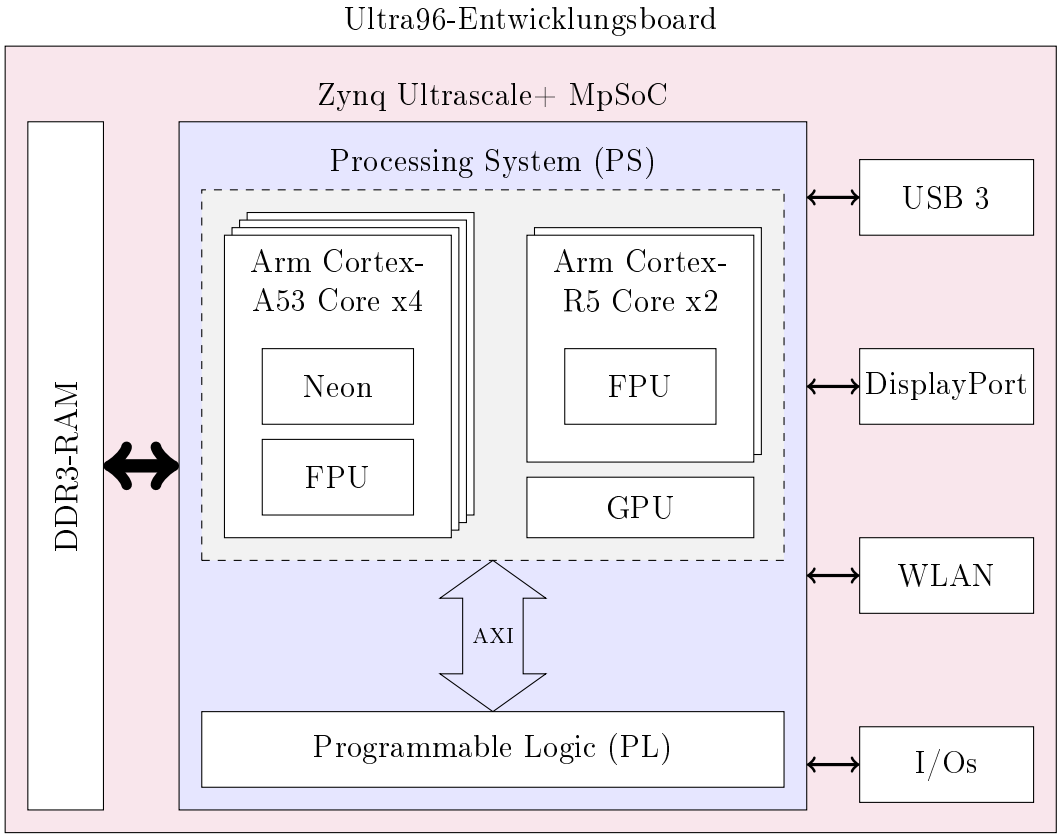

Abbildung 5.2: Die Grundlage der VADER2-Plattform [32, S. 29] [33]

Auf der Platine ist ein Baustein der Familie Zynq Ultrascale+ MpSoC des Herstellers Xilinx verbaut. System-on-Chip (SoC) bedeutet allgemein, dass verschiedene Systemkomponenten in einem einzelnen Baustein kombiniert werden.

In diesem Fall handelt es sich um eine Kombination aus unter anderem einem programmierbaren Logikteil und einem Arm-Prozessor mit vier Kernen (multi processor, MP). Jeder der vier Kerne beinhaltet einen Neon-Beschleuniger, auf den in Abschnitt 6.1.1 eingegangen wird, sowie eine Floating Point Unit (FPU). Außerdem beinhaltet der Chip einen weiteren Dual-Core Arm-Prozessor sowie einen Grafikprozessor (Graphical Processing Unit, GPU), die in dieser Arbeit allerdings nicht verwendet werden. Die Systemkomponenten sind über mehrere AXI-Busse (Advanced Extensible Interface) vernetzt. [34] [32, S. 29]

Zusammenfassend unterscheidet sich das Ultra96-Board hauptsächlich in drei Punkten von der VADER-Plattform:

1. Bei der VADER-Plattform kommt statt des Arm-Prozessors ein DSP-SoC aus je einem C66x- und Arm-Kern von Texas Instruments zum Einsatz.
2. Die Schnittstelle zwischen programmierbarer Logik und Prozessor liegt bei dem Zynq-SoC innerhalb eines einzelnen Chips, bei der VADER-Plattform an einem Steckverbinder zwischen zwei Platinen.
3. Die programmierbare Logik und der Prozessor verfügen bei der VADER-Plattform über separate Arbeitsspeicher. Ein gemeinsamer Arbeitsspeicher ist nicht effizient möglich, da alle Daten über die FPGA-DSP Schnittstelle mit relativ geringer Bandbreite fließen müssten.

Weil wie im letzten Abschnitt geschildert die Schnittstelle zwischen programmierbarer Logik (FPGA) und dem Prozessor als erheblicher Nachteil der VADER-Plattform angesehen wird, wird das Ultra96-Board im Hinblick auf die möglichen Übertragungsraten zwischen den Systemkomponenten evaluiert.

Zwischen dem SoC und dem Arbeitsspeicher, auf den sowohl die programmierbare Logik als auch das Prozessorsystem zugreifen können, wurden auf dem Ultra96-Board in einer Studie Lese- und Schreibdatenraten von jeweils über 3,5 GB/s gemessen. Auf einem anderen Entwicklungsboard, das das gleiche Zynq-SoC verwendet, wurden mit einem höherwertigen RAM-Baustein Lese- und Schreibdatenraten von jeweils über 12 GB/s gemessen. [35]

Die in der genannten Studie erreichte Datenrate zwischen dem Speichercontroller des SoCs und dem DDR-Chip des Ultra96 liegt mit 3,5 GB/s um den Faktor 20 über der theoretisch erreichbaren und um den Faktor 100 über der verifizierten Datenrate der Schnittstelle der VADER-Plattform. Sie liegt außerdem um den Faktor 13 über der benötigten Datenrate für die Übertragung vollständiger, gefilterter Zeilenbilder im SP-Fließkommaformat bei der maximalen Framerate der Spyder 3-Kamera.

Als zweiter Faktor neben den betrachteten Datenraten wird eine Abschätzung für das Verhältnis der Leistungsfähigkeiten der Prozessoren der zwei Systeme vorgenommen. Dazu wird die Laufzeit für eine FFT mit 1024 Punkten und komplexwertigen Originalsignalen in SP-Fließkommadarstellung verglichen. Der C66x-DSP benötigt für diese Operation 6,27 µs [36]. Eine Messung auf einem der vier A53-Kerne des Zynq-MpSoC ergibt eine Laufzeit von 23 µs.

Damit ist der C66x-DSP zwar bei der Berechnung einer einzelnen FFT etwa um den Faktor vier schneller, bei der Berechnung von vier solcher FFTs könnte allerdings auf dem Zynq-SoC jeder der vier Kerne eine FFT parallel ausführen, während der C66x die vier FFTs seriell ausführen müsste.

Dass mehrere FFTs parallel auszuführen sind, wird als realistisches Szenario z.B. im Kontext der DFT-Auswertung von Ortsfilterfunktionen betrachtet: Das VADER-System stellt beispielsweise bis zu acht Ortsfilterfunktionen parallel zur Verfügung [2, S. 25], die für eine Multiskalenauswertung auch parallel bearbeitet werden könnten.

Zusätzlich ist das Prozessorsystem des Zynq-SoCs durch die enge Vernetzung mit der programmierbaren Logik darauf ausgelegt, Berechnungen in Hardwarebeschleuniger auszulagern [34].

Insgesamt wird abgeschätzt, dass die Leistungsfähigkeiten der beiden Prozessorsysteme in einer ähnlichen Größenordnung liegen, der Zynq-SoC diese Leistungsfähigkeit aber besser ausnutzen kann, weil kein Flaschenhalseffekt bei der Zuführung von Daten besteht.

Abbildung 5.3 zeigt die VADER-Plattform, bestehend aus dem DSP-Entwicklungsboard oben links, die mit der VADER-Platine oben rechts über den beschriebenen Steckverbinder verbunden ist. Darunter ist das Ultra96-Entwicklungsboard zu sehen.

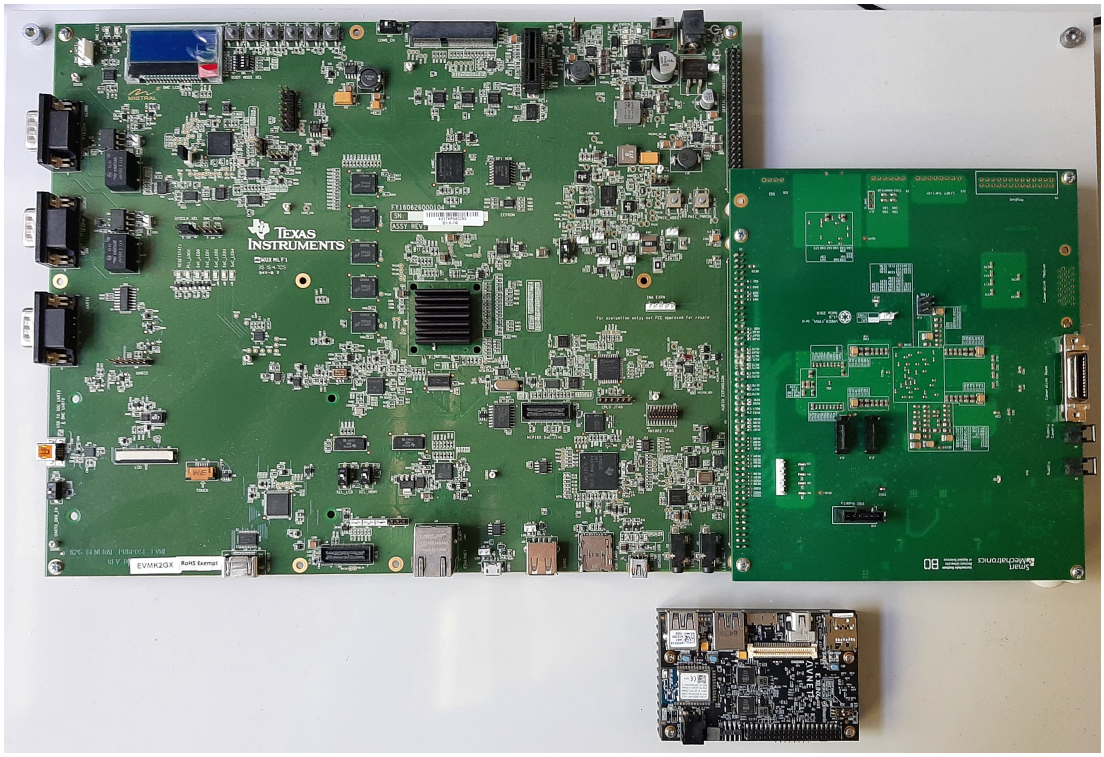

Abbildung 5.3: Die VADER-Plattform und das Ultra96-Entwicklungsboard

5.3 Betriebssystem Vader2-Linux

Grundsätzlich ist der Arm-Prozessor der Zynq-Familie herstellerseitig nicht an die Verwendung eines bestimmten oder überhaupt irgendeines Betriebssystems gebunden.

Mit PetaLinux stellt Xilinx eine Linux-Referenzdistribution und ein entsprechendes Toolkit zur Verwendung auf Zynq-Bausteinen zur Verfügung. [37]

Auch der Hersteller des Ultra96-Entwicklungsboards stellt ein passendes Linux-Abbild mit den zugehörigen Treibern für die auf der Platine verbauten Peripheriemodule zur Verfügung [38].

Bei der Abwägung möglicher Betriebssysteme wurde Linux als von den Herstellern zur Verfügung gestellte Standardlösung aus folgenden Gründen als Ausgangspunkt gewählt:

- Das gleiche Betriebssystem kann auf anderen Bausteinen der Zynq-Familie verwendet werden, was es in Zukunft ermöglicht, je nach Bedarf auf leistungsfähigere oder günstigere Bausteine umzusteigen, ohne dass dies die entwickelte Software beeinflusst.
- Auch die SoC-Bausteine des Konkurrenten Intel unterstützen Linux [39].
- Simulink Coder / Embedded Coder unterstützen Linux.

Aus diesen Gründen wäre von der Verwendung von Linux nur abgesehen worden, wenn sich Hinweise darauf ergeben hätten, dass dies der Verwirklichung einer echtzeitfähigen Implementierung des Korrelationsalgorithmus entgegenstünde oder die Weiterverwendung der Plattform in etwaigen Folgeprojekten erheblich erschweren würde.

Betriebssysteme im Allgemeinen dienen als Abstraktionsschicht zwischen einem komplexen Hardwaresystem und der oder den Anwendungen. Dabei gehört es zu den Aufgaben des Betriebssystems, den Anwendungen Ressourcen wie CPU, Arbeitsspeicher und Zugriff auf Peripherie zuzuteilen. [40]

Umgekehrt bedeutet dies aber auch, dass die Applikation, in diesem Fall das Korrelationsprogramm, nicht mehr die volle Kontrolle über die Hardware hat, was möglicherweise die Echtzeitfähigkeit beeinträchtigen könnte.

Die folgenden Abschnitte behandeln die für die VADER2-Plattform entwickelte Distribution VADER2-Linux, mit der versucht wird, einerseits die Vorteile von Linux als Standardlösung zu erhalten und andererseits eine Umgebung für den entwickelten Korrelationsalgorithmus oder bisherige und zukünftige Ortsfilteralgorithmen zu schaffen, die die Echtzeitausführung möglichst wenig beeinträchtigt.

5.3.1 Buildumgebung

Das bereits erwähnte PetaLinux-Toolkit des Herstellers Xilinx erlaubt es, vorgefertigte Linux-Referenzdistributionen für Xilinx-Bausteine zu konfigurieren und

zu erweitern oder zu verändern. PetaLinux setzt auf Yocto auf, einem Projekt der Linux Foundation, das im Baukastenprinzip das Zusammenstellen von Embedded-Linux-Distributionen erlaubt [41].

Zunächst wird abgewägt, entweder PetaLinux zu verwenden, oder eine eigene Distribution mit Yocto zu entwickeln. Dabei werden folgende Punkte berücksichtigt:

- Die PetaLinux-Tools stellen besondere Anforderungen an das Betriebssystem und dessen Version, um lauffähig zu sein [37].
- Es zeigt sich, dass PetaLinux verglichen mit Yocto beschränktere Möglichkeiten bei der Anpassung der Systemkomponenten bietet.
- Yocto ist nicht an einen Hersteller gebunden sondern kann zum Beispiel auch Linux-Distributionen für Intel SoCs erstellen, sodass ein Wechsel auf eine andere Hardware leichter möglich ist.
- Mit Yocto kann ein bottom-up-Ansatz verfolgt werden, bei der die Distribution vom absoluten Minimum durch konsekutives Hinzufügen von Komponenten aufgebaut wird, sodass eine Distribution entsteht, die nur die gewünschten Komponenten enthält [41, S. 29]. Im Gegensatz dazu wird bei PetaLinux der umgekehrte top-down Ansatz verfolgt: Aus einer umfangreichen Referenzdistribution werden unerwünschte oder unnötige Komponenten entfernt.

Nach Abwägung der genannten Punkte wird Yocto als Toolkit für die Erstellung der VADER2-Linux-Distribution gewählt.

5.3.2 Device Tree

Der sogenannte Device Tree ist eine baumförmige Datenstruktur, die Informationen über die in einem bestimmten Rechner verbaute Hardware enthält. Er wird beim Hochfahren vom Betriebssystem gelesen und interpretiert. [42]

Linux unterstützt Device Trees unter anderem für die verwendete Arm-Architektur, was zur Folge hat, dass ein bestimmtes Kernelkompilat nicht an eine bestimmte Hardwarekonfiguration gebunden ist, sondern durch die Übergabe eines veränderten Device Trees angepasst werden kann. Damit könnte beispielsweise

bei Verwendung einer neu entwickelten Hardware für das VADER-Projekt der bestehende VADER2-Linux Kernel weiterverwendet werden und lediglich die veränderten Peripheriekomponenten (z.B. größerer Arbeitsspeicher, Netzwerkschnittstellen, usw.) im Device Tree eingetragen werden. [42]

Der Device Tree wird über Yocto in die Distribution eingebunden [41, S. 275].

Wie im letzten Abschnitt beschrieben, wird aktuell das Ultra96-Board als Hardwareplattform verwendet. Ein Device Tree mit den auf dieser Platine verbauten Bausteinen wird aus dem vom Hersteller des Ultra96 zur Verfügung gestellten Linux mit kleinen Änderungen übernommen.

Eine Änderung am Device Tree, die die Verwaltung des Arbeitsspeichers betrifft, ist in Abschnitt 5.3.4 beschrieben.

5.3.3 Verwaltung der CPUs

Das zentrale Konzept eines Betriebssystem ist der sogenannte Prozess [40, S. 85].

Die laufende Instanz eines Programms wird als Prozess bezeichnet, der wiederum aus mehreren parallel ausführbaren Threads bestehen kann. Darüber hinaus kann der Kernel Threads erstellen, um Aufgaben des Betriebssystems abzuarbeiten. Da mehr Threads um Ausführung konkurrieren können, als Prozessoren zur Verfügung stehen, implementieren Betriebssysteme einen sogenannten Scheduler, der die Rechenzeit zwischen den konkurrierenden Threads verteilt. Hierfür existieren je nach Betriebssystem verschiedene Ansätze, zum Beispiel sind Echtzeit-Scheduler darauf ausgelegt, zeitlich vorhersagbar zu sein und unabhängig von anderen Einflüssen bestimmte Deadlines einzuhalten. [40] [43]

Die Unterstützung einer solchen RTOS-artigen (Real Time Operating System) Vorhersagbarkeit ist im Linux-Kernel zum aktuellen Zeitpunkt noch nicht vollständig unterstützt, wird aber entwickelt [44].

Aus diesem Grund könnte sich ein potentielles Problem ergeben, wenn ein Thread des Messalgorithmus unterbrochen und nicht rechtzeitig fortgesetzt wird, um echtzeitfähig zu bleiben. Um dieses Problem zu umgehen, wird das CPU-Isolation-Feature des Linux-Kernels verwendet.

Dieses Feature ermöglicht die Isolation einzelner Prozessorkerne von Einflüssen des Schedulers, also der regelmäßigen Übernahme der Kontrolle durch den Scheduler und der möglicherweise anschließenden Neuzuweisung der Rechenzeit an einen anderen Prozess, sowie von bestimmten Interrupts. Dies führt zu einer Aufteilung der Prozessorkerne in isolierte CPUs und sog. Housekeeping-CPUs, wobei die Housekeeping-CPUs die notwendigen Aufgaben des Betriebssystems übernehmen. [45]

VADER2-Linux sieht eine CPU (CPU0) als Housekeeping-CPU vor und isoliert die restlichen Kerne vom Scheduler und Interrupts. Der oder die Threads des Messalgorithmus können auf die restlichen drei Kerne verschoben und dort isoliert ausgeführt werden. Dieses Vorgehen hat verschiedene Gründe:

- Von den verwandten Bauteilen der Zynq-Familie mit einem Single-Core Arm-Prozessor, die auch Linux einsetzen, ist bekannt, dass ein Kern für die Erledigung der Aufgaben des Betriebssystems prinzipiell ausreichend ist.
- CPU-Isolation ist ein Standardfeature des Linux-Kernels, das keine Änderungen am Code des Betriebssystems erfordert und damit auch in zukünftigen Versionen weiterverwendet werden kann. Darüber hinaus wird davon ausgegangen, dass die Echtzeitfähigkeit des Linux-Kernels in Zukunft weiterentwickelt wird.
- Auf den drei isolierten Kernen können trotz der Isolation die Funktionen des Betriebssystem verwendet werden. Beispielsweise wird in Abschnitt 7.4 von den isolierten Kernen auf das Dateisystem eines USB3.0-Sticks zugegriffen.
- Theoretisch gehen durch die Abgabe eines Kerns für das Betriebssystem 25 % der Rechenleistung für den Algorithmus verloren (einer von vier Kernen). Andererseits ist fraglich, inwieweit der Algorithmus überhaupt von einer Parallelisierung auf vier Kernen profitieren würde (s. Kapitel 6).

  Darüber hinaus würde auch ein anderes Betriebssystem einen gewissen Overhead verursachen und der Housekeeping-Kern kann zusätzlich zu den Aufgaben des Betriebssystems weitere Programme z.B. für Diagnose- oder Konfigurationsschnittstellen ausführen. Unter den Gesichtspunkten, dass in jedem Fall ein gewisser Anteil der Rechenleistung für Nebenaufgaben reserviert werden muss, wird die Bündelung dieser Aufgaben auf einen Kern

und damit das Freihalten der restlichen drei Kerne insgesamt als sinnvolle Lösung betrachtet.

Abbildung 5.4 zeigt zwei Beispielanwendungen der VADER2-Plattform mit drei isolierten CPUs.

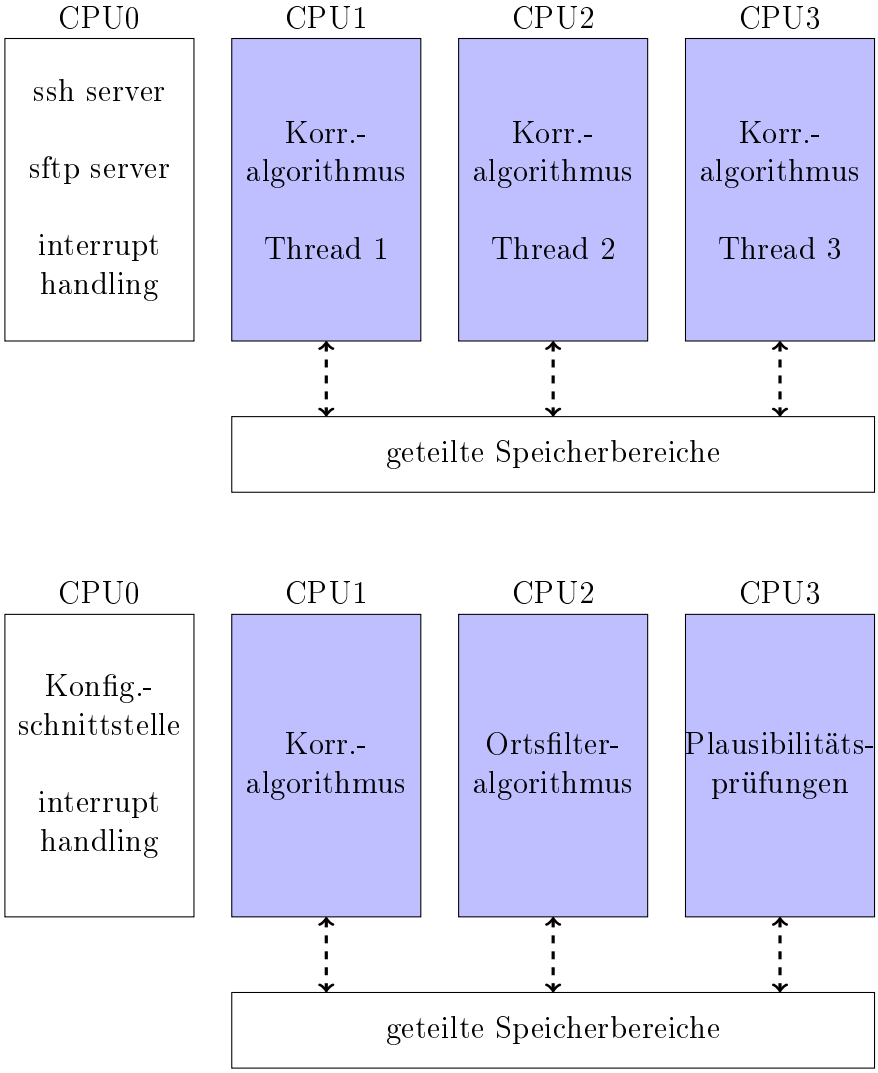

Abbildung 5.4: Zwei Beispielanwendungen für die CPUs der VADER2-Plattform

5.3.4 Arbeitsspeicher

Virtueller Speicher

Linux bietet wie andere Betriebssysteme mit virtuellem Speicher eine Abstraktion des physikalischen Arbeitsspeichers [46]. Prozesse verfügen über einen virtuellen Adressraum und sprechen den Arbeitsspeicher über virtuelle Adressen an, die zunächst einer physikalischen Adresse zugeordnet werden müssen [43].

Zuordnungen zwischen virtuellen und physikalischen Adressen erfolgen jeweils über einen kontinuierlichen Speicherbereich, der sogenannten Page, mit einer bestimmten Größe (Pagegröße) und sind in Pagetables vermerkt. Um den Vorgang der Adressumsetzung zu beschleunigen, verfügen moderne CPUs über Memory Management Units (MMUs), die den Vorgang in Hardware unterstützen. [46] [40, S. 195ff]

Durch das Konzept des virtuellen Speichers sprechen zwei Prozesse, die auf die gleiche (virtuelle) Speicheradresse zugreifen, normalerweise unterschiedliche physikalische Speicherstellen mit unterschiedlichen Inhalten an. Dies verhindert, dass ein Prozess Daten eines anderes Prozesses oder des Betriebssystems unberechtigt oder unbeabsichtigt auslesen oder verändern kann. [40, S. 41]

Andererseits ist es auch möglich, dass zwei Prozesse über unterschiedliche oder auch gleiche virtuelle Adressen einen gemeinsamen physikalischen Speicherbereich teilen [40, S. 228ff], was zum Beispiel für die Implementierung mehrerer paralleler Messalgorithmen, die auf den gleichen Eingangsdaten arbeiten, eingesetzt werden kann.

Virtueller Speicher erlaubt es außerdem, dass ein Prozess scheinbar über mehr Speicher verfügt als physikalischer Speicher verbaut ist, indem das Betriebssystem im Hintergrund Teile davon auf ein anderes Medium auslagert und nur bei Bedarf wieder in den tatsächlichen Arbeitsspeicher kopiert [40, S. 41].

Das Konzept des virtuellen Speichers ist als Abstraktion des physikalischen Speichers für die Anwendung im Allgemeinen transparent, es ist also von Seiten der Anwendung nicht unmittelbar erkennbar, welche Mechanismen seitens des Betriebssystems im Hintergrund zur Anwendung kommen [40, S. 232].

Die Transparenz des virtuellen Speichers ist im vorliegenden Fall aus verschiedenen Gründen ein potentielles Problem. Würde die Speicherverwaltung des Betriebssystems verwendet werden, um zum Beispiel Puffer für Bilddaten zu reservieren, die aus der programmierbaren Logik gefüllt werden sollen, so darf das Betriebssystem nicht mit diesem Speicherbereich interferieren während die programmierbare Logik darauf zugreift.

Darüber hinaus ist es aus praktischen Gründen von Vorteil, wenn die Bilder in einen physikalisch kontinuierlichen Speicherbereich geschrieben werden können, um keinen zusätzlichen Aufwand für Adressberechnungen in der Logik zu erzeugen. Wegen der Transparenz des virtuellen Speichers ist es aber nicht zwingend, dass ein virtuell kontinuierlicher Speicherbereich auch physikalisch kontinuierlich ist [40, S. 197]. Außerdem ist es vorstellbar, dass durch Fragmentierung des physikalischen Speichers zu einem gewissen Zeitpunkt kein ausreichend großes Stück kontinuierlicher physikalischer Speicher mehr frei ist, um die benötigten Puffer für den oder die Messalgorithmen zu reservieren.

Aus den genannten Gründen werden in VADER2-Linux ein oder mehrere physikalische Speicherbereiche dem Einfluss des Betriebssystems schon beim Hochfahren entzogen, um zu garantieren, dass diese Bereiche zu jedem Zeitpunkt an einer konstanten physikalischen Adresse zur Verfügung stehen und weder seitens des Betriebssystems anderweitig zugeteilt noch zum Beispiel zwischenzeitlich auf den Massenspeicher ausgelagert werden können. Dies wird durch die im Folgenden beschriebene Änderung im Device Tree erreicht.

Reservierter Speicher

Listing 5.1 zeigt den Ausschnitt des Device Trees, der den Arbeitsspeicher beschreibt.

```
1 memory {
2   device_type = memory;
3   reg = <0x0 0x0 0x0 0x7ff00000>;
4 };
```

Listing 5.1: Ausschnitt des Device Trees für den Arbeitsspeicher

Die ersten zwei Hexadezimalzahlen des `reg`-Feldes geben die Startadresse $(0)_{16}$ und die letzten zwei die Endadresse $(7\text{ff}00000)_{16}$ des physikalischen Speichers an [47], womit sich ein Bereich von knapp 2 GiB ergibt.

Der Linux-Kernel unterstützt reserved-memory-Knoten im Device Tree, über die Bereiche des physikalischen Speichers von der normalen Benutzung durch das Betriebssystem ausgeschlossen werden können [47], ähnlich wie mit den isolierten CPUs in Abschnitt 5.3.3 verfahren wurde. Listing 5.2 zeigt den reserved-memory Knoten, der in der vorliegenden Arbeit verwendet wird.

```
1 reserved-memory {
2   #address-cells = <2>;
3   #size-cells = <2>;
4   ranges;
5
6   imgbuf0 {
7     size = <0x0 0x60000000>;
8   };
9 };
```

Listing 5.2: Ausschnitt des Device Trees zur Reservierung eines Speicherbereichs

Die reservierten Speicherbereiche können statisch allokiert werden, indem eine Start- und Endadresse in der `reg`-Eigenschaft (analog zu Listing 5.1) angegeben wird; alternativ kann über die `size`-Eigenschaft die Größe des Bereichs ohne feste Startadresse vorgegeben werden [47].

Die letzte Variante wird gewählt, da die Startadresse des Bereichs für die betrachtete Anwendung nicht relevant ist, sofern sie sich nicht ändert. Gleichzeitig wird damit dem Kernel die Freiheit gegeben, den Speicherbereich beliebig zu platzieren und damit Kollisionen zu verhindern. Weitere Speicherbereiche, können zum Knoten hinzugefügt und über das VADER2-Kernelmodul in Prozesse gemappt werden.

5.3.5 Kernelmodul

Der Linux-Quellcode liegt offen und könnte theoretisch beliebig angepasst werden [41, S. 24]. Alternativ können sogenannte Kernel-Module programmiert werden,

die dynamisch in den Kernel geladen werden können [48] und damit im Kernelkontext, also mit sämtlichen Privilegien, ausgeführt werden.

Der Ansatz von dynamisch ladbaren Modulen hat den Vorteil, dass der Sourcecode des Kernels selbst unverändert bleibt und somit über Updates aktuell gehalten werden kann, während das Kernel-Modul auch mit älteren oder neueren Kernelversionen verwendet werden kann, sofern sich die Programmierschnittstelle nicht ändert.

Die Verwaltung des virtuellen Speichers ist wie in Abschnitt 5.3.4 erklärt Aufgabe des Betriebssystems und erfordert Privilegien des Kernels. Das VADER2-Kernelmodul stellt eine Schnittstelle zur Verfügung, über die die reservierten physikalischen Speicherbereiche in den virtuellen Speicherbereich eines oder mehrerer Prozesse gemappt werden können.

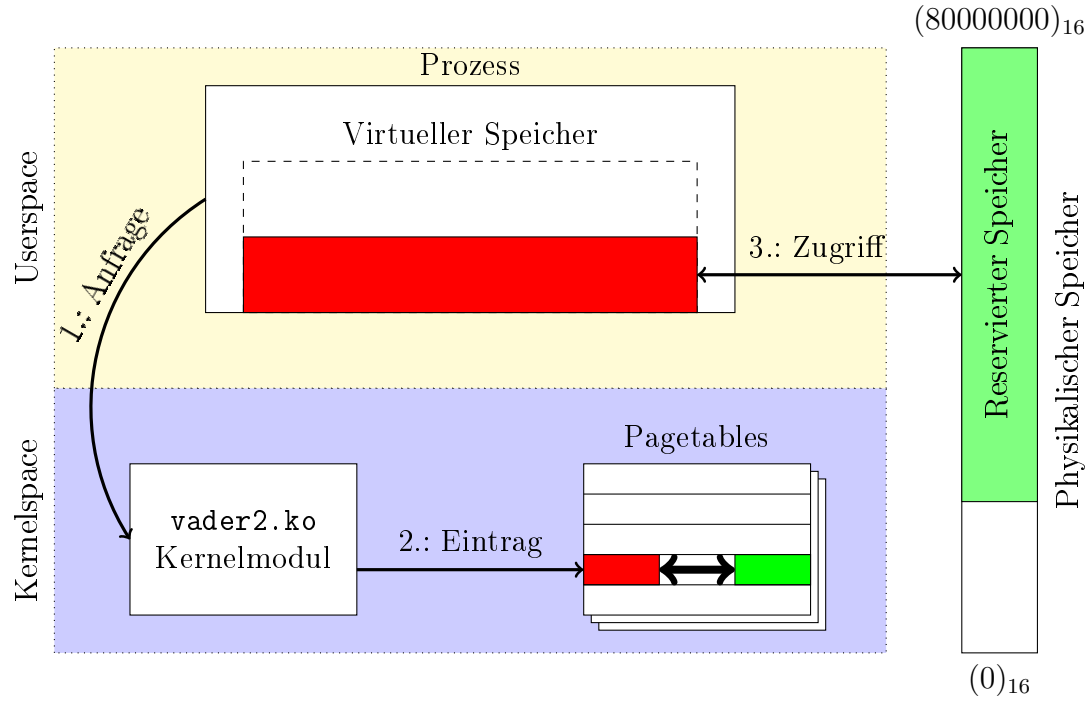

Abbildung 5.5: Vereinfachter Vorgang des Mappings des reservierten Speichers in einen User-Prozess

Abbildung 5.5 stellt den Ablauf vereinfacht schematisch dar: Der reservierte Speicher, der im Bild grün markiert ist, ist seit dem Hochfahren des Systems unverändert an einer konstanten Adresse im physikalischen Speicher vorhanden (s.

Abschnitt 5.3.4) und kann seitens der programmierbaren Logik z.B. mit Bildern gefüllt werden. Dies ist unabhängig davon, ob der reservierte Speicherbereich im Prozessorsystem in einen, mehrere oder überhaupt irgendeinen Prozess gemappt ist.

Soll vom Prozessor auf den reservierten Speicher zugegriffen werden, beispielsweise aus dem Prozess, der den Korrelationsalgorithmus ausführt, muss dieser Prozess eine Anfrage an das VADER2-Kernelmodul stellen, das daraufhin die entsprechenden Pages im virtuellen Speicher des Prozesses (rot markiert) in den Pagetables mit den physikalischen Pages des reservierten Speicher verknüpft. Anschließend können seitens des Prozesses normale Speicherzugriffe über die rot markierten virtuellen Adressen auf die grün markierten physikalischen Adressen erfolgen.

Das Mapping in mehrere Prozesse wird explizit ermöglicht, um Szenarien zu ermöglichen, in denen verschiedene Algorithmen von verschiedenen CPU-Kernen auf die gleichen Eingangsdaten zugreifen. In diesem Fall müssen die Implementierungen der Algorithmen sicherstellen, dass der Speicher entweder nur gelesen oder nur in einer Weise beschrieben wird, die die jeweils anderen Algorithmen nicht beeinflusst.

5.3.6 Software

Dieser Abschnitt führt Teile der Software auf, die in der vorliegenden Version von VADER2-Linux enthalten ist. Das Vorgehen zum Hinzufügen oder Entfernen von Softwarekomponenten in der Distribution ist in der beigelegten README-Datei erläutert.

Grob zusammengefasst werden allgemeine Entwicklungs- und Debug-Tools sowie notwendige Tools zur Verwaltung des Systems einbezogen. Als unnötig oder im Bezug auf die Leistungsfähigkeit störend betrachtete Komponenten, wie eine grafische Oberfläche, werden dabei explizit ausgeschlossen.

Software	Einsatzzweck
bash	Shell
openssh, openssh-sftp-server	Zugriff auf eine Shell über WLAN, Up- und Download von Dateien
gcc	C und C++ Compiler
libgomp	Parallelisierungsbibliothek für gcc (s. Abschnitt 6.1.2)
clang	Alternativer Compiler für verschiedene Sprachen
cmake	Buildsystem
git	Zugriff auf und Verwaltung von Sourcecode-Repositories
gdb	Debugger
perf	Performance-Messung von Programmen
python3	Skriptsprache
htop	Prozessmanager und Überwachung der Systemauslastung
devmem2	Zugriff auf physikalischen Speicher
usbutils	Tools für Zugriff auf USB-Geräte
dosfstools	Unterstützung für Dateisysteme wie z.B. FAT für USB-Sticks

Tabelle 5.1: Messergebnisse der zweiten Echtbildserie

5.3.7 Zusammenfassung

Mit VADER2-Linux wird eine minimale Linux-Distribution bereitgestellt, die in diesem und folgenden Projekten als Grundlage für die Implementierung von Geschwindigkeits- und Längensensoren dienen soll. Es werden dabei Maßnahmen getroffen, um eine durch das Betriebssystem möglichst ungestörte Umgebung für diese Ausführung von Algorithmen mit Echtzeitanforderungen zu schaffen.

Abbildung 5.6 zeigt die Auslastung des Systems nach dem Start bei der mit dem Ultra96-Board ausgelieferten Referenzdistribution.

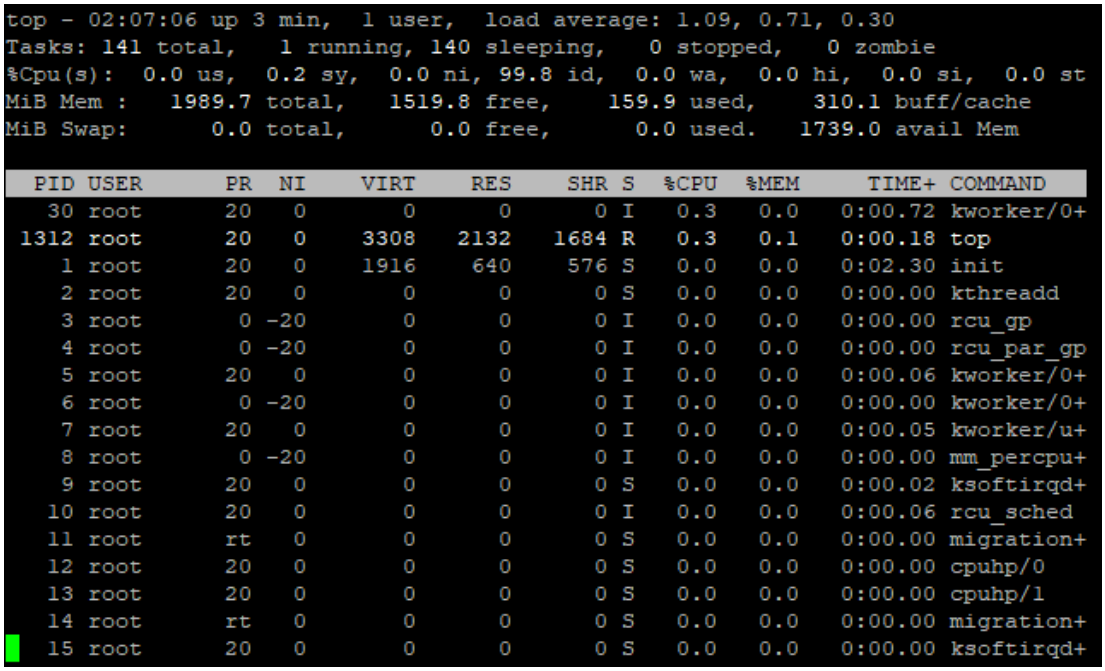

Abbildung 5.6: Ultra96-Referenzdistribution nach dem Start

In der Abbildung zu erkennen ist, dass direkt nach dem Start 141 Prozesse gestartet wurden und 160 MB Arbeitsspeicher belegt sind.

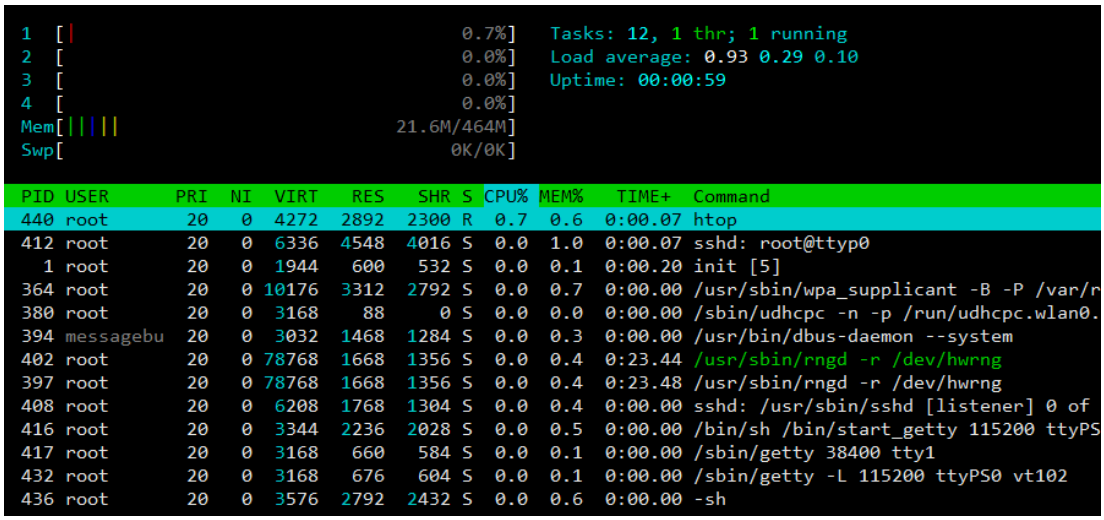

Abbildung 5.7: VADER2-Linux nach dem Start

Abbildung 5.7 zeigt die Auslastung des Systems mit VADER2-Linux nach dem

Start. Hier wurden zwölf Prozesse gestartet, die alle auf dem Housekeeping-Kern laufen.

Im Vergleich zu Abbildung 5.6, in dem 2 GB Arbeitsspeicher zur Verfügung stehen, stehen dem VADER2-Betriebssystem in Abbildung 5.7 nur knapp 500 MB zur Verfügung, von denen etwa 20 MB belegt sind.

Dass trotz der gleichen Hardware scheinbar weniger Speicher zur Verfügung steht, hängt damit zusammen, dass der restliche Speicher von der Kontrolle des Betriebssystems abgekoppelt und für den Messalgorithmus reserviert wurde (s. Abschnit 5.3.4).

5.4 CameraLink-Platine

Um die VADER2-Plattform analog zur VADER-Plattform mit einer CameraLink-Kamera verbinden zu können, wird eine entsprechende Aufsteckplatine für das Ultra96-Board entwickelt, gefertigt und bestückt.

Die Auslegung der CameraLink-Schnittstelle wird größtenteils aus dem VADER-Design übernommen. Bezüglich der Behandlung der Hochgeschwindigkeits-LVDS-Signale (Low Voltage Differential Signaling) wird auf die entsprechende Arbeit verwiesen [3].

Im Unterschied zum FPGA auf der VADER-Platine, das über 3,3 V-Eingänge verfügt [30], müssen die Spannungspegel für die Hochgeschwindigkeitsschnittstelle des Ultra96-Boards auf 1,2 V-CMOS-Pegel gebracht werden [33].

Dafür werden die 3,3 V-Signale durch Spannungsteiler auf ein Spannungsniveau von 1,2 V geteilt. Der entsprechende Schaltplan sowie das Layout der Platine sind Teil des Datenverzeichnisses (Anhang A.2).

Abbildung 5.8 zeigt die bestückte Schnittstellenplatine.

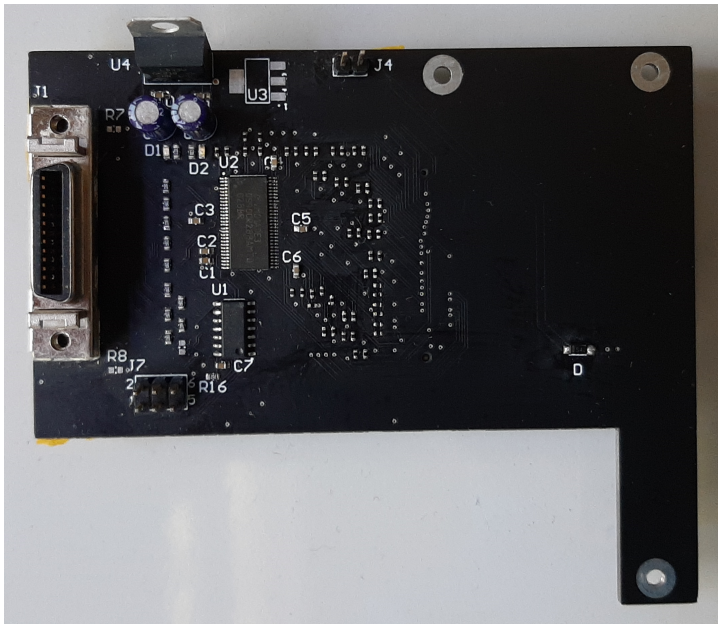

Abbildung 5.8: CameraLink-Schnittstellenplatine für die VADER2-Plattform

5.4.1 Verifikation

Im VADER-Projekt wurde das Simulink-Modell für einen CameraLink-Decoder für die Spyder 3-Kamera sowie das nachfolgende FIR-Rohbildfilter entwickelt und in der Simulation und Hardware verifiziert [2].

In der vorliegenden Arbeit wird die gleiche Kamera verwendet, weshalb das Decoder-Modul aus dem VADER-Projekt übernommen wird. Ebenfalls übernommen wird das FIR-Rohbildfilter, dieses wird aber auf die Ausgabe von SP-Fließkommasamples umgestellt.

Abbildung 5.9 zeigt einen Ausschnitt des Simulink-Modells des Kamera-Decoders und des Fließkomma-Rohbildfilters.

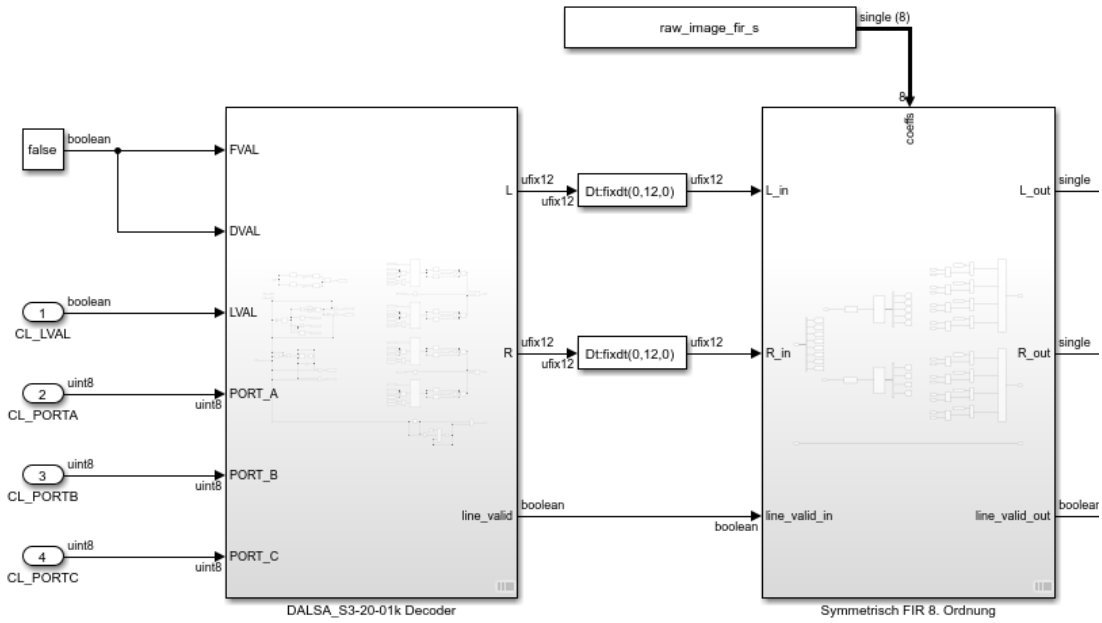

Abbildung 5.9: Simulink-Modell des Spyder 3-Kamera-Decoders und Rohbildfilters

Aus diesem Simulinkmodell wird analog zum Vorgehen im VADER-Projekt mit HDL-Coder Verilog-Code erzeugt und dieser in die programmierbare Logik des VADER2-Systems synthetisiert.

Zur Verifikation der korrekten Dekodierung der Bilddaten wird die Spyder 3-Kamera in einen Testbildmodus versetzt, der kontinuierlich Zeilenbilder in Form eines bekannten Referenzmusters ausgibt [25] (vgl. auch Verifikation der CameraLink-Schnittstelle im VADER-Projekt [2, S. 45f]).

Die Spyder 3-Kamera wird mit der Schnittstellenplatine verbunden und diese auf das Ultra96-Board aufgesteckt (Abbildung 5.10).

Abbildung 5.10: Ultra96-Board mit aufgesteckter Schnittstellenplatine und verbundenem CameraLink-Stecker

Anschließend wird das Referenzmuster im Debugger optisch und danach in MATLAB auf numerische Korrektheit überprüft:

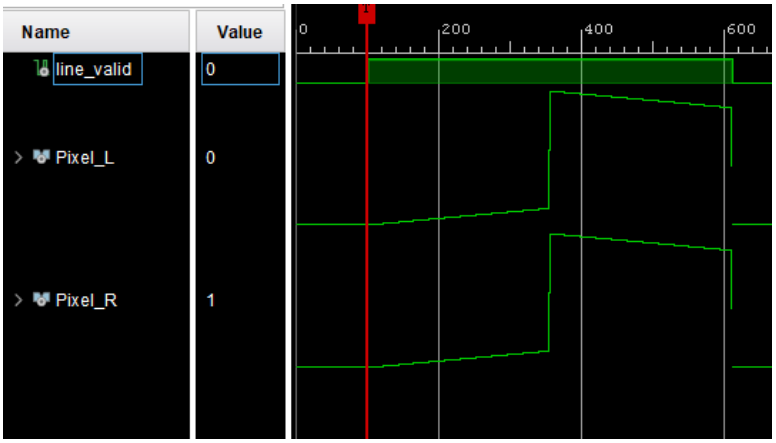

Abbildung 5.11: Referenzmuster der Kamera im Logik-Debugger

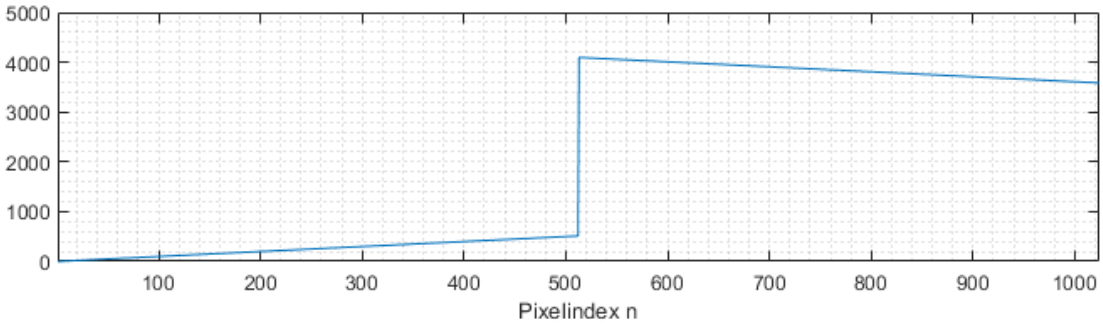

Abbildung 5.12: Referenzmuster der Kamera in MATLAB

Damit ist die CameraLink-Schnittstelle der VADER2-Plattform erfolgreich verifiziert.

Die vorliegende Arbeit verwendet keine Live-Bilder sondern aufgezeichnete Bildserien. Mit der CameraLink-Schnittstelle steht für Folgeprojekte allerdings die Möglichkeit der Verwendung der VADER2-Plattform als Framegrabber oder die Implementierung von Algorithmen, die Live-Bilder verarbeiten, offen.

6 Der Vader2-Algorithmus

Der VADER2-Algorithmus wird parallel in MATLAB und in C entwickelt, wobei keine Codegenerierung aus MATLAB zum Einsatz kommt. Dieses Vorgehen hat mehrere Gründe:

- Die grafischen Tools von MATLAB ermöglichen die Visualisierung von Signalen und Messergebnissen auf einfache Weise, was Plausibilitätsprüfungen ermöglicht und der praktischen Erfahrung nach generell zum Verständnis eines Systems beiträgt.
- MATLAB bietet für viele mathematische Operationen und Signalverarbeitungsalgorithmen vorgefertigte Funktionen an, die schnelle Entwicklungszyklen ermöglichen.
- Das Resultat einer Codegenerierung aus MATLAB ist in erster Linie Code, der im Rahmen der Möglichkeiten der Computerarchitektur numerisch äquivalent zum zugrundeliegenden Modell ist.

  Das vorrangige Ziel für den finalen C-Code des VADER2-Algorithmus ist die Ausführbarkeit in Echtzeit. Die Abstraktion zwischen dem MATLAB-Modell und daraus generiertem Code wird dabei als Problem betrachtet, weil nicht mehr die volle Kontrolle über die exakten Details der Implementierung behalten wird.

6.1 Durchführung der Benchmarks

Die in diesem Kapitel implementierten Elemente des Korrelationsalgorithmus werden auf der VADER2-Plattform, wie sie im letzten Kapitel beschrieben wurde, in C implementiert und getestet. Weil die Echtzeitfähigkeit des Algorithmus

zentrale Anforderung ist, werden für die vorgestellten Implementierung der Teilalgorithmen Benchmarks durchgeführt, die die jeweilige Laufzeit angeben.

Da für die Echtzeitfähigkeit des Algorithmus das Verhältnis aus der Geschwindigkeit der Datenverarbeitung und dem Eintreffen neuer Daten relevant ist, werden die Laufzeiten sowohl in µs als auch in Vielfachen von $T_\mathrm{s} = \frac{1}{68\,\mathrm{kHz}}$, der Frameperiode bei der maximalen Framerate der Spyder 3-Kamera, angegeben.

6.1.1 Neon / SIMD

Die Arm-Neon-Technologie ist eine Befehlssatzerweiterung für Arm-Kerne. Es handelt sich dabei um eine sogenannte SIMD-Architektur (Single Instruction, Multiple Data), was bedeutet dass einzelne Instruktionen gleichzeitig auf Datenvektoren mit mehreren gleichartigen Elementen operieren. [49, S. 95ff]

Die Neon-Register im verwendeten Arm-Core haben eine Breite von 128 bit und können damit vier SP-Fließkommazahlen fassen [50, S. 4206]. Dementsprechend ist es zum Beispiel möglich mit einer Instruktion (Single Instruction) zwei mal vier SP-Fließkommazahlen (Multiple Data) paarweise zu addieren oder zu multiplizieren.

Ein Beispiel für die Anwendung von SIMD ist in Abschnitt 6.2 bei der Bildung des Skalarprodukts zwischen zwei Vektoren für die diskrete Kreuzkorrelation gezeigt.

6.1.2 Parallelisierung

In der vorliegenden Arbeit wird zur Parallelisierung von Code-Abschnitten Open Multiprocessing verwendet (OpenMP, OMP), das in den (gcc)-Compiler integriert ist, und durch Präprozessordirektiven automatisiert die Parallelisierung bestimmter Code-Abschnitte ermöglicht [51].

```
1 float result[30000];
2 for (int i = 0; i < 30000; ++i)
3   result[i] = calculate_something(i);
```

Listing 6.1: Einfache Schleife in C

Die in Listing 6.1 gezeigte Schleife eignet sich abhängig von der Laufzeit von `calculate_something()` gut für eine Parallelisierung, weil jeder Schleifendurchlauf nur auf einen zuvor bekannten Abschnitt des `result`-Arrays schreibend zugreift.

Auf diese Weise würden sich mehrere Threads, die diese Aufgabe parallel abarbeiten, nicht gegenseitig behindern oder überschreiben. Beispielsweise könnte jeweils ein Thread auf den drei isolierten Prozessorkernen ausgeführt werden und jeder davon 10000 der insgesamt 30000 Schleifendurchläufe ausführen.

Die Sinnhaftigkeit einer Parallelisierung ist im vorgestellten Fall deshalb von der Laufzeit von `calculate_something()` abhängig, weil die Verteilung der Arbeit auf mehrere Threads selbst auch eine gewisse Zeit benötigt. Potentiell könnte so ein Fall eintreten, in dem die Verteilung der Schleifendurchläufe unter den Threads mehr Zeit in Anspruch nimmt, als die Abarbeitung der eigentlichen Berechnung.

Unter Nutzung von OMP ist für die Parallelisierung der oben gezeigten Schleife die in Listing 6.2 gezeigte Direktive vor der for-Schleife notwendig [51]:

```
1 #pragma omp parallel
2 for (int i = 0; i < 30000; ++i)
3   [...]
```

Listing 6.2: OMP-Direktive zur Parallelisierung

Anschließend kann der zugehörige Prozess über eine Umgebungsvariable auf die drei isolierten CPUs verschoben werden (Listing 6.3), was dazu führt, dass OMP die gekennzeichneten Bereiche zwischen den drei Kernen aufteilt.

```
1 $ # Je einen Thread auf den isolierten Kernen 1,2,3 erstellen
2 $ export OMP_PLACES={1},{2},{3}
3 # Die Threads sind an die genannten Kerne gebunden
4 $ export OMP_BIND_PROC=true
5 # Berechnungsprogramm ausführen
6 $ ./berechnung
```

Listing 6.3: Parametrisierung einer OMP-Parallelisierung durch Umgebungsvariablen

OMP abstrahiert die Verwaltung der Threads vollständig vom Nutzer, wodurch das Vorgehen nicht mehr exakt kontrolliert werden kann. Trotzdem wurde dieser Ansatz aus mehreren Gründen für das Projekt gewählt:

- Der Aufwand ist verglichen mit einer manuellen Erstellung und Verwaltung von Threads sowie der Verteilung der Berechnungsaufgaben zwischen ihnen gering. Die Parallelisierung einzelner Bereiche kann durch das Einfügen einer einzelnen Zeile zu- und abgeschaltet werden.
- Die Einstellungen für die Threads können nach der Kompilation über Umgebungsvariablen geändert werden. Damit ist es möglich, auf einfache Weise den Einfluss verschiedener Anzahlen von Kernen oder Threads auf die Laufzeit zu ermitteln.
- OMP lässt sich mit einer zukünftigen Code-Generierung verbinden, da einerseits z.B. die Simulink DSP-Systems Toolbox bereits OMP unterstützt [52] und darüberhinaus jede Schleife durch das Einfügen der entsprechenden Direktive parallelisiert werden kann.
- OMP wird von FFTW, einer Bibliothek zur Berechnung der schnellen Fouriertransformation (FFT) (s. Abschnitt 6.1.3), standardmäßig unterstützt und würde so die Verwendung eines einheitlichen Threading-Ansatzes im gesamten Algorithmus erlauben [53].
- Hätte sich herausgestellt, dass eine Parallelisierung mit OMP zu langsam ist, hätte zu einer manuell optimierten Threading-Implementierung gewechselt werden können.

6.1.3 FFTW

The fastest fourier transform in the west (FFTW) ist eine Bibliothek, die Implementierungen für verschiedene Arten der ein- und mehrdimensionalen FFT und IFFT für verschiedene Rechnerarchitekturen bereitstellt. Für den verwendeten Arm-Prozessor unterstützt FFTW Fließkommazahlen sowie die Neon-SIMD-Erweiterungen. [53]

FFTW kann grundsätzlich (I)FFTs beliebiger Länge berechnen, die Dokumentation weist allerdings darauf hin, dass Längen, die sich in möglichst kleine Primfak-

toren zerlegen lassen (2,3,5,7), überlicherweise die besten Ergebnisse hinsichtlich der Laufzeit erzielen. [53]

Weil sowohl die FFT als auch die IFFT jeweils für die Kreuzkorrelation im Frequenzbereich und die damit verbundene Interpolation im Frequenzbereich relevant sind (Abschnitte 6.3 und 6.5), ist die erreichbare Rechenzeit für diese Transformationen von Bedeutung für die zur Verfügung stehenden Optionen zur Implementierung des VADER2-Algorithmus.

Daher werden die Ganzzahlen zwischen 700 und 2048 faktorisiert und dabei diejenigen herausgesucht, die sich in die oben genannten, kleinen Primfaktoren zerlegen lassen.

Da in der vorliegenden Arbeit nur reellwertige Originalfunktionen behandelt werden, werden spezielle Transformationen aus der FFTW-Bibliothek verwendet: `dft_r2c_1d` (discrete fourier transform, real to complex, one-dimensional) führt eine eindimensionale Transformation vom reellwertigen Originalbereich in den Frequenzbereich durch [53]. Da bei reellwertigen Ausgangssignalen die beiden Hälften des Spektrums redundante Information beinhalten [22, S. 131], wird nur eine Hälfte des Spektrums ausgegeben, um den Vorgang zu beschleunigen.

Das Gegenstück zu dieser Transformation ist `dft_c2r_1d` (discrete fourier transform, complex to real, one-dimensional) und transformiert ein entsprechendes Halbsprektrum zurück in den Originalbereich [53].

Bei den folgenden Benchmarks wurde zunächst versucht, die Berechnungen durch eine Parallelisierung mit OMP zu beschleunigen. Dabei ergab sich allerdings eine Beschleunigung um etwa 15 % bei der Verwendung von drei Kernen statt einem Kern, was als ungünstiges Kosten zu Nutzen-Verhältnis angesehen wird.

Der Grund für diese Beobachtung könnte darin liegen, dass alle durchgeführten Transformationen vergleichsweise wenig anspruchsvoll (nur reelle Eingangsdaten, eindimensional, wenige Samples) sind, sodass die Ausführungszeit schon auf einem Kern so niedrig ist, dass der zusätzliche Aufwand, der sich durch die Verteilung der Berechnungsaufgaben auf mehrere Kerne ergibt, schlussendlich die durch die Parallelisierung theoretisch erreichbare Zeiteinsparung zunichte macht. Aus diesem Grund werden die folgenden Benchmarks auf einem Kern durchgeführt.

Tabelle 6.1 stellt aus Gründen der Übersichtlichkeit nur einen Ausschnitt der

Ergebnisse für $1000 \leq N \leq 2048$ dar. Die Rechenzeiten der IFFT und FFT gleicher Länge liegen jeweils in einem Bereich von etwa 10 % zueinander, weshalb nur ein Wert angegeben wird, der für eine symmetrische Transformation in jede Richtung veranschlagt werden kann.

Benchmark

N	Primfaktoren	Rechenzeit / µs	T_s	N	Primfaktoren	Rechenzeit / µs	T_s
1000	$2^3 \cdot 5^3$	19	1,3	1500	$2^2 \cdot 3 \cdot 5^3$	34	2,3
1008	$2^4 \cdot 3^2 \cdot 7$	22	1,5	1536	$2^5 \cdot 7^2$	24	1,6
1024	2^{10}	16	1,1	1701	$3^3 \cdot 7$	75	5,1
1152	$2^7 \cdot 3^2$	21	1,4	1728	$2^6 \cdot 3^3$	37	2,5
1200	$2^4 \cdot 3 \cdot 5^2$	22	1,5	1792	$2^8 \cdot 7$	35	2,4
1215	$3^5 \cdot 5$	46	3,1	1800	$2^3 \cdot 3^2 \cdot 5^2$	39	2,6
1280	$2^8 \cdot 5$	20	1,4	1890	$2 \cdot 3^3 \cdot 5 \cdot 7$	63	4,3
1350	$2^6 \cdot 3 \cdot 7$	35	2,4	1920	$2^7 \cdot 3 \cdot 5$	35	2,4
1400	$2^3 \cdot 5^2 \cdot 7$	30	2,0	2000	$2^4 \cdot 5^3$	40	2,7
1458	$2 \cdot 3^6$	50	3,4	2048	2^{11}	35	2,4

Tabelle 6.1: Benchmarks für verschiedene Real-to-Complex FFTs mit FFTW auf einem Prozessorkern

6.2 Kreuzkorrelation im Originalbereich

Die zeitkontinuierliche Kreuzkorrelation wurde in Abschnitt 2.5 eingeführt, für diskrete Signale geht das Integral in eine Summe über.

Gleichung 6.1 gibt die Berechnungsvorschrift der diskreten Kreuzkorrelation aus der Dokumentation der MATLAB-Funktion `xcorr` [54] wieder, die z.B. in den Implementierungen der Algorithmen in Kapitel 3 und 4 verwendet wird. Die komplexe Konjugation eines der Faktoren bei der Produktbildung wird dabei vernachlässigt, da in der vorliegenden Arbeit nur rein reelle Signale korreliert werden.

$$(x \times y)(m) = \begin{cases} \sum_{n=0}^{N-|m|-1} x(n+|m|) \cdot y(n) & \text{für} \quad m \geq 0 \\ \sum_{n=0}^{N-|m|-1} x(n) \cdot y(n+|m|) & \text{für} \quad m < 0 \end{cases} \tag{6.1}$$

Es wird angemerkt, dass die Definition der Kreuzkorrelationsfunktion in der MATLAB-Dokumentation sich in der Reihenfolge der Operanden von der Definition 2.3 in Abschnitt 2.5 unterscheidet: In der Definition aus der Literatur wird der zweite Operand um das Argument der Kreuzkorrelationsfunktion verschoben, in der Definition aus MATLAB der erste.

Wegen der unendlichen Integrationsgrenzen gilt:

$$\begin{aligned} \varphi_{a,b}(\tau) &= \int_{-\infty}^{\infty} a(t) \cdot b(t+\tau)\, \mathrm{d}t \\ &= \int_{-\infty}^{\infty} a(t-\tau) \cdot b(t)\, \mathrm{d}t = \int_{-\infty}^{\infty} b(t) \cdot a(t-\tau)\, \mathrm{d}t \\ &= \varphi_{b,a}(-\tau) \end{aligned}$$

Damit ist die Kreuzkorrelation zwar nicht kommutativ, das Kommutieren der Operanden lässt sich aber durch das Negieren des Arguments kompensieren.

Weil der VADER2-Algorithmus in MATLAB modelliert wird und die Definitionen sich nur im Vorzeichen des Arguments unterscheiden, wird im Folgenden die Definition 6.1 weiterverwendet.

Für die in Abschnitt 4.2.3 beschriebene Suche nach dem globalen Maximum der Korrelationsfunktion ist die Berechnung der KKF für mehrere m notwendig.

Für die Operation $r = a + (b \cdot c)$ existiert eine Vektorinstruktion im Neon-Befehlssatz, die die Berechnung mit vier SP-Fließkommazahlen pro Operand gleichzeitig ausführt [50, S. 5551]. Die Berechnung eines einzelnen Eintrages der Kreuzkorrelationsfunktion kann daher wie in Abbildung 6.1 gezeigt durch Vektorisierung beschleunigt werden.

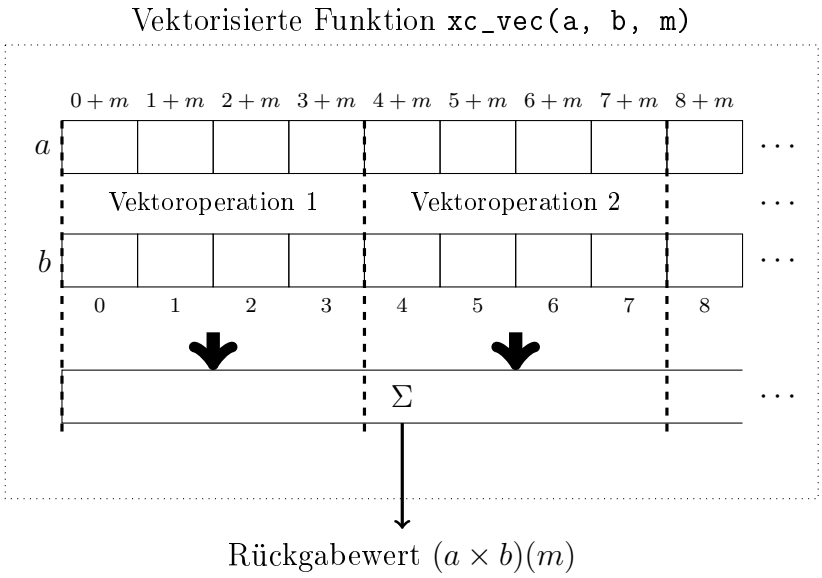

Abbildung 6.1: Vektorisierte Berechnung eines Eintrages einer Kreuzkorrelationsfunktion aus den Fließkommavektoren a und b

Für die Berechnung der gesamten Kreuzkorrelationsfunktion ist nun eine Schleife notwendig, die die vektorisierte Berechnung von $(a \times b)(m)$ für alle interessierenden m durchführt. Für diese Schleife werden zwei Varianten für einen beispielhaft zu berechnenden Bereich $0 \leq m \leq 299$ gezeigt.

Die erste Variante in Abbildung 6.2 teilt die Iterationen zwischen den drei isolierten CPU-Kernen auf.

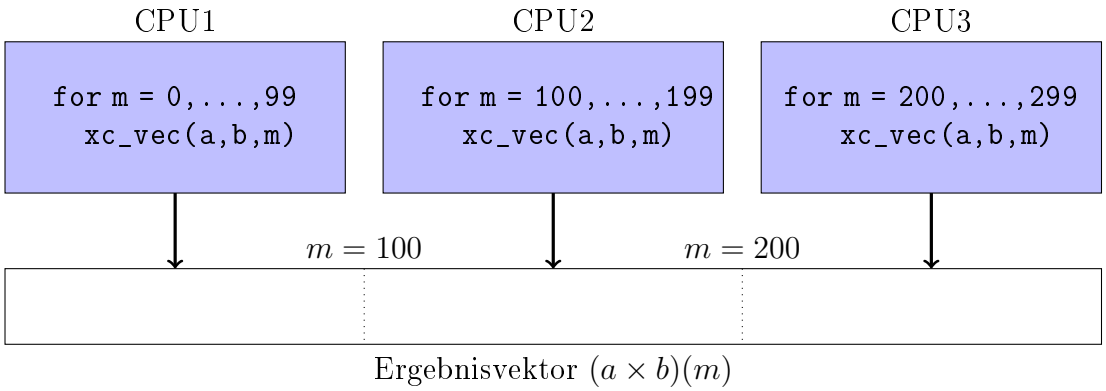

Abbildung 6.2: Parallelisierte und vektorisierte Berechnung einer Kreuzkorrelationsfunktion im Originalbereich

Die zweite Variante ohne Parallelisierung ist in Abbildung 6.3 dargestellt.

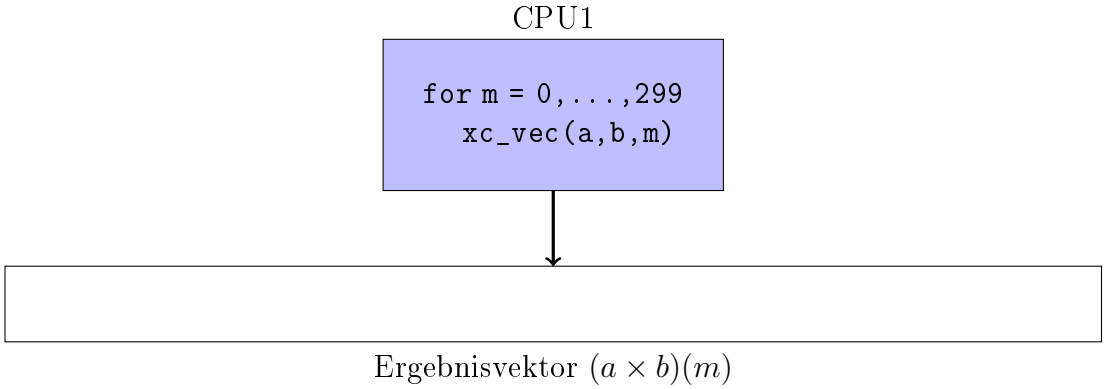

Abbildung 6.3: Vektorisierte Berechnung einer Kreuzkorrelationsfunktion im Originalbereich ohne Parallelisierung

Die beiden vorgestellten Varianten werden in ihrer Laufzeit für verschiedene Bereiche der Verschiebung m untersucht:

Bereich für m	Kerne	Rechenzeit / µs	T_s
$-1023 \leq m \leq 1023$	1	1929	131
	3	1057	72
$-400 \leq m \leq 400$	1	1197	81
	3	464	32
$-150 \leq m \leq 150$	1	516	35
	3	182	12
$50 \leq m < 60$	1	17	1,2
	3	10	0,7

Tabelle 6.2: Rechenzeiten für die diskrete Kreuzkorrelationen mit und ohne Parallelisierung

Die Ergebnisse in Tabelle 6.2 zeigen einerseits, dass eine Verkürzung der Laufzeit auf $\frac{1}{3}$ mit der verwendeten Parallelisierung auf drei Kernen statt einem Kern nicht regelmäßig erwartet werden kann.

Außerdem ist ersichtlich, dass die Berechnung der vollständigen KKF mit 2047 Einträgen selbst mit Parallelisierung über 70 Frameperioden andauert und somit z.B. nicht im Rahmen des statischen Korrelationsalgorithmus mit Chunklängen von 25 oder 50 Bildern ausführbar ist, wobei die benötigte Zeit für eine Interpolation noch nicht berücksichtigt wurde.

Selbst bei einer hypothetischen idealen Parallelisierung würde die Berechnung noch etwa $\frac{131 \cdot T_\mathrm{s}}{3} \approx 44 \cdot T_\mathrm{s}$ benötigen, wodurch bei einer Chunklänge von 50 Zeilen die diskrete Kreuzkorrelation alleine die isolierten CPUs bereits zu 88 % auslasten würde.

Aus den Untersuchungen in Abschnitt 4.2.3 ist ersichtlich, dass die Suche nach dem globalen Maximum innerhalb der KKF ab Verschiebungen von etwa 150 Pixeln schnell weniger zuverlässig wird. Aus diesem Grund könnte die Berechnung der KKF auf den Bereich $-150 \leq m \leq 150$ begrenzt werden, um damit die benötigte Rechenzeit zu reduzieren. In diesem Fall kann die KKF mit Parallelisierung in etwa zwölf Frameperioden berechnet werden.

Betrachtet man den dynamischen Korrelationsalgorithmus mit Qualitätsmerkmal (Abschnitt 4.4.1), so müssen pro berechnetem Weginkrement mehrere Kreuzkorrelationsfunktionen berechnet werden, um anschließend die beste auszuwählen.

Wenn zehn KKF jeweils für $-150 \leq m \leq 150$ berechnet werden würden, ergibt sich alleine für diese Berechnung eine Dauer von etwa $120 \cdot T_\mathrm{s}$. In dieser Zeit hat sich ein Objekt, das sich mit $5\,\mathrm{m\,s^{-1}}$ bewegt, um etwa 250 Pixel auf der Kamerazeile verschoben und liegt damit bereits außerhalb des Verschiebungsbereichs von 150 Pixeln.

Die folgenden Abschnitte untersuchen Ansätze um die benötigte Rechenzeit für die Kreuzkorrelation so weit zu senken, dass dies die Implementierung eines dynamischen Korrelationsalgorithmus mit Qualitätsmerkmal und zusätzlicher Interpolation ermöglicht.

6.3 Kreuzkorrelation im Frequenzbereich

6.3.1 Zyklische / periodische Kreuzkorrelation

Dass die Kreuzkorrelation im Frequenzbereich durchgeführt werden kann, wurde bereits am zeit- bzw. ortskontinuierlichen Fall in Abschnitt 2.5 gezeigt.

Ein praktisches Beispiel für die diskrete Kreuzkorrelation im Frequenzbereich findet sich in der aktuellen Version (R2021a) von MATLAB. Zwar wird in der Dokumentation die in Gleichung 6.1 wiedergegebene Rechenvorschrift im Originalbereich aufgeführt [54], tatsächlich wird die Berechnung intern aber auf andere Weise durchgeführt, wie Listing 6.4 zeigt [1].

```
1 m2 = findTransformLength(m);
2 X = fft(x,m2,1);
3 Y = fft(y,m2,1);
4 if isreal(x) && isreal(y)
5   c1 = ifft(X.*conj(Y),[],1,'symmetric');
6 else
7   [...]
```

Listing 6.4: Ausschnitt des internen MATLAB-Codes der Funktion `xcorr`

Dabei handelt es sich um den Ansatz der sogenannten schnellen Korrelation (Fast Correlation), der sich analog zum zeitkontinuierlichen Fall in Abschnitt 2.5 aus dem Faltungstheorem der DFT herleiten lässt [22, S. 216f]:

Für die reellwertigen Signale $x(n)$ und $y(n)$ mit der Periode N gilt danach:

$$\sum_{n=0}^{N-1} x(n+m) \cdot y(n) \circ\!\!-\!\!\bullet\, X(k) \cdot \overline{Y(k)} \quad \text{für } m = 0, ..., N-1 \tag{6.2}$$

Wegen der periodischen Signale wird diese Korrelation auch als zyklische oder periodische Kreuzkorrelation bezeichnet [22, S. 216], für die der Operator $\otimes$ eingeführt wird.

[1] `MATLAB/R2021a/toolbox/matlab/datafun/xcorr.m` ab Zeile 245

Mit der inversen DFT folgt

$$(x \otimes y)(m) = \sum_{n=0}^{N-1} x(n+m) \cdot y(n) = \text{IDFT}\left[X(k) \cdot \overline{Y(k)}\right] \tag{6.3}$$

$$= \frac{1}{N} \sum_{k=0}^{N-1} \left[X(k) \cdot \overline{Y(k)}\right] \mathrm{e}^{j2\pi km/N} \tag{6.4}$$

Gleichung 6.3 findet sich in Listing 6.4 wieder: Zunächst werden die Originalfunktionen x und y durch eine FFT in den Frequenzbereich überführt, anschließend die Spektren $X(n)$ und $\overline{Y(n)}$ multipliziert und das Ergebnis zurücktransformiert.

Setzt man die Berechnungsvorschrift der inversen DFT ein, ergibt sich Gleichung 6.4. Dabei fällt auf, dass das Argument m der Originalbereichsfunktion, das die Verschiebung zwischen den Originalfunktionen $x(n)$ und $y(n)$ beschreibt, in der Berechnung im Frequenzbereich auf der rechten Seite der Gleichung in den Exponentialterm eingeht.

Dass Verschiebungen im Originalbereich in einen Exponentialterm im Frequenzbereich übergehen, ist auch aus dem Verschiebungssatz der Fouriertransformation bekannt [22, S. 151]:

$$x(n+m) \circ\!\!-\!\!\bullet\, X(k)\mathrm{e}^{j2\pi km/N} \tag{6.5}$$

Durch Umstellung der Gleichung 6.4 ergibt sich damit eine alternative Interpretation des Vorgangs, der den direkten Bezug zur Kreuzkorrelation im Originalbereich herstellt:

$$(x \otimes y)(m) = \sum_{n=0}^{N-1} x(n+m) \cdot y(n) \qquad = \underbrace{\frac{1}{N} \sum_{k=0}^{N-1} \left[X(k) \cdot \overline{Y(k)} \right] \mathrm{e}^{j2\pi km/N}}_{\mathrm{IDFT}\left[X(k) \cdot \overline{Y(k)}\right]} \tag{6.6}$$

$$(x \otimes y)(m) = \underbrace{\sum_{n=0}^{N-1} x(n+m) \cdot y(n)}_{\text{Energieberechnung Originalbereich}} \qquad = \underbrace{\frac{1}{N} \sum_{k=0}^{N-1} \left[\underbrace{X(k) \cdot \mathrm{e}^{j2\pi km/N}}_{\mathrm{DFT}\left[x(n+m)\right]} \cdot \overline{Y(k)} \right]}_{\text{Energieberechnung Frequenzbereich}} \tag{6.7}$$

Weil der Ausdruck auf der linken Seite des Gleichheitszeichens als eine Form der Kreuzkorrelation auf einer Energieberechnung zwischen zwei Signalen unter bestimmten Verschiebungen basiert, muss auch der Ausdruck im Frequenzbereich auf der rechten Seite eine solche Energie berechnen. Dass eine Berechnung der Signalenergie auch im Frequenzbereich erfolgen kann, ist Aussage des Parseval'schen Theorems [22, S. 210].

Der Vergleich der zwei Gleichungen 6.6 und 6.7 gibt dabei bereits einen Hinweis auf die Effizienz der Fast Correlation: Die IDFT, für die in Form der IFFT eine hochoptimierte Implementierung zur Verfügung steht [22, S. 132], erfüllt in der Berechnung gleichzeitig zwei Aufgaben: Einerseits die Verschiebung von $x(n)$ um m Samples für jeden Ausgangswert und andererseits die folgende Energieberechnung.

Im Folgenden werden die Implikationen der N-Periodizitäten von $x(n)$ und $y(n)$ erörtert.

Vergleicht man die Grenzen des Summenoperators in den Gleichungen 6.1 und 6.3, fällt auf, dass die Berechnungsvorschrift für die diskrete Kreuzkorrelation bei der oberen Grenze die Verschiebung m berücksichtigt. Dies erschließt sich

dadurch, dass die diskrete Kreuzkorrelation zwei zeit- oder ortsbegrenzte Signale betrachtet, die sich mit wachsenden m so gegeneinander verschieben, dass sie nur noch teilweise zur Deckung kommen.

Aus Sicht des Berechnungsaufwandes ist dies vorteilhaft, da mit wachsenden Verschiebungen m auch weniger Produkte in die Summe eingehen. Andererseits führen große Verschiebungen dazu, dass nur noch marginale Teile der betrachteten Funktionen überhaupt in die Korrelation eingehen. Die Implikationen davon für den Korrelationsalgorithmus wurden in Abschnitt 4.2.3 behandelt.

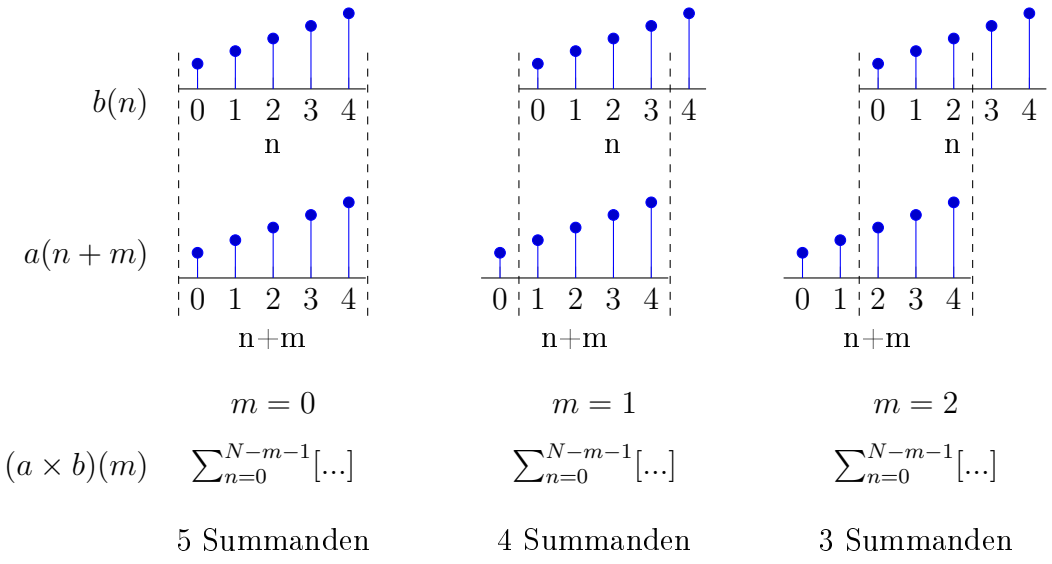

Abbildung 6.4: Operationen zur Berechnung der diskreten Kreuzkorrelation

Abbildung 6.4 zeigt die sich gegeneinander verschiebenden zeit- oder ortsbegrenzten Signale a und b und die daraus resultierende Anzahl an Summanden (und damit auch Produkten) bei der Berechnung der diskreten Kreuzkorrelation.

Für die zyklische Kreuzkorrelation ist der Vorgang in Abbildung 6.5 dargestellt, wobei x und y N-periodische Signale sind, für deren Basisperiode gilt:

$$x(n) = a(n) \quad \text{und} \quad y(n) = b(n) \quad \forall\, 0 \leq n < N$$

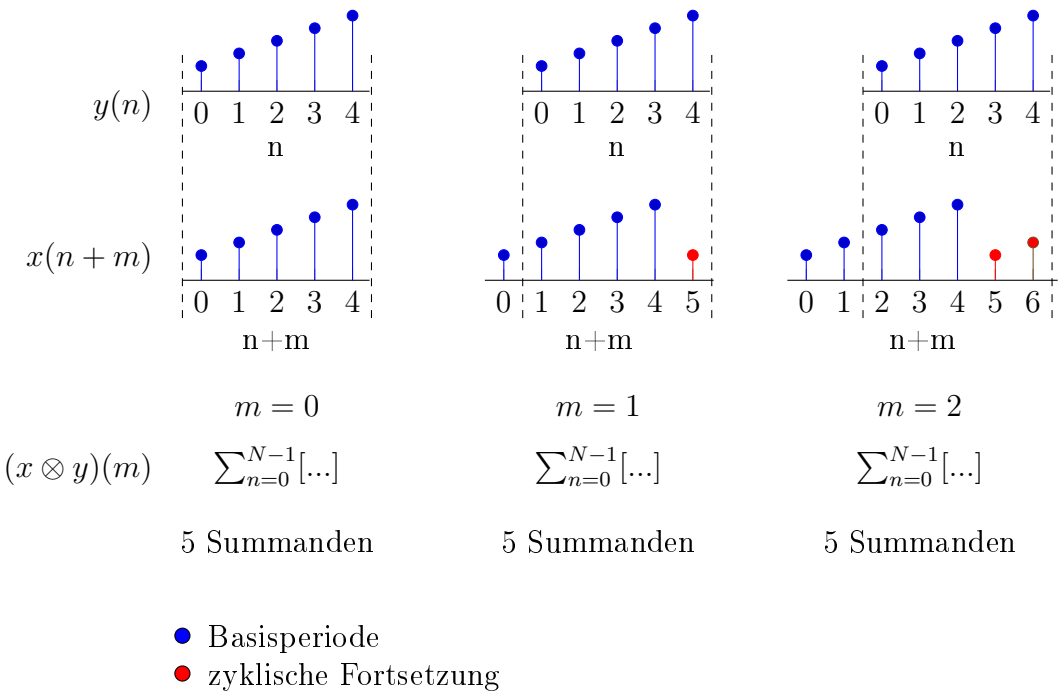

Abbildung 6.5: Operationen zur Berechnung der zyklischen Kreuzkorrelation

Aus dem Vergleich der Abbildungen 6.4 und 6.5 ist ersichtlich, dass die Periodizitäten von $x(n)$ und $y(n)$ dazu führen, dass $(x \otimes y)(m)$ und $(a \times b)(m)$ nicht die gleichen Ergebnisse liefern, weshalb die diskrete Kreuzkorrelation zunächst nicht durch die potentiell effizientere zyklische Kreuzkorrelation ersetzt werden kann.

Die N-Periodizität von $x(n)$ und $y(n)$ ist nicht frei gewählt sondern ergibt sich erzwungenermaßen aus der Verknüpfung mit den diskreten Frequenzbereichsdarstellungen $X(k)$ und $Y(k)$ über die (inverse) DFT.

Diese Tatsache wird in der Literatur damit erklärt, dass einem diskreten, periodischen Spektrum eine periodische Originalfunktion gegenübersteht [22, S. 131], lässt sich aber auch an der Berechnungsvorschrift der IDFT (vgl. Gleichung 6.6) im Zusammenhang mit der 2π-Periodizität von e^{jx} ablesen.

Um die diskrete Kreuzkorrelation durch die zyklische Kreuzkorrelation ersetzen zu können, müssen Maßnahmen getroffen werden, die im Folgenden erarbeitet werden.

Auf der Grundlage der gewonnen Erkenntnisse aus den Abbildungen 6.4 und 6.5 lassen sich zunächst zwei mögliche Ansätze formulieren:

1. Die N-Periodizität von $x(n)$ und $y(n)$ führt dazu, dass mit steigendem m immer größere Teile der Randbereiche der beiden Funktionen, also die (bzgl. der Basisperiode) ersten Samples der ersten Funktion mit den letzten Samples der zweiten Funktion, verrechnet werden, was zu einem Ergebnis führt, das von der diskreten Kreuzkorrelation abweicht.

   In Abschnitt 4.2.3 wurde gezeigt, dass Verschiebungen über etwa 150 Pixel hinaus zu evaluieren nicht mehr sinnvoll ist. Gleichzeitig wurde deutlich, dass gerade die Randbereiche der Zeilenbilder durch Einflüsse von Optik und/oder Beleuchtung wenig verwertbare Information beinhalten. Unter diesem Gesichtspunkt könnte die Abwägung getroffen werden, über die Korrelation der ohnehin wenig relevanten Randbereiche durch die zyklische Wiederholung hinwegzusehen, wenn dies im Gegenzug einen nennenswerten Vorteil bei der Berechnungskomplexität böte.

2. $x(n)$ und $y(n)$ könnten mit Null-Samples verlängert werden (sog. Zero-Padding), sodass die Periode N größer wird. Konkret müsste die Periode so lang sein, dass bei allen Verschiebungen durch die zyklische Fortsetzung nur Null-Samples und nicht Teile des eigentliches Signals auf der anderen Seite wieder in die Rechnung eingehen.

   Dieser Ansatz hat den doppelten Nachteil, dass die benötigten DFTs und IDFTs länger werden und damit auch die Berechnung $X(k) \cdot \overline{Y(k)}$ für mehr Elemente durchgeführt werden muss.

Die in Listing 6.4 gezeigte MATLAB-Implementierung wendet den zweiten Ansatz an. Wegen der zu erwartenden Ungenauigkeiten des ersten Ansatzes wird dieser zunächst verworfen und nur darauf zurückgegriffen, falls der zweite Ansatz durch die höhere Berechnungskomplexität problematisch werden würde.

Die Abbildung 6.6 zeigt das Prinzip des Zero-Paddings als Fortsetzung der Beispiele in den Abbildungen 6.4 und 6.5. Dabei werden $y(n)$ und $x(n)$ jeweils um zwei Zero-Samples am Ende zu $y_{+2}(n)$ bzw. $x_{+2}(n)$ ergänzt. Damit ist $N_{+2} = N+2$ die Periode der gepaddeten Signale.

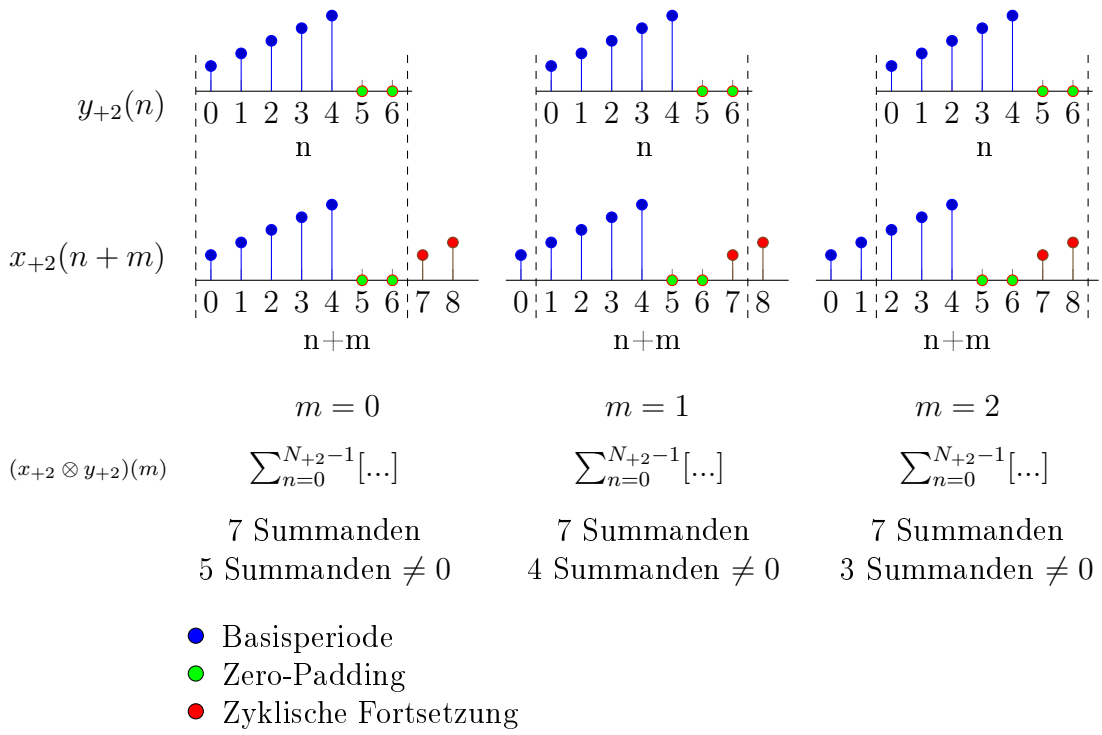

Abbildung 6.6: Operationen zur Berechnung der zyklischen Kreuzkorrelation mit Zero-Padding

Vergleicht man Abbildungen 6.6 und 6.4, so findet die Summenbildung bei der zyklischen Kreuzkorrelation zwar nun über $N+2$ Produkte statt, wegen des Zero-Paddings werden aber diejenigen Produkte zu Null, die in der diskreten Kreuzkorrelation durch die gegenseitige Verschiebung der zeit- bzw. ortsbegrenzten Signale gar nicht zustande gekommen wären.

Damit löst das Zero-Padding um zwei Samples für Verschiebungen $|m| \leq 2$ das in Abbildung 6.5 auftretende Problem der zusätzlichen Produkte verglichen mit der Originalbereichsmethode.

Mit dem Zero-Padding der beiden zu korrelierenden Funktionen ergeben sich unter Berücksichtigung bisheriger Erkenntnisse zwei weitere mögliche Ansätze für die Frequenzbereichs-Korrelation von zwei Zeilenbildern mit je 1024 Pixeln:

1. Da bekannt ist, dass Auswertungen der KKF über Verschiebungen von etwa ± 150 Pixeln hinaus nicht zuverlässig zur Maximumssuche beitragen, wird zunächst versucht, die beiden Funktionen nur mit 150 Null-Samples zu er-

gänzen, um ihre Perioden so klein wie möglich zu halten und sie gleichzeitig um bis zu 150 Samples in beide Richtungen verschieben zu können, ohne dass das eigentliche Signal durch zyklische Fortsetzung an den Rändern wieder in die Korrelation eingeht.

2. Die zweite Variante ist der allgemeine Ansatz, die Funktionen auf eine Länge von $1024 + 1023 = 2047$ aufzufüllen, um die vollständige Kreuzkorrelationsfunktion $(x \times y)(m)$ für $-1023 \leq m \leq 1023$ durch die zyklische Kreuzkorrelation berechnen zu können.

Für beide Varianten wird zunächst die Berechnungsvorschrift festgehalten:

Padding mit 150 Nullen

$$N_{+150} = 1024 + 150 = 1174$$

$$x_{+150}(n) = \begin{cases} x(n) & \text{für } 0 \leq n \leq 1023 \\ 0 & \text{für } 1024 \leq n \leq 1173 \end{cases}$$

$$y_{+150}(n) = \begin{cases} y(n) & \text{für } 0 \leq n \leq 1023 \\ 0 & \text{für } 1024 \leq n \leq 1173 \end{cases}$$

Bei der Variante mit 150 angehängten Null-Samples hat die Basisperiode der IDFT $N_{+150} = 1174$ Einträge, von denen allerdings nur $2 \cdot 150 + 1$ für die Kreuzkorrelation interessierend sind. Die entsprechenden Einträge lassen sich durch Umschreiben des Kerns der IDFT in eine Zeitverschiebung finden:

$$(x_{+150} \otimes y_{+150})(m) = \text{IDFT}\left[X_{+150}(k) \cdot \overline{Y_{+150}(k)}\right] \tag{6.8}$$

$$= \frac{1}{N_{+150}} \sum_{k=0}^{N_{+150}-1} \left[\underbrace{X_{+150}(k) \cdot \mathrm{e}^{j2\pi km/N_{+150}}}_{\text{DFT}[x_{+150}(n+m)]} \cdot \overline{Y_{+150}(k)} \right] \quad \forall\, m \geq 0 \tag{6.9}$$

Die Einträge der Korrelationsfunktion für $0 \leq m \leq 150$ lassen sich nach der Überlegung in Gleichung 6.9 direkt zu Beginn der Basisperiode der IDFT ablesen. Für negative m muss nach der Definition der diskreten Kreuzkorrelation eigentlich

$y_{+150}(n)$ um $|m|$ verschoben werden. Wegen der Periodizitäten von y_{+150} und x_{+150} kann aber stattdessen auch x_{+150} in die Gegenrichtung verschoben werden, womit die Überlegung aus Gleichung 6.9 fortgeführt wird:

$$x_{+150}(n-m) = x_{+150}(N_{+150}+n-m) \tag{6.10}$$

$$\mathrm{DFT}\big[x_{+150}(n+N_{+150}-m)\big] = X_{+150}(k)\cdot \mathrm{e}^{j2\pi k(N_{+150}-m)/N_{+150}} \tag{6.11}$$

Aus dem Vergleich der Exponentialterme in den Gleichungen 6.9 und 6.11 folgt, dass sich die Einträge der Korrelationsfunktion für negative m am Ende der Basisperiode der IDFT bei den Einträgen $N_{+150}-|m|$ finden. Zusammenfassend gilt damit für die Berechnung der diskreten KKF im Frequenzbereich:

$$(x_{+150}\otimes y_{+150})(m) = \mathrm{IDFT}\left[X_{+150}(k)\cdot\overline{Y_{+150}(k)}\right]$$

$$(x\times y)(m) = \begin{cases} (x_{+150}\otimes y_{+150})(m) & \text{für } 0\le m\le 150 \\ (x_{+150}\otimes y_{+150})(N_{+150}-|m|) & \text{für } -150\le m<0 \end{cases}$$

Padding mit 1023 / 1024 Nullen

Für die zweite oben genannte Methode, bei der das auf 1024 Einträge begrenzte Zeilenbild mit 1023 Null-Samples ergänzt wird, gelten die gleichen Überlegungen, wie im letzten Abschnitt dargelegt.

Weil so allerdings eine FFT mit 2047 Einträgen entstünde, die sich nicht in kleine Primfaktoren zerlegen lässt ($2047 = 23\cdot 89$), wird die nächstgrößere Zweierpotenz $N = 2048$ gewählt. Dies ist auch das Vorgehen in dem in Listing 6.4 gezeigten Ausschnitt aus dem internen MATLAB-Code.

Benchmark

Die zwei zuvor vorgeschlagenen Methoden für die Fast Correlation mit jeweils 150 bzw. 1024 angehängten Null-Samples werden in ihrer Laufzeit getestet.

Aus der Tabelle 6.1 werden dafür passende FFT-Längen ausgewählt: Für die Variante mit 150 angehängten Null-Samples wird die FFT mit $N = 1200$ gewählt, da dies die nächst größere Länge nach $1024 + 150$ ist. Außerdem wird eine Messung für die nächst kleinere FFT-Länge $N = 1152$ durchgeführt, woraus sich ein Padding mit $1152 - 1024 = 128$ Null-Samples ergibt. Für die Variante mit 1024 angehängten Null-Samples wird die 2048-Punkte-FFT gewählt.

Bereich für m	Kerne	Dauer diskret /		Dauer Fast Corr. /	
		µs	T_s	µs	T_s
$-1023 \leq m \leq 1023$	1	1929	131	195	13
$-176 \leq m \leq 176$	1	597	41	108	7
$-128 \leq m \leq 128$	1	446	30	102	7

Tabelle 6.3: Benchmarks für Kreuzkorrelationen im Original- und Frequenzbereich

Die Tabelle 6.3 zeigt, dass in den betrachteten Fällen die Fast Correlation schneller zu einem Ergebnis gelangt als die zuvor vorgestellte Originalbereichsmethode. Die Ergebnisse für $|m| \leq 176$ und $|m| \leq 128$ sind dabei von besonderem Interesse, weil sie den in Abschnitt 4.2.3 bestimmten Bereich der zuverlässigen globalen Maxima vollständig bzw. fast vollständig abdecken.

Gleichzeitig wird an dieser Stelle im Zusammenhang mit dem Qualitätsmerkmal die gleiche Überlegung angestellt, wie zuvor für die Berechnung im Originalbereich: Selbst wenn für das Qualitätsmerkmal pro Weginkrement nur zehn Kreuzkorrelationsfunktionen berechnet werden, sind dafür nach Tabelle 6.3 etwa $10 \cdot 7 \cdot T_s$ an Rechenzeit notwendig; eine Zeit, in der eine Oberfläche mit einer Geschwindigkeit von $5\,\mathrm{m\,s^{-1}}$ sich um etwa 150 Pixel über die Kamerazeile verschoben hat.

Durch eine Parallelisierung von mehreren gleichzeitigen Kreuzkorrelationen im Frequenzbereich könnte die benötigte Zeit weiter reduziert werden. Aus diesem Grund ist die in diesem Abschnitt vorgestellte Methode die bislang beste.

6.4 Reduced Information Preselection

Bereits in Abschnitt 4.2.3 war die Möglichkeit der Reduktion des Rechenaufwands durch eine Vorselektion in Betracht gezogen worden, wie sie zum Beispiel die Zeitkorrelationsalgorithmen in Kapitel 3 verwenden. Allerdings stehen keine anderen Algorithmen zur Verfügung, die im benötigten Zeitraster ausreichend genaue Abschätzungen von Geschwindigkeit oder Wegfortschritt liefern können, sodass diese Möglichkeit zunächst außen vor gelassen wurde.

Die Ergebnisse der Benchmarks der Kreuzkorrelationen im Original- und insbesondere im Frequenzbereich werden so interpretiert, dass sie zwar die Implementierung der in Kapitel 4 vorgeschlagenen Algorithmen nicht völlig ausschließen, dafür aber eine effiziente Parallelisierung notwendig wäre und die Hardware allein durch die Kreuzkorrelation nah an die Grenze der Leistungsfähigkeit gebracht werden würde, wobei die vorgeschlagene Subpixel-Interpolation noch nicht berücksichtigt wurde.

In den vorausgehenden Abschnitten wurde versucht, die benötigten Rechnungen so weit zu vereinfachen, dass sie in Echtzeit auf dem Arm-Prozessor ausführbar sind. Dabei wurde die Möglichkeit einer Hardwarebeschleunigung in der programmierbaren Logik, die das Prozessorsystem entlasten und den Vorgang beschleunigen könnte, noch nicht in Betracht gezogen. Für diesen Ansatz werden zunächst folgende Punkte abgewägt:

- Der Arm-Prozessor erlaubt Taktraten bis zu 1,5 GHz und je Kern bis zu vier SP-Fließkommaoperationen pro Taktzyklus (SIMD, s. Abschnitt 6.1.1) [34, S. 12]. Die theoretische Obergrenze für das Taktverteilungsnetzwerk in der programmierbaren Logik liegt bei etwa 700 MHz [55, S. 66], wobei sich in Experimenten gezeigt hat, dass ab Taktraten von etwa 500 MHz je nach Schaltungskomplexität Implementierungsprobleme auftreten.
- Die programmierbare Hardware verfügt nicht über dedizierte Fließkomma-Blöcke, diese müssten aus Standardlogik aufgebaut werden.
- Der Arm-Prozessor kann den interessierenden Ausschnitt der Kreuzkorrelationsfunktion zwischen zwei Zeilen in etwa einer Frameperiode berechnen, sofern der zu betrachtende Bereich vorher durch eine Präselektion abgeschätzt wurde (s. Tabelle 6.2, $50 \leq m < 60$).

Insbesondere der letzte Punkt wird dabei als Argument gegen eine vollständige Fließkomma-Implementierung in Hardware gewertet: Alleine die Bitbreiten von 32 bit der SP-Zahlen in Hardware zu behandeln erzeugt einen Aufwand, der im Vergleich zur benötigten Genauigkeit für eine Vorselektion als unangemessen betrachtet wird.

Diese Überlegung führt zu einem Konzept, das hier als Reduced Information Preselection (RIPSL) bezeichnet wird und bedeutet, dass die für die schnelle Eingrenzung des interessierenden Bereichs der Kreuzkorrelationsfunktion notwendigen Berechnungen auf einer möglichst kleinen Untermenge der zur Verfügung stehenden Information durchgeführt wird.

Ziel ist es dabei ausdrücklich nicht, die Fließkommaberechnung auf dem Arm-Kern zu ersetzen, sondern stattdessen die programmierbare Logik zu verwenden, um den Suchbereich so weit einzugrenzen, dass die Fließkommaberechnung im Prozessorsystem über das bei der Fast Correlation erreichte Maß hinaus beschleunigt wird.

6.4.1 Scaled Integer Preselection

Die Scaled Integer Preselection-Methode (SIPS) ist der erste vorgeschlagene RIPSL-Ansatz.

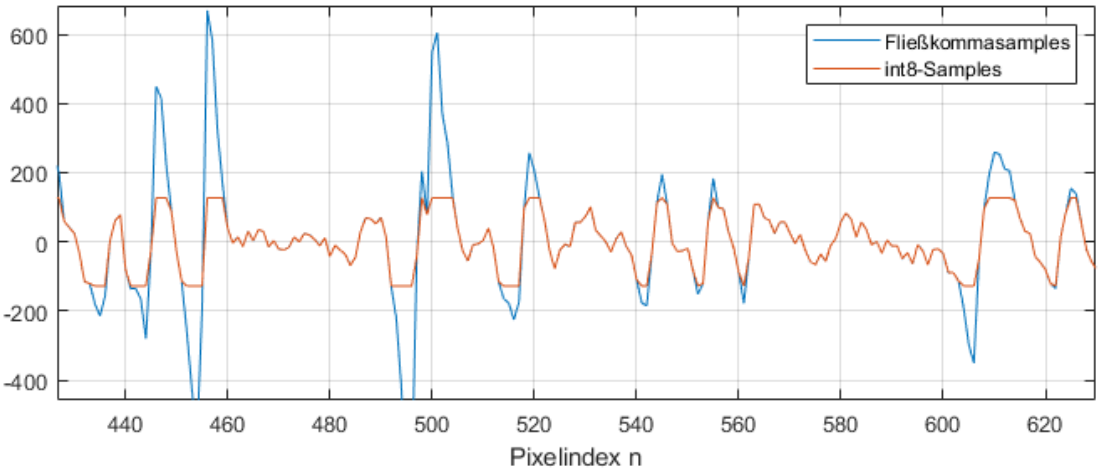

Abbildung 6.7: Vergleich eines Zeilenbilds aus Fließkommasamples und 8-Bit-Integern

Abbildung 6.7 zeigt den Auschnitt eines gefilterten Zeilenbildes in SP-Fließkommasamples und skaliert in eine 8-Bit-Ganzzahl:

Zwar ist erkenntlich, dass die 8-Bit-Werte das Signal nicht exakt wiedergeben, allerdings sind trotzdem noch Teile der Bildmerkmale zu erkennen. Daher wird erwartet, dass die Berechnung einer Kreuzkorrelation zwischen solchen ganzzahligen 8-Bit-Werten eine ausreichende Präselektion für die weitere Berechnung im Prozessorsystem liefern kann. Außerdem verfügt die programmierbare Logik über mehrere Hundert Multiplizierer [34, S. 6], die theoretisch parallel eine solche Berechnung ausführen könnten.

Als entscheidender Nachteil des SIPS-Ansatzes ist es aber zunächst notwendig, alle relevanten Samples von 32-Bit-SP-Fließkommazahlen in 8-Bit-Ganzzahlen umzuwandeln. Dies müsste entweder auf dem Arm-Prozessor geschehen, was dem ursprünglichen Zweck der Entlastung des Prozessors entgegenspräche, oder in Hardware implementiert werden, wodurch wiederum die Behandlung von Fließkommazahlen in der Logik notwendig würde. Aus diesem Grund wird SIPS zugunsten der in den nächsten Abschnitten eingeführten Konzepte verworfen.

6.4.2 Sign-Reduced Information Preselection

Modellbetrachtung

Abbildung 6.8 zeigt die Impulsantwort $g_{\mathrm{f}}(n)$ des FIR-Rohbildfilters, das in diesem Projekt aus den Vorgängerprojekten VADER und COVIDIS übernommen wurde.

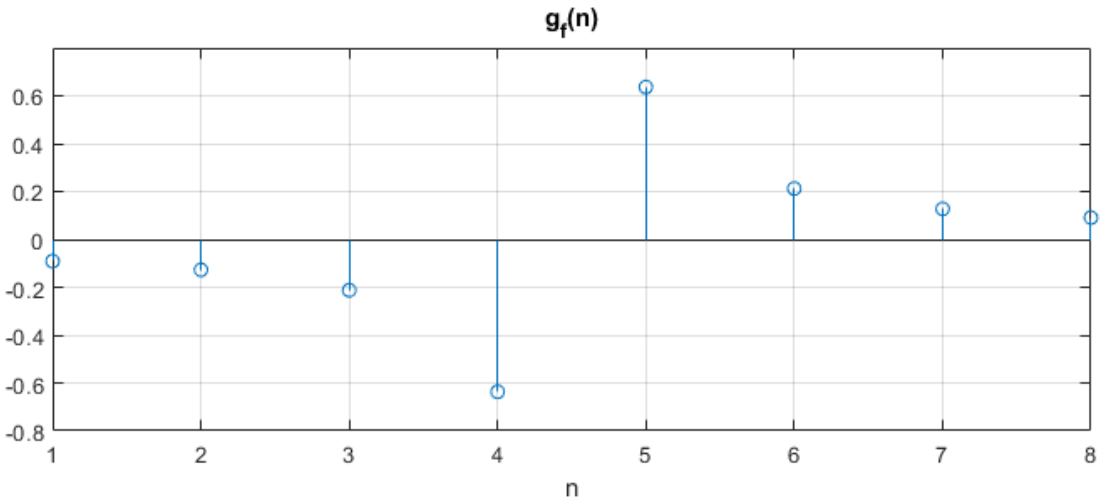

Abbildung 6.8: Impulsantwort des FIR-Rohbildfilters aus den COVIDIS- und VADER-Projekten

Weil das FIR-Filter ein LTI-System ist [56, S. 371], auf das das Superpositionsprinzip Anwendung findet [22, S. 12], können die Samples der Rohbilder beim Durchlauf des Filters getrennt betrachtet und die Teil-Systemantworten anschließend überlagert werden. Weil es sich bei den einzelnen Rohbildsamples unter diesem Gesichtspunkt lediglich um gewichtete Dirac-Impulse handelt, zeigt die Impulsantwort in Abbildung 6.8 abgesehen von einer linearen Skalierung die Teilantwort des Filters auf jedes Pixel.

Die Helligkeitswerte der einzelnen Pixel, wie sie von der Kamera ausgegeben werden, sind vorzeichenlose 12-Bit-Ganzzahlen und damit immer positiv oder Null und erzeugen so für sich betrachtet jeweils einen Ausschlag der Systemantwort nach unten und anschließend nach oben. Dabei sind die Flächen ober- und unterhalb der Nulllinie identisch, wodurch der eigentliche Zweck des Filters, der verschwindende DC-Anteil in der Systemantwort, erreicht wird.

Die Systemantwort auf einen Impuls mit einem Gewicht ungleich 0 beinhaltet damit immer auch einen Vorzeichenwechsel, sobald der Impuls die Mitte des Filters erreicht. Diese Tatsache wird für die Sign-Reduced Information Preselection (SRIPSL) ausgenutzt.

Die Signum-Funktion ist in der Mathematik für reelle Zahlen folgendermaßen definiert [22, S. 89]:

$$\operatorname{sgn}(x) = \begin{cases} +1 & \text{für x} > 0 \\ 0 & \text{für x} = 0 \\ -1 & \text{für x} < 0 \end{cases} \tag{6.12}$$

Wendet man die Signumfunktion auf ein gefiltertes Zeilenbild an, entsteht ein Muster, das von den Vorzeichenübergängen der Systemantwort des FIR-Filters abhängig ist. Abbildung 6.9 zeigt einen Ausschnitt eines gefilterten Zeilenbilds und die auf die einzelnen Samples angewendete Signumfunktion.

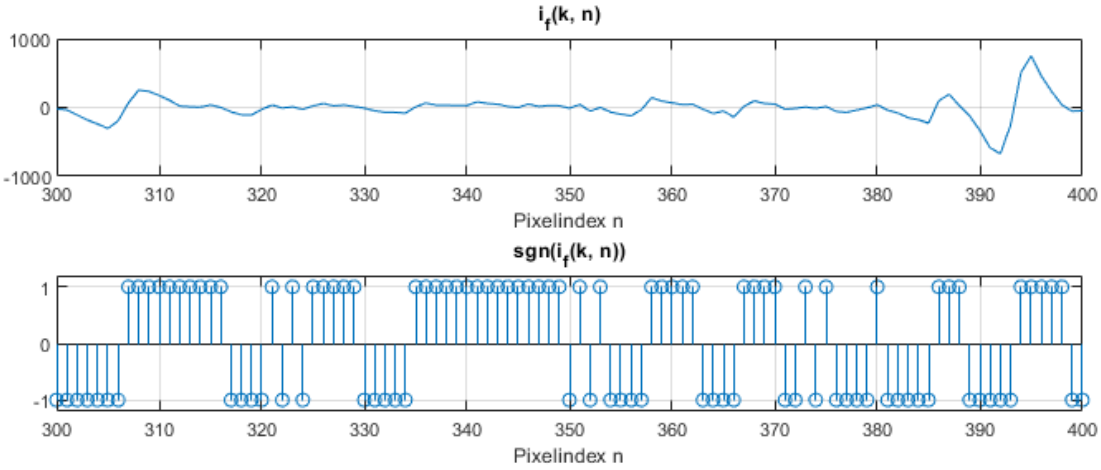

Abbildung 6.9: Ausschnitt eines gefilterten Zeilenbilds und der darauf angewendeten Signum-Funktion

Abbildung 6.10 zeigt Ausschnitte der Kreuzkorrelationsfunktionen zwischen $i_\mathrm{f}(0, n)$ und $i_\mathrm{f}(50, n)$, zwei um 50 Frameperioden auseinanderliegenden, gefilterten Zeilenbildern, sowie zwischen $\operatorname{sgn}(i_\mathrm{f}(0, n))$ und $\operatorname{sgn}(i_\mathrm{f}(50, n))$.

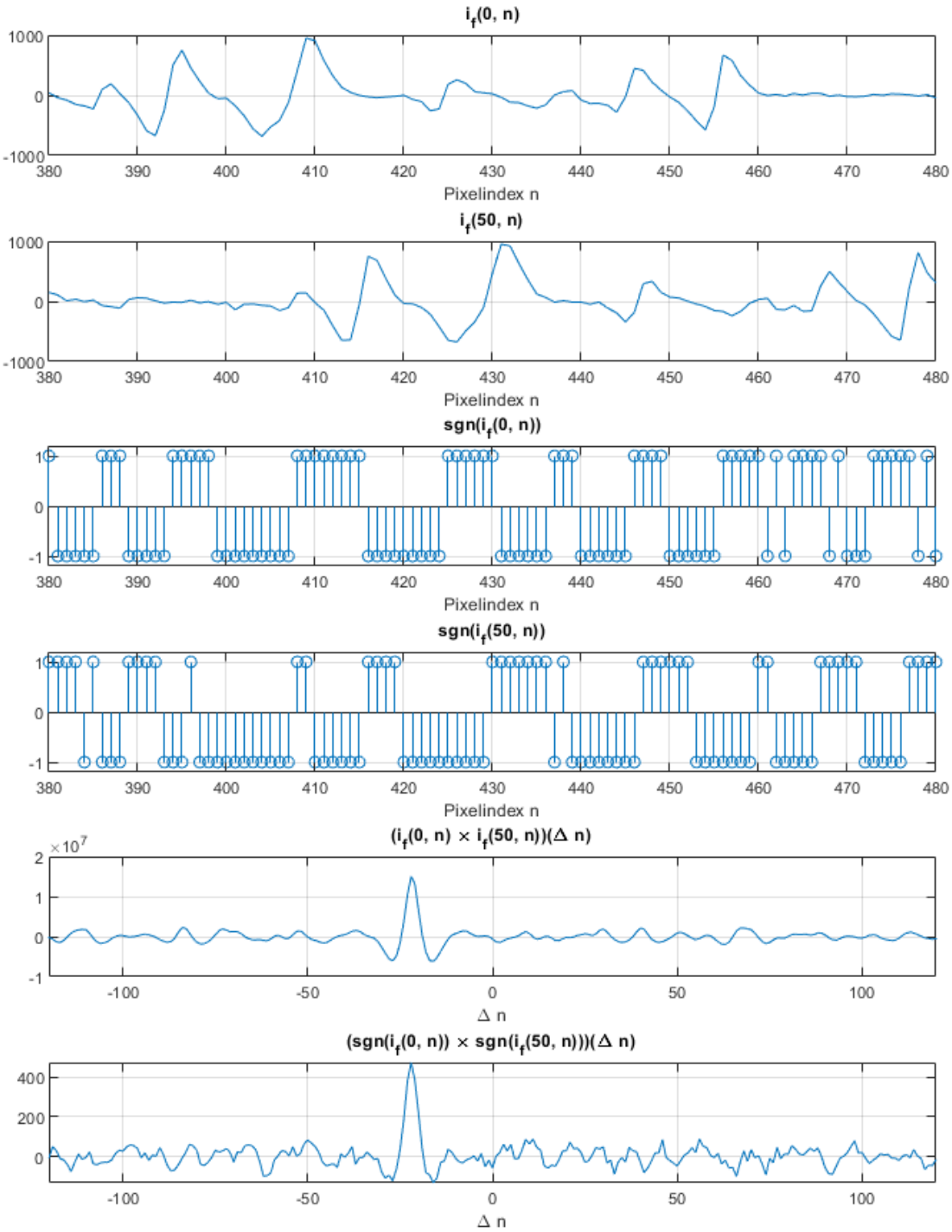

Abbildung 6.10: Normale Kreuzkorrelation im Vergleich mit SRIPSL-Kreuzkorrelation

Es fällt auf, dass der die Verschiebung Δn zwischen $i_\mathrm{f}(0,n)$ und $i_\mathrm{f}(50,n)$ anzeigende Peak in beiden Kreuzkorrelationsfunktionen zumindest näherungsweise an der gleichen Stelle sichtbar ist.

Abbildung 6.11 zeigt die vollständigen Kreuzkorrelationsfunktionen zwischen $i_\mathrm{f}(0,n)$ und $i_\mathrm{f}(50,n)$ sowie zwischen $\mathrm{sgn}(i_\mathrm{f}(0,n))$ und $\mathrm{sgn}(i_\mathrm{f}(50,n))$.

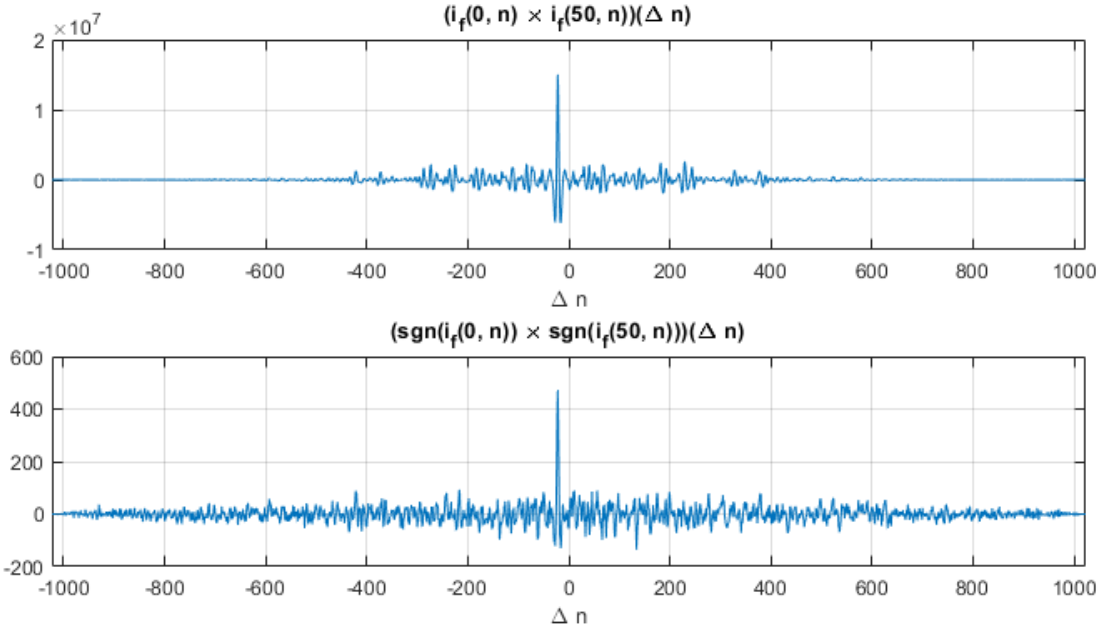

Abbildung 6.11: Kreuzkorrelationsfunktionen mit SRIPSL

Zunächst ist festzustellen, dass das Maximum der Signum-reduzierten KKF das globale Maximum der gewöhnlichen KKF erfolgreich eingrenzt. Bei der Betrachtung der zwei Kreuzkorrelationsfunktionen für $-1023 \leq \Delta n \leq 1023$ zeigen sich ähnliche Verläufe mit dem Unterschied, dass die Signum-reduzierte KKF vergleichsweise mehr Aktivität bei den betragsmäßig größeren Verschiebungen zeigt.

Dies ist darauf zurückzuführen, dass die Signum-Funktion nicht zwischen den kleinen Helligkeitsschwankungen an den Rändern des Bildes und den interessierenden, betragsmäßig größeren Helligkeitsschwankungen in der Mitte des Bildes unterscheidet: Jeder noch so kleine negative Ausschlag im gefilterten Bild wird durch die Signum-Funktion mit -1 bewertet und jeder noch so kleine positive Ausschlag mit $+1$.

Nach der Einzelfallbetrachtung in den Abbildungen 6.10 und 6.11 folgt eine statistische Betrachtung des SRIPSL-Ansatzes mit dem Ziel, festzustellen, wie zuverlässig der Peak der SRIPSL-Korrelation mit dem Peak der normalen Korrelation übereinstimmt oder diesen zumindest in einem gewissen Rahmen eingrenzt. Die Betrachtung basiert auf der Auswertung der Echtbilder aus Abschnitt 3.2.2.

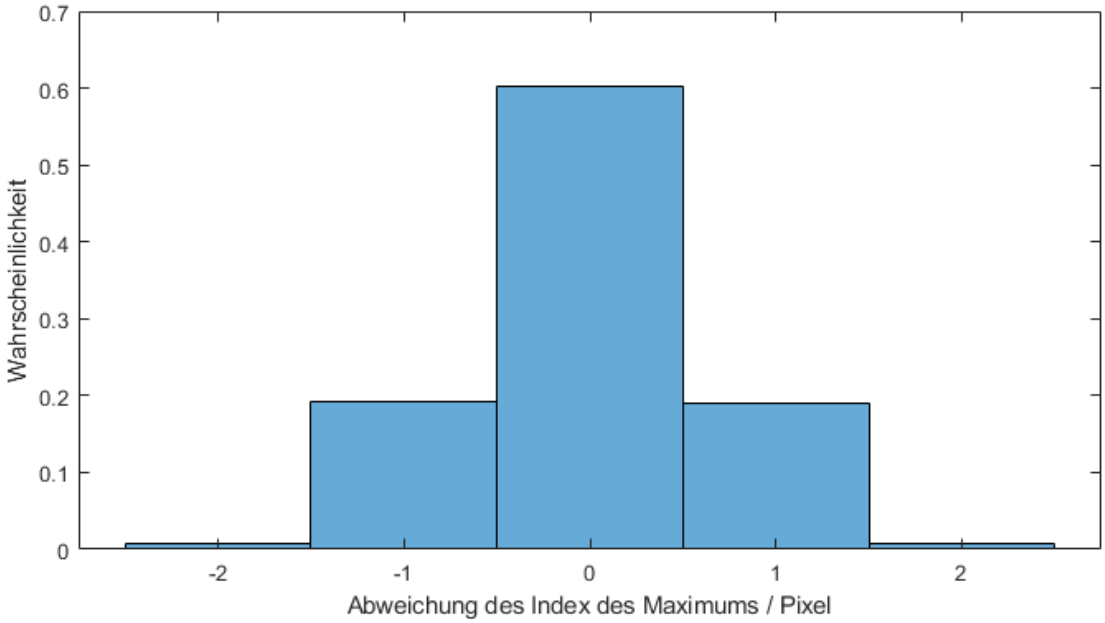

Abbildung 6.12: Histogramm der Abweichungen der Indizes der Maxima der SRIPSL-Kreuzkorrelation von der nicht reduzierten Kreuzkorrelation

Es konnte kein Fall identifiziert werden, in dem das Maximum der Signum-reduzierten KKF um mehr als zwei Pixel gegenüber dem Maximum der gewöhnlichen KKF verschoben war. Dies wird unter dem Gesichtspunkt, dass nach Tabelle 6.2 bereits die Eingrenzung des Suchbereichs auf zehn Einträge zur Berechenbarkeit der KKF binnen etwa einer Frameperiode führt, als hinreichende Präselektion betrachtet.

Hardware

Berechnet man die Kreuzkorrelation statt zwischen zwei gefilterten Zeilenbildern $i_\mathrm{f}(k_1, n)$ und $i_\mathrm{f}(k_2, n)$ zwischen $\mathrm{sgn}(i_\mathrm{f}(k_1, n))$ und $\mathrm{sgn}(i_\mathrm{f}(k_2, n))$, ergibt sich aus

Sicht der Berechnungskomplexität ein Vorteil: Da $\mathrm{sgn}(x)$ nur drei Werte annehmen kann, lässt sich die Multiplikation $\mathrm{sgn}(x) \cdot \mathrm{sgn}(y)$ vollständig in einer 3x3-Tabelle darstellen:

$\mathrm{sgn}\big(x(n+m)\big)$ \ $\mathrm{sgn}\big(y(n)\big)$	$(11)_2 = -1$	$(00)_2 = 0$	$(01)_2 = 1$
$(11)_2 = -1$	$(01)_2 = 1$	$(00)_2 = 0$	$(11)_2 = -1$
$(00)_2 = 0$	$(00)_2 = 0$	$(00)_2 = 0$	$(00)_2 = 0$
$(01)_2 = 1$	$(11)_2 = -1$	$(00)_2 = 0$	$(01)_2 = 1$

Tabelle 6.4: Berechnungstabelle für die Multiplikation von zwei Signum-Funktionen

In Tabelle 6.4 wird zu den Dezimalwerten eine Binärdarstellung im Zweierkomplement der Ein- und Ausgänge gezeigt, um zu verdeutlichen, dass eine solche Tabelle in Hardware durch eine sog. Lookup-Tabelle (LUT) mit zwei 2-Bit-Eingängen und einem 2-Bit-Ausgang realisiert werden könnte, anstatt einen Multiplizierer zu nutzen.

Gegenüber der in Abschnitt 6.4.1 vorgestellten Methode ergibt sich darüber hinaus der Vorteil, dass für die Berechnung der Signum-Funktion pro Pixel nur drei Fälle geprüft werden müssen, anstatt jeweils eine Konversion von Fließkomma zu Ganzzahl durchführen zu müssen. Die Prüfung dieser drei Fälle könnte unter vergleichsweise geringem Aufwand in der Hardware durchgeführt werden, weil dafür keine echte Fließkommaarithmetik notwendig ist.

In einer an eine Hardwareimplementierung angelehnten, vereinfachten Notation lässt sich für die in Tabelle 6.4 dargestellte Operation schreiben:

$$\mathrm{sgn}(x(n+m)) \cdot \mathrm{sgn}(y(n)) = \mathrm{LUT4x2}\Big(\mathrm{sgn}(x(n+m)), \mathrm{sgn}(y(n))\Big) \tag{6.13}$$

Die in der programmierbaren Logik real existierende LUT4-Hardwarestruktur ist eine Lookup-Tabelle mit vier 1-Bit-Eingängen, die als zwei 2-Bit-Eingänge verwendet werden können [57, S. 375].

Es ergeben sich somit $2^4 = 16$ mögliche Eingangskombinationen und damit auch 16 mögliche Ausgangswerte. Vergleicht man dies mit der Tabelle 6.4, so gibt es

für das Produkt $\mathrm{sgn}(x) \cdot \mathrm{sgn}(y)$ nur neun Fälle, da bei den 2-Bit-Eingängen der Wert $(10)_2 = -2$ ungenutzt bleibt weil er nicht zur Ergebnismenge der Signum-Funktion gehört. Schon aus diesem Grund wäre eine LUT4-Struktur nicht optimal ausgenutzt.

Weiterhin wird unter realistischer Betrachtung ein gefiltertes Zeilenbild in Fließkommadarstellung eine vernachlässigbare Anzahl an Einträgen mit dem Wert Null haben. Um am Ausgang des Rohbildfilters ein Null-Sample zu erzeugen, müssten entweder alle Eingangssamples im Schieberegister des Filters den Wert Null haben, was schon allein wegen des Grundrauschens der Kamera unwahrscheinlich ist, oder alle Eingangssamples müssten den gleichen Wert haben, was ebenfalls wegen des Rauschens der Kamera unwahrscheinlich ist.

Diese Beobachtung führt zu der Annahme, dass eine weitere Reduktion der Information auf zwei Fälle (positives Sample oder negatives Sample) möglich ist. Dieser Ansatz wird im folgenden Abschnitt verfolgt.

6.4.3 Extremely Reduced Information Preselection

Modellbetrachtung

Die Ausführungen zu SRIPSL im letzten Abschnitt haben gezeigt, dass ein auf 2 bit pro Pixel reduziertes Zeilenbild noch ausreichend Information enthält, um Ähnlichkeiten (Korrelationen) zwischen zwei Bildern ausfindig zu machen.

Die zwei Fälle unter den Signum-Samples, die eine realistische Relevanz besitzen, lassen sich in einem Bit repräsentieren, das zwei Werte annehmen kann und somit entweder auf ein positives oder negatives Sample hindeutet.

Ein ähnliches Bit steht bei Fließkommazahlen nach IEEE-754 bereits in Form des Sign-Bits b_s zur Verfügung. Es gilt $b_s = 1$, wenn die Fließkommazahl negativ oder null ist und $b_s = 0$, wenn sie positiv oder null ist. Es existieren also zwei Darstellung für die Zahl Null. [28]

Im Folgenden wird eine Notation verwendet, bei der für Ausdrücke der Aussagenlogik dem Ergebnis „wahr“ der Wert Eins und dem Ergebnis „falsch“ der Wert Null zugeordnet wird.

Ein Beispiel:

$$(a < 0) = \begin{cases} 1 & \text{für } a < 0 \\ 0 & \text{sonst} \end{cases} \tag{6.14}$$

Durch diese Festlegung lassen sich logische Ausdrücke grafisch darstellen.

Die Extremely Reduced Information Preselection-Methode (XRIPSL) basiert auf SRIPSL, ersetzt aber die Signum-Funktion $\mathrm{sgn}(x)$ durch den logischen Ausdruck $(x < 0)$. Ist x eine Fließkommazahl, dann entspricht der Ausdruck $(x < 0)$ dem Sign-Bit, sofern gilt $x \neq 0$.

Abbildung 6.13 zeigt wie Abbildung 6.10 einen Ausschnitt des Resultats einer informationsreduzierten Kreuzkorrelation im Vergleich zur ordinären Kreuzkorrelation, allerdings unter dem Einsatz der beschriebenen XRIPSL- statt SRIPSL-Methode.

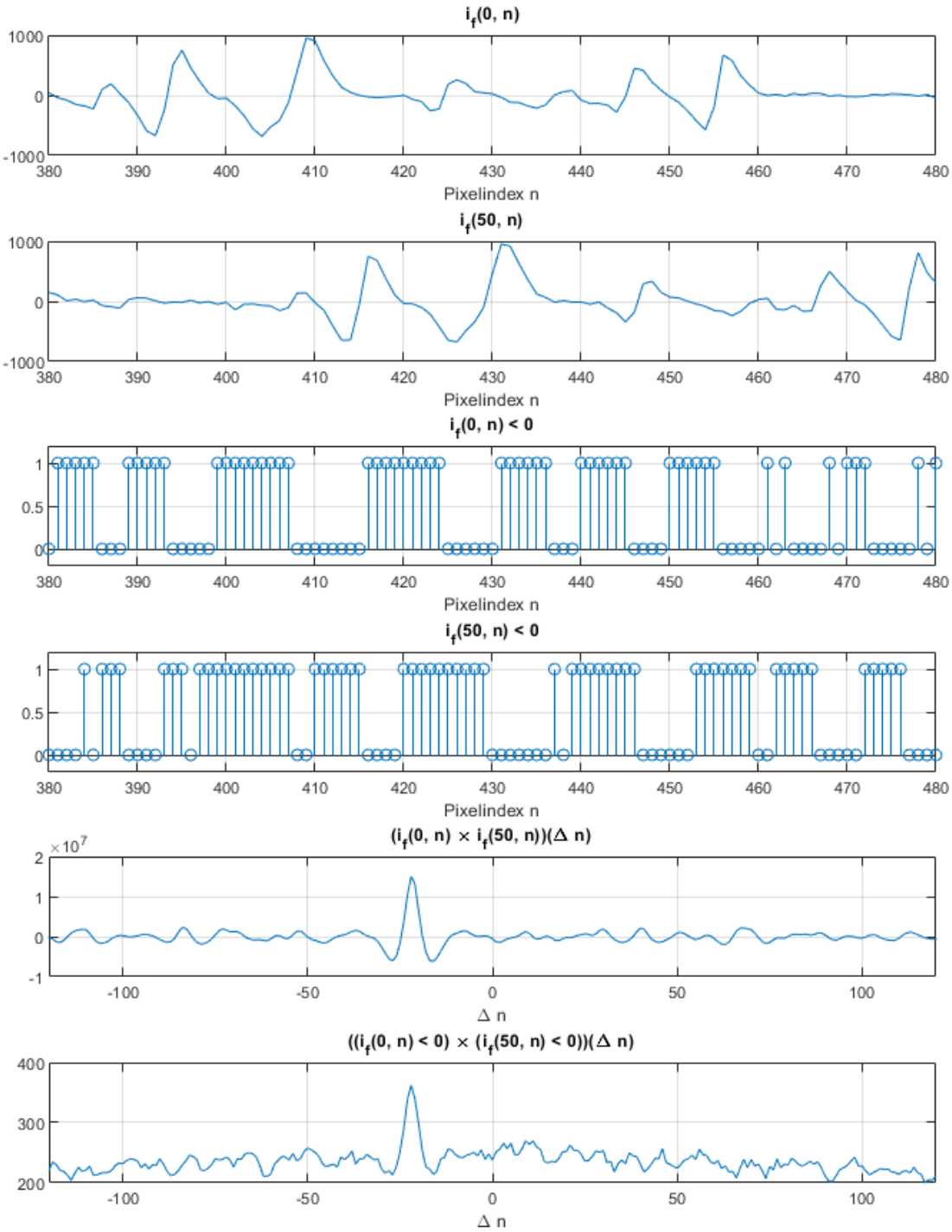

Abbildung 6.13: Normale Kreuzkorrelation im Vergleich zu XRIPSL-Kreuzkorrelation

Die gesamte XRIPSL-Kreuzkorrelationsfunktion im Vergleich zur diskreten Variante ist in Abbildung 6.14 dargestellt.

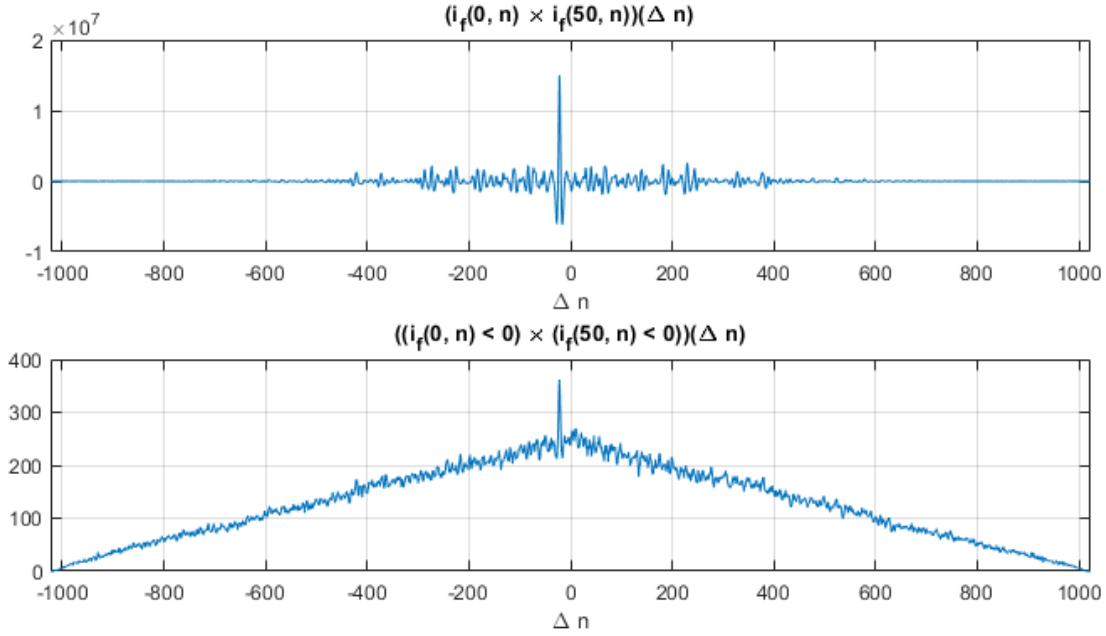

Abbildung 6.14: XRIPSL-Kreuzkorrelationsfunktion

Auffällig ist, dass die XRIPSL-KKF zu betragsmäßig größeren Verschiebungen abfällt, insgesamt also eine Art Pyramidenform hat. Dieses Phänomen war schon in Abschnitt 4.2.3 bei der Kreuzkorrelation zwischen ungefilterten Zeilenbildern beobachtet worden und ist auf den DC-Anteil der Operanden zurückzuführen.

Weil das Sign-Bit (im Gegensatz zur Signum-Funktion) immer nur die Werte Null oder Eins annehmen kann, haben die Eingangsfunktionen bei XRIPSL immer einen DC-Anteil ≥ 0, was kleine Verschiebungen systematisch bevorzugt.

Abbildung 6.15 zeigt analog zu Abbildung 6.12 eine statistische Betrachtung für die Verschiebung des Maximums einer XRIPSL-Korrelation gegenüber der ordinären diskreten Korrelation.

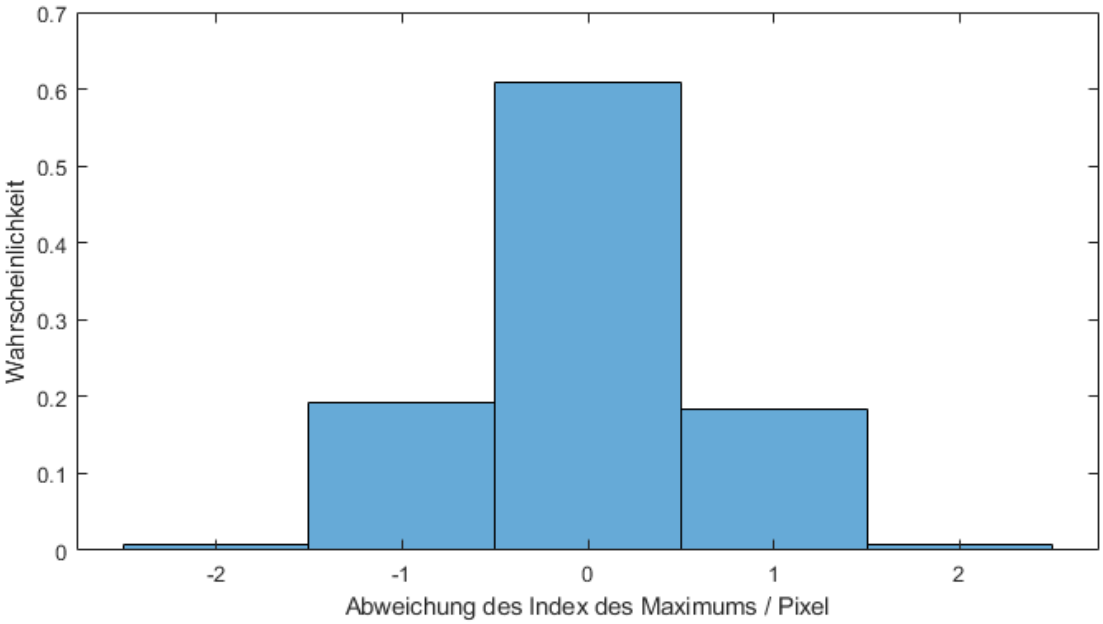

Abbildung 6.15: Histogramm der Abweichungen der Indizes der Maxima der XRIPSL-Kreuzkorrelation von der nicht reduzierten Kreuzkorrelation

Die Auswertung des Histogramms führt zum gleichen Ergebnis wie zuvor bei SRIPSL: Eine Eingrenzung der Position des Maximums auf ± 2 Pixel wird als hinreichend betrachtet.

Bezüglich der Pyramidenform der XRIPSL-KKF, die bei SRIPSL nicht auftritt, wird untersucht, inwieweit dies die Erkennung der betragsmäßig größeren Verschiebungen beeinflusst.

Zuletzt war schon bei der nicht optimierten Kreuzkorrelation in Abschnitt 4.2.3 festgestellt worden, dass Verschiebungen ab etwa 150 Pixeln nicht mehr zuverlässig als globales Maxmimum erkannt werden können. Abbildung 6.16 erweitert den Graph aus Abbildung 4.6 um Kurven für SRIPSL und XRIPSL, um diesen Aspekt vergleichend bewerten zu können.

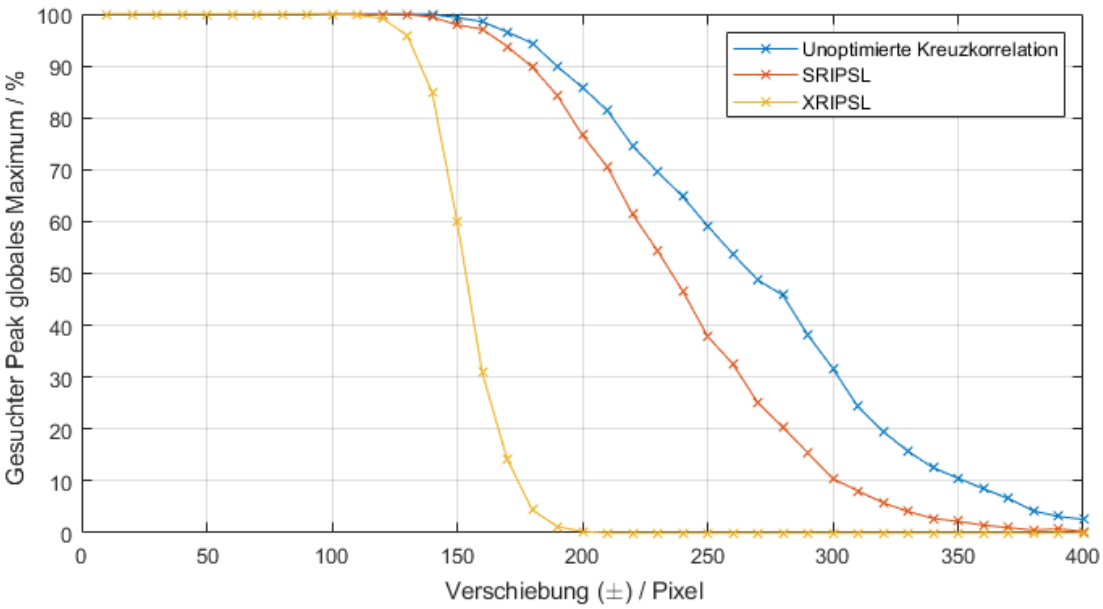

Abbildung 6.16: Wahrscheinlichkeiten für die Ermittelung der korrekten Verschiebung durch das globale Maximum bei SRIPSL und XRIPSL im Vergleich zur nicht reduzierten Kreuzkorrelation

Die Kurven in Abbildung 6.16 zeigen, dass die Wahrscheinlichkeiten, das korrekte globale Maximum durch SRIPSL und XRIPSL mit größeren Verschiebungen schneller abnehmen als bei der unoptimierten Kreuzkorrelation. Dies ist einerseits dadurch zu erklären, dass beiden Optimierungsvarianten nur ein Bruchteil der Information zur Verfügung steht und bei SRIPSL und XRIPSL die einzelnen Oberflächenmerkmale alle betragsmäßig identisch auftauchen $(-1, 0, 1)$ bzw. $(0, 1)$, während die ordinäre Kreuzkorrelation zusätzlich die Form der einzelnen Peaks in Betracht zieht.

Darüber hinaus fällt die Erkennungswahrscheinlichkeit bei XRIPSL steiler ab als bei den anderen Methoden, was auf die Pyramidenform der XRIPSL-KKF zurückzuführen ist, die weiter außen liegende Maxima systematisch benachteiligt.

Weil eine Aussage $(z < 0)$ nur zwei mögliche Ergebnisse haben kann, schrumpft die Tabelle 6.5 gegenüber der 3x3-Tabelle 6.4 von SRIPSL auf eine 2x2-Tabelle.

$x(n+m) < 0$ \ $y(n) < 0$	0	1
0	0	0
1	0	1

Tabelle 6.5: Berechnungstabelle für eine XRIPSL-Multiplikation

Eine solche Tabelle lässt sich aus Hardwaresicht in einer Lookup-Tabelle mit zwei Eingängen darstellen. Gleichzeitig fällt auf, dass diese Lookup-Tabelle eine binäre AND-Verknüpfung darstellt. Dies passt zu der Tatsache, dass eine AND-Verknüpfung im Feld der digitalen Logik auch durch den Multiplikationsoperator dargestellt wird [58, S. 89].

Die Berechnungsvorschrift für eine XRIPSL-Korrelation lässt sich damit mit einer booleschen Konjunktion schreiben:

$$\texttt{xripsl}(x, y, m) = \begin{cases} \displaystyle\sum_{n=0}^{N-|m|-1} \big[x(n+|m|) < 0\big] \wedge \big[y(n) < 0\big] & \text{für } m \geq 0 \\ \displaystyle\sum_{n=0}^{N-|m|-1} \big[x(n) < 0\big] \wedge \big[y(n+|m|) < 0\big] & \text{für } m < 0 \end{cases} \tag{6.15}$$

Als logische Ausdrücke können die Summanden nur noch die Werte Null (falsch) oder Eins (wahr) annehmen, was die Berechnung der Summe in Hardware erleichtert.

Wie bereits erwähnt müssen die Aussagen der Form $(z < 0)$ für Fließkommazahlen nicht evaluiert werden, weil das Ergebnis in Form des Sign-Bits der jeweiligen Fließkommazahl (näherungsweise, weil $z = 0$ vernachlässigt) bereits vorliegt.

XRIPSL-Implementierung in Hardware

Für die Implementierung von XRIPSL in Hardware wurden im letzten Abschnitt entscheidende Vereinfachungen hergeleitet:

- Multiplikationen tauchen nicht mehr auf und können vollständig durch Lookup-Tabellen bzw. AND-Gatter ersetzt werden.
- Eine Auswertung von Fließkommazahlen ist nicht notwendig, da die benötigte Information bereits im Sign-Bit kodiert ist.

Dies führt zu einer Struktur, wie sie in Abbildung 6.17 vereinfacht dargestellt ist.

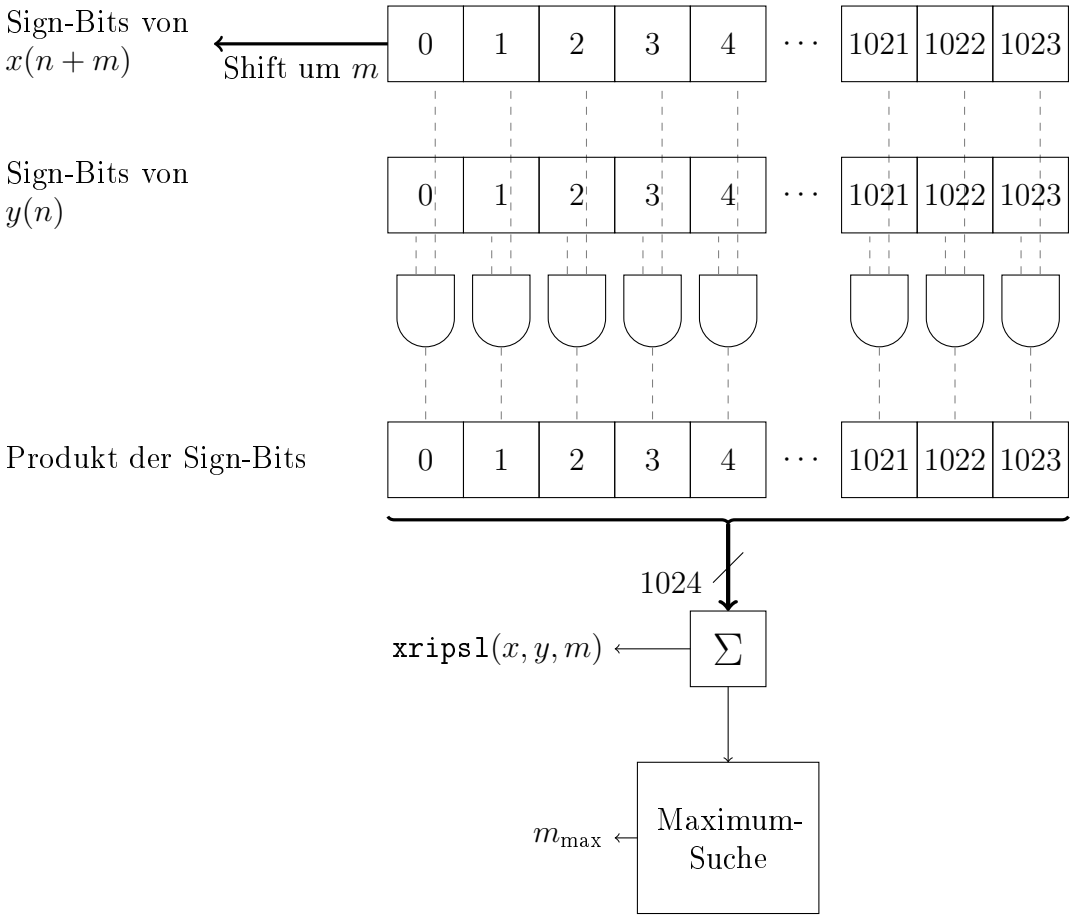

Abbildung 6.17: Parallele Multipliziererstruktur des XRIPSL-Beschleunigers

Neben dem in der Abbildung dargestellten Berechnungskern besteht das XRIPSL-Modul außerdem aus einem nicht eingezeichneten Kontrollmodul, über das der

Beschleuniger durch des Prozessorsystems angesprochen und kontrolliert werden kann; der XRIPSL-Beschleuniger führt nur auf Aufforderung seitens des Prozessorsystems Korrelationen durch und ruht ansonsten (s. Abschnitt 6.4.3).

Der Kern des XRIPSL-Beschleunigers besteht aus zwei 1024-Bit-Registern, die die Sign-Bits von $x(n)$ und $y(n)$ beinhalten. Eines der Register ist ein Schieberegister.

Im ersten Teil der Berechnung wird das Schieberegister mit jeder Taktflanke nach rechts verschoben, im zweiten Teil der Berechnung nach links. Auf diese Weise werden die Ergebnisse sowohl für positive als auch für negative m berechnet.

Das Schieberegister wird mit 400 MHz getaktet, sodass die Berechnung für $-1023 \leq m \leq 1023$ insgesamt $\frac{2048}{400\,\mathrm{MHz}} \approx 5\,\mu\mathrm{s}$ dauert.

Aus Sicht der AND-Verknüpfungen stellt sich dies als unproblematisch heraus, da diese eine Durchlaufzeit kleiner $\frac{1}{400\,\mathrm{MHz}}$ haben, sodass das Produkt am Ausgang des AND-Gates bereits vorliegt, bevor das Schieberegister weiter verschoben wird.

Etwas anderes gilt für die Summenbildung, die in Abbildung 6.17 durch Σ dargestellt ist. Im Gegensatz zum AND-Gate, das nur zwei Eingänge hat, hat das Summierungsmodul 1024 Eingänge und elf Ausgänge (die größtmögliche Summe ist 1024, die in 11 bit darstellbar ist).

Das Summierungsmodul, wie es in der Abbildung dargestellt ist, hat daher eine wesentlich längere Durchlaufzeit als $\frac{1}{400\,\mathrm{MHz}}$, was bedeutet, dass das Produktregister sich bereits wieder ändert, bevor die Summe der letzten Produkte überhaupt berechnet ist.

Für dieses Problem existieren zwei mögliche Lösungen, wobei T_Σ die Durchlaufzeit des Summierers sei:

1. Die Taktfrequenz wird auf $\frac{1}{T_\Sigma}$ verringert.
2. Die Taktfrequenz wird beibehalten und die Summenbildung in eine Pipeline umgeformt, in der jede Pipelinestufe eine Durchlaufzeit kleiner $\frac{1}{400\,\mathrm{MHz}}$ hat.

Der erste Lösungsansatz wird als für einen Hardwarebeschleuniger ungeeignet betrachtet, da er effektiv dazu führen würde, dass das gesamte XRIPS-Modul sich in der Taktung an das schwächste Glied, das Summierungsmodul, anpassen müsste. Der ursprüngliche Vorteil, über 1024 parallele AND-Gatter bei 400 MHz effektiv $4 \cdot 10^{11}$ Multiplikationen pro Sekunde durchführen zu können, ginge damit verloren.

Umgekehrt hat der zweite Ansatz abgesehen von einer höheren Schaltungskomplexität und einem leicht erhöhten Speicherbedarf keine relevanten Nachteile: Geht man von einer Pipeline mit maximal zehn Stufen aus, so ergibt sich eine Ausgangsverzögerung von $\frac{10}{400\,\mathrm{MHz}} = 25\,\mathrm{ns}$, was im Gesamtkontext der Berechnung als unerheblich betrachtet wird.

Der prinzipielle Aufbau der Summierer-Pipeline ist in Abbildung 6.18 gezeigt.

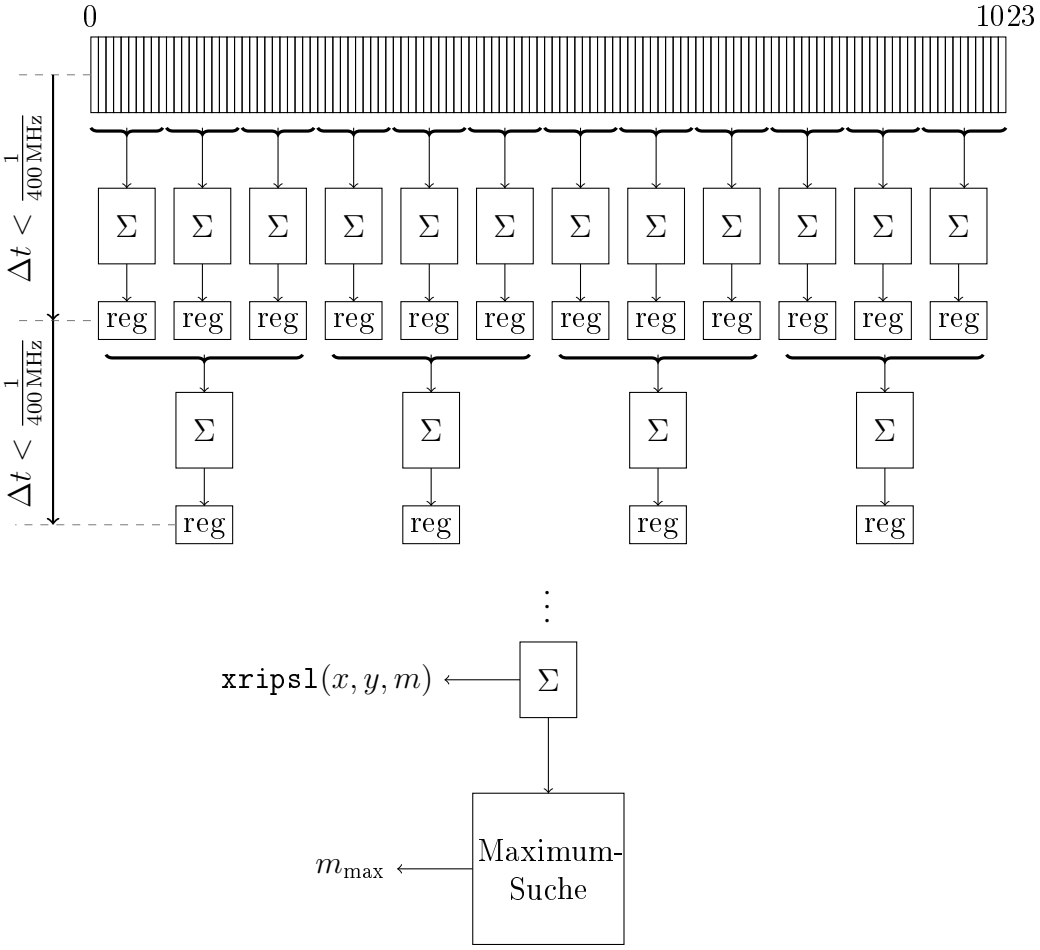

Abbildung 6.18: Summierer-Pipeline des XRIPSL-Beschleunigers

Die Summierer-Pipeline unterteilt die 1024 Summanden in Gruppen, die jeweils einzeln summiert werden und das Ergebnis in je einem Register zwischenspeichern. Dabei müssen die Gruppen so dimensioniert sein, dass das Ergebnis der letzten Summierung das nächstgelegene Speicherregister erreicht hat, bevor die

Eingangsdaten des Summierers sich durch die nächste Taktflanke bzw. das Weiterschieben des Schieberegisters ändern.

Bei der Implementierung der Pipeline wird iterativ vorgegangen: Es wird mit einer kleinen Anzahl an Stufen begonnen und so lange weitere Stufen hinzugefügt, bis die Timing-Bedingung ($\Delta t < 2{,}5\,\text{ns}$) zwischen allen Speicherregistern eingehalten werden kann.

Dem Summierer-Modul schließt sich in der Hardware direkt die Maximum-Suche an. Das Abspeichern des Ergebnisvektors `xripsl`(x, y, m) für alle m, um ihn anschließend zu durchsuchen ist nicht notwendig.

Die 2048 Takte, die ohnehin benötigt werden, um alle Produkte zu bilden, können gleichzeitig verwendet werden, um die Summen der Produkte am Ausgang mit der jeweils vorherigen Summe zu vergleichen und dabei nur m_{max}, den Index der größten Summe, zu behalten. Durch dieses Vorgehen kann die Maximum-Suche als in die Summierer-Pipeline integriert verstanden werden, weshalb sie keine zusätzlichen Taktzyklen bzw. Laufzeit kostet.

Optimierung der Schiebeoperation

In der zuletzt erörterten Allgemeinausführung von XRIPSL für 1024-Bit-Vektoren wird das Schieberegister um je 1023 Einträge nach links und rechts verschoben um somit die vollständige Kreuzkorrelationsfunktion (der Sign-Bits) zu bilden.

Aus Abbildung 6.15 ist ersichtlich, dass Verschiebungen außerhalb eines Bereichs von etwa ± 120 Pixeln bei der Verwendung von XRIPSL nicht oder kaum zum Auffinden des korrekten Maximums beitragen. Aus diesem Grund werden nur noch 120 statt 1023 Schiebeoperationen in jede Richtung durchgeführt, was die benötigten Taktzyklen um fast 90 % reduziert. Die benötigte Zeit für die Schiebeoperationen beträgt damit noch $\frac{240}{400\,\text{MHz}} = 600\,\text{ns}$.

Optimierung des Ladevorgangs

Wenn seitens des Prozessorsystems eine XRIPSL-Berechnung angestoßen wird (s. Abschnitt 6.4.3), müssen zunächst die zu korrelierenden Sign-Bits aus dem physikalischen Speicher in die zwei Register geladen werden.

Der verwendete Datenbus zum DDR-Controller ist ein 128-Bit-AXI-Bus, benötigt also $\frac{1024 \cdot 32}{128} = 256$ Taktzyklen, um eine gesamte gefilterte Bildzeile ($1024 \cdot 32\,\text{bit}$) zu übertragen. Der Bus wird mit 100 MHz getaktet (praktisch möglich sind mindestens 300 MHz [35]), woraus sich eine Übertragungszeit von mindestens 5 µs für zwei Zeilen ergibt.

Weil während des Ladevorgangs immer ganze 32-Bit-Fließkomma-Samples übertragen werden, aber nur das Sign-Bit weiterverarbeitet wird, wird der Übertragungskanal effektiv nur zu 3 % ausgenutzt. Dies ist aus zwei Gründen von Nachteil:

1. Der Bus wird für 256 Takte (zuzüglich der Takte für die Einleitung des Transfers) belegt und verlangsamt oder verhindert in dieser Zeit möglicherweise andere Transaktionen.
2. Die eigentlich benötigte Information könnte in 8 Takten übertragen werden, wodurch der Vorgang nur noch einen Bruchteil der Zeit benötigen würde.

Um das Problem zu lösen, wäre parallel zum eigentlichen Zeilenpuffer ein zweiter Puffer im physikalischen Speicher notwendig, der ausschließlich Sign-Bits enthält. Grundsätzlich könnte die CPU die Sign-Bits aus den zu korrelierenden Zeilen extrahieren und in einen separaten Puffer schreiben, was aber dem Zweck des Hardwarebeschleunigers, die CPU zu entlasten, entgegenspräche.

Stattdessen wird ein Konzept vorgeschlagen, bei dem bereits beim erstmaligen Übertragen der Pixeldaten in den physikalischen Speicher gleichzeitig ein separater Puffer mit den zugehörigen Sign-Bits gefüllt wird. Das Prinzip ist in Abbildung 6.19 dargestellt: Dabei schließt sich dem SP-Rohbildfilter (s. Abschnitt 5.4.1) ein Modul an, das hier als Sign-Bit-Kopierer bezeichnet wird und von den durchlaufenden gefilterten SP-Samples jeweils die Sign-Bits dupliziert um sie zusätzlich in einen separaten Speicherbereich zu schreiben.

Die Sign-Bits werden nicht von den restlichen Bits des SP-Samples getrennt sondern in einen eigenen kontinuierlichen Speicherbereich kopiert, sodass ein zusätzlicher Speicherbedarf von etwa 3 % entsteht. Gleichzeitig können die Sign-Bits nun aber durch ihre Kontinuität im Speicher direkt in den XRIPSL-Beschleuniger geladen werden, ohne den Bus mit den restlichen 31 bit des SP-Samples zu belasten, die anschließend sofort verworfen würden.

Dem zusätzlichen Speicherbedarf von 3 % steht eine Einsparung von $\frac{248}{256} \approx 97\,\%$ der Taktzyklen für jeden Ladevorgang einer Zeile in den XRIPSL-Beschleuniger gegenüber.

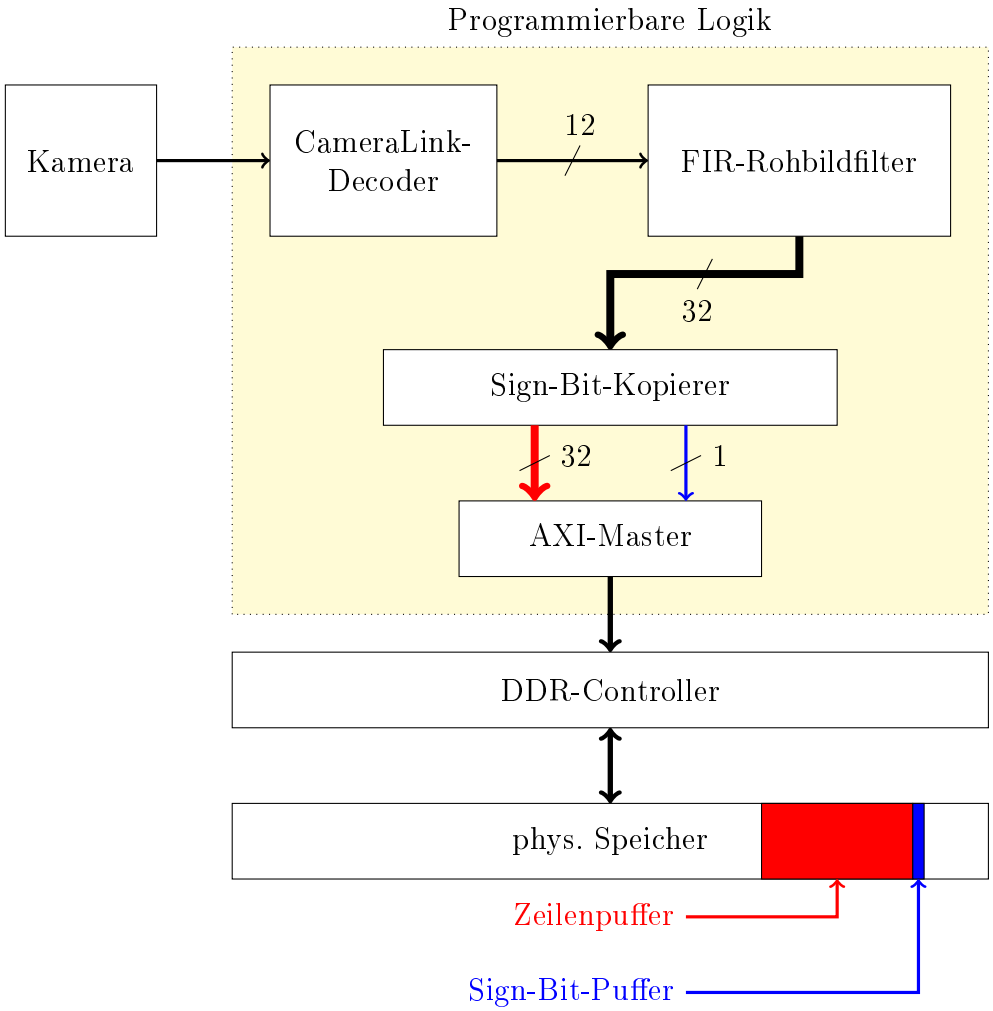

Abbildung 6.19: Erzeugung eines separaten Puffers für Sign-Bits

Laden einzelner Zeilen

Aufgrund des Blockmatching-Prinzips (s. Abschnitt 4.2.1) wird nach der Ermittelung einer Verschiebung durch eine Kreuzkorrelation zwischen den zwei gefilterten Zeilenbildern $i_f(k_1, n)$ und $i_f(k_2, n)$ eines von zwei Szenarien eintreten:

1. Die Verschiebung ist null (oder kleiner als das Verschiebungsziel). Das Zeilenbild $i_f(k_1, n)$ wird als Startpunkt für die nächste Korrelation beibehalten und das Zeilenbild $i_f(k_2, n)$ wird als Endpunkt der nächsten Korrelation durch das Zeilenbild $i_f(k_3, n)$ ausgetauscht.
2. Die Verschiebung ist ungleich null (oder größer/ gleich dem Verschiebungsziel). Das Zeilenbild $i_f(k_1, n)$ wird verworfen, das Zeilenbild $i_f(k_2, n)$, das zuvor Endpunkt der Korrelation war, ist für die nächste Korrelation der Startpunkt der Korrelation.

In beiden möglichen Fällen wird jeweils entweder das Start- oder das Endbild der Korrelation für die folgende Korrelation behalten. Daraus folgt, dass zwischen zwei aufeinanderfolgenden Korrelationen immer nur ein neues Zeilenbild geladen werden muss.

Damit lässt sich die Ladezeit des XRIPSL-Beschleunigers nach der Reduktion um 97 % durch den separaten Sign-Bit-Puffer durch das Laden jeweils nur einer Zeile nochmals halbieren.

In der in Abbildung 6.20 gezeigten Registerbank, die den XRIPSL-Beschleuniger steuert, kann daher eingestellt werden, ob das Register x, das Register y oder beide Register geladen werden sollen. Beide Register müssen nur bei der ersten Korrelation geladen werden, wenn noch gar kein Zeilenbild im Beschleuniger vorliegt.

Anbindung von XRIPSL an das Prozessorsystem

Als Hardwarebeschleuniger ist XRIPSL kein eigenständiges System sondern wird bei Bedarf vom Prozessorsystem aktiviert, um eine bestimmte Berechnung durchzuführen und das Ergebnis zurück an das Prozessorsystem zu melden. Die in den bisherigen Abbildungen gezeigte XRIPSL-Logik stellt also nur einen Teil

des XRIPSL-Beschleunigers dar, der ohne die ihn umgebende Steuerungs- und Schnittstellenlogik nicht einsetzbar ist.

Für die Steuerung des XRIPSL-Beschleunigers wird eine Registerbank bestehend aus vier 32-Bit-Registern definiert: Die ersten zwei Register werden mit den physikalischen Adressen der zu korrelierenden Vektoren gefüllt, das dritte Register dient als Rückgabewert für $m_{\max}$ und das vierte Register dient als Kontrollregister.

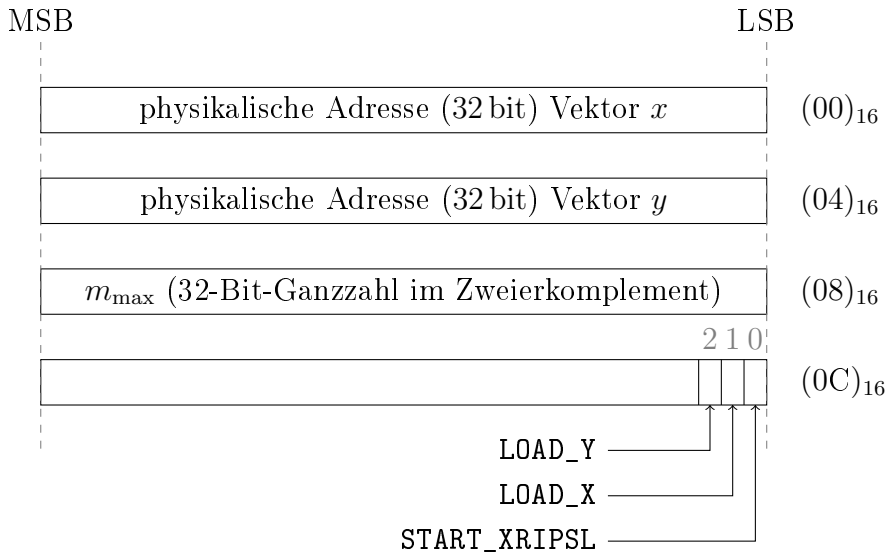

Abbildung 6.20: Register des XRIPSL-Beschleunigers

Sobald im Kontrollregister $(0C)_{16}$ das Bit `START_XRIPSL` seitens des Prozessors gesetzt wird, ist der Ablauf wie folgt:

1. Wenn das Bit `LOAD_X` im Register $(0C)_{16}$ gesetzt ist: Lade den Vektor x von der in Register $(00)_{16}$ angegebenen Adresse.
2. Wenn das Bit `LOAD_Y` im Register $(0C)_{16}$ gesetzt ist: Lade den Vektor y von der in Register $(04)_{16}$ angegebenen Adresse.
3. Führe die Korrelation durch.
4. Schreibe $m_{\max}$ in das Register $(08)_{16}$.
5. Lösche das Bit `START_XRIPSL` im Register $(0C)_{16}$ um den Abschluss des Vorgangs zu signalisieren.

Der Prozessor kann nach dem erstmaligen Setzen des `START_XRIPSL`-Bits dieses kontinuierlich abfragen, bis es wieder zurückgesetzt ist und dann das Ergebnis der Operation aus dem Ergebnisregister auslesen.

Der AXI-Bus zwischen dem physikalischen Speicher (bzw. dem Speichercontroller) und dem XRIPSL-Beschleuniger wurde in den letzten zwei Abschnitten behandelt. Um den Beschleuniger vom Prozessor aus wie beschrieben steuern zu können, wird ein zweiter AXI-Bus zwischen der XRIPSL-Registerbank und dem Prozessorsystem eingesetzt.

Das schematische Zusammenwirken der kommunizierenden Komponenten ist in Abbildung 6.21 dargestellt.

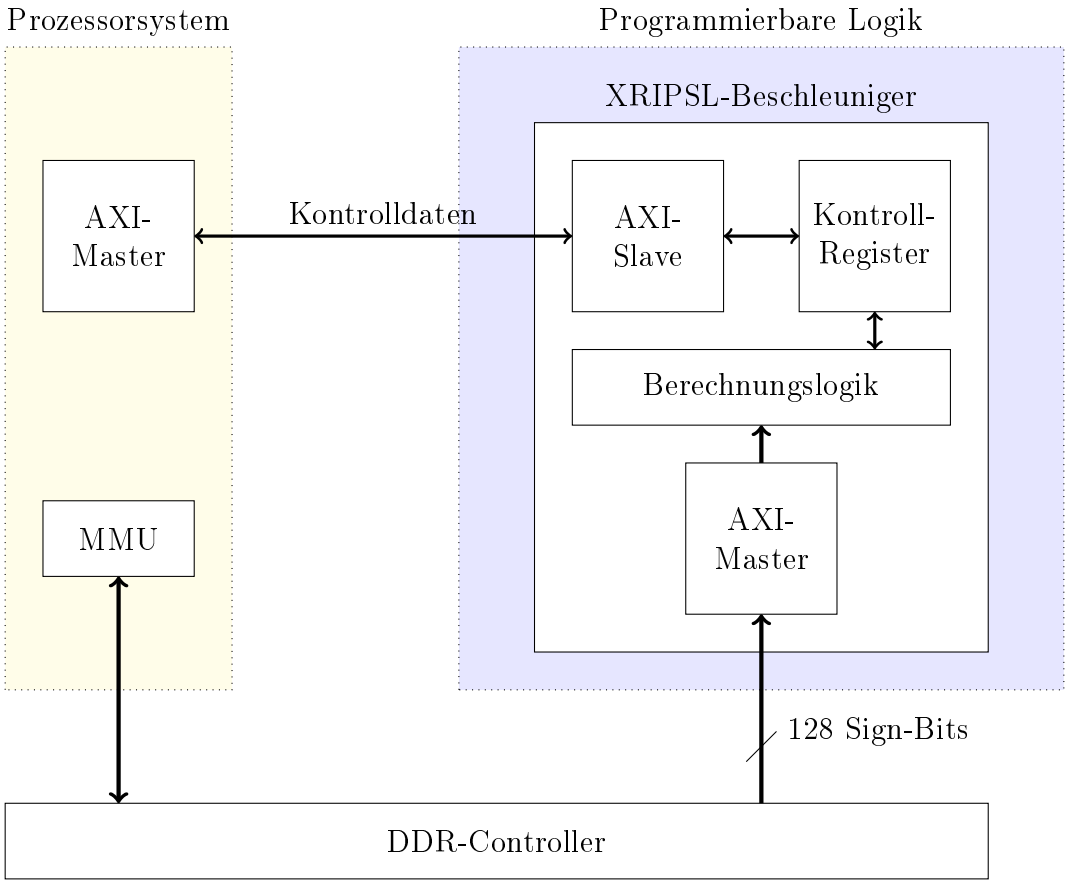

Abbildung 6.21: Anbindung des XRIPSL-Beschleunigers an das Prozessorsystem

Der Registerbank wird eine Busadresse zugewiesen. Diese funktioniert analog zu einer physikalischen Adresse mit dem Unterschied, dass nicht der Arbeitsspeicher

sondern die Registerbank mit der Adresse verknüpft ist. Dadurch lässt sich die XRIPSL-Registerbank in den virtuellen Speicherbereich eines Prozesses mappen, ähnlich wie mit dem reservierten Speicher in Abschnitt 5.3.5 verfahren wurde.

Weil die Busadresse der XRIPSL-Registerbank allerdings im Gegensatz zum reservierten Speicher nicht in einem vom Betriebssystem isolierten Bereich liegt, ist für das Mapping nicht das VADER2-Kernelmodul notwendig.

Benchmark

In Vorbereitung auf die finale Implementierung des Korrelationsalgorithmus werden die verschiedenen Varianten der Kreuzkorrelation in diesem Abschnitt bezüglich ihrer Laufzeiten getestet und zusammengefasst.

Dabei werden folgende Szenarien verglichen, wobei in allen Fällen der Bereich $-120 \leq m \leq 120$ betrachtet wird:

a) Korrelation im Originalbereich

b) Fast Correlation

c) XRIPSL

d) XRIPSL mit optimiertem Ladevorgang (separater Signbit-Puffer)

e) XRIPSL mit optimiertem Ladevorgang (separater Signbit-Puffer) und anschließender Präzisierung des Ergebnisses durch eine Korrelation im Originalbereich mit $m_{\mathrm{max,XRIPSL}} - 3 \leq m \leq m_{\mathrm{max,XRIPSL}} + 3$

Für die Benchmarks, die eine Hardwarebeschleunigung beinhalten wird die Zeitmessung jeweils aus Sicht des Prozessorsystems durchgeführt; es sind also nicht die benötigten Taktzyklen der Berechnungslogik ausschlaggebend, sondern die Zeit, die zwischen dem Start des Hardwarebeschleunigers durch den Prozessor bis zum Vorliegen des Ergebnisses am Prozessor vergeht.

Bezüglich der Szenarien ist festzustellen, dass die Szenarien a) und b) die gleichen Ergebnisse liefern: Die vollständige KKF für den definierten Bereich von m. Die Szenarien c) und d) liefern beide das XRIPSL-Maximum der KKF und das Szenario e) liefert Index und Höhe des Peaks der diskreten Kreuzkorrelation.

Wird daher nur Index und Höhe des Peaks der diskreten KKF ausgewertet, sind die Szenarien a), b) und e) gleichwertig.

Szenario	Kerne	Ladevorgänge	Dauer der Berechnung / µs	Dauer der Berechnung / T_s
a)	1	/	420	29
b)	1	/	102	7
c)	0	1	4,1	0,3
	0	2	7	0,5
d)	0	1	1,6	0,1
	0	2	2,4	0,16
e)	1	1	2,1	0,14
	1	2	2,6	0,18

Tabelle 6.6: Benchmarks der verschiedenen Korrelationsmethoden

Aus der Tabelle ergibt sich, dass durch das Szenario e) die Position und Höhe des Peaks der diskreten KKF (nicht der XRIPSL-KKF) im Bruchteil einer Frameperiode ermittelt werden kann.

6.5 Interpolation der Kreuzkorrelation im Frequenzbereich

Die in Kapitel 4 skizzierten Algorithmen sehen eine Subpixel-Interpolation der KKF vor, um die Genauigkeit der Ergebnisse zu verbessern. Die Gesamtverschiebung besteht dabei aus der Summe eines ganzzahligen Anteils, der ohne Interpolation ermittelt wird, und dem Ergebnis der Interpolation.

Die Überführung der Berechnungsvorschrift der IDFT in das Frequenzbereich-Äquivalent einer Verschiebung im Originalbereich (Gleichung 6.7) liefert den Ansatzpunkt für die DFT-Interpolation:

$$(x \otimes y)(m) = \frac{1}{N} \sum_{k=0}^{N-1} \left[X(k) \cdot \mathrm{e}^{j2\pi km/N} \cdot \overline{Y(k)} \right] \quad \forall\, m \geq 0 \tag{6.16}$$

Im Gegensatz zum diskreten Signal $x(n)$, die nur für ganzzahlige Indizes ungleich 0 ist, kann $X(k)$ durch Multiplikation mit einem Exponentialterm um beliebige Winkel verdreht werden. Für die Interpolation interessant sind dabei diejenigen Winkel, die im Originalbereich einer nicht-ganzzahligen Verschiebung entsprechen.

Ein Beispiel: Ist das interpolierte Sample $(x \otimes y)(m + 0{,}3)$ gesucht, gilt:

$$(x \otimes y)(m + 0{,}3) = \frac{1}{N} \sum_{k=0}^{N-1} \left[X(k) \cdot \mathrm{e}^{j2\pi k(m+0{,}3)/N} \cdot \overline{Y(k)} \right] \quad \forall\, m + 0{,}3 \geq 0 \tag{6.17}$$

Einerseits bietet das Vorgehen den Vorteil, dass die KKF an beliebigen Stellen subpixel-interpoliert werden kann. Der Nachteil ist, dass diese Berechnung nicht mehr durch eine IFFT beschleunigt werden kann, da die IFFT vorberechnete Phasenfaktoren für ganzzahlige m (Twiddle Factors) verwendet [14, S. 175]. Daraus folgt, dass der Ausdruck in Gleichung 6.17 für jeden Interpolationsschritt vollständig berechnet werden muss.

Nach der Einführung von XRIPSL zur Eingrenzung des Korrelationspeaks

hat sich ergeben, dass die Verschiebung m im Allgemeinen etwa im Bereich ± 120 Pixel ausgewertet werden kann. Um über diesen Bereich auf einen Zehntel Pixel zu interpolieren, ergeben sich die Zwischenwerte $\tilde{m} \in \{-120; -119{,}9; -119{,}8; \cdots, 119{,}9; 120\}$, eine Menge mit 2401 Elementen.

Weil die Interpolation nur um $\pm 0{,}5$ um das bereits ermittelte m erfolgen soll (was auf die Arbeitshypothese 1 zurückgeht), muss die Gleichung 6.17 nur für $m - 0{,}5 \leq \tilde{m} \leq m + 0{,}5$ ermittelt werden.

Da m aber im Bereich $-120 \leq m \leq 120$ liegen kann, müssen die Exponentialterme $\mathrm{e}^{j2\pi k\tilde{m}/M}$ für alle möglichen Werte von $k \cdot \tilde{m}$ vorberechnet werden, sofern deren Berechnung aus Gründen der Laufzeit nicht während der eigentlichen Interpolation erfolgen soll.

Dabei muss berücksichtigt werden, dass der Laufindex k die Werte $0, 1, \cdots, N-1$, also N unterschiedliche Werte annimmt, wobei N die Periode des diskreten Spektrums ((I)FFT-Länge) ist. Andererseits werden in dieser Arbeit nur reelle Originalsignale betrachtet, sodass es genügt, die Berechnung über eine Hälfte des Spektrums durchzuführen.

Insgesamt ergeben sich damit bei einer 1200-Punkte-FFT, wie sie für die Fast Correlation in Abschnitt 6.3.1 verwendet wurde, $(1200/2 + 1) \cdot 2401$ komplexe Exponentialterme. Werden Real- und Imaginärteil jeweils als SP-Fließkommazahl abgespeichert, ergibt sich daraus ein Speicherbedarf von etwa 10 MB.

Der Speicherbedarf von 10 MB wird nicht als problematisch angesehen, allerdings führt die Methode zu Problemen, wenn das Interpolationsraster verkleinert werden soll.

Würde das Interpolationsraster beispielsweise von 0,1 Pixel auf 0,01 Pixel verkleinert werden, verzehnfacht sich auch der Speicherbedarf auf 100 MB, was bereits etwa 5 % des überhaupt zur Verfügung stehenden und 25 % des nach Abzug des reservierten Speichers zur Verfügung stehenden Speichers ausmacht.

Wegen der ungünstigen Skalierbarkeit der vorgestellten Methode wird ein alternativer Ansatz vorgeschlagen: Da die Interpolation ohnehin nur um die bereits bekannte, ganzzahlige Verschiebung m herum ausgeführt wird, können die beiden zu korrelierenden Funktionen vor der Interpolation durch einfaches Kopieren im Speicher um m Samples gegeneinander verschoben werden, sodass die restliche

Verschiebung im Bereich $-0{,}5 \leq \tilde{m} \leq 0{,}5$ liegt. Durch diese Ansatz reduziert sich die Komplexität der Berechnung aus zwei Gründen:

1. Sofern $m \neq 0$ gilt, werden durch die Verschiebung um m am Ende bzw. Anfang des Signals m Samples "frei", die zu einem Zero-Padding umfunktioniert werden. Somit ist gar keine Verlängerung der Periode mehr notwendig und es kann die 1024-Punkte-FFT verwendet werden, die die kürzeste Laufzeit hat.
2. Je kürzer die Periode N, desto weniger Produkte sind im Frequenzbereich zu bilden.

Für eine Interpolation mit einem Zehntel-Pixelraster müssen unter Berücksichtigung dieser Punkte und Verwendung einer 1024-Punkte-FFT insgesamt $(1024/2 + 1) \cdot 11$ komplexe Exponentialterme vorberechnet werden.

Die Interpolation selbst wird dann bei bekanntem, positivem m wie folgt durchgeführt:

$$x_{\leftrightarrows}(n) = \begin{cases} x(n+m) & \text{für } 0 \leq n < 1024 - m \\ 0 & \text{für } 1024 - m < n < 1024 \end{cases} \tag{6.18}$$

$$y_{\leftrightarrows}(n) = \begin{cases} y(n) & \text{für } 0 \leq n < 1024 - m \\ 0 & \text{für } 1024 - m < n < 1024 \end{cases} \tag{6.19}$$

Die Signale $x_{\leftrightarrows}(n)$ und $y_{\leftrightarrows}(n)$ sind damit jeweils mindestens mit einem Zero-Sample gepaddet und die Verschiebungen zwischen ihnen liegt nach den getroffenen Annehmen im Bereich $\pm 0{,}5$, sodass $\tilde{m}_{\max}$ das Argument des Maximums der Funktion $f(\tilde{m})$ (Gleichung 6.21) ist, die im gewählten Interpolationsraster ausgewertet wird.

$$XY_{\leftrightarrows}(k) = X_{\leftrightarrows}(k) \cdot \overline{Y_{\leftrightarrows}(k)} \tag{6.20}$$

$$f(\tilde{m}) = \sum_{k=0}^{1023} \left[XY_{\leftrightarrows}(k) \cdot \mathrm{e}^{j2\pi k\tilde{m}/1024} \right] \tag{6.21}$$

Die Gleichung für $f(\tilde{m})$ entspricht dabei der zyklischen Kreuzkorrelation unter Wegfall der Division durch die Periode, da diese keinen Einfluss auf das Maximum hat. Das Produkt $XY_{\leftrightarrows}(k)$ wird außerdem wiederum vorberechnet (Gleichung 6.20) um danach in Gleichung 6.21 für jeden Summanden nur eine Phasendrehung bzw. eine komplexe Multiplikation statt zwei komplexen Multiplikationen durchführen zu müssen.

6.6 Zusammenfassung des Vader2-Algorithmus

Der finale VADER2-Algorithmus orientiert sich an dem in Abschnitt 4.4.1 skizzierten dynamischen Korrelationsalgorithmus mit Qualitätsmerkmal, wobei die Parameter `shift_aim`, `jump` und die Anzahl `nq` der für das Qualitätsmerkmal herangezogenen Zeilenbilder bei der Auswertung der Messreihen experimentell bzw. in der Simulation ermittelt wurden.

Abbildung 6.22 fasst den Ablauf des Algorithmus in vereinfachter Form zusammen.

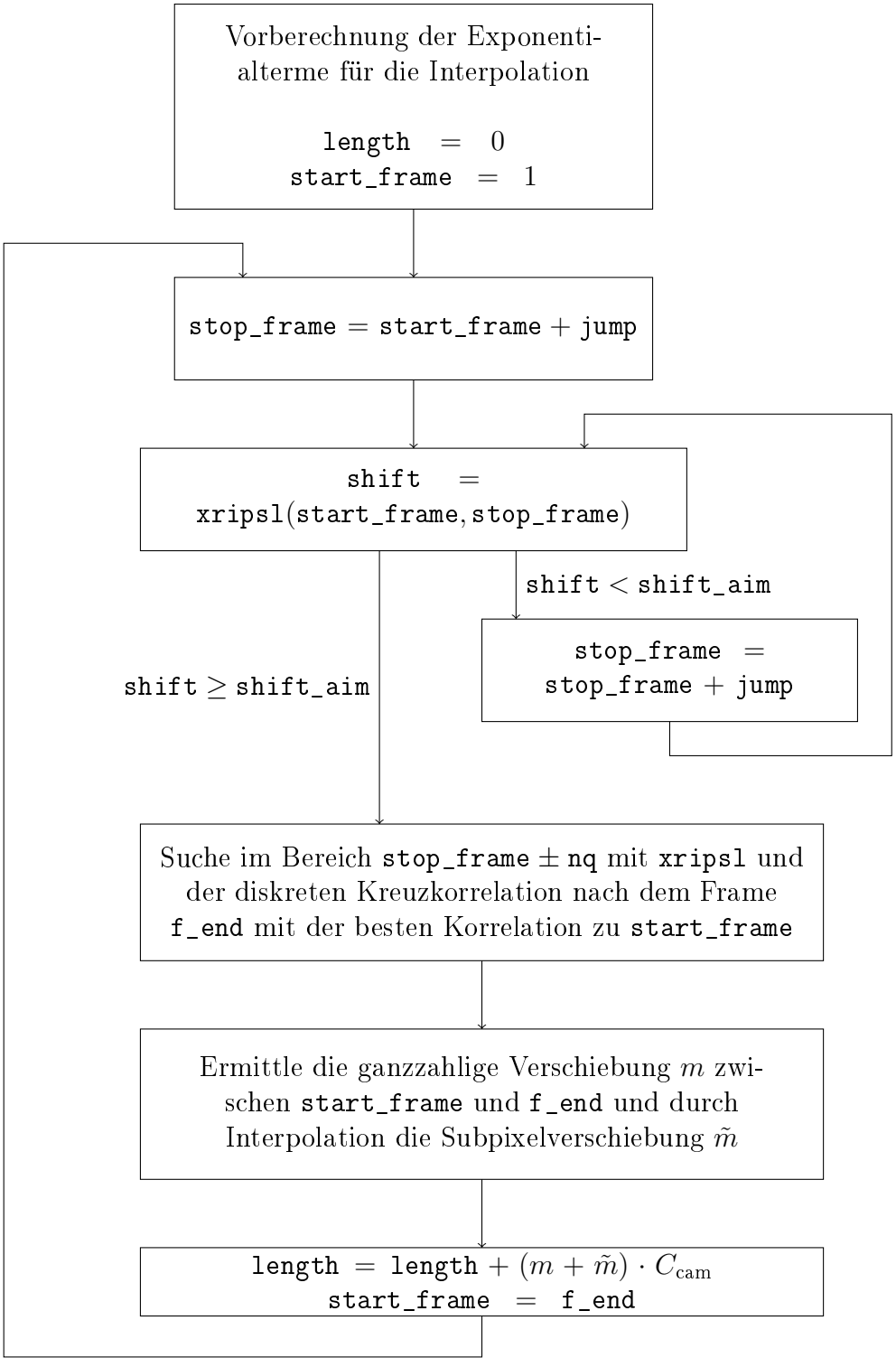

Abbildung 6.22: Vereinfachtes Ablaufdiagramm des VADER2-Korrelationsalgorithmus

7 Verifikation und Evaluation des Vader2-Algorithmus

7.1 Messaufbauten

Am Institut stehen zwei Teststände zur Verfügung: Der bereits in Kapitel 3 eingesetzte Rollenteststand sowie ein Linearteststand. Die Funktionsweise der Teststände ist in Abbildung 7.1 schematisch dargestellt.

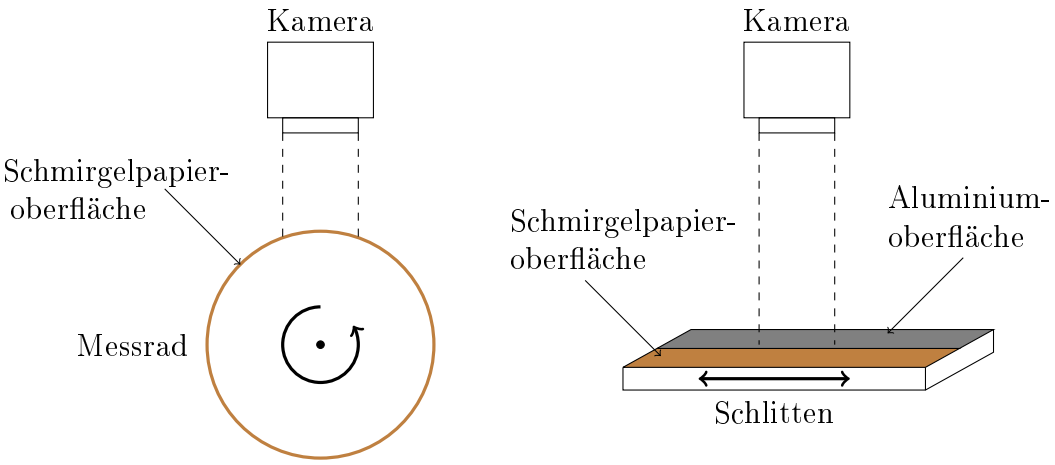

Abbildung 7.1: Rollen- und Linearteststände

Der Umfang des Messrades des Rollenteststandes ist mit Schmirgelpapier beklebt. Auf dem Schlitten des Linearteststandes ist ebenfalls Schmirgelpapier befestigt, es bedeckt aber nicht die gesamte Oberfläche, sodass auch auf dem Aluminium-Grundmaterial des Schlittens gemessen werden kann.

7.2 Messreihen

Es werden folgende Messreihen aufgenommen:

1. Rollenteststand: Strecken zwischen 1 m und 3 m bei Geschwindigkeiten zwischen $1\,\mathrm{m\,s^{-1}}$ und $5\,\mathrm{m\,s^{-1}}$ auf vier parallelen Bahnen.
2. Lineartteststand: Strecken von 1 m und -1 m bei Geschwindigkeiten zwischen $0{,}25\,\mathrm{m\,s^{-1}}$ und $1\,\mathrm{m\,s^{-1}}$ auf drei parallelen Bahnen auf der Schmirgelpapieroberfläche und auf einer Bahn auf der Aluminiumoberfläche.
3. Lineartteststand: Strecken von 10 cm und -10 cm bei Geschwindigkeiten zwischen $0{,}05\,\mathrm{m\,s^{-1}}$ und $0{,}5\,\mathrm{m\,s^{-1}}$ auf fünf parallelen Bahnen auf der Schmirgelpapieroberfläche und zwei parallelen Bahnen auf der Aluminiumoberfläche.

Die einzelnen Messreihen werden als Bildserien mittels eines Framegrabbers mit der Spyder 3-Kamera aufgezeichnet und abgespeichert. Der Wechsel zwischen den einzelnen Bahnen erfolgt durch manuelles Verschieben der Kamera senkrecht zur Bewegungsrichtung. Dabei wird versucht, durch umsichtiges Vorgehen im Rahmen der praktischen Möglichkeiten neben der Parallelverschiebung möglichst keine anderen Änderungen in das System einzubringen.

Die oben angegebenen Geschwindigkeiten sind jeweils die im Messstand eingestellten Geschwindigkeiten und geben die Zielgeschwindigkeit für den Bewegungsvorgang an. Weil jeweils von Stillstand zu Stillstand (s. hierzu auch Abschnitt 7.3) gemessen wird, beinhalten die angegebenen Strecken den Beschleunigungs- und Bremsvorgang, sodass die angegebene Geschwindigkeit eventuell nur für einen Bruchteil der Strecke erreicht wird.

Abbildung 7.2 zeigt eine Übersicht über einige Geschwindigkeitsverläufe aus den Messreihen. Teilweise wird die angegebene Geschwindigkeit beinahe über die gesamte Strecke eingehalten, teilweise nur für einen Teil der Strecke. Im oben rechts dargestellten Extremfall von $0{,}5\,\mathrm{m\,s^{-1}}$ über 10 cm wurde die Zielgeschwindigkeit überhaupt nicht erreicht.

Mit Rückblick auf die in Abschnitt 4.1 festgelegten Testbedingungen wird festgestellt, dass diese Messreihen den vereinbarten Bedingungen entsprechen: Sie beinhalten alle entweder relativ lange Beschleunigungs- und Bremsphasen oder,

wenn sie die Zielgeschwindigkeit schnell erreichen, schwanken die Geschwindigkeiten über den Verlauf der Bewegung.

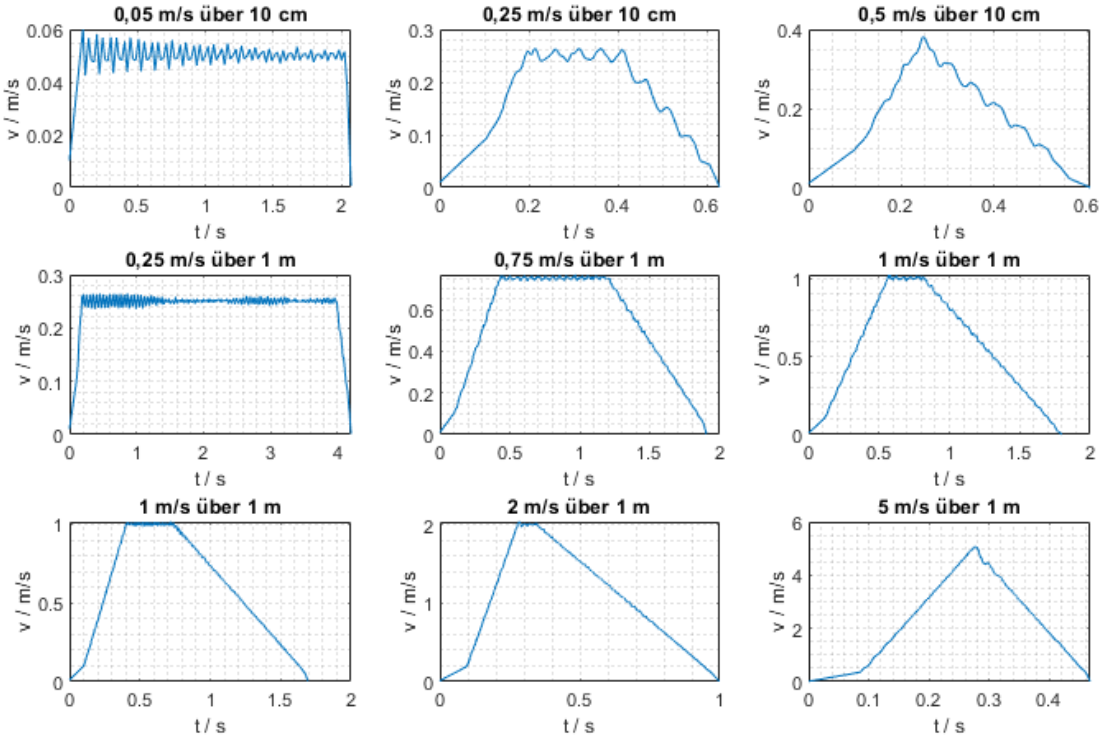

Abbildung 7.2: Beispiele für Geschwindigkeitsverläufe in den Messreihen

7.3 Stillstand

Um eine Bildserie zu erzeugen, die einen Bewegungsvorgang von Stillstand zu Stillstand zeigt, muss die Aufzeichnung der Bilder vor dem Beginn des Bewegungsvorgang gestartet werden, um keinesfalls Teile des Beschleunigungsvorgangs aus der Bildserie auszuschließen.

Der durch den Algorithmus bestimmte Beginn des Bewegungsvorganges wird als Ausgangspunkt verwendet und die Bildserie um 5000 chronologisch ältere Bilder ergänzt, die bei der Ermittelung der Echtzeitfähigkeit wieder herausgerechnet werden.

Abbildung 7.3 zeigt eine Bildserie, die nach dem beschriebenen Schema aufgezeichnet wurde.

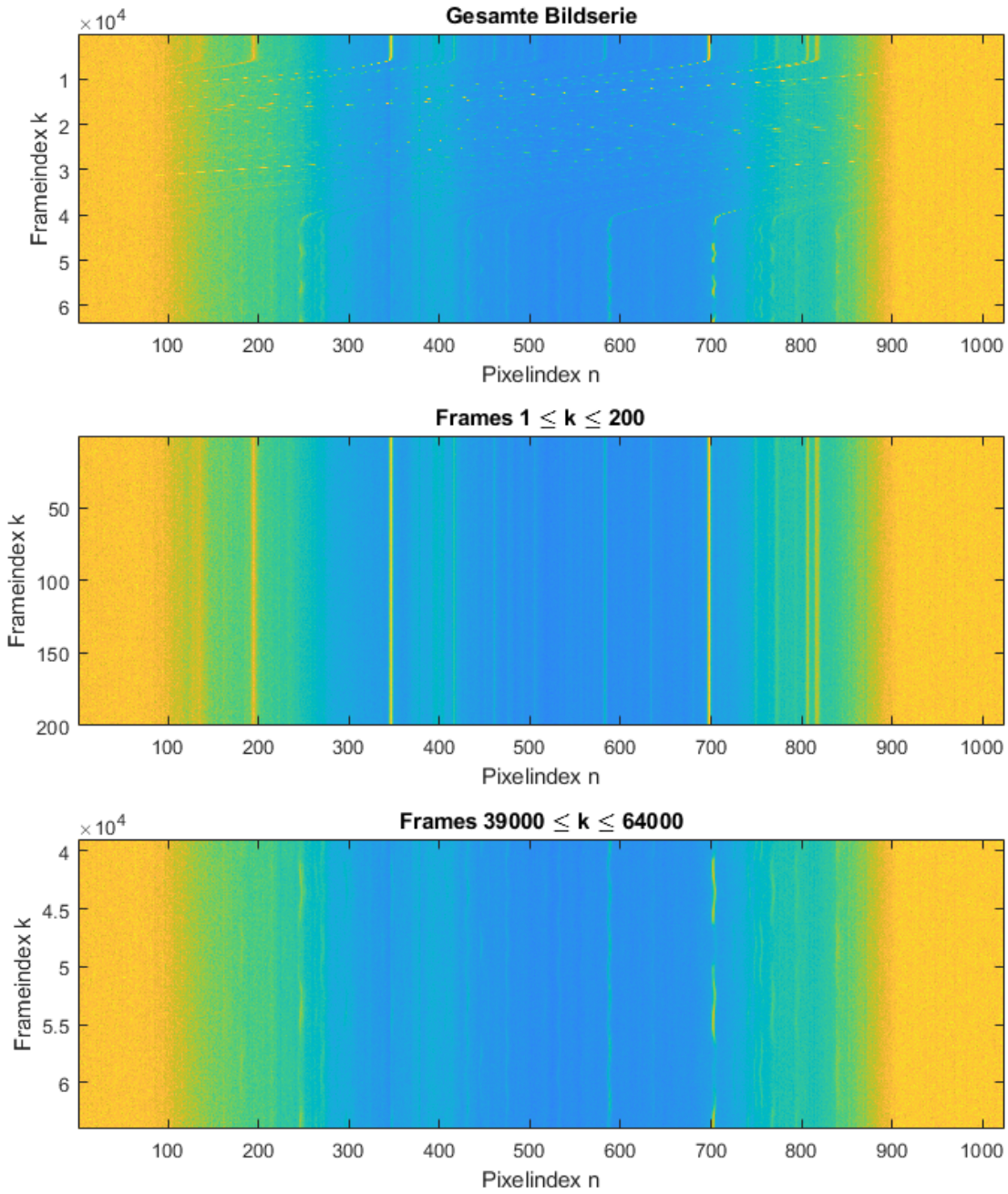

Abbildung 7.3: Phasen des Bewegungsvorgangs in einer Bildserie

Im oberen Plot ist die gesamte Bildserie einer Bewegung über 10 cm zu sehen: Sie unterteilt sich in einen etwa über 5000 Frames andauernden Bereich ohne Bewegung, dann eine deutlich erkennbare Bewegung bis etwa $k \approx 4 \cdot 10^4$ und schließlich einen Bereich in dem nach dem Abbremsen eine Schwingung zu sehen

ist.

Beim Vergleich des mittleren Plots, der tatsächlichen Stillstand zeigt, mit dem unteren Plot ist erkenntlich, dass die Schwingung über das Ende der Bildserie hinaus andauert; die Bildserie wurde aber an einem gewissen Punkt abgeschnitten, weil sich diese Schwingungen bei einzelnen Bildserien über mehrere Sekunden nicht einstellen und damit aus praktischen Gründen des begrenzten Arbeitsspeichers die Bildserie nicht bis zum vollständigen Erreichen des Stillstands aufgezeichnet werden kann.

Dies führt zu einem Problem: Der Bewegungsvorgang ist am Ende der Bildserie de facto noch nicht abgeschlossen, sodass das willkürliche oder durch die Kapazität des Arbeitsspeichers bedingte Abschneiden der Bildserie dazu führt, dass nicht mehr der gesamte Bewegungsvorgang von Stillstand zu Stillstand, wie das ursprünglich vereinbart war, ausgewertet werden kann und das Messergebnis verfälscht wird.

Für dieses Problem konnte keine ideale Lösung gefunden werden. Als Kompromiss werden 25000 Frames des Schwingungsvorgangs in die Bildserie aufgenommen und der Rest abgeschnitten. Die aus diesem Ansatz zu erwartende Messabweichung wird im Folgenden diskutiert.

Abbildung 7.4 zeigt einige Pixel über die letzten Frames der in Abbildung 7.3 gezeigten Bildserie.

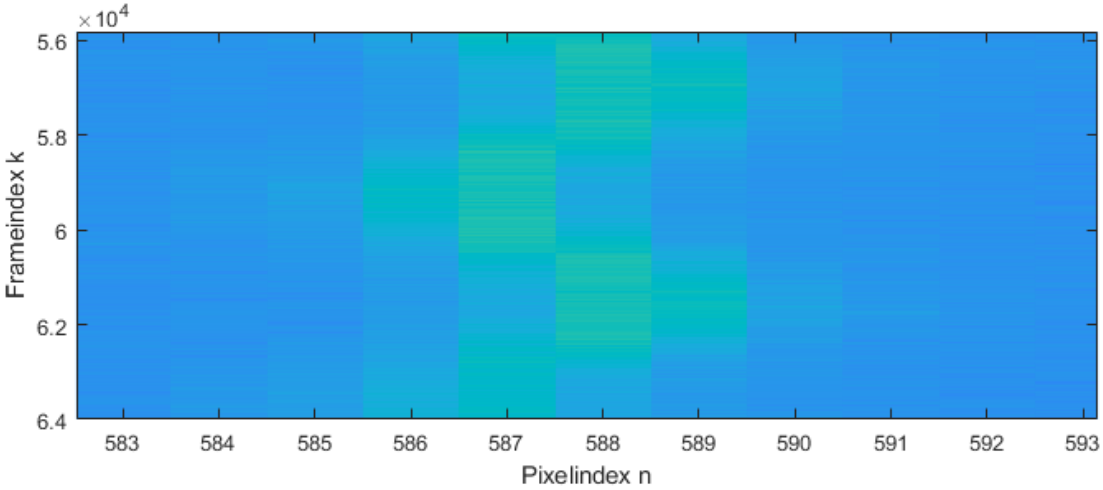

Abbildung 7.4: Oberflächenmerkmal im Schwingungsvorgang beim Abschneiden der Bildserie

Der Mittelpunkt der Schwingung des Oberflächenmerkmals scheint etwa zwischen $n = 587$ und $n = 588$ zu liegen. Je nach dem, bei welchem Frame die Bildserie abgeschnitten würde, könnte das Merkmal im letzten Frame ungefähr im Bereich $586 \leq n \leq 589$ erkannt werden.

Eine Unsicherheit von $2 \cdot W_{\mathrm{px}}$ auf der Kamerazeile entspricht über den Abbildungsmaßstab einer Unsicherheit von etwa 67 µm auf dem Messobjekt. Bezogen auf die in der betrachteten Bildserie gemessenen Strecke von 10 cm entspricht dies einer relativen Unsicherheit von 0,67 ‰. Für die Messungen bei 10 m wird die Schwingung wegen der geringeren relativen Unsicherheit (0,067 ‰) als weniger relevant betrachtet.

7.4 Messergebnisse

Um die Messungen durchzuführen, werden die gefilterten Zeilenbilder auf einen USB3.0-Stick kopiert und an die VADER2-Plattform angeschlossen. Dort werden sie geladen, die Strecke berechnet und dabei die Ausführungszeit gemessen.

Die Parameter `shift_aim` $= 30$, `jump` $= 20$ und der Suchbereich von ± 10 Frames für das Qualitätsmerkmal werden durch experimentelles Vorgehen an allen Messreihen ermittelt. Dabei wird ein Parametersatz ermittelt, der für alle Messreihen erfolgreiche Messungen durchführt.

Dieser Abschnitt stellt die Messergebnisse für die in Abschnitt 7.2 aufgeführten Messreihen grafisch dar. Die Messergebnisse wurden mit einem Korrekturfaktor um systematische Fehler korrigiert.

Tabellen mit den Ergebnissen für jede Einzelmessung inklusive der entsprechenden Laufzeiten des Algorithmus im Vergleich zur Dauer der Bildserie sind in Anhang A.1 zu finden.

7.4.1 Rollenteststand

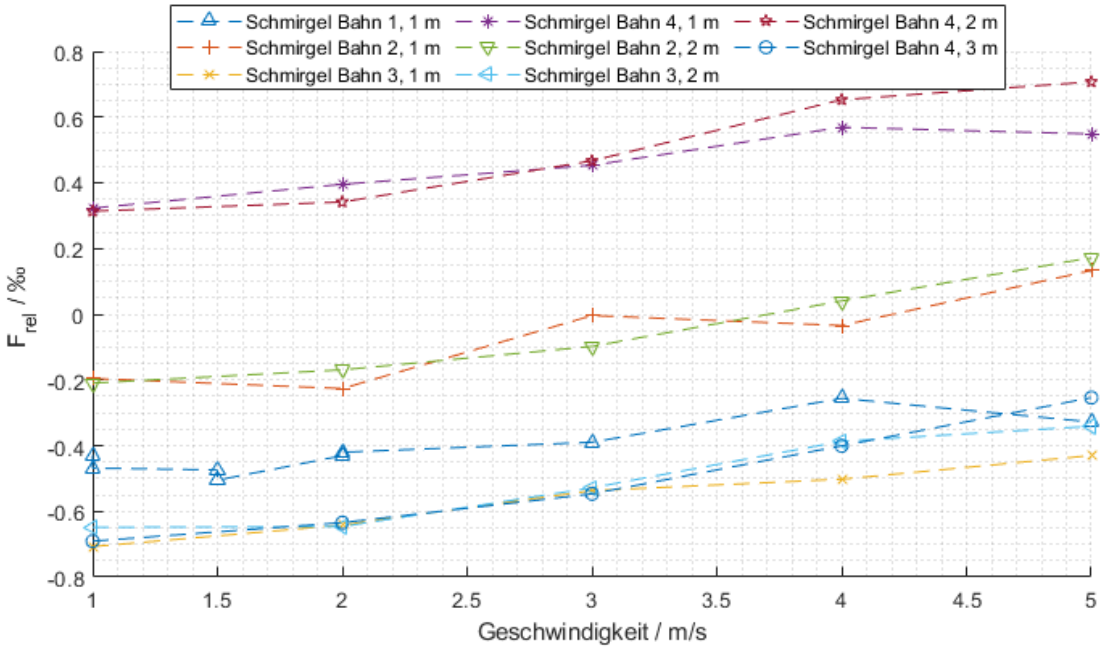

Abbildung 7.5: Messergebnisse für die Messreihe am Rollenteststand

Die Messergebnisse für die einzelnen Bahnen am Rollenteststand sind abgesehen von Bahn 4 relativ eng gruppiert. Dies wird als Hinweis auf einen systematischen Hintergrund gewertet. Für einen systematischen Hintergrund kommen zum Beispiel folgende Erklärungen in Frage:

- Die einzelnen Bahnen können tatsächlich unterschiedliche Umfänge haben, weil das Schleifpapier uneben aufgeklebt sein könnte oder keine konstante Dicke aufweist.
- Beim Verschieben der Kamera kann zum Beispiel die Optik beeinflusst worden sein, sodass der Abbildungsmaßstab oder der Winkel zwischen Kamerazeile und Bewegungsrichtung sich leicht verändert hat.

Insgesamt verbleibt für diese Messreihe eine Messunsicherheit von etwa 0,7 ‰, die sich hauptsächlich durch die Abweichungen zwischen den einzelnen Bahnen ergibt.

7.4.2 Lineartestst and - 10 cm

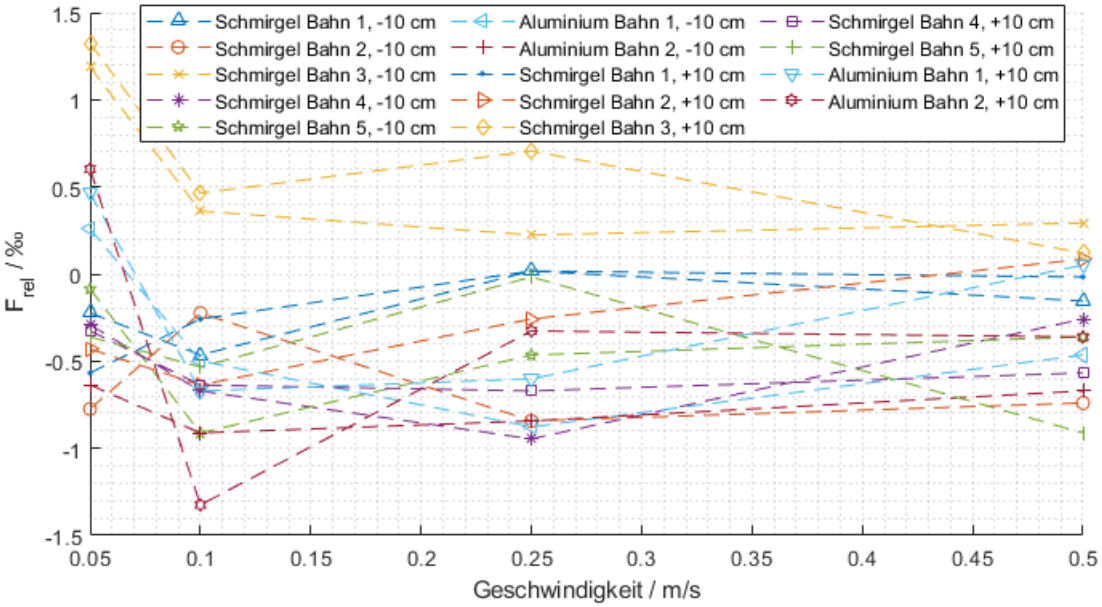

Abbildung 7.6: Messergebnisse für die Messreihe am Lineartestst and mit Strecken von 10 cm

Abgesehen von der Schmirgel Bahn 3, die sich gegenüber den anderen Bahnen systematisch abzuheben scheint, wird bei den Messreihen über 10 cm am Lineartestst and kein klares Muster erkannt.

Es ergibt sich eine Messunsicherheit von etwa 1,3 ‰ für die Messreihe. Wie in Abschnitt 7.3 erläutert, wird diese möglicherweise teilweise durch das Abschneiden der Bildserie während des Ausschwingens verursacht.

7.4.3 Lineartestst and - 1 m

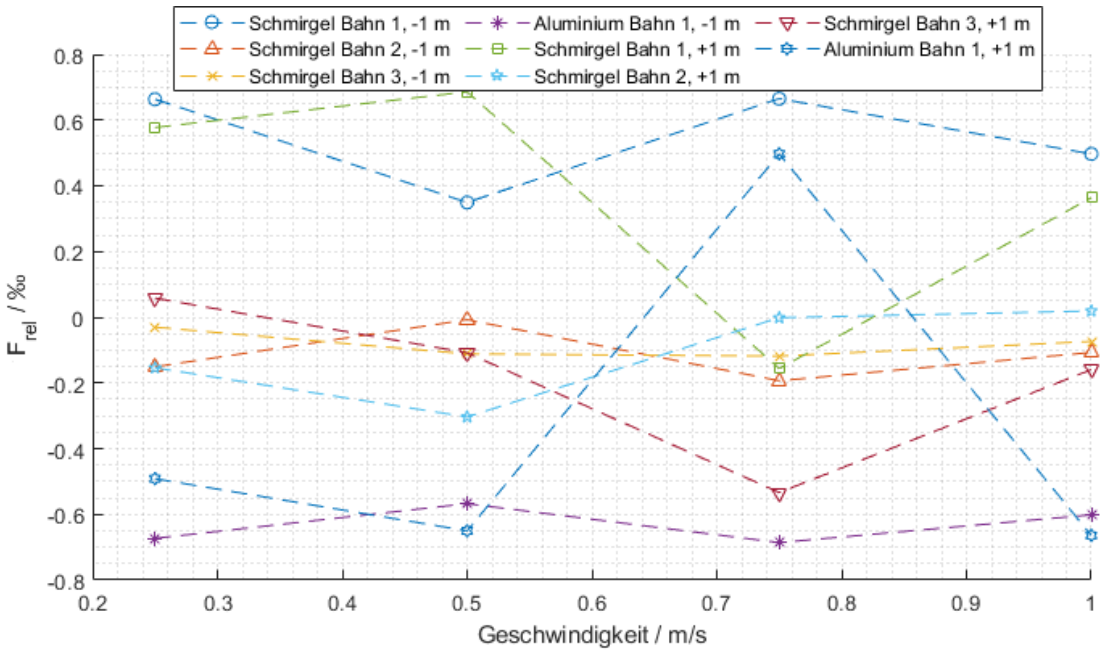

Abbildung 7.7: Messergebnisse für die Messreihe am Lineartestst and mit Strecken von 1 m

Wie beim Rollenteststand bei gemessenen Strecken von $\geq 1\,\mathrm{m}$ ergibt sich für die Strecken 1 m am Lineartestst and eine Messunsicherheit von etwa 0,7 ‰, allerdings mit weniger klaren Gruppierungen innerhalb der einzelnen Bahnen.

7.5 Bewertung der Ergebnisse

Die Bewertung der Ergebnisse erfolgt unter zweierlei Gesichtspunkten: Einerseits der Messunsicherheit und andererseits der Echtzeitfähigkeit.

Bezüglich der Bewertung der erreichten Messunsicherheit werden die Spezifikationen der Industriesensoren aus Abschnitt 2.4 zum Vergleich herangezogen. Wie dort dargelegt wird, ist eine objektive Bewertung des subjektiven Kriteriums eines „geringen Messfehlers“ ohnehin nicht möglich und ein Vergleich zu den aufgeführten Sensorsystemen muss unter der Einbringung von Annahmen zu und Interpretationen der Spezifikationen erfolgen.

Bezüglich der in dieser Arbeit erreichten Unsicherheit für eine Messlänge größer 1 m wird festgehalten:

- Die SFV-Sensor Serie „VLM“ scheidet vom Vergleich aus, weil die angegebenen Unsicherheiten für eine Messlänge von (interpretiert mindestens) 10 m spezifiziert sind, die eine Größenordnung über der in dieser Arbeit gemessenen Strecke liegt.
- Die beste Spezifikation für eine Messlänge von 1 m bei einem SFV-Sensor findet sich beim Covidis mit 0,5 ‰. Diese ist allerdings gebunden an die Bedingung einer konstanten Geschwindigkeit [1, S. 5, 104].

  Generell wird bei SFV-Systemen für die Korrespondenz zwischen Orts- und Zeitfrequenzraum von einer unbeschleunigten Bewegung ausgegangen [1, S. 95], weshalb angenommen wird, dass die gleiche Bedingung auch für andere Ortsfiltersysteme gilt.
- Die angegebene Unsicherheit von 0,3 ‰ ohne Angabe von Bedingungen für die „Laserspeed“-Serie von BETA LaserMike wird für die Diskussion im Sinne des Sensors optimistisch interpretiert und als Vergleichswert hingenommen.

  Ob es sich bei dem Wert um einen Maximalfehler oder eine statistisch ermittelte Größe handelt, wird dabei nicht weiter diskutiert, um nicht weitere Spekulationen einzuführen.

Die in den aufgeführten Punkten genannten Messunsicherheiten liegen in der gleichen Größenordnung wie die in dieser Arbeit ermittelte Maximalunsicherheit von 0,7 ‰ und gelten für die Systeme mit den *geringsten* spezifizierten Messunsicherheiten. Unter diesem Gesichtspunkt wird die in dieser Arbeit erzielte Unsicherheit als *gering* beurteilt.

Für die Messungen auf einer Strecke von 10 cm bleibt zum Vergleich nur der „Laserspeed“-Sensor, der keine Bedingungen und damit auch keine Mindest-Messstrecke angibt. Unter Berücksichtigung der Tatsache, dass andere Sensoren regelmäßig Mindestlängen von $1-10$ m angeben, wird auch hier die rein subjektive Einschätzung getroffen, dass die in dieser Arbeit erreichte Maximalunsicherheit von 1,3 ‰ als *gering* einzuschätzen ist, wenn eine optimistisch ausgelegte Messun-

sicherheit von 0,3 ‰ die *geringste* spezifizierte unter den angebotenen Systemen ist.

Die Echtzeitfähigkeit wird für alle gemessenen Strecken bis 1 m, für die der Algorithmus nach den Vorüberlegungen in Abschnitt 4.1 parametrisiert wurde, unter Verwendung nur eines Prozessorkerns erreicht. Sie wird auch für die längeren Strecken von 2 m und 3 m erreicht, bei höheren Geschwindigkeiten allerdings nur unter Einsatz aller Prozessorkerne.

Nicht erheblich für die Beurteilung, ob das Projektziel eines *geringen* Messfehlers erreicht wurde, aber dennoch ein relevanter Aspekt der Ergebnisbeurteilung im Hinblick auf mögliche Folgeprojekte ist die Tatsache, dass alle hier vorgenommenen Vergleiche den in dieser Arbeit entwickelten Einzelalgorithmus auf der einen Seite vollständigen Sensorsystemen auf der anderen Seite gegenüberstellen.

Bei der Parametrierung des Algorithmus werden konservative Werte verwendet, um in keiner Messreihe das Risiko eines Totalausfalls der Messung einzugehen, um schlussendlich eine *geringe* Maximalabweichung angeben zu können. Möglicherweise führt eine aggressivere Parametrisierung des Algorithmus in Verbindung mit weiteren Plausibilitätsprüfungen oder Redundanzsystemen in Zukunft insgesamt zu besseren Ergebnissen.

8 Fazit und Ausblick

Die vorliegende Arbeit untersucht einen zuvor modellbasiert entwickelten Algorithmus zur Geschwindigkeitsmessung durch Korrelation von Ortsfilterfunktionen anhand von Echtbildern. Dabei ergeben sich zwar zum Großteil Messabweichungen unterhalb $\pm 0{,}5\,‰$, es werden aber auch im Modell nicht vorhergesagte Phänomene beobachtet. Aus den Beobachtungen werden Arbeitshypothesen für Folgeprojekte zur Weiterentwicklung des Modells und des Algorithmus abgeleitet.

Der Hauptteil der Arbeit befasst sich mit der Entwicklung und Implementierung eines echtzeitfähigen Algorithmus zur Längenmessung bewegter Objekte mittels Korrelation zwischen Bildern einer Zeilenkamera. Dafür wird eine Quad-Core Linux-Plattform für ein SoC mit programmierbarer Logik und Prozessorsystem entwickelt.

Der Algorithmus wird in Hardware beschleunigt und erreicht Echtzeitfähigkeit für gemessene Strecken ab 1 m bei Zielgeschwindigkeiten bis $5\,\mathrm{m\,s^{-1}}$ und maximalen Messunsicherheiten von etwa $0{,}7\,‰$, sowie für gemessene Strecken von 10 cm bei Zielgeschwindigkeiten bis $0{,}5\,\mathrm{m\,s^{-1}}$ und maximalen Messunsicherheiten von etwa $1{,}3\,‰$.

Je nach Messsituation nutzt der Algorithmus die Leistungsfähigkeit der entwickelten Mehrkern-Plattform nur zu einem kleinen Teil aus, was die Möglichkeit eröffnet, weitere Algorithmen parallel in das System zu integrieren und mit dem entwickelten Korrelationsalgorithmus zu verbinden. So ist dieser explizit für kurze Strecken und beschleunigte Bewegungen sowie niedrige Geschwindigkeiten ausgelegt und würde sich damit z.B. in Zukunft durch ein Ortsfrequenzfiltersystem, das in diesen Bereichen Schwächen aufweist, sinnvoll auf der entwickelten Plattform ergänzen lassen.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Arno Bergmann. »Verbesserung des Ortsfrequenzfilterverfahrens zur kamerabasierten Messung der translatorischen Geschwindigkeit bewegter Objekte«. Dissertation. Ruhr-Universität Bochum, 2010.

[2] Felix Schneider. »Modellbasierte Entwicklung einer FPGA-Logik für Echtzeit-Bildverarbeitung«. Bachelorarbeit. Hochschule Bochum, 2018.

[3] Sean William Dalton. »Design and Implementation of High-Speed Electronics for a Spatial Filter Velocimeter«. Masterarbeit. Hochschule Bochum, 2018.

[4] Tobias Kanigowski. »Experimentelle Verifizierung optischer Velocimeter mit LabVIEW«. Bachelorarbeit. Hochschule Bochum, 2016.

[5] Lukas Gehrmann. »Industrielles Testsystem für optische Velocimeter«. Bachelorarbeit. Hochschule Bochum, 2016.

[6] Feynman, Leighton und Sands. *The Feynman Lectures on Physics*. URL: `https://www.feynmanlectures.caltech.edu/`.

[7] Yoshihisa Aizu und Toshimitsu Asakura. *Spatial Filtering Velocimetry: Fundamentals and Applications*. 2006.

[8] *Proton InteliSENS SLmini-i4*. `https://protonproducts.com/wp-content/themes/pro-child/assets/BrochureSLmini-i4series-Latest-05.19.21.pdf`. Accessed: 05/2021.

[9] Martin Schaeper. *Mehrdimensionale Ortsfiltertechnik*. Springer Vieweg, 2014.

[10] Felix Schneider. »Simulation einer Geschwindigkeitsmessung durch Kreuzkorrelation zwischen Ortsfilterfunktionen«. Projektarbeit. Hochschule Bochum, 2020.

[11] Helmut Knebl. *Algorithmen und Datenstrukturen*. 2. Auflage. Springer Vieweg, 2021.

[12] Thomas Ottmand und Peter Widmayer. *Algorithmen und Datenstrukturen*. 5. Auflage. Spektrum, 2012.

[13] Jürgen Eichler. *Physik*. 4. Auflage. Vieweg, 2011.

[14] Martin Meyer. *Signalverarbeitung*. 9. Auflage. Springer Vieweg, 2021.

[15] *ASTECH VLM500*. `https://www.astech.de/en/products-home/velocity-and-length-sensors-vlm/vlm500.html`. Accessed: 10/2020.

[16] Rainer Parthier. *Messtechnik*. 8. Auflage. Springer Vieweg, 2016.

[17] Thomas Mühl. *Elektrische Messtechnik*. 6. Auflage. Springer Vieweg, 2020.

[18] Kistler Gruppe. *Correvit SFII*. `https://www.kistler.com/files/document/000-812d.pdf`, 2019.

[19] Kistler Gruppe. *Correvit L-Motion*. `https://www.kistler.com/files/document/003-279d.pdf`, 2019.

[20] *ProSpeed LSV-2100 - Datasheet*. Polytec. 2020.

[21] *LASERSPEED PRO LENGTH and SPEED GAUGE*. NDC Technologies. 2019.

[22] Jens-Rainer Ohm und Hans Dieter Lüke. *Signalübertragung*. 12. Auflage. Springer Vieweg, 2014.

[23] Arno Bergmann und Felix Schneider. *Projektauftrag VADER - Transit Time Correlation Analysis*. 2019.

[24] *Piranha 2 Camera Link User's Manual*. DALSA. 2011.

[25] *Spyder 3 Camera Link User's Manual*. DALSA. 2008.

[26] Sean Dalton. *SFV VADER Lastenheft Version 1.0*. Hochschule Bochum. 2018.

[27] Benjamin Degener. »Erstellung einer vollautomatischen Testumgebung in MATLAB/Simulink für Ortsfrequenzfiltersensoren«. Diplomarbeit. Fachhochschule Gelsenkirchen, 2006.

[28] »IEEE Standard for Floating-Point Arithmetic«. In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019).

[29] The MathWorks Inc. *Fixed-Point Designer Documentation*. 5–2021. URL: https://de.mathworks.com/help/fixedpoint/index.html.

[30] Sean Dalton. *Vader FPGA Board Schaltplan und Layout*. 2018.

[31] *SPRSP07F - 66AK2G1x Multicore DSP+ARM KeyStone II System-on-Chip (SoC)*. Texas Instruments. 2019.

[32] *UG1085 - Zynq UltraScale+ Device TRM*. Xilinx. 2020.

[33] *Ultra96 SBC V2 Schematic*. AVNET. 2019.

[34] *DS891 - Zynq UltraScale+ MPSoC Data Sheet: Overview*. Xilinx. 2019.

[35] »Unexpected Diversity: Quantitative Memory Analysis for Zynq UltraScale+ Systems«. In: *International Conference on Field-Programmable Technology (FPT)*. Okt. 2019.

[36] *SPRAC13 - TI DSP Benchmarking*. Texas Instruments. 2016.

[37] *UG1144 (v2020.1) - PetaLinux Tools Documentation*. Xilinx. 2020.

[38] *Using the Ultra96*. 6–2021. URL: https://www.96boards.org/documentation/consumer/ultra96/ultra96-v2/.

[39] *Intel Linux Development Center SoC FPGA and Nios II Processor*. 6–2021. URL: https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-guidance/linux-developer.html.

[40] Andrew S. Tanenbaum. *Modern Operating Systems*. Fourth Edition. Pearson Education International, 2015.

[41] Rudolf J. Streif. *Embedded Linux Systems with the Yocto Project*. Prentice Hall, 2016.

[42] kernel.org. *Linux and the Devicetree*. 6–2021. URL: https://www.kernel.org/doc/html/latest/devicetree/usage-model.html.

[43] Daniel P. Bovet und Marco Cesati. *Understanding the Linux Kernel*. Third Edition. O'REILLY, 2005.

[44] The Linux Foundation. *The Real Time Linux collaborative project*. 6–2021. URL: https://wiki.linuxfoundation.org/realtime/start.

[45] kernel.org. *The kernel's command-line parameters*. 6–2021. URL: https://www.kernel.org/doc/html/v5.4/admin-guide/kernel-parameters.html.

[46] kernel.org. *The Linux kernel user's and administrator's guide - Memory Management*. 5–2021. URL: https://www.kernel.org/doc/html/latest/admin-guide/mm/concepts.html.

[47] kernel.org. *Linux Kernel Documentation - Reserved memory regions*. 5–2021. URL: https://www.kernel.org/doc/Documentation/devicetree/bindings/reserved-memory/reserved-memory.txt.

[48] *modprobe(8) - Linux man page*. 6–2021. URL: https://linux.die.net/man/8/modprobe.

[49] *ARM Cortex-A Series Programmer's Guide Version 4.0*. ARM. 2013.

[50] *Arm Architecture Reference Manual - Armv8, for Armv8-A architecture profile*. ARM. 2021.

[51] GNU. *GNU libgomp*. 6–2021. URL: https://gcc.gnu.org/onlinedocs/libgomp/.

[52] The MathWorks Inc. *Multicore Simulation and Code Generation of Dataflow Domains*. 6–2021. URL: https://de.mathworks.com/help/dsp/ug/multicore-simulation-and-code-generation-of-dataflow-systems.html.

[53] Massachusetts Institute of Technology. *FFTW User Manual*. 6–2021. URL: http://fftw.org/doc/index.html.

[54] *MATLAB Documentation - xcorr*. The MathWorks, Inc. 2021. URL: https://de.mathworks.com/help/matlab/ref/xcorr.html.

[55] *DS925 - Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics*. Xilinx. 2020.

[56] Ottmar Beucher. *Signale und Systeme: Theorie, Simulation, Anwendung*. 3. Auflage. Springer Vieweg, 2019.

[57] *UltraScale Architecture Libraries Guide*. Xilinx. 2018.

[58] Winfried Gehrke u. a. *Digitaltechnik*. 7. Auflage. Springer Vieweg, 2016.

A Anhang

A.1 Messergebnistabellen

A.1.1 Rollenteststand

Material	Bahn	v / $\mathrm{m\,s^{-1}}$	s / m	s_{meas} / m	F_{rel} / ‰	Laufzeit / %	
						1 CPU	3 CPUs
Schmirgel	1	1,0	1	0,99957	-0,429	29	21
		1,0	1	0,99953	-0,468	29	21
		1,5	1	0,99953	-0,474	39	30
		1,5	1	0,99950	-0,5	39	29
		2,0	1	0,99957	-0,43	46	33
		2,0	1	0,99958	-0,416	46	34
		3,0	1	0,99961	-0,389	52	38
		4,0	1	0,99974	-0,256	68	50
		5,0	1	0,99967	-0,328	68	50
Schmirgel	2	1,0	1	0,99980	-0,19	34	25
		1,0	2	1,99958	-0,209	41	31
		2,0	1	0,99977	-0,23	61	44
		2,0	2	1,99966	-0,17	82	59
		3,0	1	0,99999	-0,004	50	37
		3,0	2	1,99980	-0,01	69	51
		4,0	1	0,99996	-0,03	67	49
		4,0	2	2,00008	0,04	107	78
		5,0	1	1,00013	0,13	68	50

		5,0	2	2,00034	0,17	98	72
Schmirgel	3	1,0	1	1,00032	0,32	35	26
		1,0	2	2,00063	0,32	42	31
		2,0	1	1,00039	0,39	63	46
		2,0	2	2,00068	0,34	82	60
		3,0	1	1,00045	0,45	49	36
		3,0	2	2,00093	0,47	70	52
		4,0	1	1,00056	0,56	67	49
		4,0	2	2,00130	0,65	108	79
		5,0	1	1,00054	0,54	70	51
		5,0	2	2,00141	0,7	104	76
Schmirgel	4	1,0	1	0,99929	-0,7	35	26
		1,0	2	1,99870	-0,65	42	31
		1,0	3	2,99791	-0,69	45	33
		2,0	1	0,99935	-0,64	62	45
		2,0	2	1,99870	-0,65	83	60
		2,0	3	2,99809	-0,63	93	68
		3,0	1	0,99946	-0,53	49	37
		3,0	2	1,99894	-0,53	69	51
		3,0	3	2,99835	-0,55	79	59
		4,0	1	0,99949	-0,54	67	49
		4,0	2	1,99922	-0,38	108	79
		4,0	3	2,99880	-0,4	133	97
		5,0	1	0,99956	-0,43	70	52
		5,0	2	1,99931	-0,34	106	78
		5,0	3	2,99923	-0,25	129	95

Tabelle A.1: Messergebnisse für die Messreihe am Rollenteststand

A.1.2 Lineartestand - 10 cm

Material	Bahn	v / $\mathrm{m\,s^{-1}}$	s / m	s_{meas} / m	F_{rel} / ‰	Laufzeit / %	
						1 CPU	3 CPUs
Schmirgel	1	0,05	-0.1	-0.099971	0.29	< 10	< 10
		0,05	0.1	0.099936	-0.64	< 10	< 10
		0,1	-0.1	-0.099947	0.53	< 10	< 10
		0,1	0.1	0.099967	-0.33	< 10	< 10
		0,25	-0.1	-0.099995	0.05	< 10	< 10
		0,25	0.1	0.099995	-0.05	< 10	< 10
		0,5	-0.1	0.099967	-0.33	< 10	< 10
		0,5	0.1	-0.099978	0.22	< 10	< 10
Schmirgel	2	0,05	-0.1	-0.099916	0.84	< 10	< 10
		0,05	0.1	0.099950	-0.50	< 10	< 10
		0,1	-0.1	-0.099971	0.29	< 10	< 10
		0,1	0.1	0.099929	-0.71	< 10	< 10
		0,25	-0.1	-0.099909	0.91	< 10	< 10
		0,25	0.1	0.099967	-0.33	< 10	< 10
		0,5	-0.1	-0.099919	0.81	< 10	< 10
		0,5	0.1	0.100002	0.02	< 10	< 10
Schmirgel	3	0,05	-0.1	-0.100112	-1.12	< 10	< 10
		0,05	0.1	0.100139	1.39	< 10	< 10
		0,1	-0.1	-0.100029	-0.29	< 10	< 10
		0,1	0.1	0.100040	0.40	< 10	< 10
		0,25	-0.1	-0.100015	-0.15	< 10	< 10
		0,25	0.1	0.100064	0.64	< 10	< 10
		0,5	-0.1	-0.100022	-0.22	< 10	< 10
		0,5	0.1	0.100012	0.12	< 10	< 10

Schmirgel	4	0,05	-0.1	-0.099964	0.36	< 10	< 10
		0,05	0.1	0.099960	-0.40	< 10	< 10
		0,1	-0.1	-0.099926	0.74	< 10	< 10
		0,1	0.1	0.099929	-0.71	< 10	< 10
		0,25	-0.1	-0.099899	1.01	< 10	< 10
		0,25	0.1	0.099926	-0.74	< 10	< 10
		0,5	-0.1	-0.099967	0.33	< 10	< 10
		0,5	0.1	0.099936	-0.64	< 10	< 10
Schmirgel	5	0,05	-0.1	-0.099985	0.15	< 10	< 10
		0,05	0.1	0.099957	-0.43	< 10	< 10
		0,1	-0.1	-0.099902	0.98	< 10	< 10
		0,1	0.1	0.099940	-0.60	< 10	< 10
		0,25	-0.1	-0.099947	0.53	< 10	< 10
		0,25	0.1	0.099991	-0.09	< 10	< 10
		0,5	-0.1	-0.099957	0.43	< 10	< 10
		0,5	0.1	0.099902	-0.98	< 10	< 10
Alu	1	0,05	-0.1	-0.100019	-0.19	< 10	< 10
		0,05	0.1	0.100040	0.40	< 10	< 10
		0,1	-0.1	-0.099943	0.57	< 10	< 10
		0,1	0.1	0.099926	-0.74	< 10	< 10
		0,25	-0.1	-0.099905	0.95	< 10	< 10
		0,25	0.1	0.099933	-0.67	< 10	< 10
		0,5	-0.1	-0.099954	0.46	< 10	< 10
		0,5	0.1	0.099998	-0.02	< 10	< 10

Alu	2	0,05	-0.1	-0.099929	0.71	< 10	< 10
		0,05	0.1	0.100053	0.53	< 10	< 10
		0,1	-0.1	-0.099902	0.98	< 10	< 10
		0,1	0.1	0.099861	-1.39	< 10	< 10
		0,25	-0.1	-0.099909	0.91	< 10	< 10
		0,25	0.1	0.099967	-0.33	< 10	< 10
		0,5	-0.1	-0.099926	0.74	< 10	< 10
		0,5	0.1	0.099957	-0.43	< 10	< 10

Tabelle A.2: Messergebnisse für die Messreihe über Strecken von 10 cm am Linearteststand

A.1.3 Lineartestst and - 1m

Material	Bahn	v / m s^{-1}	s / m	s_{meas} / m	F_{rel} / ‰	Laufzeit / %	
						1 CPU	3 CPUs
Schmirgel	1	0,25	-1	-1,0007	-0,66	15	11
		0,25	1	1,0006	0,58	14	11
		0,5	-1	-1,0003	-0,35	26	18
		0,5	1	1,0007	0,68	23	18
		0,75	1	1,0007	0,66	24	20
		1,0	-1	-1,0005	0,49	27	20
		1,0	1	1,0004	0,36	25	20
Schmirgel	2	0,25	-1	-0,9998	0,15	15	11
		0,25	1	0,9998	-0,15	15	11
		0,5	-1	-1,0000	0,007	25	18
		0,5	1	0,9997	-0,3	23	18
		0,75	-1	-0,9998	0,15	28	21
		0,75	1	0,9998	-0,19	24	20
		1,0	-1	-0,9999	0,1	27	20
		1,0	1	1,0000	0,07	25	20
Schmirgel	3	0,25	-1	-0,99996	0,03	15	11
		0,25	1	1,00006	0,058	14	11
		0,5	-1	-0,99988	0,11	25	18
		0,5	1	0,99989	-0,1	23	19
		0,75	-1	-0,99999	0,001	28	21
		0,75	1	0,99988	-0,12	24	20
		1,0	-1	-0,99992	0,07	28	20
		1,0	1	0,99984	-0,15	25	20

Alu	1	0,25	-1	-0,99931	0,69	15	11
		0,25	1	0,99950	-0,49	14	11
		0,5	-1	-0,99943	0,57	25	18
		0,5	1	0,99935	-0,65	23	18
		0,75	-1	-0,99946	0,53	28	21
		0,75	1	0,99931	-0,69	24	20
		1,0	-1	-0,99939	0,6	27	20
		1,0	1	0,99933	-0,66	25	20

Tabelle A.3: Messergebnisse für die Messreihe über Strecken von 1 m am Linearteststand

A.2 Datenverzeichnis

1 C-Code/

- Wisdom/
- matlab_export.c
- matlab_export.h
- README.txt
- taskstartcpu.c
- vader2_length.c

2 Dokumentation/

- MA_Schneider.pdf

3 MATLAB_Simulink/

- cam_input.slx
- fftxcorr_single.m
- vader2_length.m

4 Platine_CameraLink/

- CameralinkUltra96_HS_connector.xlsx
- Ultra96CameraLink.zip