Wilhelm Büchner Hochschule
Private Fernhochschule Darmstadt

Fraunhofer MEVIS

Wilhelm Büchner Hochschule

Darmstadt

Fachbereich Informatik

Bachelor-Thesis:

Leistungssteigerung in der medizinischen Bildregistrierung durch Mehrkern-Signalprozessoren

vorgelegt von:
Roelof Berg
Technische Informatik
Matrikel-Nummer 877492
Adalbert-Stifter-Str. 19
23562 Lübeck

Erstgutachter	Prof. Dr. rer. nat. Ralph Lausen
Institut	Fraunhofer MEVIS in Lübeck
Fachbetreuer	Jan Rühaak, Lars König
Abgabe	02.08.2012

Abstract

Deutsch

Untersucht wird der Einsatz digitaler Signalprozessoren (DSP) für die Bildregistrierung. Nach einer Einführung in die Bildregistrierung wird zunächst ein mathematisches Verfahren für die rigide Registrierung zweidimensionaler Graustufenbilder entworfen. Es wird gezeigt, wie dieses Verfahren algorithmisch auf einem verteilten Rechnersystem so umgesetzt werden kann, dass eine effiziente Berechnung ermöglicht wird. In Zusammenhang mit den daraus gewonnenen Erkenntnissen wird ein Beispielsystem entwickelt, das Bildregistrierung auf vier Hochgeschwindigkeits-DSPs mit je acht, also insgesamt 32 Rechenkernen, durchführt. Die Geschwindigkeit dieses DSP-basierten Systems wird abschließend messtechnisch mit der Geschwindigkeit herkömmlicher PC-Technik verglichen. Dabei stellt sich heraus, dass durch den Gebrauch von DSPs eine zur PC-Technik konkurrenzfähige Rechenleistung nur dann erreicht wird, wenn der Overhead zur Bildübertragung weitestgehend reduziert wird. Eine Bildregistrierung kann mit DSP-Bausteinen innerhalb von wenigen Sekunden durchgeführt werden, bei gleichzeitiger Überlegenheit in puncto Energie- und Raumbedarf gegenüber der PC-Technik.

English

In this paper, the use of digital signal processors (DSP) for image registration is studied. After an introduction into image registration, a mathematical method for the rigid registration of two-dimensional greyscale images is initially outlined. It will be shown how this method can be implemented algorithmically on a distributed computer system in a way that enables efficient calculation. The findings will help to develop an example system that performs image registration on four high-speed DSPs with eight calculation cores each, that is 32 in total. In the final section, the speed of this DSP-based system will be metrologically compared with the speed of traditional PC technology. As a result, it appears that the use of DSPs only achieves a computing power that is able to compete with PC technology if the overhead for image transfer is being reduced to the greatest possible extent. With DSP components, an image registration can be performed within a few seconds, simultaneously outrunning PC technology in terms of energy consumption and space requirement.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Das Fraunhofer-Institut für Bildgestützte Medizin MEVIS erforscht in der 'Projektgruppe Bildregistrierung' Methoden für bildgestützte Diagnose- und Therapieverfahren[1]. Ein Forschungsschwerpunkt ist die Erforschung effizienter Methoden der Bildregistrierung. Darunter versteht man die Überlagerung von zwei oder mehr Bildern derselben Szene, die zu verschiedenen Zeitpunkten, aus unterschiedlichen Ansichten und/oder mit verschiedenen Sensoren aufgenommen wurden.[2]

Die elektronische Bildregistrierung in einem Computer ist ein rechenintensiver Vorgang. Da die Berechnungsdauer im Sekundenbereich liegt, entsteht beim medizinischen Personal eine Wartezeit, bis das Registrierungsergebnis angezeigt werden kann. Zukünftige Medizingeräte können optimiert werden, indem diese Wartezeit durch effizientere Verfahren zur Berechnung einer Bildregistrierung reduziert wird. Dies kann sowohl eine verzögerungsarme Bilddarstellung als auch eine höhere Bildauflösung zur Folge haben, da die Rechenzeit für eine Bildregistrierung mit zunehmender Bildauflösung steigt.

Bisher wurde in der Projektgruppe MEVIS Bildregistrierung auf herkömmlicher PC-Technik erforscht. In der vorliegenden Bachelor-Thesis zum Studiengang 'Technische Informatik' soll nun der Frage nachgegangen werden, ob durch den Einsatz von digitalen Signalprozessoren (DSP) Vorteile für die Entwicklung von Medizingeräten mit integrierter Bildregistrierung entstehen.

Methodisch wird zunächst aus der Vielzahl an mathematischen Möglichkeiten, mit deren Hilfe eine Bildregistrierung durchgeführt werden kann, ein mathematisches Verfahren entwickelt, das sich für die Analyse der Geschwindigkeit einer DSP-basierten Bildregistrierung eignet. Die daraus resultierenden Formeln können weder der Literatur noch dem Internet entnommen werden, vielmehr müssen sie eigens und speziell auf die besondere Fragestellung hin zugeschnitten werden, wofür umfangreiche mathematische Arbeit leisten ist. Die so ermittelten Rechenschritte werden in den entsprechenden Abschnitten ausführlich nachvollzogen werden. Im Anschluss an die Entwicklung des mathematischen Ansatzes wird ein Vorschlag zur Überführung in einen Algorithmus, unter Berücksichtigung dessen, dass die Untersuchung auf einem verteilten System aus vier DSPs mit je acht Rechenkernen, also insgesamt 32 Rechenkernen, erfolgen soll, unterbreitet. Hier werden

1 Homepage des Instituts: www.mevis-hl.fraunhofer.de

2 „Image registration is the process of aligning two or more images of the same scene taken at different times, from different viewpoints and/or by different sensors."[FM08, S.1]

bereits Konzepte für eine spätere hohe Rechengeschwindigkeit erarbeitet, wie z.B. die Verhinderung von Pipelinehemmnissen im DSP oder die Reduktion von Overhead bei der Datenübertragung.

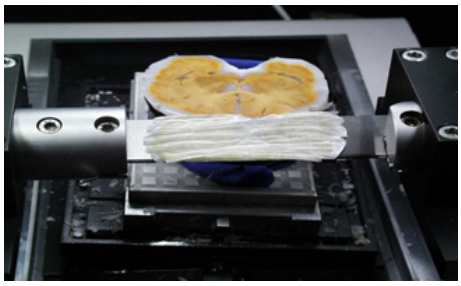
Nach der konzeptionellen Arbeit werden der Hardwareaufbau sowie die softwareseitige Lösung vorgestellt. Für den messtechnischen Vergleich der DSP-basierten mit einer PC-basierten Lösung sind drei Softwareprodukte notwendig: Erstens eine Software, die lokal auf einem PC die Registrierung berechnet und die Zeit misst; sie dient später dem Performancevergleich mit der DSP-Software. Zweitens fungieren zwei weitere Softwareprodukte als Client-Server-System, das die Berechnung der Bildregistrierung verteilt auf vier DSPs und einem PC durchführt. Es wird gezeigt, welche Konzepte geeignet sind, eine Bildregistrierung unter den gegebenen Bedingungen erfolgreich berechnen zu können. Dabei werden auch jene Wege, von denen aufgrund der Erfahrungen in dieser Arbeit abgeraten werden muss, aufgeführt.

Den Abschluss der Arbeit bildet die Analyse der so erreichten Rechengeschwindigkeit; wobei die Rechengeschwindigkeit einer DSP-basierten Lösung mit der von Standard-PCs bis in höchste Leistungsklassen verglichen wird. Die Unterschiede werden analysiert sowie deren Ursachen ermittelt. Daraus wird eine Prognose bezüglich der Anzahl an DSP-Bausteinen ermittelt, die notwendig sind, um einen PC zu ersetzen und unter welchen hardwaretechnischen Bedingungen dies erfolgen muss.

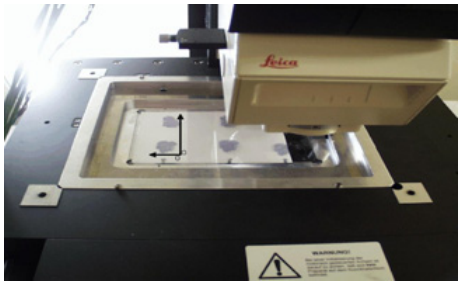
Zunächst wird damit begonnen, das Verfahren zur Bildregistrierung darzulegen.

2 Bildregistrierung

Die Zielsetzung der Bildregistrierung kann an einem Beispiel aus der medizinischen Forschung, erläutert werden:[1] In der medizinischen Forschung werden biologische Objekte mittels histologischer Schnitte analysiert. Bei dieser Technik wird ein biologisches Objekt zunächst gehärtet (z.B. durch Einlegen in Paraffin, oder durch Gefrieren). Anschließend wird es mit einem Mikrotom in sehr dünne, flächenartige Schnitte zerteilt (Abbildung 2.1 links).



(a) Mikrotom



(b) Mikroskop

Abbildung 2.1: Anwendungsbeispiel: Histologische Schnitte eines menschlichen Gehirns

Die Schnitte werden separat mit einem digitalen Mikroskop in digitale Bilddaten umgewandelt (Abbildung 2.1 rechts).

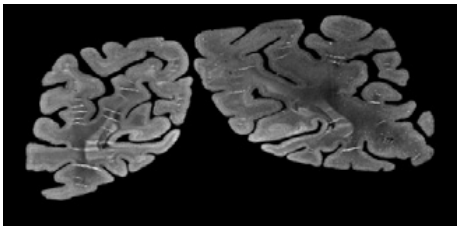
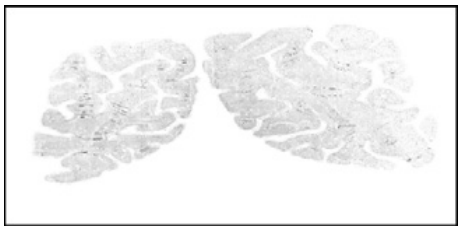


Abbildung 2.2: Digitaldaten eines Schnittexemplars

Da das Mikrotom sie manuell auf den Objektträger des Mikroskops befördert, können die digitalen Schnitte zueinander verdreht und verschoben sein. Ferner kann durch eine Verformung beim Schnitt eine Stauchung entstehen. In Abbildung 2.3 wird links das Differenzbild zweier aufeinanderfolgender histologischer Schnitte dargestellt: je heller

1 Die Bilder 2.1 bis 2.4 sind mit Genehmigung der Autoren entnommen aus [FM08, Seite 3]. Ursprünglich stammen sie aus [Sch95], dessen Autor die Verwendung in der vorliegenden Arbeit genehmigt hat.

(a) Differenzbild ohne Registrierung



(b) Verbesserung im Differenzbild

Abbildung 2.3: Distanzmaß zweier aufeinanderfolgender Schnitte

die Bildpunkte, desto höher, je dunkler, desto niedriger die Übereinstimmung beider Bilder.

Aufgabe der Bildregistrierung ist es in diesem Beispiel, automatisiert eine Transformation zu finden, welche die Bilddaten zu dem vorhergehenden Schnitt abgleicht. Mithilfe des Registrierungsalgorithmus sollen die Bildunterschiede zweier, aufeinanderfolgender Schnitte minimiert werden. In Abbildung 2.3 erkennt man, dass das Differenzbild nach der Registrierung eine höhere Übereinstimmung beider Bilder anzeigt. Sind alle histologischen Schnitte miteinander registriert worden, können die Bilddaten aneinandergereiht werden, um eine 3D-Darstellung des biologischen Objektes zu erhalten (Abbildung 2.4).

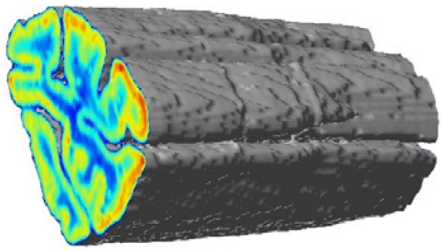
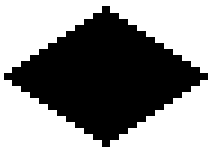


Abbildung 2.4: Zusammengesetzte 3D-Daten

2.1 Optimierungsproblem

In der Bildregistrierung wird eine Transformation ermittelt, die ein Templatebild optimal einem Referenzbild überlagert, was sich durch das Erreichen eines minimalen Distanzmaßes auszeichnet.[2] Ein Distanzmaß ist dabei die auf einen Skalar reduzierte Bewertung der Ähnlichkeit zweier Bilder. Die Bildregistrierung findet das minimale Distanzmaß über mehrere Recheninterationen hinweg, wobei Schritte durchlaufen werden, wie sie beispielhaft in Abbildung Abbildung 2.6 gezeigt werden. Dort soll ein um 45° verdrehtes Quadrat als Templatebild (Abbildung 2.5 links) mit einem unverdrehten Quadrat als Referenzbild

2 Genaue Quellenangaben folgen in der mathematischen Abhandlung.



(a) Templatebild (wird transformiert)

(b) Referenzbild (soll erreicht werden)

Abbildung 2.5: Zwei Beispielbilder, das linke soll auf das rechte registriert werden

(a) 5° (b) 20° (c) 40° (d) 45°

Abbildung 2.6: Sukzessive Approximation auf das minimale Distanzmaß

(Abbildung 2.5 rechts) registriert werden. Ein Optimierungsalgorithmus verändert die Registrierungsparameter sukzessive, bis näherungsweise eine optimale Überlagerung und damit ein minimales Distanzmaß erreicht wird.

Abbildung 2.7 zeigt den Verlauf des Distanzmaßes zwischen beiden Bildern im Verhältnis zum Drehwinkel, wobei man deutlich das gesuchte Minimum bei 45° erkennen kann.

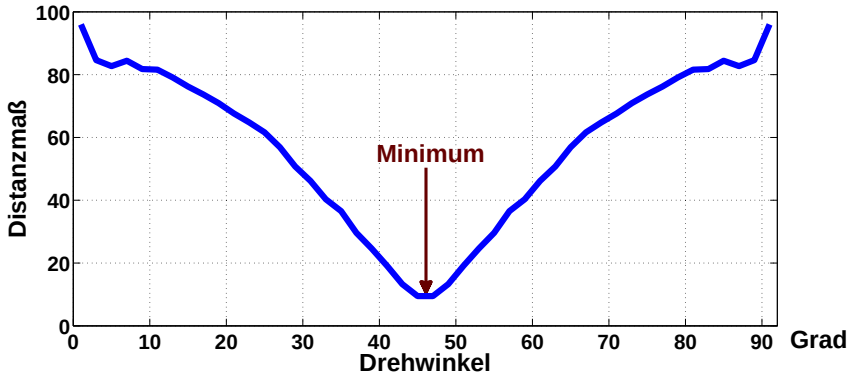


Abbildung 2.7: Veränderung des Distanzmaßes bei Rotation des Templatebilds

Bei einer vollständigen Bildregistrierung werden nicht nur das Minimum des hier exemplarisch dargestellten Rotationswinkels ermittelt, sondern zeitgleich je nach Art des Algorithmus weitere Parameter, wie beispielsweise eine horizontale und eine vertikale Verschiebung.

2.2 Mathematische Ansätze

Es gibt eine Vielzahl unterschiedlicher mathematischer Ansätze in der Bildregistrierung.[3] Für die Erforschung von DSP-Architekturen wird hier der von Jan Modersitzki in seinem Buch 'Fair - Flexible Algorithms for Image Registration'[4] verwendete Ansatz „discretize then optimize“[5] gewählt, der numerische Optimierung mit Newton- (bzw. Quasi-Newton-) Verfahren einsetzt.[6] Für eine Abbildung in DSPs wird angenommen, dass der Einsatz von bewährten und einfachen numerischen Grundverfahren, wie dem Newton-Verfahren, ein hohes Potenzial für eine effiziente Implementierung in einem Digitalrechner bietet. In der vorliegenden Arbeit soll gezeigt werden, dass mit diesem Ansatz eine Bildregistrierung pro Pixel parallelisierbar und mit einer geringen Anzahl an Mikroprozessorinstruktionen (pro Pixel) berechnet werden kann.

Ein „discretize then optimize“ Ansatz beinhaltet die Lösung eines Optimierungsproblems.[7] Für die Ermittlung der Zielfunktion des Optimierungsverfahrens wird zunächst eine Transformation definiert, welche das Templatebild modifiziert. Dazu könnte beispielsweise eine rigide Abbildung gewählt werden, welche das Bild einer Rotation sowie einer Translation unterzieht. Aufgabe des Newton-Verfahrens ist nun, optimale Eingabeparameter für diese Transformation zu finden, so dass, wie zuvor beschrieben, das Distanzmaß minimiert werden kann.

2.3 Bildregistrierung nach Modersitzki

Innerhalb des „discretize then optimize“-Ansatzes der Bildregistrierung existieren viele verschiedene Verfahren und mathematische Alternativen. Für diverse Anwendungsfälle können aus einem breiten Repertoire an Grundbausteinen unterschiedliche, individuell zugeschnittene Algorithmen entwickelt werden. Auch für die hier durchgeführte Untersuchung des DSP-Chips wird ein maßgeschneidertes Verfahren eigens hergeleitet, das auf die Stärken der Hardwarearchitektur abgestimmt, günstig im Speicherverbrauch und pro Pixel parallel berechenbar ist sowie auf Teilbilder angewandt werden kann. Ferner wird es auf eine hohe Auslastung der DSP-Chips optimiert. Dies wird durch Verfahren mit geringer Komplexität erreicht, um Speicherzugriffe und Cache Misses zu reduzieren. Im Folgenden soll die Herleitung einer für DSPs optimalen Formel explizit nachvollzogen werden.

3 Eine Übersicht findet sich in [ZF03]
4 [Mod09] ist kostenfrei als PDF-Version im Internet erhältlich als Teil der Software 'FAIR: Flexible Algorithms for Image Registration - Software and Apps' unter [SIA]
5 [Mod09, S.12]
6 In Abschnitt 2.9 wird näher darauf eingegangen.
7 „The solution of an optimization problem is a set of allowed values of the variables for which the objective function assumes an 'optimal' value."[Gil81, S.1]'

2.4 Mathematische Bausteine

Prinzipiell eignen sich für die Bildregistrierung eine Reihe von Optimierungsverfahren, Interpolationsmethoden sowie unterschiedliche Distanzmaße. Für eine Implementierung muss jeweils ein Verfahren ausgewählt werden. Um das volle Potenzial des DSP-Assemblerbefehlssatzes auszureizen, was u.A. eine Reduktion der langsameren Speicherzugriffe erforderlich macht, werden Verfahren gewählt, die einen bestmöglichen Kompromiss aus Einfachheit, Anzahl der Speicherzugriffe und mathematischer Leistungsfähigkeit bilden. Modersitzki beschreibt mehrere Bausteine, die für eine derartige Bildregistrierung in Frage kommen und aus denen in dieser Arbeit eine Auswahl getroffen wird.[8]

2.4.1 Bildformat

Als exemplarischer Anwendungsfall soll die Registrierung von zweidimensionalen, histologischen Schnitten in Graustufen dienen. Dieser dient als Startpunkt der Erforschung von DSPs in der Bildregistrierung. Komplexere Szenarien für den Einsatz von DSPs mit höherdimensionalen Bildern oder Aufnahmen mit unterschiedlichen Sensoren stellen eine Erweiterung dar, zu deren weiterer Erforschung die hier gelieferten Erkenntnisse beitragen können.

2.4.2 Interpolation

Ein Pixel kann mathematisch als Punkt angesehen werden, ist weder rund, noch quadratisch und hat somit keine Ausdehnung.[9] Deswegen kommt bei der Bildtransformation immer ein Interpolationsverfahren zur Anwendung. Die einfachste Form wäre eine Nächster-Nachbar-Interpolation, welche auch dann - möglicherweise unbewusst - zur Anwendung kommt, wenn dem eigentlich ausdehnungslosen Pixel eine rechteckige Fläche zugewiesen wird. Da diese Interpolation allerdings ungünstig differenzierbar ist (entweder 0 oder eine Unstetigkeitsstelle), wird die etwas aufwendigere bilineare Interpolation gewählt[10]. Die Ableitung der bilinearen Interpolation weist zwar ebenfalls Unstetigkeitsstellen auf, diese sind jedoch vernachlässigbar klein. Zwischen den Unstetigkeitsstellen ist die bilineare Interpolation stetig differenzierbar; dort kann aus vier benachbarten Pixeln der Gradient berechnet werden. Die bilineare Interpolation wird in Unterabschnitt 2.7.3 mathematisch beschrieben.

8 [Mod09]
9 [ZG10, S.7]
10 Interpolationsverfahren im Kontext der Bildregistrierung sind beschrieben in [Mod09, S.24 ff.]

2.4.3 Distanzmaß

Ein einfaches Distanzmaß ist die Fehlerquadratsumme. Es werden die Farbwerte aller Pixel zwischen dem transformierten Templatebild und dem Referenzbild miteinander verglichen. Verglichen wird jeweils ein Pixel im transformierten Templatebild mit dem jeweils anderen Pixel gleicher Position im Referenzbild. Die Ergebnisse jedes Vergleichs werden einzeln quadriert und schließlich insgesamt aufsummiert. Vorteil dieses Distanzmaßes ist die algorithmische Einfachheit.

Es gibt komplexere Verfahren, welche nicht die Farbwerte, sondern die Gradienten vergleichen. Diese eignen sich für die Registrierung von Bilddaten verschiedener Sensoren. Da hier monochrome histologische Schnitte untersucht werden, die mit demselben Sensor digitalisiert wurden, reicht eine Beurteilung der Luminanzwerte bereits aus. Deswegen wird als Distanzmaß die in Unterabschnitt 2.7.4 erläuterte Fehlerquadratsumme zugrunde gelegt.

2.4.4 Optimierung

Modersitzki beschreibt, dass sich das im „discretize then optimize“-Ansatz enthaltene Optimierungsproblem mit dem Gauß-Newton-Verfahren lösen lässt.[11] Das Verfahren hat den Vorteil, dass der Gradient der Zielfunktion zur Ermittlung des Minimums ausreicht, ohne dass die Hesse-Matrix benötigt wird, wie beim klassischen Newton-Verfahren, deswegen soll es hier Verwendung finden.

2.5 Zielfunktion

Für die Lösung des Optimierungsproblems wird zunächst die genaue Zielfunktion ermittelt. Modersitzki liefert die Zielfunktion, indem er den Vorgang der Bildregistrierung wie folgt beschreibt.[12]

„Given two images $\mathcal{R}, \mathcal{T} : \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}$, find a transformation $y : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that

$$\mathcal{J}[y] = \mathcal{D}[\mathcal{T}[y], \mathcal{R}] + \mathcal{S}[y] \overset{y}{\rightarrow} min“ \tag{2.1}$$

11 Darauf wird in Abschnitt 2.9 ausführlich eingegangen. Entnommen aus [Mod09, S.77].
12 [Mod09, S.11]

Die dabei von Modersitzki verwendete Nomenklatur der Variablen und Funktionen ist in Tabelle 2.1 dargestellt. Weil dieses Kapitel wesentlich auf Modersitzkis Publikationen aufbaut, wird seine Nomenklatur[13] übernommen.

d	Raumdimension
$\Omega \subset \mathbb{R}^d$	Relevante Region
$\mathcal{T}, \mathcal{R} : \Omega \subset \mathbb{R}^d \to \mathbb{R}$	Template und Referenzbild
$y : \Omega \to \mathbb{R}^d$	Transformation
$\mathcal{T}[y]$	transformiertes Bild, $\mathcal{T}[y](x) = \mathcal{T}(y(x))$
$\mathcal{T}^{linear}$	lineare Interpolation eines Farbwertes aus mehreren Pixeln
$\mathcal{D}$	Distanzmaß
$\mathcal{S}$	Regularisierer
$\mathcal{J}$	Zusammengesetzte Zielfunktion
w	Satz von Transformationsparametern
w_i	Einzelner Transformationsparameter

Tabelle 2.1: Nomenklatur nach Modersitzki

In Tabelle 2.2 findet sich eine Übersicht über zusätzliche Nomenklatur[14]. Die Einträge werden im weiteren Text näher erläutert.

m	Breite von Template- und Referenzbild in Pixeln ($m_\mathcal{T} = m_\mathcal{R}$)
n	Breite von Template- und Referenzbild in Pixeln ($n_\mathcal{T} = n_\mathcal{R}$)
x	Kartesische Koordinaten, je m horizontal und n vertikal ($\mathbb{R}^{mn \times 2}$)
$\vec{x_i}$	Paar aus horizontaler und vertikaler Komponente einer kartesischen Koordinate ($\mathbb{R}^2$)
x_i	horizontale ($i <= mn$) oder vertikale ($i > mn$) Komponente einer kartesischen Koordinate ($\mathbb{R}^1$)
$\vec{s}$	Suchrichtung für den Gauß-Newton-Algorithmus
$J_\mathcal{F}$	Jacobi-Matrix einer Funktion $\mathcal{F}$
$\mathcal{P}$	abbilden rigider Registrierungsparameter auf affine Parameter
$\mathcal{A}$	bilden eines Koordinatenvektors aus w
$\mathcal{D}_r$	bilden der euklidischen Distanz zweier Bilder auf Elementebene ($\mathbb{R}^{mn} \to \mathbb{R}^{mn}$)
$\mathcal{D}_s$	aufsummieren aller Matrizenelemente zu einem Wert ($\mathbb{R}^{mn} \to \mathbb{R}$)
w_r	Satz von Transformationsparametern für eine rigide Registrierung
w_a	Satz von Transformationsparametern für eine affine Registrierung

Tabelle 2.2: Zusätzliche Nomenklatur

Vernachlässigt man den von Modersitzki vorgeschlagenen Regularisierer $\mathcal{S}$, kommt man auf folgendes Optimierungsproblem:

$$\mathcal{D}[\mathcal{T}[y], \mathcal{R}] \xrightarrow{y} min \tag{2.2}$$

13 Entnommen aus [Mod09, S.11, S.25, S.49].

14 Diese Nomenklatur ist, wenn auch nicht immer direkt entnommen, so doch zumindest geprägt von Modersitzkis Nomenklatur in [Mod09] sowie Besprechungen mit den Fachbetreuern König und Rühaak.

Der Regularisierer dient dazu, unerwünschte Eigenschaften der Transformation zu reduzieren[15]. Da ein möglichst einfacher Algorithmus zum Einsatz kommen soll, um die Analyse der DSP-Bausteine in den Vordergrund zu stellen, genügt eine Registrierung ohne Regularisierer. Für die Zielfunktion nach Gleichung 2.2 wird die Kurzschreibweise $\mathcal{D}(w)$ vorgenommen. Sie bringt zum Ausdruck, dass der wesentliche Aspekt der Zielfunktion die Berechnung des Distanzmaßes $\mathcal{D}$ anhand der Registrierungsparameter w ist.

Bezüglich der Bildtransformation y unterscheidet man zwischen einer rigiden und einer affinen Registrierung.[16] Bei der rigiden Registrierung ist die Bildtransformation auf Rotation und Translation begrenzt, eine affine Registrierung erlaubt zusätzlich die Skalierung und die Scherung.[17] Dabei benötigt die rigide Registrierung $w_r \in \mathbb{R}^3$ Parameter, während die affine Registrierung $w_a \in \mathbb{R}^6$ Parameter verwendet. In den folgenden Formeln wird eine zu transformierende Koordinate, bestehend aus einer vertikalen und einer horizontalen Komponente, mit $\vec{x}_i$ bezeichnet.

2.5.1 Rigide Registrierung

Die rigide Registrierung schränkt die Bildregistrierung auf die Operationen Rotation und Translation ein. Als Registrierungsparameter werden

$$w_r = \begin{bmatrix} w_{r1} & w_{r2} & w_{r3} \end{bmatrix} \tag{2.3}$$

verwendet. Dabei wird eine Koordinate $\vec{x}_i$ durch die Vorschrift

$$x_i(w_r) = \begin{bmatrix} \cos(w_{r1}) & -\sin(w_{r1}) \\ \sin(w_{r1}) & \cos(w_{r1}) \end{bmatrix} \cdot \vec{x}_i + \begin{bmatrix} w_{r2} \\ w_{r3} \end{bmatrix} \tag{2.4}$$

transformiert.

2.5.2 Affine Registrierung

Für eine affine Registrierung werden die Registrierungsparameter

$$w_a = \begin{bmatrix} w_{a1} & w_{a2} & \cdots & w_{a6} \end{bmatrix} \tag{2.5}$$

15 Pohl verwendet hier den Terminus „Bestrafen“ [Poh11, S.5 f.]
16 Definitionen und Formeln sinngemäß entnommen aus [Mod09, S.49 f.]
17 [Wen04, S.16]

verwendet. Einer Koordinate $\vec{x}_i$ wird bei affiner Registrierung durch die Vorschrift

$$x_i(w_a) = \begin{bmatrix} w_{a1} & w_{a2} \\ w_{a4} & w_{a5} \end{bmatrix} \cdot \vec{x}_i + \begin{bmatrix} w_{a3} \\ w_{a6} \end{bmatrix} \tag{2.6}$$

transformiert.

Die Zielfunktion gemäß Gleichung 2.2 ist bei der rigiden Registrierung eine Abbildung von $\mathbb{R}^3$ auf $\mathbb{R}$ und bei der affinen Registrierung eine Abbildung von $\mathbb{R}^6$ auf $\mathbb{R}$. Vergleicht man Gleichung 2.4 mit Gleichung 2.6, so wird ersichtlich, dass die rigide Registrierung ein Spezialfall der affinen Registrierung ist. Dieser Umstand wird zur weiteren Vereinfachung genutzt. Es wird die Abbildung $\mathcal{P}$ als

$$\begin{gathered} \mathcal{P} : \mathbb{R}^3 \to \mathbb{R}^6 \\ \begin{bmatrix} \cos(w_{r1}) & -\sin(w_{r1}) \\ \sin(w_{r1}) & \cos(w_{r1}) \end{bmatrix} \cdot \vec{x}_i + \begin{bmatrix} w_{r2} \\ w_{r3} \end{bmatrix} \to \begin{bmatrix} w_{a1} & w_{a2} \\ w_{a4} & w_{a5} \end{bmatrix} \cdot \vec{x}_i + \begin{bmatrix} w_{a3} \\ w_{a6} \end{bmatrix} \end{gathered} \tag{2.7}$$

definiert. Dadurch braucht in den folgenden Überlegungen nur der rigide Fall betrachtet zu werden, der affine Fall dagegen ist automatisch in den Überlegungen enthalten, was sich in Gleichung 2.9 mathematisch noch exakter zeigen wird.

2.6 Gradient der Zielfunktion

Für das Gauß-Newton-Optimierungsverfahren wird der Gradient der Zielfunktion benötigt. Zur Bestimmung des Gradienten wird die Zielfunktion weiter konkretisiert. Die Zielfunktion $\mathcal{D}(w)$ bildet anhand der Registrierungsparameter w das Distanzmaß zwischen transformiertem Templatebild $\mathcal{T}[y]$ und dem Referenzbild $\mathcal{R}$. Zur Bildung des Distanzmaßes sind mehrere, aufeinanderfolgende Abbildungen notwendig. Eine Übersicht über die notwendigen Abbildungen für die Berechnung zweidimensionaler Bilder der Größe $m \times n$ findet sich in Tabelle 2.3.

$\mathcal{P} : \mathbb{R}^3 \to \mathbb{R}^6$	Abbilden rigider Registrierungsparameter w_r auf affine w_a
$\mathcal{A} : \mathbb{R}^6 \to \mathbb{R}^{2mn}$	Koordinaten für die Transformation $\mathcal{T}[y]$ aus w_a berechnen (Kartesische $\mathbb{R}^2$-Korrdinaten, deswegen $2mn$)
$\mathcal{T}[y] : \mathbb{R}^{2mn} \to \mathbb{R}^{mn}$	Ermitteln der Bildwerte pro Pixel in $\mathcal{T}[y]$ je $\mathbb{R}^2$-Eingangskoordinate aus $\mathcal{A}$ über bilineare Interpolation aus $\mathcal{T}$
$\mathcal{D}_r : \mathbb{R}^{mn} \to \mathbb{R}^{mn}$	Bilden der Residuum-Matrix (Euklidische Distanz pro Bildpunkt)
$\mathcal{D}_s : \mathbb{R}^{mn} \to \mathbb{R}$	Aufsummieren Residuum-Matrix zu einem Wert

Tabelle 2.3: Einzelabbildungen der Distanzmaß-Berechnung in der Reihenfolge ihrer Ausführung

Demnach ist für die Funktion

$$\begin{aligned}&\mathbb{R}^3 \xrightarrow{\mathcal{P}} \mathbb{R}^6 \xrightarrow{\mathcal{A}} \mathbb{R}^{2mn} \xrightarrow{\mathcal{T}[y]} \mathbb{R}^{mn} \xrightarrow{\mathcal{D}_r} \mathbb{R}^{mn} \xrightarrow{\mathcal{D}_s} \mathbb{R}\\ &\mathcal{D}(w_r) = \mathcal{D}_s(\mathcal{D}_r(\mathcal{T}[y](\mathcal{A}(\mathcal{P}(w_r)))))\end{aligned} \tag{2.8}$$

der Gradient zu bilden. Nach der verallgemeinerten Kettenregel kann dies als ein Produkt der Jacobi-Matrizen

$$\begin{aligned}\bigtriangledown\mathcal{D}(w_r) = J_{\mathcal{D}(w_r)} =& J_{\mathcal{D}_s}(\mathcal{D}_r(\mathcal{T}[y](\mathcal{A}(\mathcal{P}(w_r))))) \cdot J_{\mathcal{D}_r}(\mathcal{T}[y](\mathcal{A}(\mathcal{P}(w_r)))) \cdot \\ & J_{\mathcal{T}[y]}(\mathcal{A}(\mathcal{P}(w_r))) \cdot J_{\mathcal{A}}(\mathcal{P}(w_r)) \cdot J_{\mathcal{P}}(w_r)\end{aligned} \tag{2.9}$$

dargestellt werden. Gradienten seien in dieser Arbeit als Zeilenvektoren dargestellt, so entspricht der Gradient $\bigtriangledown\mathcal{D}(w_r)$ der Jacobi-Matrix $J_{\mathcal{D}(w_r)}$. Diese Gleichung gilt für die rigide Registrierung. Soll eine affine Registrierung durchgeführt werden, so wird die Jacobi-Matrix $J_{\mathcal{P}}(w_r)$ weggelassen und der Gradient

$$\begin{aligned}\bigtriangledown\mathcal{D}(w_a) = J_{\mathcal{D}(w_a)} =& J_{\mathcal{D}_s}(\mathcal{D}_r(\mathcal{T}[y](\mathcal{A}(\mathcal{P}(w_r))))) \cdot J_{\mathcal{D}_r}(\mathcal{T}[y](\mathcal{A}(\mathcal{P}(w_r)))) \cdot \\ & J_{\mathcal{T}[y]}(\mathcal{A}(\mathcal{P}(w_r))) \cdot J_{\mathcal{A}}(w_a)\end{aligned} \tag{2.10}$$

verwendet.

2.7 Herleitung der Jacobi-Matrizen

Die Berechnung der in Gleichung 2.9 zu multiplizierenden Jacobi-Matrizen wird nun einzeln und ausführlich hergeleitet. Zunächst wird die zugehörige Funktion (bzw. Funktionenschar) konkretisiert; darauf aufbauend kann dann die Jacobi-Matrix ermittelt werden. Alle hier berechneten Jacobi-Matrizen sowie deren Produkt sind über eine Taylor-Reihenentwicklung auf Korrektheit überprüft worden.[18]

2.7.1 Die Funktion $\mathcal{P}(w_r)$

Die Funktion $\mathcal{P}$ dient der Anwendung der Rigiditätskriterien auf eine ansonsten affine Registrierungsberechnung. Die drei Registrierungsparameter für eine rigide Registrierung w_r (s. Gleichung 2.3) werden abgebildet auf die sechs Registrierungsparameter

18 Das entsprechende Verfahren zum Überprüfen der Nicht-Gültigkeit einer Ableitung ist beschrieben in [Mod09, S.32].

einer affinen Registrierung w_a (s. Gleichung 2.5). Nach Gleichung 2.7 wird die Funktion

$$\begin{aligned}\mathcal{P} :&\mathbb{R}^3 \to \mathbb{R}^6 \\ \mathcal{P}(w_r) = w_a = &\begin{bmatrix} w_{a1}(w_{r1}, w_{r2}, w_{r3}) \\ w_{a2}(w_{r1}, w_{r2}, w_{r3}) \\ \vdots \\ w_{a6}(w_{r1}, w_{r2}, w_{r3}) \end{bmatrix} = \begin{bmatrix} cos(w_{r1}) \\ -sin(w_{r1}) \\ w_{r2} \\ sin(w_{r1}) \\ cos(w_{r1}) \\ w_{r3} \end{bmatrix}\end{aligned} \tag{2.11}$$

gewonnen. Davon lässt sich die Jacobi-Matrix

$$\begin{aligned}&J_{\mathcal{P}} \in \mathbb{R}^{6\times 3} \\ J_{\mathcal{P}}(\mathcal{P}(w_r)) = &\begin{bmatrix} \frac{\partial w_{a1}}{\partial w_{r1}} & \frac{\partial w_{a1}}{\partial w_{r2}} & \frac{\partial w_{a1}}{\partial w_{r3}} \\ \frac{\partial w_{a2}}{\partial w_{r1}} & \frac{\partial w_{a2}}{\partial w_{r2}} & \frac{\partial w_{a2}}{\partial w_{r3}} \\ \vdots & \vdots & \vdots \\ \frac{\partial w_{a6}}{\partial w_{r1}} & \frac{\partial w_{a6}}{\partial w_{r2}} & \frac{\partial w_{a6}}{\partial w_{r3}} \end{bmatrix} = \begin{bmatrix} -sin(w_{r1}) & 0 & 0 \\ -cos(w_{r1}) & 0 & 0 \\ 0 & 1 & 0 \\ cos(w_{r1}) & 0 & 0 \\ -sin(w_{r1}) & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\end{aligned} \tag{2.12}$$

bilden.[19]

2.7.2 Die Funktion $\mathcal{A}(\mathcal{P})$

Durch die Abbildung $\mathcal{A}$ werden die Parameter der affinen Bildregistrierung in eine Menge von Zielkoordinaten transformiert. Das Ergebnis ist eine Matrix, die für jeden Bildpunkt des transformierten Bilds $\mathcal{T}[y]$ die X- und die Y-Koordinate enthält, an deren Stelle in einem späteren Schritt im Originalbild $\mathcal{T}$ der Farbwert (bilinear interpoliert) entnommen werden wird (s. Abbildung 2.8).

Die Koordinaten werden nach X- und Y-Koordinate getrennt abgelegt. Es wird vereinbart, dass die erste Matrizenhälfte die X-, die zweite Hälfte die Y-Koordinaten enthält. $\mathcal{A}$ lässt

19 Das Ergebnis stimmt überein mit der Darstellung in [Mod09, S.53].

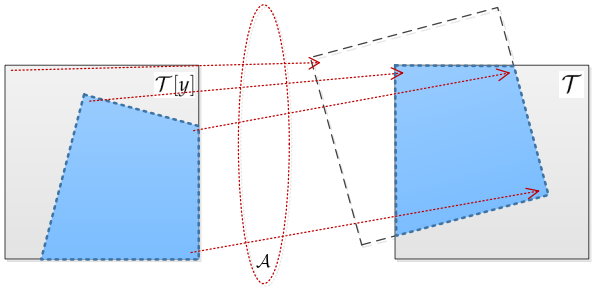


Abbildung 2.8: $\mathcal{A}$ weist jedem Bildpunkt in $\mathcal{T}[y]$ eine Koordinate aus $\mathcal{T}$ zu

sich damit mathematisch als

$$\mathcal{A} : \mathbb{R}^6 \rightarrow \mathbb{R}^{2mn}$$
$$\mathcal{A}(\mathcal{P}(w_r)) = \begin{bmatrix} \mathcal{A}_1(w_a) \\ \mathcal{A}_2(w_a) \\ \vdots \\ \mathcal{A}_{2mn}(w_a) \end{bmatrix} \tag{2.13}$$

beschreiben. Die Koordinaten werden so gebildet, dass zunächst ein äquidistantes Koordinatengitter x mit Ursprung in der Bildmitte sowie der Größe $\mathbb{R}^{mn\times 2}$ generiert wird. Die Koordinaten sollen jeweils auf die Mitte eines Bildpunkts zeigen, wie in der Matrix

$$\begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \\ \cdots & -1.5,-1.5 & -0.5,-1.5 & 0.5,-1.5 & 1.5,-1.5 & \cdots \\ \cdots & -1.5,-0.5 & -0.5,-0.5 & 0.5,-0.5 & 1.5,-0.5 & \cdots \\ \cdots & -1.5,0.5 & -0.5,0.5 & 0.5,0.5 & 1.5,0.5 & \cdots \\ \cdots & -1.5,1.5 & -0.5,1.5 & 0.5,1.5 & 1.5,1.5 & \cdots \\ \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2.14}$$

skizziert. Dieses Gitter wird sodann nach Vorschrift in Gleichung 2.6 anhand der Parameter w_a transformiert. Das Ergebnis ist eine Liste von Koordinaten, die später verwendet werden, um an transformierter Position Bildpunkte zu entnehmen. Trennt man Gleichung 2.6 nach X- und nach Y-Koordinaten (unter Berücksichtigung der Aufteilung im Zielvektor: X von 1 bis mn, Y von mn+1 bis 2mn) wird die Gleichung

$$\mathcal{A}_i(w_a) = \begin{cases} w_{a1}x_i + w_{a2}x_{mn+i} + w_{a3} & i \leq mn \\ w_{a4}x_{i-mn} + w_{a5}x_i + w_{a6} & i > mn \end{cases} \tag{2.15}$$

erhalten. Von Gleichung 2.15 können nun alle partiellen Ableitungen gebildet werden, um

die Jacobi-Matrix zu erhalten:

$$
\begin{aligned}
J_{\mathcal{A}} &\in \mathbb{R}^{2mn\times 6} \\
J_{\mathcal{A}}(\mathcal{A}(\mathcal{P}(w_r))) &= \begin{bmatrix} \frac{\partial \mathcal{A}_1}{\partial w_{a1}} & \frac{\partial \mathcal{A}_1}{\partial w_{a2}} & \dots & \frac{\partial \mathcal{A}_1}{\partial w_{a6}} \\ \frac{\partial \mathcal{A}_2}{\partial w_{a1}} & \frac{\partial \mathcal{A}_2}{\partial w_{a2}} & \dots & \frac{\partial \mathcal{A}_2}{\partial w_{a6}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{A}_{2mn}}{\partial w_{a1}} & \frac{\partial \mathcal{A}_{2mn}}{\partial w_{a2}} & \dots & \frac{\partial \mathcal{A}_{2mn}}{\partial w_{a6}} \end{bmatrix} \\
&= \begin{bmatrix} x_1 & x_{mn+1} & 1 & 0 & 0 & 0 \\ x_2 & x_{mn+2} & 1 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{mn} & x_{2mn} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & x_{mn+1} & 1 \\ 0 & 0 & 0 & x_2 & x_{mn+2} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & x_{mn} & x_{2mn} & 1 \end{bmatrix}
\end{aligned}
\tag{2.16}
$$

2.7.3 Die Funktion $\mathcal{T}[y](\mathcal{A})$

Durch die Operation $\mathcal{T}$[y] wird die zuvor in $\mathcal{A}$ erhaltene Menge von realwertigen, transformierten Koordinaten über bilineare Interpolation in Bildwerte umgewandelt. Dabei werden für das Ermitteln eines Bildwerts der Zielmatrix insgesamt vier Bildwerte des Templatebilds $\mathcal{T}$ berücksichtigt.

$$
\begin{aligned}
\mathcal{T}[y] &: \mathbb{R}^{2mn} \to \mathbb{R}^{mn} \\
\mathcal{T}[y](\mathcal{A}(\mathcal{P}(w_r))) &= \begin{pmatrix} \mathcal{T}[y]_1(\mathcal{A}) \\ \mathcal{T}[y]_2(\mathcal{A}) \\ \vdots \\ \mathcal{T}[y]_{mn}(\mathcal{A}) \end{pmatrix} = \begin{pmatrix} \mathcal{T}_1^{linear}(\mathcal{A}_1, \mathcal{A}_{mn}) \\ \mathcal{T}_2^{linear}(\mathcal{A}_2, \mathcal{A}_{mn+1}) \\ \vdots \\ \mathcal{T}_{mn}^{linear}(\mathcal{A}_{mn}, \mathcal{A}_{2mn}) \end{pmatrix}
\end{aligned}
\tag{2.17}
$$

Modersitzki beschreibt die Gleichung

$$
\mathcal{T}^{linear}(\chi) = \sum_{\kappa\in\{0,1\}^d} dataT(p+\kappa) \prod_{i=1,\dots,d} (\xi^i)^{\kappa^i} (1-\xi^i)^{(1-\kappa^i)} \tag{2.18}
$$

$$
\tag{2.19}
$$

für die lineare Interpolation.[20] Dabei steht d für die Anzahl räumlicher Dimensionen, ξ für den Abstand des Interpolationspunkts nach links sowie oben zum nächsten Referenzpixel. Die Funktion $dataT(p)$ soll den Zugriff auf einzelne Bildwerte an der Stelle p beschreiben.

20 [Mod09, S.25]

Für $d = 2$, wird die Gleichung

$$\begin{aligned}\mathcal{T}^{linear}(x,y) = \mathcal{B}_{x,y} \cdot (1-\xi_x) \cdot (1-\xi_y) + \mathcal{B}_{(x+1),y} \cdot \xi_x \cdot (1-\xi_y) \\ + \mathcal{B}_{x,(y+1)} \cdot (1-\xi_x) \cdot \xi_y + \mathcal{B}_{(x+1),(y+1)} \cdot \xi_x \cdot \xi_y\end{aligned} \tag{2.20}$$

gewonnen.[21] Dabei soll hier anstelle von Modersitzkis Schreibweise $dataT(p)$ die Syntax $\mathcal{B}_{x,y}$ verwendet werden und anstelle von ξ^i als Benennung des Abstands von der linken, oberen Ecke ξ_x und ξ_y. Über diese Interpolationsvorschrift wird auf die Bilddaten des Templatebilds $\mathcal{T}$ anhand der im vorigen Schritt berechneten Transformationskoordinaten $\mathcal{A}$ so zugegriffen, dass im Ergebnis das Templatebild $\mathcal{T}$ um die Transformation y modifiziert wird. Das Ergebnis ist das transformierte Templatebild $\mathcal{T}[y]$.

Vektorisierter Zugriff

Die Bildwerte von $\mathcal{T}$ werden algorithmisch über einen vektorisierten Zugriff adressiert. Unter einem vektorisierten Zugriff soll hier verstanden werden, dass die einzelnen Zeilenvektoren einer Matrix zu einem Vektor konkateniert werden. Dabei wird der erste Vektoreintrag mit dem Elementindex 1 adressiert. Vorteil dieses Verfahrens ist, dass die Jacobi-Matrizen maximal zweidimensional werden.

$$\begin{bmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,m} \\ e_{2,1} & e_{2,2} & \cdots & e_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ e_{n,1} & e_{n,2} & \cdots & e_{n,m} \end{bmatrix} \rightarrow \begin{bmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,m} & e_{2,1} & e_{2,2} & \cdots & e_{n,m} \end{bmatrix} \tag{2.21}$$

Die Pixel werden linear über einen Index 1...mn adressiert. Dafür soll die Syntax $\mathcal{T}_{(i)}$ gewählt werden. Ferner wird die Schreibweise $\mathcal{T}(\lfloor \mathcal{A}_{ix} \rfloor, \lfloor \mathcal{A}_{iy} \rfloor)$ definiert. Sie soll für die Entnahme eines Bildwerts aus dem Templatebild $\mathcal{T}$ über einen vektorisierten Zugriff an der horizontalen Koordinate $\lfloor \mathcal{A}_{ix} \rfloor$ sowie der vertikalen Koordinate $\lfloor \mathcal{A}_{iy} \rfloor$ nach der Vorschrift

$$\mathcal{T}(\lfloor \mathcal{A}_{ix} \rfloor, \lfloor \mathcal{A}_{iy} \rfloor) = \mathcal{T}_{(\lfloor \mathcal{A}_{iy}+n/2-0{,}5 \rfloor \cdot m + \lfloor \mathcal{A}_{ix}+m/2+0{,}5 \rfloor)} \tag{2.22}$$

stehen.

Die Addition von $n/2$ bzw. $m/2$ ist notwendig, weil der Koordinatenursprung der in $\mathcal{A}_i$ enthaltenen Koordinaten in der Bildmitte von $\mathcal{T}$ liegt. Der Ursprung der Indexierung für den vektorisierten Zugriff auf $\mathcal{T}$ liegt jedoch beim linken, oberen Bildpunkt.[22]

21 Wie schon die vorangegangene Formel lässt sich auch diese Umformung aus [Mod09, S.25] entnehmen.

22 Für die Implementierung in Software sei angemerkt, dass die Floor Operation in der Reihenfolge vorzuziehen ist, weil im Integer-Format bestimmte Rundungsfehler nicht vorkommen können. Z.B.

In Abschnitt 3.1 wird näher auf die Problematik eingegangen werden, im Algorithmus ein Überschreiten der Bildränder beim vektorisierten Zugriff zu vermeiden. Für die rein mathematische Darstellung sei dieses Detail zunächst vernachlässigt. Es ergibt sich die Formel

$$\begin{aligned}\mathcal{T}_i^{linear}(\mathcal{A}_i, \mathcal{A}_{mn+i}) =& \mathcal{T}(\lfloor\mathcal{A}_i\rfloor, \lfloor\mathcal{A}_{mn+i}\rfloor) \cdot (1-\xi_x) \cdot (1-\xi_y) \\ &+ \mathcal{T}(\lfloor\mathcal{A}_i\rfloor + 1, \lfloor\mathcal{A}_{mn+i}\rfloor) \cdot \xi_x \cdot (1-\xi_y) \\ &+ \mathcal{T}(\lfloor\mathcal{A}_i\rfloor, \lfloor\mathcal{A}_{mn+i}\rfloor + 1) \cdot (1-\xi_x) \cdot \xi_y \\ &+ \mathcal{T}(\lfloor\mathcal{A}_i\rfloor + 1, \lfloor\mathcal{A}_{mn+i}\rfloor + 1) \cdot \xi_x \cdot \xi_y \end{aligned} \tag{2.23}$$

für die Interpolation. Dabei gelten für ξ die Gleichungen

$$\xi_x = (\mathcal{A}_i + 0.5) \mod 1 \tag{2.24}$$

$$\xi_y = (\mathcal{A}_{mn+i} + 0.5) \mod 1 \tag{2.25}$$

wegen der um einen halben Pixel verschobenen Gitterstützpunkte (s. Matrix 2.14).

Die analytische Ableitung der bilinearen Interpolation erfolgt durch partielle Ableitung von Gleichung 2.23. Für die partielle Ableitung zur horizontalen Koordinate $\partial\mathcal{A}_i$ kann diese Gleichung wegen der Richtungsgleichheit nach ξ_x partiell abgeleitet werden. Für die partielle Ableitung zur vertikalen Koordinate $\partial\mathcal{A}_{mn+i}$ wird entsprechend partiell nach ξ_y abgeleitet.

$$\begin{aligned}\Delta_x(i) =& \frac{\partial\mathcal{T}[y]_i}{\partial\mathcal{A}_i} = \frac{\partial\mathcal{T}_i^{linear}}{\partial\xi_x} \\ =& \mathcal{T}(\lfloor\mathcal{A}_i\rfloor + 1, \lfloor\mathcal{A}_{mn+i}\rfloor + 1) \cdot \xi_y - \mathcal{T}(\lfloor\mathcal{A}_i\rfloor, \lfloor\mathcal{A}_{mn+i}\rfloor + 1) \cdot \xi_y \\ &+ \mathcal{T}(\lfloor\mathcal{A}_i\rfloor, \lfloor\mathcal{A}_{mn+i}\rfloor) \cdot (\xi_y - 1) - \mathcal{T}(\lfloor\mathcal{A}_i\rfloor + 1, \lfloor\mathcal{A}_{mn+i}\rfloor) \cdot (\xi_y - 1) \end{aligned} \tag{2.26}$$

$$\begin{aligned}\Delta_y(mn+i) =& \frac{\partial\mathcal{T}[y]_{mn+i}}{\partial\mathcal{A}_{mn+i}} = \frac{\partial\mathcal{T}_{mn+i}^{linear}}{\partial\xi_y} \\ =& \mathcal{T}(\lfloor\mathcal{A}_i\rfloor + 1, \lfloor\mathcal{A}_{mn+i}\rfloor + 1) \cdot \xi_x - \mathcal{T}(\lfloor\mathcal{A}_i\rfloor + 1, \lfloor\mathcal{A}_{mn+i}\rfloor) \cdot \xi_x \\ &+ \mathcal{T}(\lfloor\mathcal{A}_i\rfloor, \lfloor\mathcal{A}_{mn+i}\rfloor) \cdot (\xi_x - 1) - \mathcal{T}(\lfloor\mathcal{A}_i\rfloor, \lfloor\mathcal{A}_{mn+i}\rfloor + 1) \cdot (\xi_x - 1) \end{aligned} \tag{2.27}$$

In diesem Zusammenhang sei angemerkt, dass die Ableitung der bilinearen Interpolation verschwindend kleine Unstetigkeitsstellen an jedem Pixel besitzt. In Gleichung 2.26 sowie Gleichung 2.27 werden bewusst keine Fallunterscheidungen zur Behandlung dieser Unstetigkeitsstellen vorgesehen. Dadurch sind die Gleichungen eine Approximation der par-

kommt im Zusammenspiel von Gleichung 2.22 und Gleichung 2.27 die Operation $b = a + 1$ zum Einsatz. Wird diese Operation mit Integerzahlen $b = floor(a) + 1$ ausgeführt, wird im Definitionsbereich a zuverlässig um den Betrag 1 inkrementiert. Bei partieller Fließkommarechnung $b = floor(a + 1)$ kann eine Inkrementierung jedoch ausbleiben, wenn a vor der Berechnung bereits nahe einer natürlichen Zahl ist und durch Rundungsfehler nur um den Betrag 0.99999999 inkrementiert wird.

tiellen Ableitung, die direkt auf einer (verschwindend kleinen) Unstetigkeitsstelle eine Steigung zurückliefert, die direkt neben der Unstetigkeitsstelle gilt.

Durch Einsetzen der partiellen Ableitungen aus Gleichung 2.26 und Gleichung 2.27 wird die Jacobi-Matrix

$$J_{\mathcal{T}}[y] \in \mathbb{R}^{mn \times 2mn}$$

$$J_{\mathcal{T}[y]}(\mathcal{A}(\mathcal{P}(w_r))) = \begin{bmatrix} \frac{\partial \mathcal{T}[y]_1}{\partial \mathcal{A}_1} & \frac{\partial \mathcal{T}[y]_1}{\partial \mathcal{A}_2} & \dots & \frac{\partial \mathcal{T}[y]_1}{\partial \mathcal{A}_{2mn}} \\ \frac{\partial \mathcal{T}[y]_2}{\partial \mathcal{A}_1} & \frac{\partial \mathcal{T}[y]_2}{\partial \mathcal{A}_2} & \dots & \frac{\partial \mathcal{T}[y]_2}{\partial \mathcal{A}_{2mn}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{T}[y]_{mn}}{\partial \mathcal{A}_1} & \frac{\partial \mathcal{T}[y]_{mn}}{\partial \mathcal{A}_2} & \dots & \frac{\partial \mathcal{T}[y]_{mn}}{\partial \mathcal{A}_{2mn}} \end{bmatrix}$$

$$= \begin{bmatrix} \Delta_x(1) & 0 & 0 & \dots & 0 & \Delta_y(mn+1) & 0 & 0 & \dots & 0 \\ 0 & \Delta_x(2) & 0 & \dots & 0 & 0 & \Delta_y(mn+2) & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \dots & \Delta_x(mn) & 0 & 0 & 0 & \dots & \Delta_y(2mn) \end{bmatrix} \tag{2.28}$$

bestimmt.[23]

2.7.4 Die Funktion $\mathcal{D}_r(\mathcal{T}[y])$

In Unterabschnitt 2.4.3 wurde als Distanzmaß die Fehlerquadratsumme[24] ausgewählt. Die Berechnung der Fehlerquadratsumme wird in zwei Arbeitsschritte aufgeteilt. Einmal die elementweise Matrixoperation $\mathcal{D}_r = 0.5 \cdot (\mathcal{T}[y] - \mathcal{R})^2$, die hier als Residuumsfunktion bezeichnet werden soll, da diese Funktion das Residuum der Fehlerquadratsumme berechnet.[25] Der zweite Arbeitsschritt ist die im nachfolgenden Abschnitt dargestellte elementweise Aufsummierung des Residuums $\mathcal{D}_s$.

Die von Modersitzki vorgestellte Formel zur Berechnung der Fehlerquadratsumme[26] soll

23 Diese Darstellung stimmt mit [Mod09, S.31] überein.
24 In der englischsprachigen Literatur als „sum of squared differences“ bzw. „SSD“ oder im diskreten Fall „discretized SSD“ bezeichnet. Ausführlich beschrieben in [Mod09, S.72 ff.].
25 Die mathematische Definition des Residuums einer Fehlerquadratsumme findet sich in [NW06, S.245].
26 [Mod09, S.72]

hier mit einem Faktor 0.5 multipliziert werden, wodurch die Matrix

$$\begin{aligned}\mathcal{D}_r :&\mathbb{R}^{mn} \to \mathbb{R}^{mn} \\ \mathcal{D}_r(\mathcal{T}[y](\mathcal{A}(\mathcal{P}(w_r)))) &= \begin{bmatrix} \mathcal{D}_{r1}(\mathcal{T}[y]) \\ \mathcal{D}_{r2}(\mathcal{T}[y]) \\ \vdots \\ \mathcal{D}_{rmn}(\mathcal{T}[y]) \end{bmatrix} = \begin{bmatrix} 0.5 \cdot (\mathcal{T}[y]_1 - \mathcal{R}_1)^2 \\ 0.5 \cdot (\mathcal{T}[y]_2 - \mathcal{R}_2)^2 \\ \vdots \\ 0.5 \cdot (\mathcal{T}[y]_{mn} - \mathcal{R}_{mn})^2 \end{bmatrix}\end{aligned} \tag{2.29}$$

gewonnen wird. Dabei wird die Funktion des Algorithmus als solches nicht beeinträchtigt. Bei der weiter unten beschriebenen Implementierung des Gauß-Newton-Algorithmus kann dieser Faktor bei den Armijo-Line-Search-Kriterien und bei den Stop-Kriterien so berücksichtigt werden, dass keine negativen Auswirkungen entstehen. Der Faktor bewirkt eine numerisch weniger aufwendig zu berechnende Jacobi-Matrix, da sich der Faktor und die Quadrierung bei einer Ableitung aufheben. Die gesuchte Jacobi-Matrix

$$\begin{aligned} J_{\mathcal{D}_r} &\in \mathbb{R}^{mn \times mn} \\ J_{\mathcal{D}_r}(\mathcal{T}[y](\mathcal{A}(\mathcal{P}(w_r)))) &= \begin{bmatrix} \frac{\partial \mathcal{D}_{r1}}{\partial \mathcal{T}[y]_1} & \frac{\partial \mathcal{D}_{r1}}{\partial \mathcal{T}[y]_2} & \cdots & \frac{\partial \mathcal{D}_{r1}}{\partial \mathcal{T}[y]_{mn}} \\ \frac{\partial \mathcal{D}_{r2}}{\partial \mathcal{T}[y]_1} & \frac{\partial \mathcal{D}_{r2}}{\partial \mathcal{T}[y]_2} & \cdots & \frac{\partial \mathcal{D}_{r2}}{\partial \mathcal{T}[y]_{mn}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{D}_{rmn}}{\partial \mathcal{T}[y]_1} & \frac{\partial \mathcal{D}_{rmn}}{\partial \mathcal{T}[y]_2} & \cdots & \frac{\partial \mathcal{D}_{rmn}}{\partial \mathcal{T}[y]_{mn}} \end{bmatrix} \\ &= \begin{bmatrix} \mathcal{T}[y]_1 - \mathcal{R}_1 & 0 & 0 & \cdots & 0 \\ 0 & \mathcal{T}[y]_2 - \mathcal{R}_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \mathcal{T}[y]_{mn} - \mathcal{R}_{mn} \end{bmatrix} \end{aligned} \tag{2.30}$$

wird durch Bildung der partiellen Ableitungen erhalten.

2.7.5 Die Funktion $\mathcal{D}_s(\mathcal{D}_r)$

Das zuvor berechnete Residuum in Matrixform wird durch die Funktion $\mathcal{D}_s$ elementweise aufaddiert. Als Ergebnis wird das gesuchte Distanzmaß

$$\begin{aligned}\mathcal{D}_s :&\mathbb{R}^{mn} \to \mathbb{R} \\ \mathcal{D}_s(\mathcal{D}_r(\mathcal{T}[y](\mathcal{A}(\mathcal{P}(w_r))))) &= \mathcal{D}_s(\mathcal{D}_r) = \mathcal{D}_{r1} + \mathcal{D}_{r2} + ... + \mathcal{D}_{rmn}\end{aligned} \tag{2.31}$$

erhalten. Daraus wird die Jacobi-Matrix

$$J_{\mathcal{D}_s} \in \mathbb{R}^{1\times mn}$$
$$J_{\mathcal{D}_s}(\mathcal{D}_r(\mathcal{T}[y](\mathcal{A}(\mathcal{P}(w_r))))) = \begin{bmatrix} \frac{\partial \mathcal{D}_s}{\partial \mathcal{D}_{r1}} & \frac{\partial \mathcal{D}_s}{\partial \mathcal{D}_{r2}} & \dots & \frac{\partial \mathcal{D}_s}{\partial \mathcal{D}_{rmn}} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}}_{\text{mn Einträge}} \tag{2.32}$$

berechnet.

2.8 Multiplikation der Jacobi-Matrizen

Nach Gleichung 2.9 werden die einzelnen Jacobi-Matrizen miteinander multipliziert, um die gesuchte Jacobi-Matrix $J_{\mathcal{D}(w_r)}$ der Zielfunktion zu erhalten.

$$\bigtriangledown \mathcal{D}(w_r) = \underbrace{J_{\mathcal{D}(w_r)}}_{\mathbb{R}^3} = \underbrace{J_{\mathcal{D}_s}}_{\mathbb{R}^{1\times mn}} \cdot \underbrace{J_{\mathcal{D}_r}}_{\mathbb{R}^{mn\times mn}} \cdot \underbrace{J_{\mathcal{T}}[y]}_{\mathbb{R}^{mn\times 2mn}} \cdot \underbrace{J_{\mathcal{A}}}_{\mathbb{R}^{2mn\times 6}} \cdot \underbrace{J_{\mathcal{P}}}_{\mathbb{R}^{6\times 3}} \tag{2.33}$$

Das Ergebnis der Multiplikation der zuvor erhaltenen Jacobi-Matrizen[27] ist der gesuchte Gradient

$$\bigtriangledown \mathcal{D}(w_r) = J_{\mathcal{D}(w_r)} = \begin{bmatrix} \frac{\partial \mathcal{D}(w_r)}{\partial w_{r1}} & \frac{\partial \mathcal{D}(w_r)}{\partial w_{r2}} & \frac{\partial \mathcal{D}(w_r)}{\partial w_{r3}} \end{bmatrix} \in \mathbb{R}^3 \tag{2.34}$$

der Zielfunktion $\mathcal{D}(w_r)$. Die drei partiellen Ableitungen nach w_{r1}, w_{r2} und w_{r3} zeigen für die Position der aktuellen Registrierungsparameter w_r die Richtungen an, in deren Gegenrichtung der steilste Abstieg zum Minimum der Zielfunktion liegt. Für den affinen Fall gilt

$$\bigtriangledown \mathcal{D}(w_a) = J_{\mathcal{D}(w_a)} = \begin{bmatrix} \frac{\partial \mathcal{D}(w_a)}{\partial w_{a1}} & \frac{\partial \mathcal{D}(w_a)}{\partial w_{a2}} & \frac{\partial \mathcal{D}(w_a)}{\partial w_{a3}} & \frac{\partial \mathcal{D}(w_a)}{\partial w_{a4}} & \frac{\partial \mathcal{D}(w_a)}{\partial w_{a5}} & \frac{\partial \mathcal{D}(w_a)}{\partial w_{a6}} \end{bmatrix} \in \mathbb{R}^6 \tag{2.35}$$

entsprechend.

2.9 Optimierungsverfahren

Wie eingangs geschildert, kommt das Gauß-Newton-Verfahren zum Einsatz. Das Gauß-Newton-Verfahren ist ein Quasi-Newton-Verfahren, welches gültig ist für Optimierungsprobleme bezüglich einer Zielfunktion zur Berechnung einer Fehlerquadratsumme. Für eine ausführliche Beschreibung des Gauß-Newton-Verfahrens sei auf Standardliteratur für numerische Optimierungsverfahren verwiesen.[28] Beim klassischen Newton-Verfahren im

27 Siehe Gleichungen 2.12, 2.16, 2.28, 2.30 sowie 2.32
28 Z.B. [NW06, S.254 ff.] oder [Gil81, S.134 ff.].

Kontext eines Optimierungsproblems einer Funktion $\mathbb{R}^{d>1} \rightarrow \mathbb{R}$ werden deren Ableitung und die Hesse-Matrix benötigt. Beim Gauß-Newton-Verfahren hingegen kann gezeigt werden, dass die Approximation der Hesse-Matrix

$$\bigtriangledown^2 J_{\mathcal{D}_r \mathcal{TAP}} \approx J^{\mathrm{T}}_{\mathcal{D}_r \mathcal{TAP}} J_{\mathcal{D}_r \mathcal{TAP}} \tag{2.36}$$

ausreicht.[29] Sie ist aus der Residuumsfunktion zu berechnen,[30], deren Jacobi-Matrix hier mit $J_{\mathcal{D}_r \mathcal{TAP}}$ bezeichnet wird (siehe Gleichung 3.1 sowie Gleichung 3.3). Es ist pro Iteration das Gleichungssystem

$$J^{\mathrm{T}}_{\mathcal{D}_r \mathcal{TAP}} J_{\mathcal{D}_r \mathcal{TAP}} \cdot \vec{s} = -J_{\mathcal{D}(w_r)} \tag{2.37}$$

zu lösen, um die Suchrichtung $\vec{s}$ zu erhalten. Die nächste Iteration wählt entlang der Suchrichtung anhand der Armijo-Bedingung[31] eine sinnvolle Schrittweite, um den Punkt für die nächste Iteration zu bestimmen. Dabei wird beginnend von der Schrittweite $\frac{1}{2^{i=0}}$ bis zur minimalen Schrittweite $\frac{1}{2^{i=10}}$ eine günstiger Startwert ermittelt. Ist $\frac{1}{2^{i=10}}$ kein ausreichender Startwert, wird ein Fehler angenommen und der Algorithmus wird mit einer Fehlermeldung abgebrochen.

Das iterative Gauß-Newton-Verfahren wird bei Erreichen der folgenden Stop-Kriterien beendet,[32] wenn entweder alle Kriterien aus 1 bis 3 erfüllt sind oder eines aus 4 und 5.

1. Das Distanzmaß hat sich nur noch sehr gering geändert.
2. Der Suchvektor $\vec{s}$ hat nur noch eine sehr geringe Vektorlänge.
3. Ger Gradient $J_{\mathcal{D}(w_r)}$ hat nur noch eine sehr geringe Vektorlänge.
4. Das Distanzmaß hat die Präzisionsgrenze des Rechners unterschritten.
5. Die maximale erlaubte Anzahl an Iterationen wurde überschritten.

Im diesem Kapitel ist eine maßgeschneiderte Formel entstanden, die sich für die algorithmische Implementierung auf einem speicherbegrenzten und parallel arbeitenden Rechner eignet. Prinzipiell ist es möglich, einzelne mathematische Bausteine auszutauschen, wenn es der Anwendungsfall erfordert. Beispielsweise kann anstelle der bilinearen Interpolation auch eine Spline-Interpolation eingesetzt werden. Dazu müsste lediglich ihr Gradient zur Bildung der Jacobi-Matrix $\mathcal{T}[y](\mathcal{A})$ verwendet werden. Es folgt die algorithmische Umsetzung der mathematischen Überlegungen.

29 Eine Herleitung dieser Approximation findet sich in [NW06, S.245-247]
30 Residuumsfunktion siehe Unterabschnitt 2.7.4 Beleg für deren Verwendung siehe [Mod09, S.76].
31 Armijo Line Search siehe [NW06, S.37-41].
32 Kriterien aus [Mod09, S.78], der sie aus [Gil81] entnommen hat.

3 Algorithmische Umsetzung

Um das zuvor entworfene Verfahren auf einem Rechner ausführbar zu machen, sind die mathematischen Überlegungen in einen Computeralgorithmus zu überführen. Dabei ist zu beachten, dass die Berechnung der Zielparameter w innerhalb möglichst geringer Zeit und unter Nutzung der begrenzten Ressourcen, wie Speicher oder Rechenkerne, stattfindet. Insbesondere die Programmbereiche beeinflussen die Geschwindigkeit der Gesamtberechnung, die für jeden Bildpunkt einzeln und damit sehr oft ausgeführt werden (Gleichung 3.1, Gleichung 3.2 und Gleichung 3.3). Diesen performancekritischen Bereichen wird bei der Performanceoptimierung besondere Aufmerksamkeit zuteil.

3.1 Randwertproblem

Die Bilddaten des Referenz- und des Templatebilds verfügen in der Darstellung im Rechner über endliche Dimensionen. Über den Bildrand hinaus sind die Bildpunkte nicht definiert. Benötigt der Algorithmus Bilddaten aus diesem undefinierten Bereich, muss dem Rechner eine Vorschrift vorliegen, welchen Farbwert er zur weiteren Berechnung verwenden soll. Es wird vereinbart, dass in so einem Fall auf den Farbwert 0, also einen Farbwert mit geringster Luminanz, entsprechend der Farbe Schwarz, ausgewichen werden soll. Eine einfache algorithmische Umsetzung dieser Vorschrift wäre, vor jedem vektorisierten Zugriff auf einen Bildpunkt zu überprüfen, ob dessen Position innerhalb der bekannten Bilddaten liegt. Fällt diese Prüfung negativ aus, wird keine Speicherstelle ausgelesen, sondern der Farbwert 0 verwendet.

Dieses Verfahren hat jedoch den Nachteil, dass eine bedingte Verzweigung auf Assembler-Ebene einen Sprungbefehl[1] auslösen kann. Durch einen Sprungbefehl kann es wiederum zu einem Pipelinehemmnis kommen. Die Pipeline eines Mikroprozessors arbeitet Teilschritte aufeinanderfolgender Assemblerkommandos fließbandartig[2] ab: Beispielsweise verarbeitet eine 3-stufige Pipeline drei Assemblerbefehle gleichzeitig. Der neueste Assemblerbefehl wird eingelesen, während der vorhergehende Befehl zeitgleich dekodiert wird und der Befehl hiervor befindet sich bereits in der Ausführung. So kann in jedem Prozessortakt ein Assemblerbefehl beendet werden, obwohl sich die Bearbeitungsdauer

1 Das Versetzen des Programmzählers an eine entfernte Ausführungsposition. Die Arbeitsweise eines Mikroprogramms und des Programmzählers ist beschrieben in [Tan06, S.743 ff.]

2 [Tan06, S.80 ff.]

eines Befehls über drei Takte erstreckt. Ein Pipelinehemmnis entsteht, wenn nicht mehr in jedem Taktzyklus ein Assemblerbefehl beendet werden kann. Dabei wird zwischen betriebsmittel-, daten- und kontrollflussabhängigen Pipelinehemmnissen unterschieden.[3] Eine Betriebsmittelabhängigkeit entsteht bei einem Ressourcenkonflikt, z.B. dem gleichzeitigem Zugriff auf dasselbe Prozessorregister oder denselben Bus. Von einer Datenabhängigkeit spricht man, wenn ein Datum als Eingabeparameter benötigt wird, das jedoch von einem vorhergehenden Befehl noch nicht geliefert wurde. Eine Kontrollflussabhängigkeit tritt bei Sprungbefehlen auf. Werden Assemblerbefehle übersprungen, werden die Pipelineeinträge, die vor dem Sprungkommando liegen, überflüssig. Die Kommandos am Zielort der Sprungfunktion müssen allerdings erst neu in die Pipeline aufgenommen werden, und es dauert einige Taktzyklen (genau so viele Taktzyklen wie die Pipeline Stufen hat), bis wieder ein Befehl vollständig abgearbeitet werden kann. Die praktische Konsequenz einer hohen Anzahl von Pipelinehemmnissen in einem Algorithmus liegt in einer Erhöhung der Berechnungsdauer.

Wird das Randwertproblem gelöst, ohne ein kontrollflussabhängiges Pipelinehemmnis zu erzeugen, steigt folglich die Rechengeschwindigkeit. Für den Algorithmus zur Bildregistrierung kann präzise ermittelt werden, wie weit im größtmöglichen Fall über den Bildrand hinaus der Farbwert eines Pixels eingelesen wird. Zu untersuchen sind Zugriffe auf das Templatebild $\mathcal{T}$ und auf das Referenzbild $\mathcal{R}$. Auf das Referenzbild wird nur lesend durch die Funktion $\mathcal{D}_r$ (s. Unterabschnitt 2.7.4) zugegriffen und zwar genau einmal für jeden vorhandenen Bildpunkt. Für Zugriffe auf das Referenzbild muss demnach nicht kontrolliert werden, ob der Bildrand überschritten wird. Für die Implementierung im Algorithmus resultiert daraus, dass kein if-Statement benötigt wird, was wiederum die Möglichkeit eines kontrollflussabhängigen Pipelinehemmnisses eliminiert.

Auf das Templatebild wird lesend durch die Funktion $\mathcal{T}[y]$ zugegriffen, dabei werden Koordinaten $\mathcal{A}$ verwendet, die außerhalb der bekannten Bilddaten liegen können (s. Abbildung 2.8). Weil die Koordinaten $\mathcal{A}$ lediglich die um die Transformationsparameter w gedrehten und verschobenen Gitterpunkte aller Pixel des Referenzbilds darstellen, kann über eine Anwendung der Transformation y auf die Eckkoordinaten des Referenzbilds der größtmögliche Zugriff über den Bildrand hinaus berechnet werden. In Abbildung 3.1 ist dieser Bereich dargestellt, ferner wird eine Bounding-Box entlang des Koordinatensystems um den größtmöglich lesend zugegriffenen Bereich eingezeichnet.

Umgibt man die Bilddaten des Templatebilds mit einem Rand aus schwarzen Pixeln, dessen Größe der Bounding-Box entspricht (vgl. Abbildung 3.2, wird der gesamte Bildrand dahingehend erweitert, dass ein Lesebefehl ähnlich wie beim Referenzbild niemals einen Pixel außerhalb der definierten Bilddaten adressieren wird. Dadurch kann jeder Lesebefehl auf die Bilddaten unmittelbar ausgeführt werden, ohne zu prüfen, ob der - nun weiter

3 Eine ausführliche Darstellung findet sich in [Bä10, S.68 ff.].

(a) Zugriff über den Rand (b) Bounding-Box

Abbildung 3.1: Bounding-Box für bekannte Transformationsparameter w

entfernte - Bildrand überschritten wurde. Auf diese Weise kann es auch für die Bilddaten des Templatebilds nicht mehr zu einem kontrollflussabhängigen Pipelinehemmnis kommen.

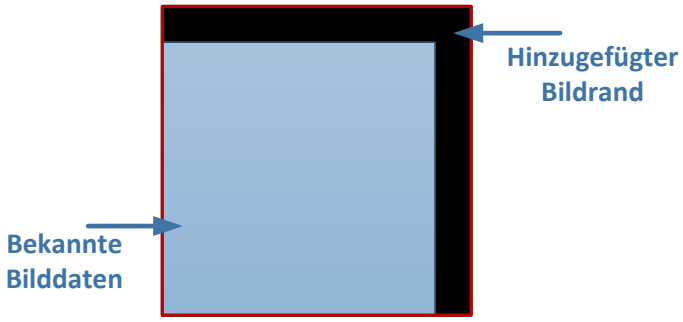


Abbildung 3.2: Künstlicher Rand gegen Pipelinehemmnisse

Generell ist es aus den erläuterten Gründen von Vorteil, jede Form von Verzweigung in dem performancekritischen Teil des Algorithmus, der pro Pixel ausgeführt wird, zu vermeiden.

Geeignete Bilddaten

Das hier vorgestellte mathematische Verfahren lässt sich auf beliebige zweidimensionale Bilddaten anwenden, solange sie in Graustufen vorliegen. Dabei ist zu beachten, dass nicht jedes Registrierungsproblem lösbar ist. Es wurde beobachtet, dass Kombinationen aus Referenz- und Templatebild existieren, für die der Algorithmus nicht zum optimalen Distanzmaß konvergiert, z.B. weil er in einem lokalen Minimum gefangen ist. Zur Vorbeugung derartiger Probleme, können die Bilddaten vor der Registrierung einem Gauß- oder Mittelwertfilter unterzogen werden, um unbedeutende Bildkanten zu eliminieren. Außerdem werden gute Ergebnisse erzielt, wenn eine Farbinvertierung vorgenommen wird, falls die Bildränder heller sein sollten als der Bildinhalt. Dies ist darauf zurückzuführen, dass außerhalb des Bildbereichs liegende Pixel mit der Farbe Schwarz definiert sind. Wäre der Bildrand hell, würde bei der Drehung eine unerwünscht starke Kante von der Farbe Weiß

zur Farbe Schwarz in das Bild aufgenommen werden, welche die Registrierung negativ beeinflussen würde.

3.2 Gradientenbildung ohne LSI-Filter

Die spärlich besetzte Jacobi-Matrix $J_{\mathcal{T}[y]}(\mathcal{A})$ wird in Unterabschnitt 2.7.3 per analytischer (partieller) Ableitung berechnet. Dabei enthalten die beiden besetzten Diagonalen jeweils die Bildgradienten in X- und in Y-Richtung des transformierten Templatebilds (vgl. Gleichung 2.28). Für die algorithmische Berechnung dieser auf die Matrixdiagonalen abzulegenden Bildgradienten wäre die Anwendung von LSI-Filternn möglich, da die Verwendung eines LSI-Filters ein Standardverfahren zur Berechnung eines Bildgradienten durch eine diskrete Faltung ist[4]. Die dabei benötigten Faltungsoperatoren müssten zur Laufzeit entsprechend den Registrierungsparametern anzupassen sein, denn die Ausrichtung der Ableitungsbildung unterscheidet sich bei $w_1 \neq 0°$ von der Ausrichtung des Koordinatensystems. Soll stattdessen ein vordefinierter Faltungsoperator verwendet werden, so ist nicht er an die Transformationsparameter w anzupassen, sondern der Gradient von $\mathcal{T}$ **vor** der Transformation parallel zu den Koordinatenachsen zu berechnen. Die aus der horizontalen und der vertikalen Ableitung resultierenden Bilder müssten dann allerdings später transformiert und interpoliert werden, um sie in die Jacobi-Matrix $J_{\mathcal{T}[y]}(\mathcal{A})$ kopieren zu können, was einen hohen Rechenaufwand bedeutet.

Die Verwendung eines LSI-Filters ist somit zumindest ähnlich aufwendig, wie eine analytische Bildung der Jacobi-Matrix, was deutlich wird, wenn man die geringe Anzahl der mathematischen Grundoperationen der Gleichungen 2.27, 2.26 sowie 2.20 betrachtet. Weil die Filteroperatoren auf interpolierte Daten zugreifen würden, entstünde zudem ein Präzisionsverlust mit der Gefahr einer schlechteren Konvergenzrate. Deswegen wird für die algorithmische Implementierung der analytischen Ableitung der Vorzug gegeben; LSI-Filter kommen nicht zum Einsatz. Im folgenden Abschnitt wird gezeigt werden, dass bei einer pixelbasierten Verarbeitung ein explizites Berechnen der Jacobi-Matrix $J_{\mathcal{T}[y]}(\mathcal{A})$ ohnehin unvorteilhaft ist.

3.3 Reduktion von Speicherzugriffen

Moderne Prozessoren rechnen schneller, als Daten von/zum Hauptspeicher transferiert werden können.[5] Insbesondere bei großen Bilddaten, welche die Speicherkapazität des

4 Linear-Shifting-Invariant Filter und Faltungsoperatoren werden ausführlich beschrieben in [Jä93, Kapitel 4.2]

5 "... Speicher viel langsamer sind als CPUs, was sich - relativ gesehen - jedes Jahr verschlimmert."[Tan06, S.318]

L2-Cache überschreiten, bilden die Speicherzugriffe einen entscheidenden Faktor für die Rechengeschwindigkeit. Auf die Besonderheiten des L2-Caches und L3-Speichers wird in Abschnitt 5.4.2 näher eingegangen. Vorweg sei gesagt, dass eine Reduktion von Speicherzugriffen die Ausführungsgeschwindigkeit eines Algorithmus erhöht, da ansonsten viele Speichertransfers zwischen L2-Cache und L3-Speicher erfolgen müssten.

Besonders viele Speicherzugriffe entstehen bei der Berechnung auf Grundlage von großen Matrizen. In der vorhergehenden mathematischen Darstellung kommt es zu Jacobi-Matrizen bis zur Größenordnung $\mathbb{R}^{mn \times 2mn}$ für die Matrix $J_{\mathcal{T}}[y]$. Bei einem Bild der Dimension 3000x3000 Pixel mit einem Byte Speicherverbrauch pro Pixel würde eine derartige Matrix ca. 147 Terabyte Speicher verbrauchen. Berücksichtigt man, dass es sich um eine, wie in Gleichung 2.28 gezeigte, spärlich besetzte Matrix mit zwei besetzten Diagonalen der Länge mn handelt, bleibt immer noch ein Speicherverbrauch von 17,17 Megabyte. Alleine für das Anlegen und wieder Auslesen dieser Matrix würden 36 Millionen Speicherzugriffe erforderlich werden.

3.3.1 Matrixfreie Berechnung pro Pixel

Die vorangegangene, noch unveröffentlichte Studie der Fachbetreuer Lars König und Jan Rühaak hat für einen zwar komplexeren, aber ähnlichen Algorithmus[6] ergeben, dass auf Matrizenoperationen verzichtet werden kann, wenn die in Gleichung 2.33 dargestellte Multiplikation analytisch vorberechnet wird. Ohne die konkreten Jacobi-Matrizen bereits zu kennen, kann eine Berechnungsvorschrift erhalten werden, welche pro Pixel auszuführen ist. Das Vorhalten von Zwischenschritten in Matrixform ist nicht erforderlich. Obwohl hier dieselbe Formel auf jeden Pixel angewandt wird handelt es sich nicht um einen LSI-Filter, da zwei verschiedene Bilder in die Formel einfließen und weil als Ergebnis keine Bilddaten erhalten werden.

Um Gleichung 2.33 in eine solche pro Pixel ausführbare Form zu bringen, wird eine Berechnungsreihenfolge vorgeschlagen, die sich günstig auf die Parallelisierung auswirkt. Es wird zuerst $J_{\mathcal{D}_r\mathcal{T}\mathcal{A}} = J_{\mathcal{D}_r} \cdot J_{\mathcal{T}}[y] \cdot J_{\mathcal{A}}$, danach $J_{\mathcal{D}_r\mathcal{T}\mathcal{A}\mathcal{P}} = J_{\mathcal{D}_r\mathcal{T}\mathcal{A}} \cdot J_{\mathcal{P}}$ (nur notwendig bei rigider Registrierung) berechnet. Erst als letzter Schritt erfolgt die Berechnung von $J_{\mathcal{D}(w_r)} = J_{\mathcal{D}_s} \cdot J_{\mathcal{D}_r\mathcal{T}\mathcal{A}\mathcal{P}}$.

6 Bildregistrierung auf 3D-Daten mit Spline-Interpolation und NGF-Distanzmaß

Draus resultieren die Gleichungen

$$\underbrace{J_{\mathcal{D}(w_r)}}_{\mathbb{R}^3} = \underbrace{J_{\mathcal{D}_s}}_{\mathbb{R}^{1\times mn}} \cdot \underbrace{J_{\mathcal{D}_r\mathcal{TA}}}_{\mathbb{R}^{mn\times 6}} \cdot \underbrace{J_{\mathcal{P}}}_{\mathbb{R}^{6\times 3}}$$

$$J_{\mathcal{D}_r\mathcal{TA}} = \begin{bmatrix} r_1\Delta_{x1}X_1 & r_1\Delta_{x1}X_{mn+1} & r_1\Delta_{x1} & r_1\Delta_{y1}X_1 & r_1\Delta_{y1}X_{mn+1} & r_1\Delta_{y1} \\ r_2\Delta_{x2}X_2 & r_2\Delta_{x2}X_{mn+2} & r_2\Delta_{x2} & r_2\Delta_{y2}X_2 & r_2\Delta_{y2}X_{mn+2} & r_2\Delta_{y2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ r_{mn}\Delta_{xmn}X_{mn} & r_{mn}\Delta_{xmn}X_{2mn} & r_{mn}\Delta_{xmn} & r_{mn}\Delta_{ymn}X_{mn} & r_{mn}\Delta_{ymn}X_{2mn} & r_{mn}\Delta_{ymn} \end{bmatrix} \tag{3.1}$$

$$J_{\mathcal{D}_r\mathcal{TAP}} = \begin{bmatrix} \psi_1 & J_{\mathcal{D}_r\mathcal{TA}}(1,3) & J_{\mathcal{D}_r\mathcal{TA}}(1,6) \\ \psi_2 & J_{\mathcal{D}_r\mathcal{TA}}(2,3) & J_{\mathcal{D}_r\mathcal{TA}}(2,6) \\ \vdots & \vdots & \vdots \\ \psi_{mn} & J_{\mathcal{D}_r\mathcal{TA}}(mn,3) & J_{\mathcal{D}_r\mathcal{TA}}(mn,6) \end{bmatrix} \tag{3.2}$$

$$\begin{aligned} \psi_i = &(-sin(w_{r1}) \cdot J_{\mathcal{D}_r\mathcal{TA}}[i,1]) \cdot (-cos(w_{r1}) \cdot J_{\mathcal{D}_r\mathcal{TA}}[i,2]) \\ &\cdot (cos(w_{r1}) \cdot J_{\mathcal{D}_r\mathcal{TA}}[i,4]) \cdot (-sin(w_{r1}) \cdot J_{\mathcal{D}_r\mathcal{TA}}[i,5]) \end{aligned} \tag{3.3}$$

wobei $J_{\mathcal{D}_r\mathcal{TA}}(i,k)$ den Zugriff auf die Matrix $J_{\mathcal{D}_r\mathcal{TA}}$ in Zeile i und Spalte k bezeichnet. Bei einer affinen Registrierung entfällt die Multiplikation mit $J_{\mathcal{P}}$. Die drei Spalten von $J_{\mathcal{D}_r\mathcal{TAP}}$ (bzw. im affinen Fall die sechs Spalten von $J_{\mathcal{D}_r\mathcal{TA}}$) werden zur Approximation der Hesse-Matrix verwendet. Ferner werden diese Spalten einzeln aufsummiert, um $J_{\mathcal{D}(w_r)}$ zu erhalten, denn eine spaltenweise Aufsummation entspricht der Multiplikation mit $J_{\mathcal{D}_s}$ (sieheGleichung 2.32). Auch dieser Vorgang kann pro Pixel ausgeführt werden, das Ergebnis kumuliert sich dann von Pixel zu Pixel auf. In Anhang A.4 ist eine Matlab-Implementierung des so erhaltenen Algorithmus zu finden. Über eine doppelte For-Schleife wird horizontal- und vertikal auf jeden Pixel zugegriffen, um die Gleichungen 3.1, 3.2 sowie 3.3 numerisch zu lösen. Es ist gut erkennbar, dass auf Zwischenschritte in Matrixform komplett verzichtet werden kann.

Die Nicht-Anwendbarkeit des Kommutativgesetzes bei den verwendeten Matrizen bleibt bei diesem Verfahren berücksichtigt. Die Komplexität des so erhaltenen Algorithmus entspricht $\mathcal{O}(n)$[7] bezogen auf die Anzahl der Pixel mn für die einmalige Berechnung von Distanzmaß, Jacobi- und Hessematrix. Die Gleichung 3.1 ist vollständig parallelisierbar, theoretisch könnte jeder Einzelne Eintrag in $J_{\mathcal{D}_r\mathcal{TA}}$ parallel berechnet werden. Dies ist ein optimaler Ausgangspunkt für die Implementierung auf einer parallelen DSP-Architektur.

7 Das O-Kalkül wird beschrieben unter [Hof09, S.340 ff.]

3.4 Parallelisierung

Der in Kapitel 2 entworfene mathematische Algorithmus ist bereits auf eine einfache Parallelisierbarkeit hin entwickelt. Die Mathematik ist innerhalb einer Iteration pro Pixel parallel ausführbar. Für eine Implementierung auf mehreren DSP-Chips ist eine zweistufige Hierarchie zu beachten. Die erste Parallelisierungshierarchie unterteilt die Daten und die Rechenverantwortung an die einzelnen DSP-Chips. In der zweiten Parallelisierungshierarchie werden pro Chip die Berechnungen mittels der Rechenkerne weiter unterteilt. Beispielweise würde eine Verteilung an vier DSP-Chips à acht Rechenkerne auf eine Verteilung an $4 \cdot 8 = 32$ Recheneinheiten hinauslaufen.

3.4.1 Parallelisierung über mehrere DSP-Chips

Jeder DSP-Chip verfügt über einen privaten Speicherbereich, auf den die einzelnen Rechenkerne gemeinsam zugreifen können. Um die Anforderungen an RAM-Speicher und Netzwerkbandbreite zu reduzieren, ist es von Vorteil, lediglich eine Mindestmenge an Bilddaten auf die DSP-Chips zu verteilen. Zu diesem Zweck wird für die rigide Bildregistrierung ein Verfahren vorgestellt, welches irrelevante Teile der Bilddaten von der Übertragung ausschließt sowie Bilddaten nur einmal für alle Iterationen überträgt. Bei den folgenden Iterationen werden die bereits übermittelten Daten wiederverwendet und für die Bildregistrierung basierend auf dem Referenzbild so eingeteilt, dass die Gesamtfläche in möglichst quadratische und gleich große Bereiche unterteilt wird. Stehen etwa vier DSPs zur Verfügung und sind die Bilddaten quadratisch,[8] so kann der Bildbereich symmetrisch in vier Quadrate aufgeteilt werden, wie Abbildung 3.3 verdeutlicht.

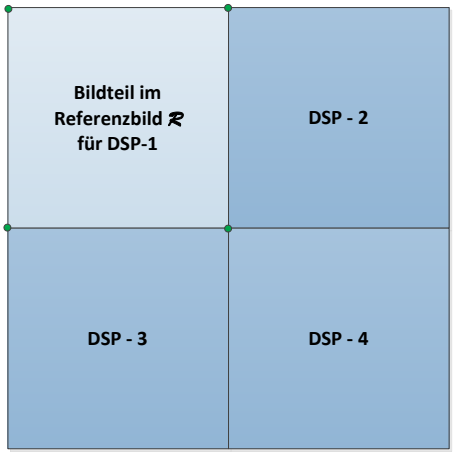


Abbildung 3.3: Verantwortlichkeit des ersten DSP bezogen auf das Referenzbild

8 Dies wird später bei der Performancemessung im Testaufbau der Fall sein.

Die vier grün gekennzeichneten Eckkoordinaten markieren den Verantwortungsbereich des ersten DSP-Chips bezogen auf das Referenzbild. Die quadratische Fläche zwischen diesen Koordinaten wird an ihn übertragen, wofür keine weiteren Daten des Referenzbilds benötigt werden. Der Bereich im Templatebild, der während der Registrierung mit dem Referenzbild verglichen wird, ist anhand der Registrierungsparameter w gedreht und verschoben. Transformiert man die in Abbildung 3.3 grün markierten Koordinaten mithilfe der Registrierungsparameter w, erhält man jene Fläche im Templatebild, die mit dem Referenzbild verglichen wird, um das Distanzmaß zu erhalten. Für eine Iteration (bei bekanntem w) könnten auf diese Weise die Bilddaten der in Abbildung 3.4 abgebildeten Fläche des Templatebilds vom DSP-Chip benötigt werden.

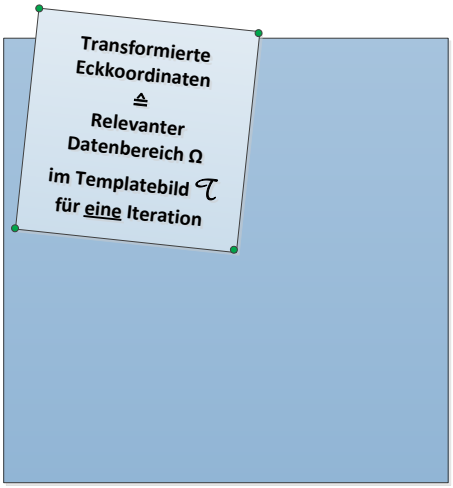


Abbildung 3.4: Transformierte Eckkoordinaten definieren die vom Templatebild benötigten Daten

Indem eine Bildregistrierung über mehrere Iterationen hinweg ausgeführt wird, muss dieser Bereich erweitert werden, damit nicht zwischen jeder Iteration neue Bilddaten des Templatebilds an den DSP zu übertragen sind. Ein mögliches Verfahren für die Erweiterung dieses Bildbereichs zur Verwendung in mehreren Iterationen wird wie folgt vorgeschlagen:

1. Auf den aktuellen Rotationswinkel der Registrierungsparameter w wird ein Worst-Case-Offset aufgeschlagen und die Eckkoordinaten damit berechnet.
2. In gleicher Weise wird ein Worst-Case-Offset subtrahiert; die daraus resultierenden Eckkoordinaten werden berechnet.
3. Liegen die Winkel 45○, 135○, 225○ oder 315○ innerhalb der Worst-Case-Winkel, werden auch deren Eckkoordinaten berechnet, weil es sich um lokale Maxima bezüglich der Bildausmaße handelt.
4. Aus den so berechneten Eckkoordinaten werden jeweils die äußersten (von der Mitte des relevanten Bildviertels gesehen) ausgewählt.

5. Hat man so die maximal durch Rotation zu berücksichtigenden Bilddaten ausgewählt, wird noch die maximal zu erwartende Translation (relativ zu den aktuellen in w) als Radius um die Koordinaten aufgeschlagen.

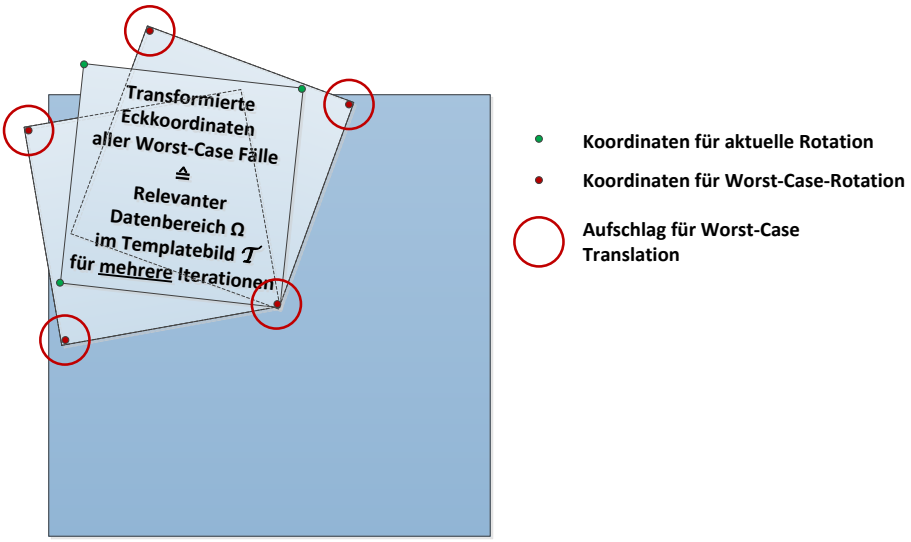


Abbildung 3.5: Für mehrere Iterationen wird eine Worst-Case-Annahme getroffen

In Abbildung 3.5 ist das Verfahren in einer Skizze dargestellt. Die grünen Punkte kennzeichnen die Startparameter w, die bei der ersten (bzw. nächsten) Iteration den relevanten Bildbereich markieren. Die roten Punkte kennzeichnen die u.A. durch Rotation entstandenen Worst-Case-Eckkoordinaten. Die roten Kreise um die ebenfalls roten Eckkoordinaten bezeichnen den Aufschlag, der durch die Worst-Case-Translation zu erwarten ist. Als Ergebnis kann über eine Bounding-Box der Bereich des Templatebilds ermittelt werden, der an den entsprechenden DSP zu übertragen ist (vgl. Abbildung 3.6). Können die maximalen Registrierungsparameter gut eingegrenzt werden, ist eine Reduktion der insgesamt zu übertragenden Datenmenge sowie des Speicherbedarfs auf etwa die Hälfte möglich.[9]

Auf diese Weise brauchen die Bilddaten des Templatebilds nur partiell und lediglich einmal an den DSP übertragen zu werden. Zwischen jeder Iteration ist zu prüfen, ob die Registrierungsparameter w noch im erlaubten Bereich liegen. Kommt es dazu, dass die Registrierungsparameter der nächsten Iteration den zulässigen Rotations- oder Translationsoffset überschreiten, wird anhand der aktuellen Parameter w eine neue Worst-Case-Bounding-Box berechnet und deren Bilddaten an den DSP-Chip übertragen. Die Worst-Case-Offsets für Rotation und Translation sind abhängig vom Anwendungsfall so zu wählen, dass eine derartige Mehrfachübertragung nicht zu erwarten ist.

Es sei noch erwähnt, dass die Bounding-Box für die bikubische Interpolation im Registrierungsalgorithmus um einen Pixel pro Rand erweitert wird. Ferner wird sie auf den Bereich

9 In einer Matlab-Simulation konnte eine Datenreduktion auf 53 % beobachtet werden, wenn maximal 10° Rotation und maximal 10 % Translation erlaubt sind.

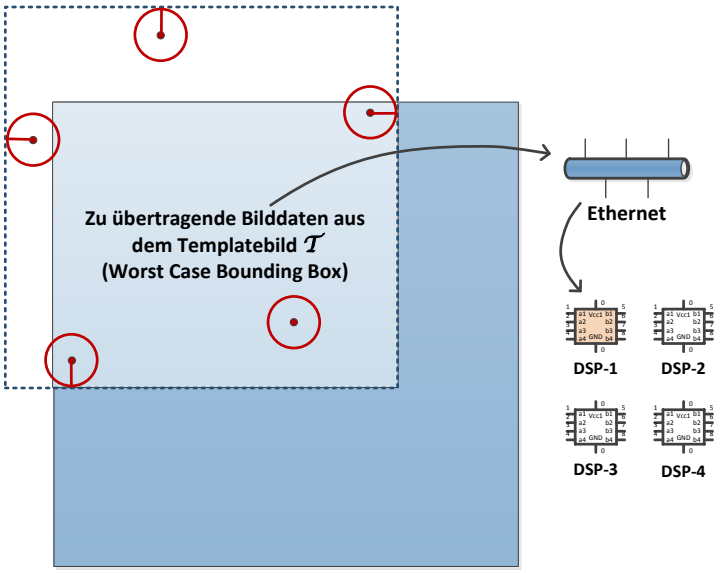


Abbildung 3.6: Bounding-Box zur Reduktion der Datenübertragung

an verfügbaren Bilddaten geclipped, wie am linken sowie am oberen Rand des Beispiels in Abbildung 3.6. Außerdem werden die hier berechneten Worst-Case-Koordinaten für die in Abschnitt 3.1 dargestellte Technik gegen Pipelinehemmnisse wiederverwendet. Beim Empfang der Bilddaten wird im Speicher des DSP automatisch ein schwarzer Bildrand um die Bilddaten des Templatebilds hinzugefügt, welcher der Worst-Case-Bounding-Box (ohne Clipping) entspricht.

3.4.2 Parallelisierung pro DSP-Chip an seine Rechenkerne

Ist ein DSP mit Bilddaten versorgt, können die Rechenkerne (beim TI C6678 acht Stück) unabhängig von den Nachbarkernen die Berechnung einer Registrierungsiteration durchführen. Dafür werden die zu berechnenden Bildbereiche logisch auf die Rechenkerne verteilt. Jeder Rechenkern hat lesenden Zugriff auf alle Bilddaten des DSP-Chips. Das Ergebnis wird pro Rechenkern in einen privaten Speicherbereich geschrieben und die Bildbereiche werden den Rechenkernen anhand einer vertikalen Aufteilung zugewiesen, damit Cache-Misses vermieden werden. In Abbildung 3.7 befindet sich eine derartige Aufteilung, die beim zeilenweisen Zugriff auf das Referenzbild minimale Cache-Misses zur Folge hat. Bei einer sequenziellen Verarbeitung, die Byte pro Byte immer auf die jeweils folgende Speicherstelle zugreift, ist die Auslastung der Caches und Pipelines im DSP ideal.

Das Ergebnis jeder parallelen Recheninstanz ist ein auf den Bildteil bezogenes Distanzmaß ($\mathbb{R}^1$), ein Gradient ($\mathbb{R}^3$) und das zur Approximierung der Hesse-Matrix notwendige Produkt aus Jacobi-Matrizen ($\mathbb{R}^{3\times3}$). Diese Teilergebnisse der Rechenkerne werden zunächst auf jedem DSP-Chip aufsummiert, anschließend von einer zentralen Recheninstanz

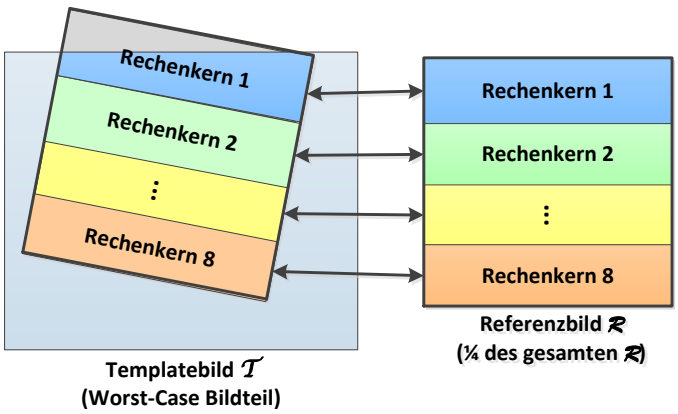


Abbildung 3.7: Parallele Berechnung auf acht Rechenkernen über ein Viertel des Referenzbilds

gesammelt und zum Ergebnis einer Iteration aufsummiert.

Nach Klärung der algorithmischen Vorgehensweise wird im folgenden Kapitel der Hardwareaufbau für die algorithmische Implementierung vorgestellt.

4 Hardwareaufbau

Der zuvor beschriebene Algorithmus wird für die messtechnische Untersuchung auf einer Architektur aus parallel arbeitenden DSPs ausgeführt.

4.1 DSP

DSPs sind Mikroprozessoren für die spezifischen Anforderungen der digitalen Signalverarbeitung. Sie sind gekennzeichnet durch:[1]

- spezielle Rechenwerke (insbesondere Multiplizier-Akkumuierwerke) mit großen Registersätzen
- mehrfache, komplexe Adressrechenwerke
- vielfache Bussysteme
- besondere Steuereinheiten (z.B. zur Schleifensteuerung in Hardware)

Diese Besonderheiten ermöglichen eine hochperformante Algorithmenausführung in der digitalen Signalverabeitung. Insbesondere die Multiplizier-Akkumulierwerke beschleunigen die in der digitalen Signalverarbeitung häufige Multiply-Accumulate-Operation $a \leftarrow a + (b \cdot c)$.

4.1.1 Hochleistungs-DSP

Bei Hochleistungs-DSPs wird die Performance durch weitere Maßnahmen gesteigert:[2]

- superskalare und VLIW-DSPs
- Multiprozessor-Kopplung von DSPs (häufig über 100 parallele DSPs)[3]
- SIMD (Single Instruction Multiple Data) Befehlssätze

1 Vorstehende Definition sowie die folgenden Listeneinträge sind entnommen aus [Bä10, S.49]
2 Die ersten beiden Einträge sind entnommen aus [Bä10, S.436], der letzte Eintrag folgt aus dem Datenblatt [SPR11f]
3 [Bä10, S.436, „viele - oft über 100 - DSPs zu einem Mehrprozessor-System miteinander zu verbinden“]

Ein **superskalarer DSP** ist ein DSP, der mehrere Befehle gleichzeitig in einem Taktzyklus an die Ausführungseinheiten des Prozessorbausteins übergeben kann. So könnten beispielsweise eine Integer- sowie eine Gleitkommaberechnung und gleichzeitig ein Speicherzugriff parallel abgearbeitet werden.[4]

Bei der **VLIW**-Technologie (Very Long Instruction Word) werden - im Gegensatz zu den bekannteren Prozessorarchitekturen RISC (Reduced Instruction Set Computer) und CISC (Complex Instruction Set Computer) - mehrere Instruktionen in eine einzelne Prozessoroperation zusammengefügt. Alle Befehle dieser Operation werden vom DSP parallel ausgeführt. Wobei unterschiedliche Befehle gemischt verwendet werden können. Der Compiler ist verantwortlich dafür, dass nur solche Befehle in einem VLIW zusammengefasst werden, die ohne negative Konsequenzen parallelisierbar sind.[5]

Verfügt ein Prozessor über **SIMD**-Instruktionen (Single Instruction-stream Multiple Datastream[6]), so besteht die Möglichkeit, einen identische Befehlsstrom auf unterschiedliche Speicherstellen gleichzeitig ausführen[7], z.B. elementweises Multiplizieren eines ganzen Vektors.

Über die **Mutiprozessor-Kopplung** wird eine Parallelisierung über die Grenzen eines DSP-Bausteins hinweg erreicht, hierbei bearbeiten mehrere eigenständige DSP-Bausteine dieselbe Aufgabe.[8]

4.1.2 Vergleich zu GPU-Prozessoren

Für HPC-Anwendungen (High-Performance-Computing) werden zunehmend GPUs (Graphics Processing Unit) eingesetzt.[9] Eine GPU ist ein Prozessor, der für Berechnungen digitaler Computergrafik optimiert ist. Da solche Berechnungen im Bereich der dreidimensionalen Grafik sehr aufwendig sind, verfügen GPUs über eine sehr hohe Rechenperformance. Über herstellerspezifische Schnittstellen können beliebige mathematische Algorithmen auf einer GPU ausgeführt werden.[10] Aktuelle GPU Prozessoren haben laut Texas Instruments (TI) eine höhere Rechenperformance als C6678-DSPs.[11] Dem stehen jedoch folgende Vorteile der DSPs gegenüber:

- Ein DSP wird mit gewöhnlichem C/C++ Quellcode programmiert[12]. Der Quellcode eines GPU-Programms unterscheidet sich dagegen deutlich, da der Algorithmus an

4 [Tan06, S.583]
5 [DS98, 383]
6 [Tan06, S.84]
7 [Bä10, S.440]
8 [Bä10, S.436]
9 [Sti12]
10 Z.B. mittels der CUDA Schnittstelle der Firma NVIDIA [NVI]
11 [Sab11]
12 Dies wurde in der vorliegenden Studie erfolgreich nachvollzogen.

die Schnittstelle CUDA[13] angepasst werden muss.

- Ein DSP ist dafür entwickelt, um in ein produktspezifisches PCB (Printed Circuit Board) Layout integriert zu werden. Eine GPU wird in der Regel als fertige Erweiterungskarte bezogen.[14]
- Ein DSP ist dafür geeignet, auch als Hauptprozessor zu fungieren.[15]
- Für DSPs gibt es frei erhältliche Hardware-Referenzdesigns.[16]
- Die Datenblätter eines DSP beschreiben sehr detaillierte Aspekte für die Hardware-, Systemsoftware- und Anwendungsentwicklung.[17]

4.1.3 Raum- und Energieanforderungen

Ein Vergleich zwischen DSPs und GPUs ist von Texas Instruments unter [Sab11] veröffentlicht. Darin wird der Schluss gezogen, DSP-Technik sei GPUs bezüglich der Rechenperformance unter-, aber in puncto Raum- sowie Energiebedarf überlegen. Es stellt sich die Frage, ob diese Überlegenheit der DSPs auch einem Vergleich mit der PC-Technik stand hält.

Die bestehende Hauptplatine eines Medizingerätes muss zur Aufnahme zusätzlicher DSP-Chips nur um wenige Zentimeter vergrößert werden. Die in dieser Studie betrachteten PCs auf Intel[18]-Basis sind dagegen erheblich größer. Es gibt Industrie-PCs[19] in unterschiedlichen Ausmaßen, dabei sind sie immer größer, als die Fläche eines oder weniger DSP-Chips, weil insbesondere bei Hochleistungsrechnern eine komplexe Platine mit umfangreicher Infrastruktur notwendig ist. In Abbildung 4.1 wird dieser Unterschied optisch ersichtlich. Dargestellt ist das DSP Modell C6678 von Texas Instruments sowie ein Industrie-PC des

13 [NVI]

14 Eine Recherche bei einem der weltgrößten Lieferaten für Elektronikbauteile DigiKey (www.digikey.com) belegt diese Einschätzung. C6678-DSPs sind in einer Vielzahl an elektronischen Bauformen erhältlich. NVIDIA GPU Bauteile sind nicht erhältlich, vermutlich weil diese nur fest verbaut in Erweiterungskarten erhältlich sind.

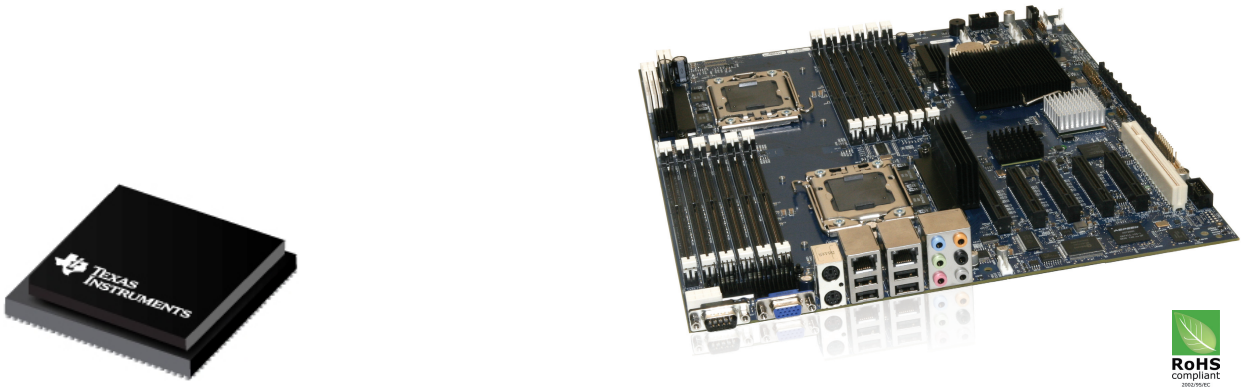
15 Die vorliegende Arbeit beweist, dass ein DSP-Baustein wie in Abschnitt 5.4 dargestellt ein eigenständiges Betriebssystem samt Anwenderprogramm auf dem C6678-Evaluierungsmodul ausführen kann, ohne dass ein weiterer Mikroprozessor als Hauptprozessor fungiert.

16 Beispielsweise das Referenzdesign für den in dieser Studie verwendeten C6678-DSP unter [SNV12]

17 Auf der Begleit-CD sowie in der Bibliographie finden sich eine Fülle von umfangreichen und detaillierten Datenblättern, die für den C6678-DSP vom Hersteller zur Verfügung gestellt werden. Das gerade einmal zwei Seiten starke Datenblatt für die GPU 'Tesla M2090' von NVIDIA belegt beispielhaft den Unterschied.[NVI11]

18 Hersteller von PC-Prozessoren, Firmenhomepage: www.intel.de

19 Wegen der medizinischen Zulassungen sowie der Notwendigkeit einer langjährigen Verfügbarkeit von Neu- und Ersatzteilen können in Medizingeräten keine Konsumenten-PCs verbaut werden, es werden Industrie-PCs verwendet.



(a) TI C6678 (24x24 mm) (b) Kontron KTC5520-EATX (305x330 mm)

Abbildung 4.1: Optischer Vergleich zwischen DSP-Chip und PC-Platine

renommierten Herstellers Kontron[20] mit zwei 6-Kern CPUs von Intel.[21] Ein ähnliches Hochleistungssystem mit insgesamt 12 PC-Rechenkernen wird später für den Performancevergleich herangezogen werden. Es werden auch Industrie-PCs mit kleineren Ausmaßen angeboten[22], doch ist vor der Auswahl zu prüfen, ob ein System mit geringen Ausmaßen nicht auch über eine geringere Rechenleistung verfügt.

Weiterhin bemerkenswert ist der geringe Energieverbrauch der DSPs im Gegensatz zu alternativen Lösungen mit PC- oder GPU-Technik, die einen wesentlich höheren Energieverbrauch aufweisen. Ein Vergleich der schnellsten, in der vorliegenden Arbeit verwendeten Vergleichs-CPU mit dem C6678-DSP, sowie einem aktuellen GPU-Prozessor ist in Tabelle 4.1 wiedergegeben. Dabei wirkt sich ein hoher Energieverbrauch auch auf den Raumbedarf aus, da ein Mehr an Energieaufnahme auch zu mehr Abwärme sowie einer aufwendigeren Stromversorgung führt. Schlimmstenfalls ist bei sehr hohem Energieverbrauch eine Klimaanlage im medizinischen Behandlungsraum zu installieren, wenn die Abwärme die Raumtemperatur zu stark erhöht.

Typ	Hersteller	Beispielmodell	Energieverbrauch
GPU	NVIDIA	Tesla M2090	225 Watt[1]
CPU	Intel	E5645	80 Watt[2]
DSP	TI	C6678	10 Watt[3]

[1] [Hei]
[2] [Int]
[3] [Tex12] sowie „delivering [...] in just 10W“ [SPR11g]

Tabelle 4.1: Vergleich im Energieverbrauch zwischen CPU, GPU und DSP

20 www.kontron.de
21 Bilder und Maße mit freundlicher Genehmigung der Hersteller entnommen aus [Kon12], [SPR11f] sowie [Texe].
22 Beispielsweise der 160x160x25 mm große ’fit-PC 3’ [Com]

DSP-basierte Lösungen sind somit prädestiniert für energieeffiziente, kleine oder gar tragbare Geräte.

4.2 Der C6678-DSP von Texas Instruments

Zur Erforschung des Einsatzes von Multicore-DSPs in der medizinischen Bildregistrierung eignet sich der C6678-DSP von Texas Instruments, denn es handelt sich dabei um einen hoch getakteten (1,25 GHz), superskalaren Hochleistungs-DSP mit Mehrkerntechnologie und VLIW- sowie SIMD-Befehlssatz.[23] Dank der Mehrkerntechnologie arbeiten acht unabhängige DSP-Rechenkerne parallel auf derselben integrierten Schaltung ($N_k = 8$), wobei sich die einzelnen Rechenkerne den Speicher und die Peripherie teilen. Infolge der VLIW-Technologie kann jeder Rechenkern pro Taktzyklus bis zu $N_o = 8$ Operationen gleichzeitig ausführen. Verwendet man Multiply-Accumulate Operationen, sind das je zwei Fließkommaberechnungen pro Operation ($N_{fpo} = 2$). Bei einer Taktrate von $f = 1,25GHz$ kommt man auf die im Datenblatt von Texas Instruments angegebene Maximalperformance v_{max}

$$\begin{aligned} v_{max} &= N_k \cdot N_o \cdot N_{fpo} \cdot f \\ v_{max} &= 8 \cdot 8 \cdot 2 \cdot 1,25GHz \\ v_{max} &= 160GFLOPS^{24}proDSP \end{aligned} \tag{4.1}$$

für 32-Bit-Fließkomma-Arithmetik.[25] In Tabelle 4.2 findet sich eine Übersicht über weitere technische Daten des C6678-DSP.[26]

Taktgeschwindigkeit	1,25 GHz
Rechenkerne	8
SIMD-Vektorgröße	128 Bit
L1 Programmspeicher (SRAM/Cache)	256 KB
L1 Datenspeicher (SRAM/Cache)	256 KB
L2 Cache (vereinheitlicht)	4096 KB
Speichercontroller	maximal 8 GB DDR3 RAM
Bauteildimensionen	24 mm x 24 mm

Tabelle 4.2: Kennwerte des C6678-DSP von TI

23 Spezifikation siehe [SPR11f]
24 GFLOPS = Milliarden Fließkomma-Instruktionen pro Sekunde
25 „20 GFLOP/Core for Floating Point @ 1.25 GHz“ ergibt bei acht Kernen 160 GFLOPS[SPR11f, S.11]
26 Entnommen aus Datenblatt C6678-DSP [SPR11f] sowie Datenblatt DDR3-Controller [SPR11a].

4.2.1 C6678-Evaluierungsmodule

Texas Instruments stellt Evaluierungsmodule (EVM) des C6678-DSP-Chips für Rapid Prototyping bereit, die nur in geringen Stückzahlen an Soft- und Hardwareentwickler vertrieben werden.[27] Für einen Systementwickler ermöglichen diese Module, die Fähigkeiten der Chips auszuprobieren, bevor oder während die Hardware, für die der Chip eingesetzt werden soll, entwickelt wird.

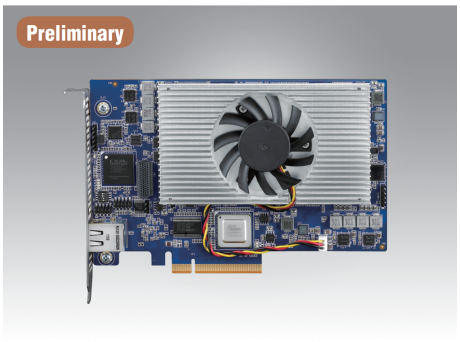
Das EVM enthält einen C6678-DSP-Chip, 512 MB Speicher, wichtige Schnittstellen sowie die Energieversorgung. Das Modul verfügt über einen 170 pin B+ AMC (Advanced Mezzanine Card) Anschluss.[28] Ein kompletter JTAG-Controller[29] ist bereits onboard integriert, ein externer JTAG-Controller deswegen unnötig; Kennwerte sind Tabelle 4.3 zu entnehmen

DSP	1 Stück C6678 (s. Tabelle 4.2)
RAM	512 MB
Flash	64 MB NAND, 16 MB NOR
Netzwerk	2 Stück Gigabit Ethernet (einmal RJ45, einmal über AMC)
JTAG	integrierter JTAG über USB (Typ: XDS100)
RS232	sowohl dediziert als auch über USB
Erweiterungen	170 pin B+ AMC (siehe Text) sowie 80-Pin-Stecker mit SPI, UART etc.
Hispeed Busse	Hyperlink, PCIe (nur über AMC, kein Steckkartenbetrieb möglich)
Energieverbrauch	31,2 Watt

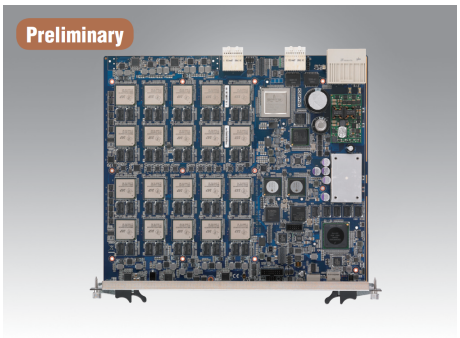
Tabelle 4.3: C6678-Evaluierungsmodul von TI

Es sei angemerkt, dass ein Bezug der Evaluationsmodule in hohen Stückzahlen von Texas Instruments nicht möglich ist. Für eine industrielle Verwendung der C6678-DSP-Chips wird ein Hersteller von Medizingeräten im Regelfall die DSP-Chips in die eigene Elektronik integrieren. Es besteht auch die Möglichkeit, fertig integrierte Module zu erwerben. In Abbildung 4.2[30] sind zwei Produkte der Firma Advantech exemplarisch abgebildet.[31] Zum Zeitpunkt der Drucklegung dieser Arbeit sind diese Module noch nicht verfügbar, daher sind die hier verwendeten Datenblätter und Fotos des Herstellers als „preliminary“ (vorläufig) markiert. Links ist das Produkt DSPC-8681 mit PCI-Express-Schnittstelle zu sehen: Es enthält vier Stück C6678-DSP, 1 GB Ram pro DSP und eignet sich für den unkomplizierten

27 Das Datenblatt findet sich in [SPR11d].
28 Anschluss für Hardwareerweiterungen, Kurzspezifikation siehe [PIC04], die ausführliche Spezifikation ist kostenpflichtig beim Herausgeber erhältlich.
29 Schnittstelle für Mikroprozessor Debugging der Joint Test Action Group nach IEEE 1149.1, siehe [Tan06, S.614]
30 Bilder entnommen mit freundlicher Genehmigung des Herstellers aus den vorläufigen Datenblättern [Adv11a] sowie [Adv11b].
31 Diese Firma baut auch im Auftrag von Texas Instruments das oben erwähnte Evaluierungsmodul TMDXEVM6678L.



(a) DSPC-8681



(b) DSPA-8901

Abbildung 4.2: Industriell verfügbare Komponenten mit C6678-DSPs

Einbau in Medizingeräte, die auf herkömmlicher PC-Technik basieren. Der Energieverbrauch liegt bei Worst-Case 54W. Alternativ eignet sich diese PCIe-Karte für den Einbau in Labor-PCs an Forschungsarbeitsplätzen. Rechts ist das Produkt DSPA-8901 als Blade im PICMG 3.0/3.1-Format mit 20 DSP-Chips, 0,5 GB Ram pro DSP und 350 Watt Worst-Case-Energieverbrauch für die gesamte Platine abgebildet.

4.3 Versuchsaufbau für die messtechnische Analyse

Die Messung der Performance von DSPs in der medizinischen Bildregistrierung ist ein wesentlicher Aspekt dieser Forschungsarbeit. Für diese Messung wird ein bestimmtes Verfahren angewendet, auf das vorab eingegangen werden soll.

Es handelt sich beim zuvor vorgestellten C6678 um ein sehr junges Produkt, bei dem zum Zeitpunkt dieser Studie noch keine Industriekomponenten (wie DSPC-861 oder DSPC-8901 s.Abbildung 4.2) zur Verfügung stehen. Es existieren lediglich die Evaluierungsmodule von TI(s. Unterabschnitt 4.2.1), weswegen vier Evaluierungsmodule vom Typ „TMDXEVM6678L EVM“ für die Performancemessung verwendet werden. Die mit Gigabit Ethernet ausgestatteten, gehäuselosen Platinen werden auf einer ESD-Schutzmatte platziert und in Sterntopologie über vier dedizierte Cat6a-Ethernetleitungen (gekreuzte Kabel) mit dem Client-PC verbunden. Als Client-PC kommt ein PC im 19-Zoll-Format mit Intel Xeon Dualcore 3065 Prozessor zum Einsatz. Der Client-PC wird für die netzwerktechnische Sternstruktur mit vier Stück 1-GBit-Ethernet-Interface ausgestattet. Desweiteren werden die USB-Schnittstellen der Evaluierungsmodule (die das Konsolen- und das JTAG-Signal transportieren) über einen USB-Hub mit dem Client-PC verbunden (s. Abbildung 4.3).

Im Testaufbau werden alle Geräte mit Energie versorgt, der Client-PC wird an ein Terminal (Maus, Tastatur sowie Bildschirm) angeschlossen. In Abbildung 4.4 findet sich eine Fotografie des zur Messung verwendeten Testaufbaus.

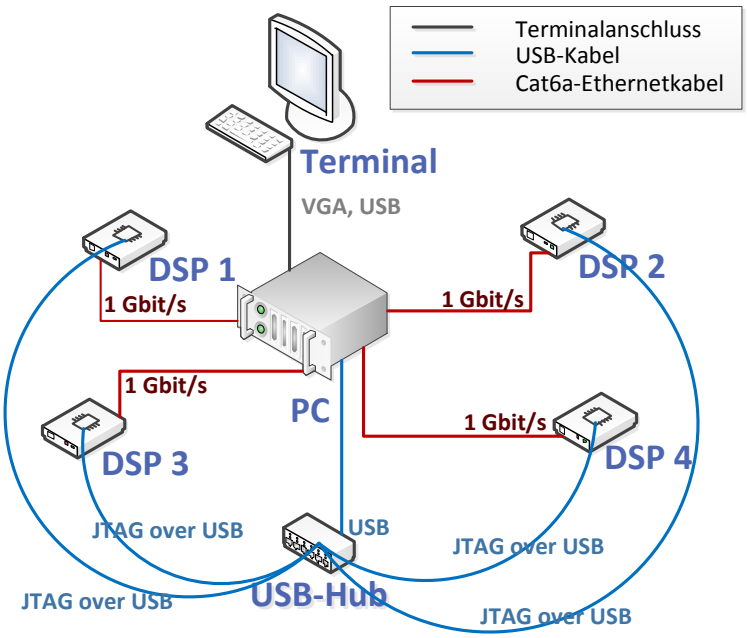


Abbildung 4.3: Testaufbau schematisch

Abbildung 4.4: Testaufbau mit vier DSP-Evaluierungsmodulen

4.3.1 JTAG-Konfiguration

Bedeutend für den Betrieb einer DSP-Lösung ist der Bootvorgang. Nach dem Einschalten der Stromversorgung ist es notwendig, dass die DSP-Chips entweder selbständig ihre Programme aus einem Festspeicher[32] laden oder von einer externen Instanz, wie einem Mikrocontroller, mit einem Programm versorgt werden. Am Einsprungpunkt dieses Programms beginnt der Bootloader.[33]

32 Alternativ von einem Netzwerkserver. Bootoptionen siehe [SPR11f]
33 Der Bootloader des C6678-DSP ist dokumentiert unter [SPR12a].

Die Erstellung eines solchen Bootloaders ist bei DSPs mit mehr, als einem Rechenkern mit hohem Aufwand verbunden.[34] Der Bootvorgang besteht dabei aus zwei Stufen: Anfangs wird, wie zuvor beschrieben, ein Programm in den ersten Rechenkern geladen, das umgehend mit der Ausführung des Bootloaders beginnt. Dieser trägt Sorge für die Versorgung der übrigen Rechenkerne mit ihren (ggf. unterschiedlichen) Programmen, die daraufhin parallel zueinander in ihrem jeweiligen Einsprungspunkt ausgeführt werden. Da während der Entwicklung des Programms dessen Aufteilung auf die Rechenkerne noch modifiziert werden kann, empfiehlt es sich, den Bootloader erst zum Projektende zu erstellen. Ein weiteres Argument dafür liefert der Umstand, dass ein Teil des Bootvorgangs aus Systeminitialisierungen besteht, die vom Softwareentwickler aufgabenspezifisch zu programmieren sind. Da es während der Softwareerstellung noch zu Änderungen am Umfang verwendeter Betriebssystemkomponenten kommen kann, würde eine zu frühe Implementierung der Systeminitialisierungen zu permanenten Nachbesserungen führen.

Im Vorfeld wurde darauf hingewiesen, dass ein C6678-Evaluierungsmodul bereits über einen integrierten JTAG-Controller für das Debugging über die USB-Schnittstelle verfügt. Neben klassischen Debugging-Aufgaben, wie das Setzen von Haltepunkten oder das Auslesen von Speicher, ist es über diesen JTAG-Controller auch möglich, ein Programm auf mehreren DSP-Rechenkernen zu starten, ohne dass zuvor ein Bootloader programmiert wurde. Die Systeminitialisierungen werden dabei scriptgesteuert vorgenommen, wofür viele passende Beispielscripte von Texas Instruments verfügbar sind, auf die sich die vorliegende Arbeit im Wesentlichen beschränkt. Für die reine Analyse eines DSP-Programms genügt ein Softwarestart über den JTAG-Controller. Ein Bootloader muss erst später von den Industrieunternehmen geschrieben werden, welche die Forschungsergebnisse Produkte überführt, die in der Lage sein müssen, selbsttätig zu booten.

Die onboard JTAG-Adapter der C6678-Evaluierungsmodule tragen interne Seriennummern, die alle identisch sind. Um mehrere DSP-Chips über JTAG zu kontrollieren, muss jedoch eine eindeutige Nummer vergeben sein. Deswegen ist es für den Testaufbau dieser Arbeit notwendig, die Seriennummern von drei der vier Evaluierungsmodule zu ändern, was durch ein Flashen der JTAG-Firmware erfolgt.[35] Dabei ist höchste Vorsicht geboten, da der Flashvorgang über die JTAG-Schnittstelle selber ausgeführt wird: Schlägt er fehl, steht diese nicht mehr zur Verfügung und ein weiterer Versuch ist unmöglich.

Eine gefahrlosere Alternative zum Flashvorgang ist die Verwendung von einem dedizier-

34 Texas Instruments stellt hierfür die 'Multicore Application Deployment (MAD) Utilities' bereit, siehe [Texd].

35 Siehe FAQ (Frequently Asked Questions) im Internet von Texas Instruments zum XDS100 Debugger [Texf]

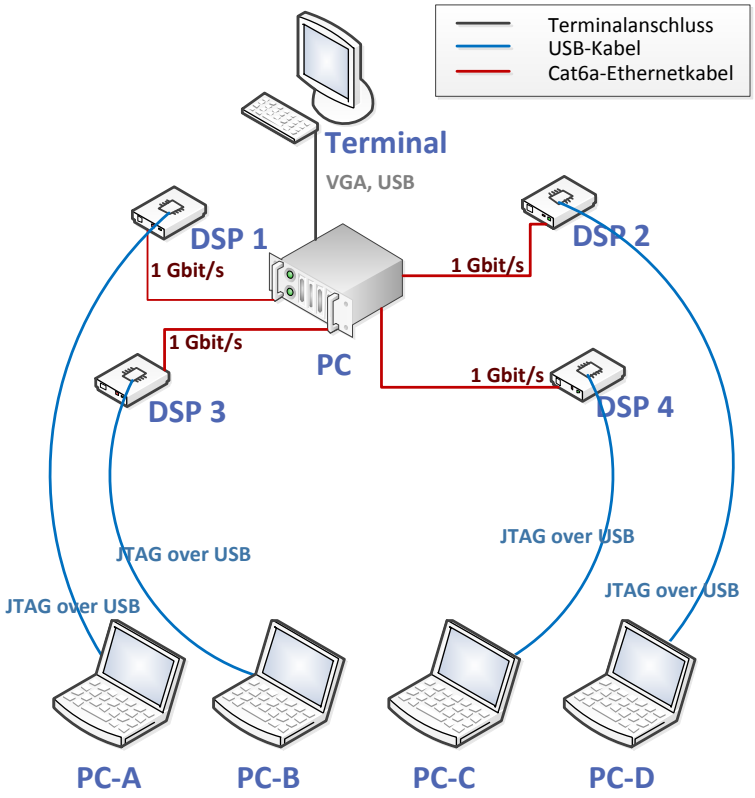


Abbildung 4.5: Alternativer Testaufbau mit dedizierten JTAG-Verbindungen

Funktion	Anzahl	Hardware	Technische Details
DSP	4	EVMC6678	DSP-Evaluierungsmodul, 512 MB Ram, 1 GHz Taktfrequenz, acht Rechenkerne pro DSP
JTAG-PC	4	Dell Vostro	Notebook mit USB 2.0, 2,33 GHz Intel Core i5 (Quadcore), 6 GByte Ram
Client-PC	1	HP ProLiant	19-Zoll-Server mit fünf mal 1 GBit/s Ethernet, 2,66 GHz Intel Xeon Dualcore 3065, 2 GByte Ram

Tabelle 4.4: Hardwareausstattung für die Performancemessung

ten Rechner pro DSP-Evaluierungsmodul. Der Testaufbau wäre dann entsprechend zu modifizieren, wie in Abbildung 4.5 dargestellt. Ein weiterer Vorteil des alternativen Testaufbaus ist, dass die Programme vierfach parallel auf die Zielhardware geladen werden können, was den Programmstart beschleunigt. Für die unten stehende messtechnische Untersuchung wird ein derartiger Aufbau gewählt mit der in Tabelle 4.4 beschriebenen Hardwareausstattung.

4.3.2 Netzwerkkonfiguration

Im zuvor skizzierten Testaufbau werden die DSP-Chips über Ethernet in einer Sterntopologie[36] mit dem Client-PC verbunden, damit jeder DSP mit der maximal möglichen Datenrate des Bussystems angesprochen werden kann. Bei der Verwendung von Gigabit Ethernet kann eine Geschwindigkeit von 1000 MBit/s pro Verbindung erreicht werden.[37]: Demnach steht insgesamt eine digitale Bandbreite von 4000 MBit/s, bei vier dedizierten Leitungen zur Verfügung. Jedes DSP-Evaluierungsmodul wird über eine Zweipunkt-Leitung mit je einem Ethernetanschluss des Client-PC verbunden. Um zu entscheiden, über welchen Anschluss ein ausgehendes Datenpaket gesendet werden muss, befragt der Client-PC seine interne Routingtabelle.[38] Sie wird vom Betriebssystem des Client-PC automatisch aufgebaut, wobei u.A. die IP-Konfiguration aller Ethernetanschlüsse ausgewertet wird. Damit ein eindeutiges Routing gewährleistet ist, wird auf dem Client-PC jedem Ethernetanschluss ein eindeutiges Subnetz zugewiesen. In jedem Subnetz befinden sich lediglich zwei Geräte: erstens der Client-PC und zweitens das DSP-Evaluierungsmodul. Demzufolge bietet sich ein privates Class-C-Netz mit der Netzmaske 255.255.255.0 an, denn dieses unterstützt bis zu 254 Netzwerkknoten.[39] In Tabelle 4.5 wird eine Netzwerkkonfiguration passend zu Abbildung 4.5 dargestellt.

Gerät	**IP-Adresse(n)**	**Netz(e)**	**Netzmaske(n)**
DSP-1	192.168.177.2	192.168.177.0	255.255.255.0
DSP-1	192.168.178.2	192.168.178.0	255.255.255.0
DSP-1	192.168.179.2	192.168.179.0	255.255.255.0
DSP-1	192.168.180.2	192.168.180.0	255.255.255.0
Client-PC	192.168.177.1	192.168.177.0	255.255.255.0
	192.168.178.1	192.168.178.0	255.255.255.0
	192.168.179.1	192.168.179.0	255.255.255.0
	192.168.180.1	192.168.180.0	255.255.255.0

Tabelle 4.5: IP-Konfiguration des Aufbaus für die Performancemessung

Die in Abbildung 4.5 aufgeführten Laptops sind über DHCP (Dynamic Host Configuration Protokoll)[40] mit dem Hausnetz verbunden, wodurch ein Dateiaustausch und der Internetzugriff stattfinden kann. Eine netzwerktechnische Verbindung zu den DSP-Evaluierungsmodulen besteht nicht.

Der zuvor beschriebene Hardwareaufbau benötigt für den Betrieb ein Softwareprogramm, welches im folgenden Kapitel ausführlich beschrieben wird.

36 [Tan06, S.647 ff.]

37 [Tan06, S.602]. Im Versuchsaufbau wurden CAT-6a STP Kabel eingesetzt, eine Übersicht über Twisted Pair Kupferkabel findet sich in [BDH05, S.201].

38 Routing und Routingtabelle werden erläutert in [ST10, S.157 ff.]

39 IP-Netzklassen werden behandelt in [ST10, S.70 ff.]

40 Über DHCP werden Endgeräte automatisiert konfiguriert, insbesondere kann darüber automatisch eine IP-Konfiguration vergeben werden, siehe [Bor98, S.309 ff.]

5 Softwareimplementierung

Eine Analyse von DSPs in der Bildregistrierung, insbesondere die Messung von Performancedaten, erfordert die Erstellung einer umfangreichen Software. Die DSP-Chips werden mit einem Betriebssystemcode versorgt und mit einem Datenübertragungsprotokoll sowie dem Registrierungsalgorithmus versehen. Zur Steuerung der DSP-Chips wird eine PC-Software benötigt, welche die DSP-Chips über ein Datenübertragungsprotokoll mit Bilddaten und Berechungsvorschriften versorgt. Ferner wird parallel zur Implementierung des Algorithmus auf einem DSP eine weitere Implementierung des Algorithmus in eine PC-Software notwendig, um die Performance der DSP-Software mit einer lokalen Berechnung auf einem PC zu vergleichen.

Die in dieser Arbeit erstellte Software ist in zwei Projekte aufgeteilt: ein Projekt für die DSP-Software und eines für die PC-Software. Aus Tabelle 5.1 wird der Umfang einer solchen Implementierung ersichtlich.[1]

Projekt	C/C++ Dateien	Matlab-Dateien	Codezeilen	Kommentarzeilen
DSP-Software	54	3	3326	3048
PC-Software	116	23	7058	4347

Tabelle 5.1: Umfang der Software in Dateien und Code- sowie Kommentarzeilen

In den Anhängen A.1 sowie A.2 finden sich Auszüge aus den Softwareengineering-Dokumenten, der während der Untersuchung entstandenen Messsoftware.

5.1 Software für die messtechnische Erforschung

Der Registrierungsalgorithmus wird so auf die beiden Projekte der Messsoftware aufgeteilt, dass die wenig intensiven Berechnungen für den Gauß-Newton-Algorithmus und den Armijo-Line-Search lokal auf einem PC ausgeführt werden. Sobald im Rahmen des Gauß-Newton- oder Armijo-Verfahrens eine Berechnung des Distanzmaßes (SSD), der Jacobi-Matrix oder der Hesse-Matrix benötigt wird, werden diese Berechnungsaufgaben, wie in Abbildung 5.1 dargestellt, auf die DSP-Chips verteilt.

1 Ermittelt wurden Codezeilen ohne Kommentar- und Leerzeilen mit der quelloffenen Software 'cloc'. Projekthomepage: cloc.sourceforge.net.

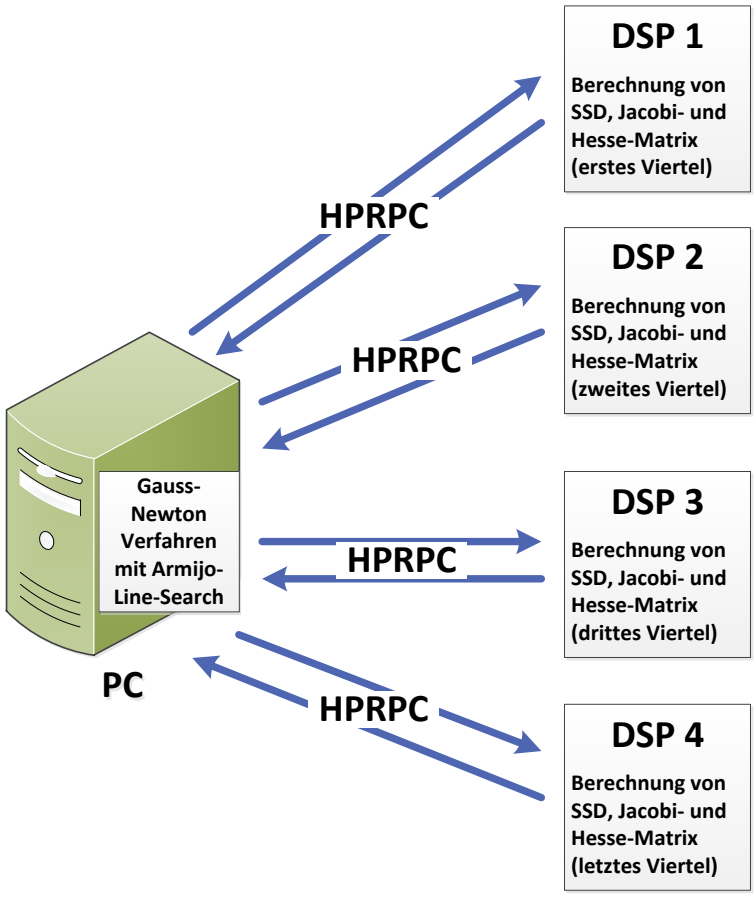


Abbildung 5.1: Verteilung des Registrierungsalgorithmus

Die DSP-Chips benötigen für ihre Berechnungsaufgaben Teile der Bilddaten, wie in Unterabschnitt 3.4.1 dargestellt, die zum Programmstart der PC-Software an die DSP-Software verteilt werden. Dabei kommt das weiter unten in Unterabschnitt 5.2.1 beschriebene HPRPC-Protokoll zum Einsatz. Nachdem es sich beim C6678-DSP um Multicore-DSP-Chips mit acht Rechenkernen handelt, ist eine weitere Aufgabe der DSP-Software die Berechnungen in acht gleichmäßige Berechnungsbereiche aufzuteilen (vgl. Par5) sowie die Teilergebnisse zu summieren.

Im Rahmen der vorliegenden Arbeit ist ein per Kommandozeile zu steuerndes Programm völlig ausreichend. In Abbildung 5.2 findet sich eine Bildschirmkopie der zu diesem Zweck erstellten Software. Es lassen sich Informationen über die Bilddateien, die Anzahl der DSP-Rechenkerne und die Netzwerkadressen der DSP-Chips eingeben. Ferner kann über das in Unterabschnitt 3.4.1 vorgestellte Verfahren Einfluss darauf genommen werden, wie viel Spielraum an maximaler Rotation und Translation bestehen soll, ohne dass neue Bilddaten übertragen werden müssen. Darüberhinaus können auch Stop-Kriterien des Gauß-Newton-Algorithmus spezifisch eingestellt werden.

Damit die Bedienung trotz der vielfältigen Einstellungsmöglichkeiten intuitiv und einfach bleibt, sind für die meisten Parameter sinnvolle Defaultvorgaben angelegt. Bereits die

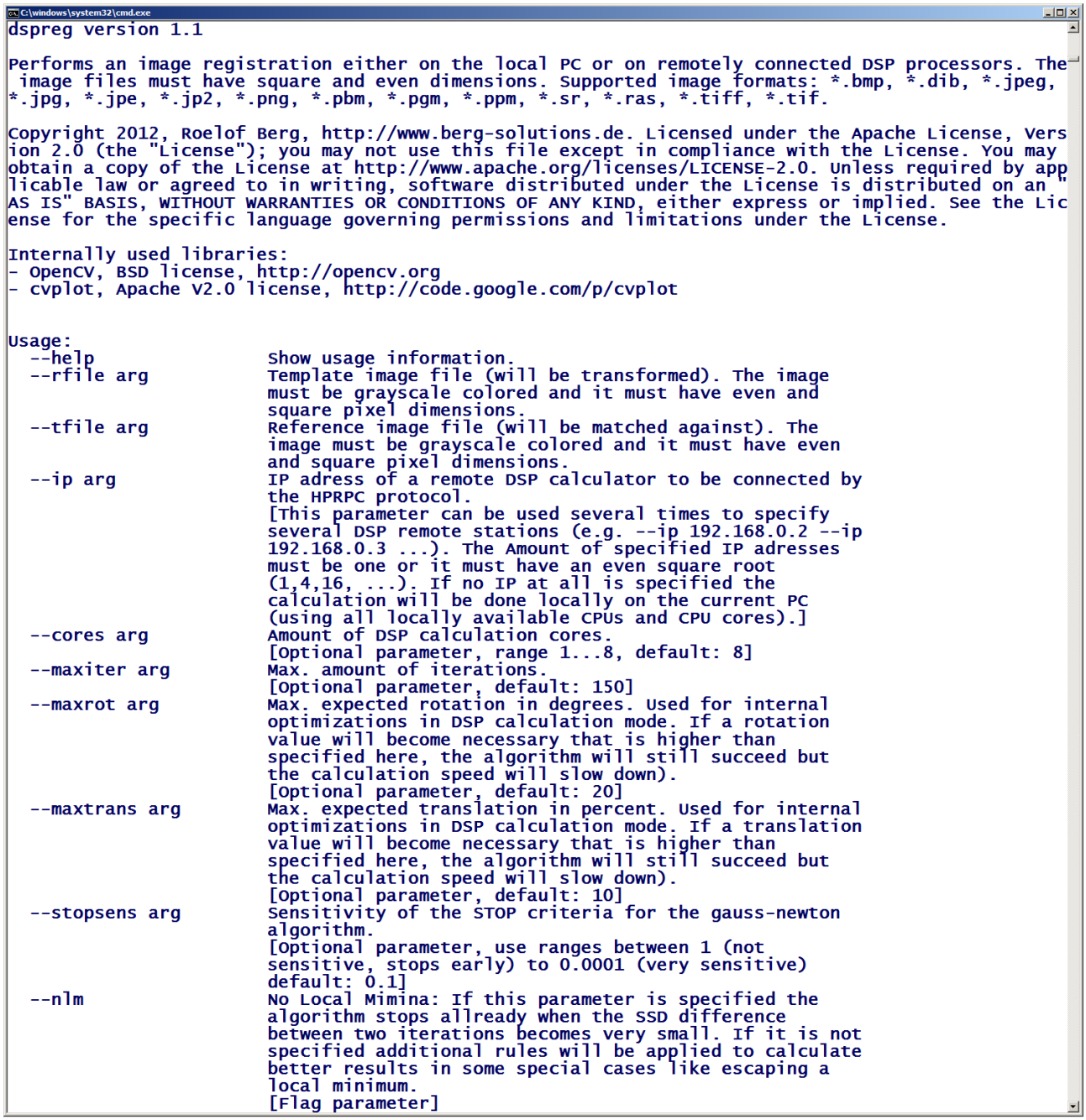


Abbildung 5.2: Kommandozeilenoptionen der PC Software

Eingabe der Bilddaten sowie der Netzwerkadressen der DSP-Chips reichen für einen Programmstart aus. Eine Beispielausgabe ist in Abbildung 5.3 abgebildet.

Da Bilddaten verarbeitet werden, ist neben der Konsolen- auch eine grafische Bildausgabe notwendig, vgl. Abbildung 5.4[2] sowie Abbildung 5.5[3]. Abgebildet werden das Templa-

2 Die Bilddaten wurden entnommen aus dem Softwarepaket FAIR (kostenfrei im Internet erhältlich unter [SIA]) mit Genehmigung des Urhebers dieser Software Jan Modersitzki von Fraunhofer Institut MEVIS in Lübeck sowie mit Genehmigung des Urhebers der Bilder Oliver Schmitt, mittlerweile tätig am Institut für Anatomie in Rostock, der sie in seiner Habilitationsschrift [Sch95] veröffentlichte.

3 Die medizinischen Bilddaten wurden zur Verfügung gestellt von Dennis Doleschel vom Lehrstuhl für

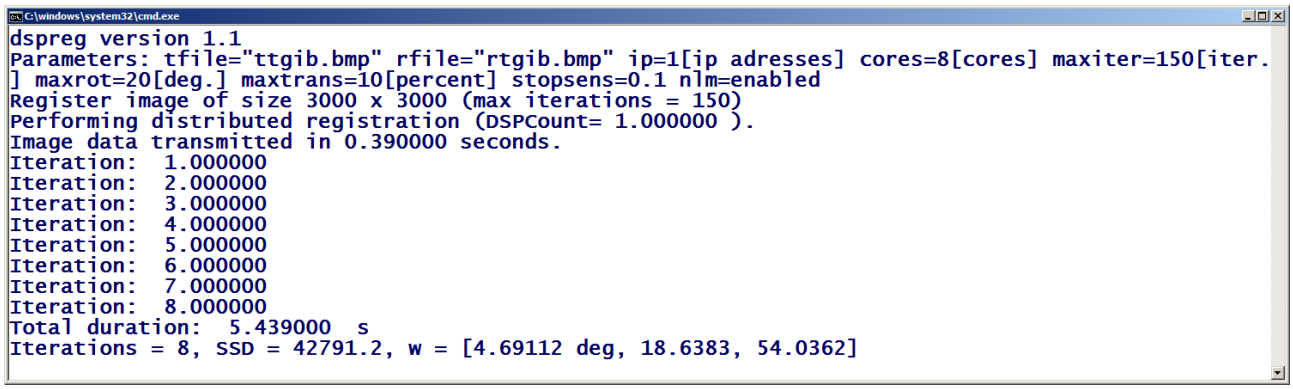


Abbildung 5.3: Kommandozeilenausgabe der PC Software

tebild (l.o.), das Referenzbild (m.o.), das registrierte Ausgabebild (r.o.). In der unteren Zeile werden Distanzbilder sowie das Distanzmaß dargestellt. Wie in Unterabschnitt 3.4.1 kennzeichnet die Pixelfarbe die Abweichung zwischen Referenz- und Templatebild in einem Punkt. Je dunkler die Pixelfarbe, desto stärker ist die Abweichung zwischen den Bildern. Links ist das ursprüngliche Distanzbild abgebildet, mittig das Distanzbild nach der Registrierung.

Unten rechts ist eine Graphendarstellung über den Verlauf des SSD-Distanzmaßes über die einzelnen Registrierungsiterationen.[4]. Es ist gut sichtbar, dass das Distanzmaß in den Beispielen mit jeder Iteration abnimmt, bis der Algorithmus verlassen wird.

5.1.1 Rapid Prototyping in Matlab

Es ist von Vorteil, die mathematischen Algorithmen zunächst in einer Rapid-Prototyping-Umgebung, wie Matlab, zu entwickeln und später auf die Zielsprache (hier C++) zu portieren. Dies hat den Grund, dass in einer Rapid-Prototyping-Umgebung eine stärkere Fokussierung auf die rein mathematisch-algorithmischen Aspekte möglich ist, ohne von Aspekten der Systemsoftware, wie z.B. der Speicherverwaltung und der Performanceoptimierung, ausgebremst zu werden. Einen weiteren Vorteil bietet die einfache Integrierbarkeit mathematischer Prüfverfahren, die lediglich während der Entwicklung benötigt werden. Als Beispiel sei hier die, bereits in Abschnitt 2.7 erwähnte, numerische Prüfung auf Korrektheit der Gradienten genannt.

In einer Rapid-Prototyping-Umgebung können die mathematischen, hardwareunabhängigen Teile des Algorithmus optimiert werden. Bei dem hier vorgestellten Algorithmus bedeutet das vor allem, die Verwendung von Matrizen weitestgehend zu reduzieren,

Experimentelle Molekulare Bildgebung des Universitätsklinikums Aachen

4 Erstellt mit der quelloffenen Software CVPlot, erhältlich unter [Cha] Leider unterstützt diese Software keine textuelle Beschreibung von Abszisse sowie Ordinate.

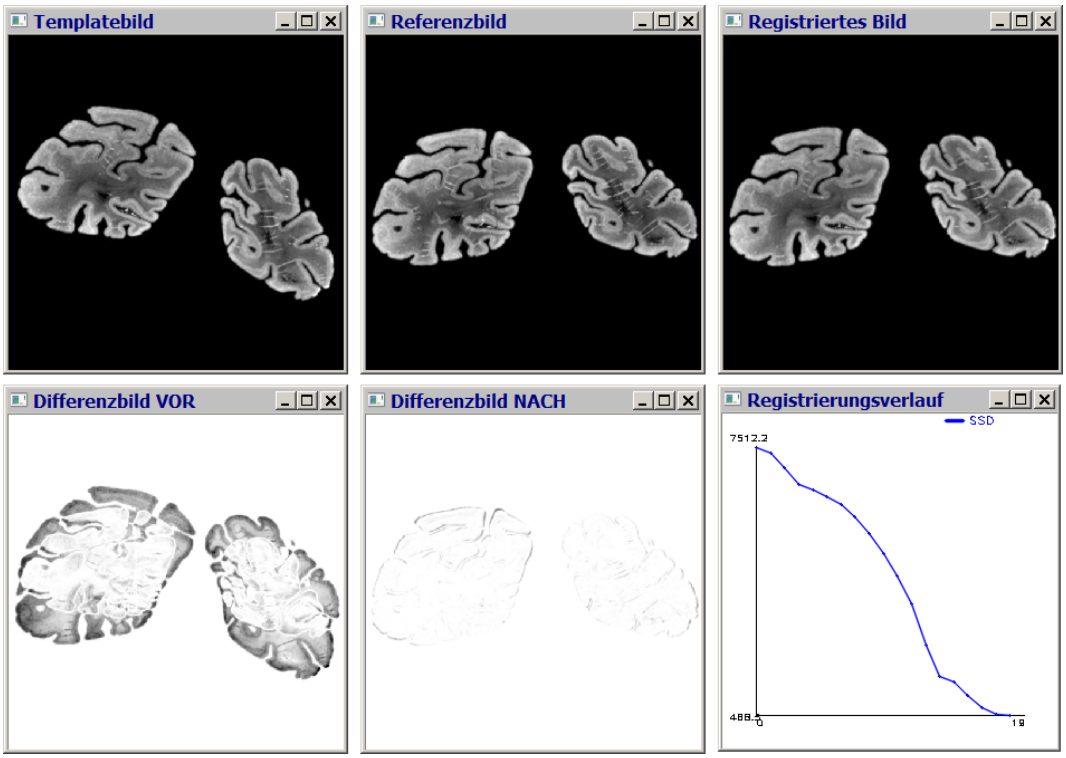


Abbildung 5.4: Bildausgabe der PC Software - Gehirn

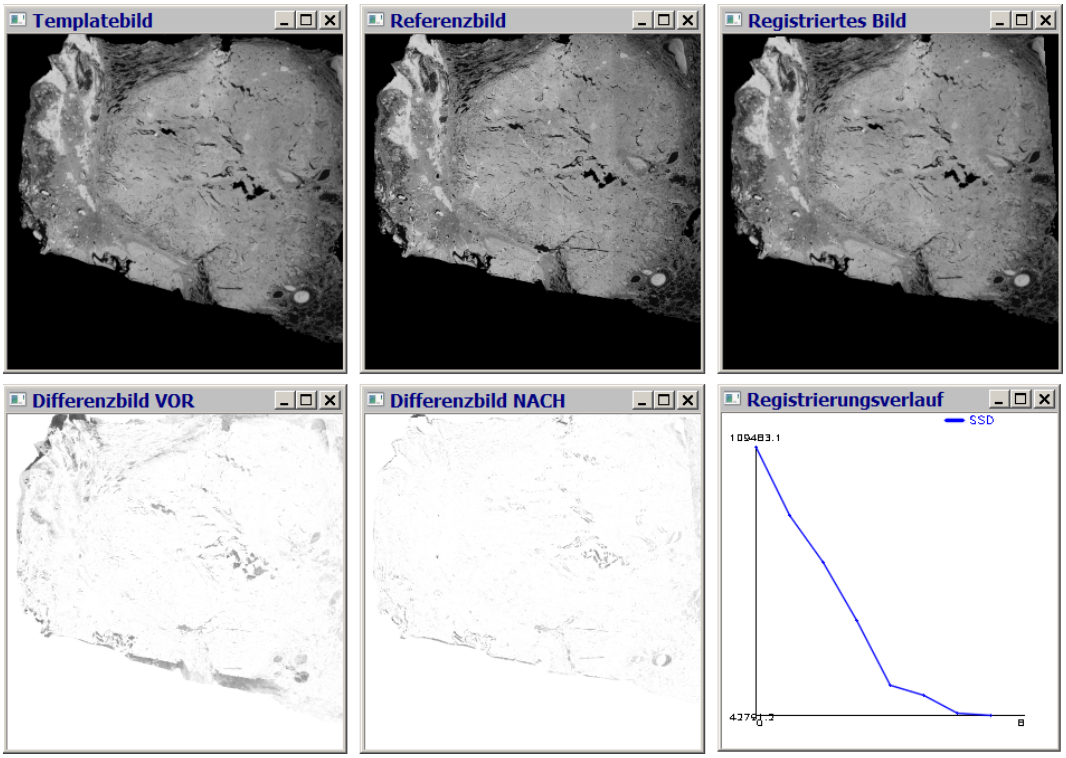


Abbildung 5.5: Bildausgabe der PC Software - Tumor

wie in Abschnitt 3.3 vorgestellt. Es ist schon in der Rapid-Prototyping-Umgebung möglich, die Verteilung des Algorithmus zu simulieren. So können Host- und Target-Code in Matlab in getrennten m-Files programmiert werden. In diesem Fall ruft der Simula-

tionscode für den Host mehrmals den Target-Code auf, was in Matlab zwar seriell und nicht parallel geschieht, jedoch für eine Funktionsprüfung (bzw. für eine prototypische Entwicklung) akzeptabel ist. In Abbildung 5.6 ist eine exemplarische Ausgabe einer Rapid-Prototyping-Implementierung dargestellt: Oben befinden sich Referenz- und Templatebild, unten das fertig registrierte Bild sowie der Verlauf des Distanzmaßes über die einzelnen Iterationen.

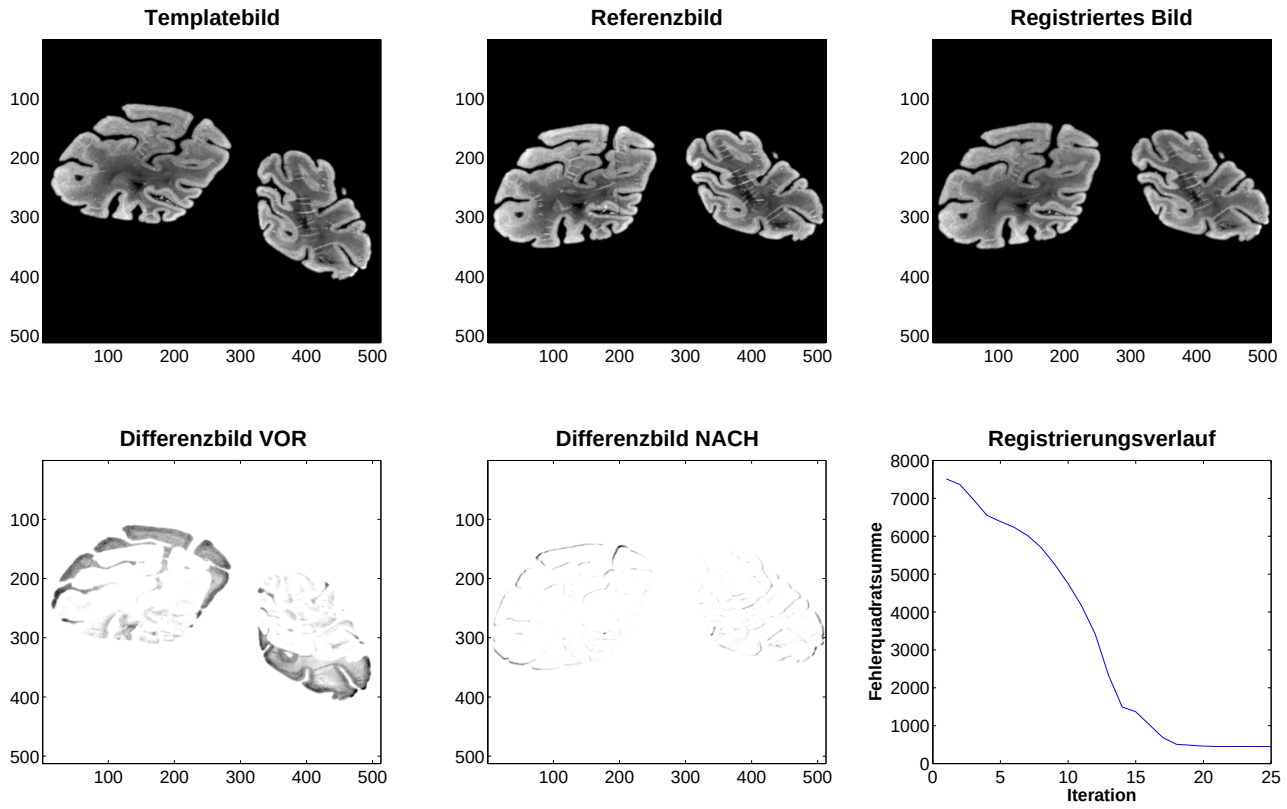


Abbildung 5.6: Prototyp in Matlab

5.1.2 Matlab-Coder

Die Portierung in eine geeignete Programmiersprache kann in Matlab mit dem Softwarepaket Matlab-Coder[5] automatisiert werden. Die von Matlab-Coder ausgegebenen C++ Dateien können direkt in die jeweilige Entwicklungsumgebung für die einzelnen Softwaremodule eingebunden werden. Dabei ist zu beachten, dass es sich um einen verteilten Algorithmus handelt. Das heißt, die mit Matlab-Coder generierten Dateien müssen aufgeteilt werden in solche, die zur DSP-Serversoftware gehören sowie in jene, die in die PC-Clientsoftware eingebunden werden müssen. Hierfür ist es von Vorteil, wenn die spätere Verteilung des Algorithmus bereits in der Rapid-Prototyping-Umgebung

5 Beschreibung des Herstellers im Internet unter [Thec]

vorgenommen wurde (z.B., wie zuvor demonstriert, über getrennte m-Dateien in Matlab).

Um C++ Quellcode aus Matlab-Quellcode zu generieren muss in Matlab-Coder zunächst eine Matlab-Coder-Projektdatei über das Menükommando 'File/New/Code Generation Project' angelegt werden.[6] Die Projektdatei trägt die Dateiendung '.prj' und wird automatisch neben den m-Dateien angelegt, die den Matlab-Quellcode enthalten. Auf der rechten Seite des Matlab-Hauptfensters wird die Projektkonfiguration eingeblendet. Hier sind vielfältige Einstellmöglichkeiten zu finden, insbesondere der Ausgabetyp mit den in Tabelle 5.2 aufgelisteten Auswahloptionen. Für die gegebene Problemstellung ist der Typ 'Static Library' zu wählen, da der generierte Code fest als Teilkomponente in eine andere Software einkompiliert wird.[7]

Ausgabetyp	**Erläuterung**
C/C++ Static Library	Statische Bibliothek zur Verlinkung in andere Software während des Kompilierens.
C/C++ Dynamic Library	Laufzeitbibliothek, die dynamisch von anderer Software geladen werden kann.
MEX function	Kompillierter Matlab-Code zur Einbindung in Matlab-Scripte.
C/C++ Executable	Eigenständig ausführbare Programmdatei.

Tabelle 5.2: Ausgabetypen von Matlab-Coder

Für die Entwicklung von DSP-Software empfiehlt sich neben dem Produkt Matlab-Coder das Zusatzprodukt Matlab-Embedded-Coder[8], welches Matlab um eine hardwarespezifische Codegenerierung mittels angepasster 'Code Replacement Libaries' erweitert. Unter Nutzung von Matlab-Embedded-Coder ist für die Generierung der PC-Software als Zielhardware 'Generic / Matlab HOST Computer' sowie als Code Replacement Library 'C89/C90 (ANSI)' einzustellen. Dies kann in den zuvor erwähnten Projekteigenschaften am rechten Bildrand von Matlab über die Schaltfläche mit dem Zahnradsymbol vorgenommen werden (siehe Abbildung 5.7 rechts, oben). Für die DSP-Software sind 'Texas Instruments C6000' als Hardware sowie 'TI C66X' (falls nicht vorhanden 'TI C67X') als Code Replacement Library einzustellen, wie in Abbildung 5.8 dargestellt.

Neben den Projekteinstellungen ist auch eine Modifikation des Matlab-Quellcodes notwendig. Weil Matlab eine typenlose Sprache ist und C++ eine typenbehaftete, benötigt der Codegenerator Typinformationen für alle verwendeten Variablen. Die Typen der Eingabevariablen einer Matlab-Funktion sowie die der globalen Variablen werden in den Matlab-

6 Es wird davon ausgegangen, dass Matlab sowie ein zu Matlab-Coder kompatibler C/C++ Compiler bereits auf dem System installiert sind.

7 In der später dargestellten Firmware besteht keine Möglichkeit, dynamische Laufzeitbibliotheken nachzuladen, da aus Effizienzgründen kein Filesystem eingesetzt werden soll.

8 Beschreibung des Herstellers im Internet unter [Theb]

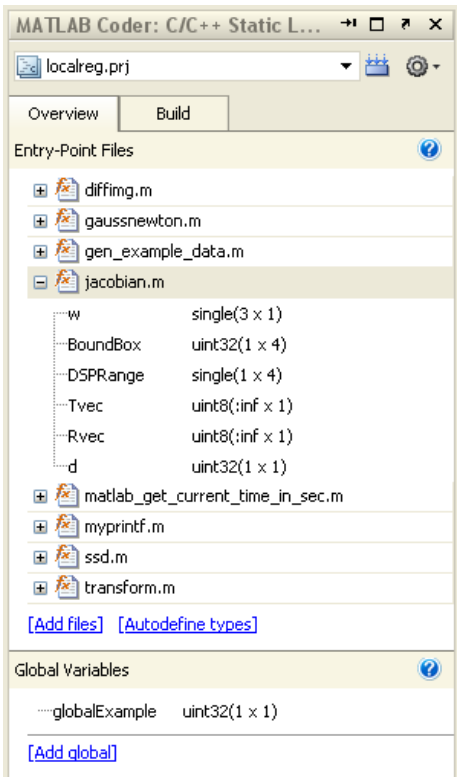


Abbildung 5.7: Definition von Variablentypen in Matlab-Coder

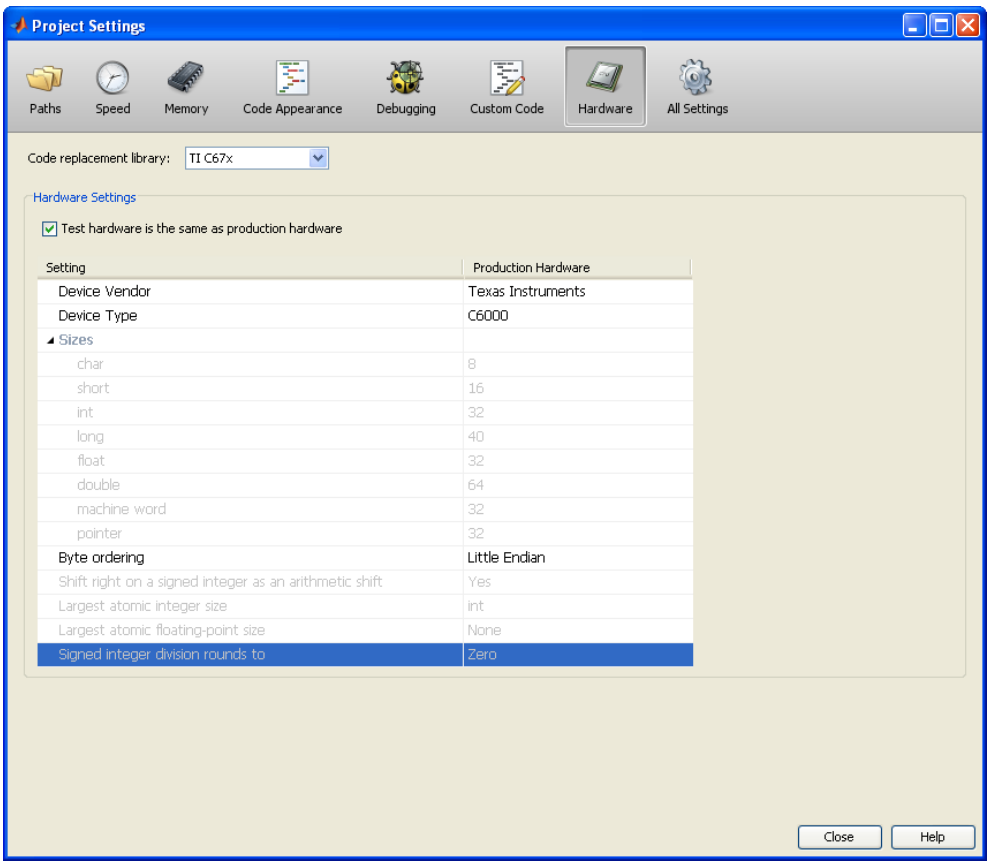


Abbildung 5.8: Matlab-Embedded-Coder-Einstellungen für Texas Instruments DSPs

Projekteinstellungen eingestellt, wie in Abbildung 5.7 dargestellt. Die Typen der Ausgabe- sowie der lokalen Variablen müssen im Quellcode mit dem Matlab-Casting-Operator beim erstmaligen Auftreten definiert werden. Für alle weiteren Stellen ist Matlab-Coder der Variablentyp dann bekannt. Handelt es sich um einen Vektor oder eine Matrix, so dient der Befehl 'zeros' gleichzeitig zur Initialisierung sowie zur Typenzuweisung. Folgendes Codebeispiel ist eine Matlab-Funktion vor der Typendefinition:

```
function transmitImageData(DSPCount, w, da, dtr, Tvec, Rvec, d)

tic;  %Berechnungsdauer aufzeichnen. Timer starten.

dmax=d/2-0.5;              %Ende der Koordinaten A
dmin=-dmax;                %Beginn der Koordinaten A
s=d/2+0.5;                 %Shifting, um negative Koordinaten in
    Matrixbereich zu bringen
angles=zeros(6,1);         %Winkel fuer Worst-Case-Bounding-Box
i=3;                       %Laufvariable (drei Winkel bereits bekannt)

%Winkel berechnen
%... (code hier Gekuerzt) ...

%Simulationsmodus, simuliert Datentransfer zu DSP ueber globale Variable
GlobAngles=angles;

toc;  %Timer stoppen, Berechnungsdauer ausgeben.
```

Für Matlab-Coder wird der Code wie folgt angepasst:

```
function transmitImageData(DSPCount, w, da, dtr, Tvec, Rvec, d)

coder.extrinsic('tic', 'toc');
tic;  %Berechnungsdauer aufzeichnen. Timer starten.

dmax=single((single(d)/2)-0.5);   %Ende der Koordinaten A
dmin=single(-dmax);               %Beginn der Koordinaten A
s=single(single(d)/2+0.5);        %Shifting, um negative Koordinaten in
    Matrixbereich zu bringen
angles=zeros(6,1,'single');       %Winkel fuer Worst-Case-Bounding-Box
i=uint32(3);                      %Laufvariable (drei Winkel bereits
    bekannt)

%Winkel berechnen
%... (code hier Gekuerzt) ...

%Verzweigung je nachdem, ob Matlab Coder im Spiel ist
```

```
16 if 1==isempty(coder.target)
17     %Simulationsmodus, simuliert Datentransfer zu DSP ueber globale Variable
18     GlobAngles=angles;
19 else
20     %Echter Modus. C-Funktion matlab_c_sendToTarget() aufrufen, welche das
21     %target kontaktieren und den Vektor 'angles' uebertragen wird
22     coder.ceval('matlabSendToTarget', angles);
23 end

25 toc;  %Timer stoppen, Berechnungsdauer ausgeben.
```

Aus dem Beispiel wird auch ersichtlich, dass der Matlab-Coder über das Schlüsselwort 'coder' gesteuert werden kann. Im Beispiel wird mittels 'coder.extrinsic' definiert, dass die Befehle 'tic' und 'toc' von Matlab-Coder ignoriert werden sollen. Das ist notwendig, weil sie in der Codegenerierung nicht unterstützt werden und zu einer Fehlermeldung führen würden. Mittels 'coder.target' ist eine Fallunterscheidung möglich, so dass bestimmte Zeilen nur bei der Codegenerierung oder nur bei der Ausführung in Matlab ausgewertet werden sollen. So ist es möglich, dieselbe Codebasis in Matlab ausführbar zu halten und gleichzeitig daraus Quellcode zu generieren. Über 'coder.ceval' schließlich können aus dem generierten Matlab-Code global definierte C-Funktionen des Programms aufgerufen werden, das den generierten Code beinhalten wird. Im Beispiel wird dies für die Datenübertragung verwendet, da im Matlab-Quellcode keine Möglichkeit besteht, die Netzwerkfunktionen des Betriebssystems anzusprechen.

Manuelle Nachbearbeitung

Bei der gegebenen Problemstellung fällt auf, dass Matlab-Coder zwar lesbaren und schnell ausführbaren Quellcode generiert, jedoch muss der Quellcode innerhalb des performancekritischen Teils, der pro Pixel ausgeführt wird, nachbearbeitet werden. Zwei Nachbearbeitungen sind notwendig:

1. Matlab übersetzt die Operation $a \cdot a$, welche für die Berechnung des Distanzmaßes benötigt wird, mit einer 'powf'-Funktion. Hier bewirkt eine Anpassung des generierten Quellcodes auf eine Multiplikation vom Typ $a \cdot a$ eine deutliche Steigerung der Ausführungsgeschwindigkeit.
2. Matlab verwendet unnötige Rundungsbefehle, wie 'roundf'. Diese Befehle enthalten intern Verzweigunsanweisungen, welche zu einem kontrollflussabhängigen Pipelinehemmnis führen können. Durch Casting-Operatoren auf Integer-Werte können diese aufwendigen Operationen vermieden werden, was die Ausführungsgeschwindigkeit erheblich verbessert.

5.2 Verteilte Informationsverarbeitung

Bei der in Abbildung 5.1 dargestellten verteilten Berechnung werden Daten zwischen den DSP-Chips und dem PC, der die grafische Benutzeroberfläche darstellt, über Ethernet ausgetauscht. Im folgenden Abschnitt wird dafür ein Protokoll vorgeschlagen, das über ein beliebiges streambasiertes, bidirektionales Businterface die Kommunikation zu den DSP-Chips herstellt. Dabei soll das Protokoll nicht auf Ethernet limitiert sein, sondern auch auf hardwarenahen Bussystemen anwendbar sein. Beispielsweise kann in einem eingebetteten System, bei dem Mikrocontroller und DSP-Chips auf derselben Platine angeordnet sind, das von den C6678-DSPs unterstützte Hyperlink-Prokokoll zum Einsatz kommen. Wird eine PCI-Express-Karte, wie in Abbildung 4.2, verwendet, wird diese Kommunikation über PCI-Express durchgeführt. Die weiteren Ausführungen beziehen sich auf eine ethernetbasierte Variante, es sei jedoch angemerkt, dass sich die Überlegungen genauso auf streambasierte, bidirektionale Kanäle anderer Bussysteme mit mehr Hardwarenähe sowie höherer Bandbreite übertragen lassen.

5.2.1 HPRPC-Protokoll

Das Protokoll soll die Bezeichnung HPRPC tragen. Diese Abkürzung steht für „High Performance Remote Procedure Call Protokoll“. Es bietet einige folgende, die in den anschließenden Abschnitten erläutert werden.

Verzicht auf Endian-Umcodierung

Herkömmliche Netzwerkprotokolle übertragen die Daten im Big-Endian-Format. Kommunizieren zwei Little-Endian-Rechner miteinander, werden die Bytes vor der Netzwerkübertragung in das Big-Endian-Format umgewandelt und beim Empfänger wieder zurück transformiert.[9] Beim hier vorgestellten Hardwareaufbau handelt es sich ausschließlich um Little-Endian-Gegenstellen, weswegen eine Übertragung im Big-Endian-Format unnötiger Performance-Overhead, der vermieden werden kann, ist. Insbesondere beim Betrieb des HPRPC-Protokolls über ein hardwarenahes Bussystem ist eine solche Konvertierung zu vermeiden, damit die Bytes ohne weitere Datenverarbeitung direkt per DMA (Direct Memory Access[10]) an den Protokollumsetzer übermittelt werden können.

9 Unterschiede in der Bytereihenfolge sind beschrieben unter [Tan06, S.89]
10 Direkter Speicherzugriff ohne Mitwirkung des Prozessors nach [Tan06, S.120].

Die C6678-DSPs von Texas instruments lassen sich sowohl im Big- als auch im Little-Endian-Modus betreiben.[11] Der DSP muss entsprechend dem Endian-Modus der Mikrocontroller-Gegenstelle betrieben werden. Bei einer Kommunikation zu einem Intel-PC ist Little-Endian einzustellen.

Geringer Protokolloverhead

Um höchste Performance zu erreichen, wird der Protokollheader auf acht Byte begrenzt, die ausschließlich die notwendigsten Informationen übertragen. Auf den Transfer redundanter Informationen, wie beispielsweise der Protokollversion in jedem Datenpaket, wird verzichtet. Das Gesamtsystem ist so zu konfigurieren, dass auf beiden Gegenstellen dieselbe Protokollversion zum Einsatz kommt. Zur Laufzeit werden keine Versionsnummern überprüft, um den Overhead der Protokollverarbeitung weitestgehend zu reduzieren.

Übertragung von Binärdaten

Daten werden zwischen den Gegenstellen ohne Umcodierung direkt im Binärformat übertragen, um den Zeitaufwand für den Datenversand zu dezimieren; dies gilt auch für Fließkommazahlen. Das Fließkommaformat des Mikrocontrollers muss dem der DSPs entsprechen. In Bezug auf die Kommunikation zwischen einem Intel-PC und einem C6678-DSP sind beide kompatibel, da ihre Plattformen den IEEE-Standard 754 verwenden.[12]

HPRPC über Ethernet

Wird das HPRPC-Protokoll über Ethernet übertragen, wird wegen der hohen Verbreitung das IP-Protokoll empfohlen. Als Transportschicht eignet sich das verbindungsorientierte TCP-Protokoll, das sicherstellt, dass die Daten vollständig und in korrekter Reihenfolge beim Empfänger ankommen.[13] In Abbildung 5.9 ist die Einbindung in das OSI-Referenzmodell[14] abgebildet.

11 [SPR11f]
12 IEEE 754 Norm siehe [Tan06, S.732 ff.], IEEE 754 Kompatibilität belegt unter [SPR11f] für den C6678-DSP sowie unter [Sev98] für Intel Prozessoren.
13 TCP (Transmission Control Protocol) wird ausführlich beschrieben in [ST10, S.297-454]
14 Beschrieben unter [Bor98, S.17 ff.]

OSI Layer 5-7	HPRCP
OSI Layer 4	TCP
OSI Layer 3	IP
OSI Layer 2	Ethernet
OSI Layer 1	CAT6a TP RJ45

Abbildung 5.9: Einordnung des HPRPC-Protokolls in das OSI-Referenzmodell

HPRPC über hardwarenahe Busprotokolle

Bei hardwarenahen Busprotokollen ist eine verbindungsorientierte Transportschicht nicht notwendig, da diese Protokolle auf sehr kurzen Hardware-Busleitungen betrieben werden und die Paketreihenfolge oder die Paketanzahl nicht wie beim IP-Protokoll durch zwischengeschaltete Router und Switches beeinflusst werden können. Etwa unterstützt der C6678-DSP das Hyperlink-Protokoll. Das HPRPC-Protokoll jedoch kann direkt in das Hyperlink-Protokoll eingebettet werden.

5.2.2 Protokollformat

Das HPRPC-Protokoll erlaubt das bidirektionale Senden von Paaren aus Anfrage (Request) und Antwort (Response). Es ist ein asynchrones Protokoll: Es dürfen mehrere Requests gleichzeitig bearbeitet werden und unbeantwortet sein. Über ein Indexfeld (Callindex) wird die Referenzierung eines Response-Pakets zum zugehörigen Request-Paket vorgenommen. In den Tabellen Tabelle 5.3 sowie Tabelle 5.4 wird eine Übersicht des Protokollformats gegeben.

Pos.	**Länge**	**Datentyp**	**Bedeutung**
1	2 Byte	uint16_t	Opcode (MSB 0)
2	2 Byte	uint16_t	Callindex (MSB 0)
3	4 Byte	uint32_t	Datalength
4	n Byte	uint8_t[]	Data

Tabelle 5.3: HPRPC Request

Pos.	**Länge**	**Datentyp**	**Bedeutung**
1	2 Byte	uint16_t	Callindex (MSB 1)
2	2 Byte	uint16_t	Returncode
3	4 Byte	uint32_t	Datalength
4	n Byte	uint8_t[]	Data

Tabelle 5.4: HPRPC Response

Das MSB des ersten Bytes zeigt an, ob es sich um ein Request- (MSB 0) oder ein Response-Paket (MSB 1) handelt. Da das MSB in der Response im Callindex eingetragen wird, ist der Zahlenraum der in einem Request-Paket verwendbaren Indizes auf den Bereich von 15 Bit begrenzt; das MSB des Callindex wird im Request auf 0 gesetzt. Bei einer Übertragung zwischen zwei Big-Endian-Systemen ist das zur Kennzeichnung verwendete MSB an Bit Nummer 0 des Datenpakets zu finden. Bei Little-Endian-Systemen befindet es sich wegen der Bytevertauschung dagegen an Bit Nummer 9 des Datenstroms.

Ein Response-Paket enthält einen Antwortcode im Returncode-Datenfeld und gegebenenfalls Antwortdaten, wie Berechnungsergebnisse. Wird ein von 0 (0=Success) verschiedener Antwortcode übermittelt, handelt es sich um einen Fehlercode (vgl. Tabelle 5.5). In diesem Fall befinden sich im Datenfeld keine Antwortdaten, sondern eine 8-Bit-ASCII[15] Zeichenfolge mit der textuellen Fehlerbeschreibung.

Pos.	Länge	Datentyp	Bedeutung
1	2 Byte	uint16_t	Callindex (MSB 1)
2	2 Byte	uint16_t	Returncode = Errorcode (>0)
3	4 Byte	uint32_t	Stringlength
4	n Byte	uint8_t[]	Error message

Tabelle 5.5: HPRPC Error Response

Alle in dieser Arbeit verwendeten Anfrage-Antwort-Paare inklusive deren Opcodes, Datenfelder sowie Returncodes werden in Anhang A.6 aufgelistet.

5.2.3 Datenkompression

Der Einsatz von DSPs lohnt sich nur dann, wenn die Summe aus Berechnungszeit und Overhead zur Übertragung der Bilddaten an die DSPs sowie zur Rückübertragung der Rechenergebnisse geringer ist als die Berechnungszeit einer lokalen Berechnung auf einem Mikroprozessor. Aus diesem Grund ist neben der Reduktion der Rechenzeit auch die des Overheads für die Datenübertragung essenziell. In Unterabschnitt 3.4.1 ist ein Verfahren dargestellt, das pro DSP-Chip ausschließlich den relevanten Ausschnitt der Bilddaten extrahiert. Die Datenmenge eines solchen Ausschnitts kann mit Bild-Kompressionsverfahren weiter verringert werden.

Für den Transfer der Bilddaten im HPRPC-Protokoll wird eine Lauflängencodierung[16] vorgeschlagen. Gerade bei Bildern, die einen hohen Anteil einfarbiger Flächen enthalten, arbeitet diese Methode besonders effektiv, da aufeinanderfolgende, gleichlautende

15 ASCII ist beschrieben unter [Tan06, S.144 ff.]
16 Bildkompressionsverfahren, beschrieben in [Str09, S.78]

Speicherstellen stark komprimiert werden. Eine Messung auf dem C6678-DSP hat ergeben, dass die Lauflängendecodierung aufgrund ihrer Einfachheit so schnell ausgeführt werden kann, dass die Empfangsdaten schon während des Kopierens vom Betriebssystemspeicherbereich in den Anwendungsspeicherbereich dekomprimiert werden können.[17] Datenempfang und Datenkompression können so synchron ablaufen, sofern die Dekompression schnell genug erfolgt. Somit resultiert aus der Dekompression kein Zeitverlust.

Bei einer derartigen Technik aus gleichzeitiger Dekompression und Datenempfang besteht jedoch die Gefahr von Pufferüberläufen im Netzwerkstack, wenn die Empfangspuffer schneller mit Netzwerkdaten bestückt werden, als die Dekompression neue Empfangspuffer abarbeiten kann. Eine zusätzlich zur Lauflängencodierung angewandte, verlustlose Kompression durch eine Entropiecodierung, welche die positionsunabhängige Symbolhäufigkeit berücksichtigt (z.B. Huffman-Codierung[18]), würde eine solche Gefahr des Pufferüberlaufs wegen des größeren Rechenaufwands erhöhen. Gleichzeitig wären bei einem nach der Lauflängencodierung gleichmäßigerem Histogramm[19] keine hohen Kompressionsraten erreichbar, weil die Entropie eines mit hoher Gleichmäßigkeit verteilten Datenstroms nicht mehr weit genug von der maximalen Entropie entfernt ist. Bei verlustbehafteten Kompressionsverfahren (z.B. Wavelet- oder DCT-basierte Verfahren[20]) ist die Gefahr eines Pufferüberlaufs wegen des hohen Rechenaufwands noch deutlich größer.

Eine Abwägung zwischen Dekodierungszeit bzw. Risiko eines Pufferüberlaufs und Kompressionsrate bzw. Netzwerkübertragungszeit weist für die in der späteren Performancemessung verwendeten Beispieldaten auf den Vorteil einer reinen Lauflängencodierung hin. Als Beispieldaten werden dort eine Auswahl an histologischen Schnitten mit großen, einfarbigen Flächen am Randbereich verwendet, die durch eine Lauflängencodierung auf 28,9% ($\mathcal{R}$) sowie 29,3% ($\mathcal{T}$) der Ursprungsdaten komprimiert werden. Berücksichtigt man zusätzlich die in Unterabschnitt 3.4.1 erläuterte Datenreduktion um etwa 50 %, die durch Teilübertragung der ausschließlich relevanten Bilddaten entsteht, erzielt man bei den Beispieldaten eine Reduktion der Netzwerkbelastung um 85% bei gleichzeitigem Ausschluss eines Pufferüberlaufs.

17 Texas Instruments stellt spezielle No-Copy-Socket-Operationen bereit, die einer Anwendung einen direkten Zugriff auf den Empfangspuffer im Netzwerkstack ermöglichen.[SPR12c, S.49] Damit kann die Anwendung Einfluss auf den Kopiervorgang vom Netzwerkstack in den Anwendungsspeicher nehmen.

18 [Huf52]

19 Bei histologischen Schnitten kann angenommen werden, dass die Digitalisierung aufgrund der Bildqualität das volle Helligkeitsspektrum ausnutzen wird. Nach Eliminierung gleichmäßiger Flächen durch die Lauflängencodierung ist bereits eine hohe Entropie erreicht.

20 [Str09, S.158 ff., S. 194 ff.]

5.3 PC-Clientsoftware

Wie eingangs erwähnt, sind eine DSP-Software und eine PC-Software zu erstellen. Es handelt sich zwischen PC- und DSP-Software um eine Client-Server-Beziehung[21], da die Programmablaufsteuerung von der PC-Software vorgenommen wird und die DSP-Software nur auf Anfrage des PCs eine Berechnung ausführt. Die PC-Clientsoftware arbeitet in zwei Modi: Sie ist einerseits in der Lage eine Registrierung lokal auf allen Rechenkernen des PCs durchzuführen, andererseits bietet sie die Möglichkeit, eine Registrierung unter Einbezug von über TCP verbundenen DSPs auszurechnen. Dadurch wird ein Vergleich der Rechenperformance eines PCs mit der Rechenperformance eines PCs mit zugeschalteten DSPs möglich.

In dem Modus, der eine Registrierung rein lokal auf dem PC durchführt, fallen folgende Arbeitsschritte an:

1. Bilddateien laden.
2. Gauß-Newton-Algorithmus mit Armijo-Line-Search durchführen.
3. Innerhalb dieses Algorithmus lokal auf allen Rechenkernen parallel die SSD, Jacobi- und (approximierte) Hesse-Matrix berechnen.
4. Nach Berechnungsende die grafischen Ergebnisse auf dem Bildschirm und die Rechenergebnisse auf der Konsole darstellen (vgl. Abbildung 5.4).
5. Messung und Ausgabe der für die Berechnung benötigten Zeit.

In dem anderen Modus, welcher entfernte DSPs einbezieht, werden anhand der per Kommandozeile übergebenen Parameter folgende Arbeitsschritte ausgeführt:

1. Bilddateien laden.
2. Bilddateien in Teilbilddaten für jeden DSP untergliedern (vgl. Unterabschnitt 3.4.1).
3. Teilbilddaten per Lauflängencodierung komprimieren.
4. Teilbilddaten über das HPRPC-Protokoll an alle DSPs verteilen.
5. Gauß-Newton Algorithmus mit Armijo-Line-Search durchführen.
6. Innerhalb dieses Algorithmus über das HPRPC-Protokoll die entfernte Berechnung von SSD, Jacobi- und (approximierter) Hesse-Matrix auf den DSPs beauftragen.
7. Nach Berechnungsende die grafischen Ergebnisse auf dem Bildschirm und die Rechenergebnisse auf der Konsole darstellen (vgl. Abbildung 5.4).

21 [Sta05, S.114 f.]

8. Messung und Ausgabe der für die Berechnung benötigten Zeit sowie der für die Übertragung der Bilddaten benötigten Zeit (Overhead).

Die in Kapitel 6 ausgewiesenen Performancedaten stammen aus Messungen mit einer eigens für diese Arbeit entwickelten DSP-Server- und PC-Clientsoftware. Die DSP-Serversoftware ist in der Programmiersprache C++ geschrieben, weil für die C6678-DSP-Chips ein C++ Compiler von Texas Instruments als Teil des kostenfreien Produkts 'Code Composer Studio' zur Verfügung gestellt wird.[22] Da die Sprache C++ für die DSP-Serversoftware eingesetzt wird, ist es sinnvoll, auch die PC-Clientsoftware in C++ zu erstellen, um nicht mehrere Programmiersprachen vermischen zu müssen. Ein weiterer Grund für die Wahl von C++ ist die Möglichkeit den in Unterabschnitt 5.1.2 vorgestellten Matlab-Coder so zu konfigurieren, dass C++ Code[23] generiert wird.

5.3.1 Entwicklungsumgebung

Die PC-Clientsoftware ist in 'Microsoft Visual C++'[24] erstellt. Es handelt sich dabei um eine integrierte Entwicklungsumgebung inklusive einem i86-spezifischen Compiler. Das daraus resultierende Programm lässt sich unter Microsoft Windows direkt und unter Linux über den Emulator 'Wine HQ'[25] ausführen.

Bibliotheken

Aus Gründen der zeitlichen Effizienz während der Softwareentwicklung werden mehrere Bibliotheken gebraucht:

- **BOOST**:[26] Kostenfreie Bibliothek, die teilweise in den Standard C++ V.11 aufgenommen wurde.[27].Verwendet werden Funktionen für Stringoperationen und das Parsing der Kommandozeile. Nutzbar unter der Boost Software License.[28]
- **OpenCV**:[29] Kostenfreie Bibliothek für Computervisualisierung. Genutzt werden die Möglichkeiten für das Laden und Anzeigen von Bildern. Unter BSD-Lizenz[30] veröffentlicht.

22 Code Composer Studio ist kostenfrei bei der Verwendung von XDS100 JTAG Emulatoren. Produktwebseite siehe [Texb].
23 Es handelt sich um C Code in einem C++ Gewand.
24 [Micb]
25 [Unb]
26 [Kar06]
27 [Gri11, S.30]
28 [BD]
29 [its]
30 [Ope]

- **cvplot**:[31] Bibliothek zur grafischen Darstellung verbundener Funktionswerte in einem OpenCV-Fenster. Leider bietet die Bibliothek keine Möglichkeit an, Abszisse und Ordinate zu beschriften, wie in den Abbildungen 5.4 sowie 5.5 an der SSD-Funktion ersichtlich. Über die Apache Lizenz V2.0[32] verwendbar.

5.3.2 Multicore-Parallelisierung mittels OpenMP

Im lokal rechnenden Modus werden die Berechnungen von SSD, Jacobi- und (approximierter) Hesse-Matrix durch den Einsatz von OpenMP parallelisiert. OpenMP wird von Microsoft Visual C++[33] unterstützt. Gleim und Schüle definieren OpenMP wie folgt:

„OpenMP (Open Multi-Processing) ist eine Spracherweiterung für die parallele Programmierung in C/C++ und Fortran [...]. Den Kern bilden spezielle Compileranweisungen, auch Direktiven oder Pragmas genannt, die in den Quelltext eines Programms eingefügt werden. Damit werden zum Beispiel die Bereiche eines Programms markiert, die parallel ausgeführt werden sollen.“[34]

Die entsprechenden Pragmas werden manuell in den aus Matlab generierten Code eingepflegt. An einem Code-Beispiel soll die Anwendung von OpenMP verdeutlicht werden. Es ist aus der von Matlab generierten Funktion zur Berechnung des SSD-Distanzmaßes entnommen.

```
1   int32_T i=0;
2   for (X_mni = 0; X_mni < i2 ; X_mni++)
3   {
4     b_X_mni = DSPRange[2] + (real32_T)X_mni;
5     for (X_i = 0; X_i < i3; X_i++)
6     {
7       // SSD berechnen ...

9       i++;
10    }
11  }
```

Um eine Parallelisierung auf allen Rechenkernen eines PCs durchzuführen ist in diesem Beispiel folgende Änderung des Quellcodes notwendig:

31 [Cha]
32 [Thea]
33 Bei dieser Studie wurde die Version 2008 verwendet und die Compileroption ’/openmp’ aktiviert.
34 [GS11, S.231]

```
1  #pragma omp parallel for reduction(+:SSD) shared(FP_idx_0, FP_idx_1, ↩
       FP_idx_3, FP_idx_4, y, width, i2, i3, w, BoundBox, DSPRange, pTData, ↩
       pRData, d) private(X_mni, b_X_mni, X_i)
2    for (X_mni = 0; X_mni < i2 ; X_mni++)
3    {
4      b_X_mni = DSPRange[2] + (real32_T)X_mni;
5      for (X_i = 0; X_i < i3; X_i++)
6      {
7        int32_T i = (X_mni*i3) + X_i;

9        // SSD berechnen ...
10     }
11   }
```

Der Compiler erkennt aus der Pragma-Anweisung, dass die äußere For-Schleife parallelisiert auszuführen sind. Ferner wird dem Compiler über das Schlüsselwort *reduction* mitgeteilt, dass die Variable SSD über alle Rechenkerne aufsummiert wird. Der Compiler wird intern Sorge dafür tragen, dass jeder Rechenkern eine eigene Kopie der Variable SSD bekommt. Die Kopien werden automatisch nach Ausführung aller Schleifendurchläufe über alle Kerne aufsummiert.

Mit dem Schlüsselwort *shared* wird dem Compiler mitgeteilt, auf welche Speicherstellen alle Kerne gleichzeitig zugreifen dürfen. Vor allem konstante Variablen mit reinem Lesezugriff können hier gefahrlos angegeben werden, da für einen reinen Lesezugriff einer unveränderlichen Variable keine kritischen Abschnitte[35] notwendig sind. Das Schlüsselwort *private* bewirkt, dass für die dort angegebenen Variablen pro Rechenkern eine private Kopie erstellt wird. Dieser Bereich bietet sich für beschreibbare Variablen an. Weil jeder Rechenkern nur auf seine eigene Kopie schreibend zugreift, sind auch hier keine kritischen Abschnitte notwendig. Das vermeiden von kritischen Abschnitten kommt der Ausführungsgeschwindigkeit zugute, da es nicht zu Wartesituationen kommt. Die für diese Studie erzeugte Software kommt völlig ohne kritische Abschnitte aus.

Das Hinzufügen der Pragma-Anweisung ist im obigen Beispiel noch nicht ausreichend, um die Parallelisierung korrekt durchzuführen. Die Variable *i* wird in der sequenziellen Variante bei jedem inneren Schleifendurchlauf inkrementiert. In OpenMP ist dies nicht mehr möglich, da die For-Schleifen parallel von mehreren Rechenkernen ausgeführt werden. Diese Problematik wird dadurch gelöst, dass der Zählerstand von *i* in jedem Schleifendurchlauf aus den Laufvariablen der beiden For-Schleifen (X_mni und X_i) berechnet wird.

35 Beschrieben unter [Sta05, S.242 ff.]

5.3.3 Parallele Datenübertragung mittels nicht blockierender Sockets

Die verteilte Ausführung auf entfernten DSP-Chips birgt die Gefahr, dass der zeitliche Overhead zur Übertragung der Bilddaten an die DSP-Chips so groß wird, dass sich insgesamt eine unvorteilhafte Ausführungszeit ergibt. In Abschnitt 4.3 sind hardwaretechnische Maßnahmen zur Reduzierung des Overheads dargestellt, insbesondere die Verwendung dedizierter Ethernetverbindungen in Sternstruktur. Um die Bandbreite dieser parallelen Ethernetleitungen auszulasten, ist es erforderlich, dass die Daten an alle angeschlossenen DSPs gleichzeitig versendet werden.

Es ist von Vorteil, in der Messsoftware die Datenkommunikation zu den DSP-Instanzen nicht mit blockierenden Sockets[36] durchzuführen. Blockierende Sockets halten den Programmfluss solange an, bis der Netzwerkvorgang beendet wurde. Sie zu parallelisieren bedeutet deswegen, dass pro DSP ein eigener Thread gestartet werden müsste, was unweigerlich zur Verwendung des Patterns "Thread-Per-Object"(hier ein Thread pro DSP) führt. In der Fachliteratur[37] wird dieses Pattern als Anti-Pattern beschrieben, da es die Skalierbarkeit eines Systems auf wenige, gleichzeitig erlaubte Gegenstellen dadurch einschränkt, dass ein System nicht beliebig viele Threads starten kann, ohne an Speicher und Ausführungsgeschwindigkeit zu verlieren. Eine bessere Alternative ist es, nicht blockierende Sockets zu verwenden, damit der ausführende Thread beim Aufruf von Socketfunktionen, wie Verbinden, Senden und Empfangen, nicht angehalten wird. Dadurch können mehrere Datenverbindungen zeitgleich von diesem Thread aus kontrolliert werden. Im Falle von TCP/IP kann unter Microsoft Windows über die Winsock-Bibliothek ein blockierendes Socketverhalten eingestellt werden.[38]

5.4 Embedded DSP-Serversoftware

Wie zu Beginn des Kapitels geschildert, wird neben der PC-Clientsoftware auch eine DSP-Serversoftware benötigt. Als Server startet sie einen Socket, zu dem sich der Client-PC verbinden kann. Die Serversoftware arbeitet nur auf direkte Anweisung über ein HPRPC-Kommando. Es werden folgende Kommandos angeboten:

1. **Store Image Data**: Der Client-PC kann mit diesem HPRPC-Kommando Bilddaten an die DSP-Serversoftware übertragen. Es werden Teilbilddaten (vgl. Unterab-

36 Blockierende und nicht blockierende Primitive siehe [Sta05, S.671]

37 „A recurring design error in would-be extreme systems is implementing [...] Thread per User, Thread Per Session, Thread Per Request or even Thread per Object. For example some firms have designed call servers that spawn a thread for each call. [...] 'Thread Per ...' patterns are rarely appropriate unless threads usually perform blocking operations[...]" [Uta05, S.102]

38 Dokumentation der ioctlsocket Funktion aus der Winsock-Bibliothek zur Steuerung des blockierenden Verhaltens siehe [Mica]

schnitt 3.4.1) lauflängencodiert übertragen, um die benötigte Netzwerkbandbreite zu reduzieren.

2. **Caclulate SSD**: Berechnet das Distanzmaß anhand aktueller Transformationsparameter w.

3. **Calculate SSD_Jacobian_Hessian**: Berechnet sowohl das Distanzmaß als auch die Jacobi- sowie die (approximierte) Hesse-Matrix.

Das Kommando 'Store Image Data' wird pro Registrierungsvorgang im Idealfall nur einmal zu Anfang aufgerufen. Es enthält gemäß des in Unterabschnitt 3.4.1 erörterten Verfahrens im Normalfall genügend Bilddaten für alle folgenden Iterationen. Sollten die übertragenen Bilddaten nicht ausreichen, weil im Laufe der Iterationen die Transformationsparameter so groß werden, dass die bei der Auswahl des Bildbereichs angenommenen Dimensionen überschritten werden, so wird von der PC-Clientsoftware erneut 'Store Image Data' mit einem aktualisierten Bildbereich aufgerufen.

Die Kommandos zur Berechnung von SSD, Jacobi- und (approximierter) Hesse-Matrix starten die zeitintensiven Berechnungen, die durch den Einsatz von DSP-Chips beschleunigt werden sollen. Hier wird für jeden Pixel eine Berechnungsvorschrift ausgeführt. Bei einem quadratischen Bild der Dimensionen 3000x3000 Pixel sind das beispielsweise 9.000.000 Ausführungen der Berechnungsvorschrift. Diese Berechnungen werden gleichmäßig auf alle DSP-Rechenkerne verteilt und die aufsummierten Endergebnisse als HPRPC-Antwort (Response-Paket) an die PC-Clientsoftware zurückgegeben.

5.4.1 Entwicklungsumgebung

Die integrierte Entwicklungsumgebung Code Composer Studio von Texas Instruments enthält einen für C6678-DSPs optimierten Compiler sowie JTAG-Treiber für die onboard JTAG-Debugging-Schnittstelle der Evaluationsmodule.[39] Es ist eine auf Eclipse[40] basierende und speziell auf die Bedürfnisse eines DSP-Entwicklers abgestimmte Umgebung. Dementsprechend finden sich zu den Evaluierungsmodulen passende GEL-Dateien[41] mit Memory-Maps und Initialisierungsscripten. Eine GEL-Datei beinhaltet hardwarespezifische Definitionen, wie Adressbereiche des Memory-Mapped-IO[42]. Es ist von Vorteil, wenn für die Evaluierungsmodule bereits vorgefertigte GEL-Dateien vorhanden sind, da die Erstellung einer hardwarespezifischen GEL-Datei so erst im fortgeschrittenen Entwicklungsprozess für die konkrete Zielhardware zu erfolgen braucht.

39 [Texb]
40 [Ecl]
41 General Extension Language, beschrieben unter [Texc]
42 Bei Memory-Mapped-IO sind Peripheriebausteine direkt über den Daten- und Adressbus ansteuerbar, siehe[Cat05, S.7, S.299].

Bibliotheken

Texas Instruments bietet zahlreiche frei verwendbare Bibliotheken. In Zusammenhang mit der durchgeführten Untersuchung wurde sowohl dieses Angebot als auch eine Drittanbieter-Implementierung einer BLAS-kompatiblen (Basic Linear Algebra System) Bibliothek evaluiert.[43] Dabei erwiesen sich folgende Bibliotheken für die vorliegende Problemstellung als geeignet:

- **TI Multicore Software Development Kit**: Basisbibliothek für die Betriebssystemplattform.
- **TI Network Development Kit**: Treiber und Funktionen für die Socket-Programmierung.
- **TI Inter Processor Communication Kit**: Funktionen für den Datenaustausch zwischen mehreren DSP-Rechenkernen.

Desweiteren stellt Texas Instruments hardwareoptimierte Bibliotheken für Matrizenalgebra und Bildverarbeitung zur Verfügung. Der in Kapitel 2 vorgestellte Ansatz ist jedoch so speziell, dass sich weder diese Bibliotheken noch die, wie in Unterabschnitt 5.4.3 vorgestellt, aktuelle OpenMP-Implementierung von Texas Instruments dafür eignet.

5.4.2 Partitionierung der Applikation

Software für den C6678-DSP unterscheidet sich von herkömmlicher PC-Software in Hinblick auf die parallele Abarbeitung von Programmen über mehrere Rechenkerne. Bei paralleler PC-Software muss der Anwendungsprogrammierer keine explizit zu programmierende Logik bezüglich der Speicherverwaltung oder der Verteilung von Binärcode auf die Rechenkerne erstellen (vgl. Unterabschnitt 5.3.2). Beim C6678-DSP hingegen ist die Anwendungssoftware so eng mit der Systemsoftware verbunden, dass sowohl der Quellcode für die Verwaltung des Mehrkernbetriebs geschrieben als auch die Betriebssystemkonfiguration angepasst werden muss. Die hierfür notwendigen Arbeitsschritte seien hier kurz skizziert.

Speicherpartitionierung

Software für den C6678-DSP wird in C (oder C++) entwickelt und ist wie jedes C-Programm im Arbeitsspeicher in die in Tabelle 5.6 aufgeführten Speicherbereiche unterteilt.[44]

43 GDD9000-CBLAS von Sundance Digital Signal Processing Inc, siehe [Sun].
44 [Tan06, S.768 ff.], [Sta05, S.67 ff.] sowie [Str98, S.906]

Bereich	Funktion
.stack	Stapelspeicher für Aufrufparameter, lokale Variablen sowie Rücksprungadressen.
.heap	Freispeicher, aus dem Speicherplatz explizit angefordert werden muss.
.data	Enthält Speicher, dessen Initialisierung vom Compiler vorgenommen wird (z.B. konstante Variablen).
.bss	Block Started By Symbol, enthält uninitialisierten Speicher. Hier wird vom Compiler Speicherplatz reserviert, der später vom Programm zu initialisieren ist.
.text	Enthält die Prozessorbefehle bzw. das auszuführende Programm.

Tabelle 5.6: Speicherbereiche eines C-Programms

Das C6678-Evaluierungsmodul verfügt über die in Tabelle 5.7 gelisteten Arten von RAM-Speicher.[45]

Speicherart	Speichermenge	Eigenschaften
L1	64 Kilobyte	Schnellster Speicher, für jeden Rechenkern privat.
L2	512 Kilobyte	Sehr schneller Speicher, für jeden Rechenkern privat.
L2S	4 Megabyte	Der Buchstabe 'S' seht für shared. Sehr schneller Speicher, gemeinsamer Zugriff für alle Rechenkerne.
MSMCS	4 Megabyte	Multicore-Shared-Memory, schneller Speicher, gemeinsamer Zugriff für alle Rechenkerne.
L3S	512 Megabyte	Größter und langsamster Speicher, gemeinsamer Zugriff für alle Rechenkerne.

Tabelle 5.7: Speicherarten des C6678-Evauierungsmoduls

Für den Betrieb einer Mehrkernanwendung sind auf dem C6678-DSP alle Speicherbereiche genau zu konfigurieren. Dabei muss für jeden Speicherbereich (mindestens für die in Tabelle 5.6 gelisteten) eingestellt werden, auf welcher Speicherart (siehe Tabelle 5.7) er reserviert werden soll. Diese Einstellungen lassen sich in beliebig tiefer Granularität vornehmen, so können selbst einzelnen C++ Variablen eine Speicherart über den Compiler mittels eines Pragma-Kommandos zugewiesen werden. Gleichzeitig können beliebige Heaps angelegt und deren Speicherart in der Betriebssystemkonfiguration festgelegt werden.

Grundsätzlich gilt es, einen guten Kompromiss zu finden, indem die Ressourcen nach der Häufigkeit ihres Zugriffs auf die jeweilige Speicherart konfiguriert werden. Dabei ist zu

45 Entnommen aus dem Datenblatt [SPR11d]. Eine Übersicht über L1 bis L3 Speicher findet sich in [Noe05, S.243]. Beim C6678-Evaluierungsmodul gibt es keinen L3-Cache, der L3-Speicher ist laut Datenblatt direkt mit den 512 Megabyte umfassenden DDR-3-Speicherbausteinen verbunden.

beachten, dass die Speicherkapazität einer Speicherart beim C6678-DSP umso geringer ist, je schneller der Speicher arbeitet. Eine solche Speicherkonfiguration als Teil der vollständigen Betriebssystemkonfiguration der Messsoftware geht aus Anhang A.3 hervor; die zusätzlich erforderliche Plattformkonfiguration ist aus Abbildung 5.10 ersichtlich.[46] Für eine HPC-Anwendung sind in Abbildung 5.10 die drei Eingabeelemente der untersten Dialogzeile von Bedeutung: Ist hier für Code- (.text-Segment), Data-(.bss-Segment), und Stack-Memory als Speicherart der L2-Ram eingestellt, ist eine optimale Performance zu erwarten, da zumindest die reine Programmausführung (ohne Heapzugriff) keinen Zugriff auf langsamen MCMS-Ram bzw. L3-Ram benötigt.

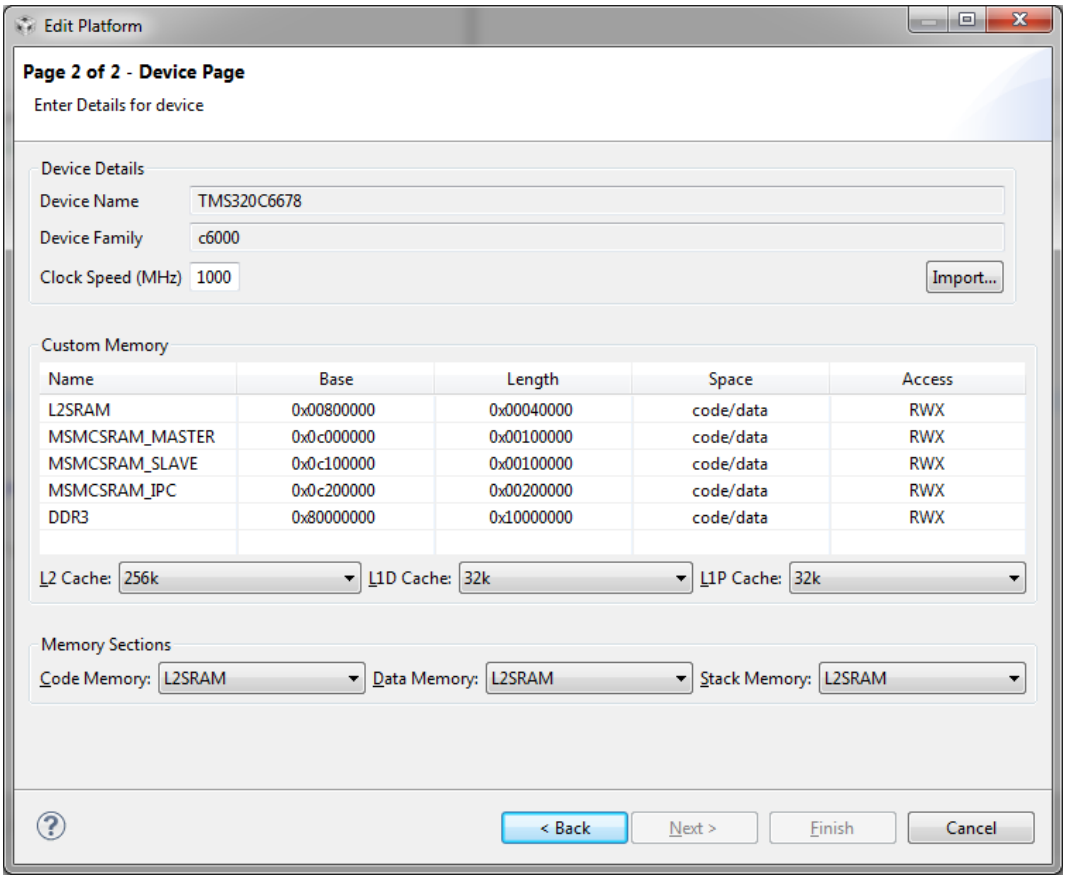


Abbildung 5.10: Speicherkonfiguration als Teil der Plattformkonfiguration

Einzig die Bilddaten müssen permanent im L3-Ram vorgehalten werden, weil die erforderliche Speichermenge nur im L3-Ram angeboten werden kann. In der Konfigurationsdatei erfordert dies einen Eintrag in die Speicherkonfiguration, dabei steht der Bezeichner 'DDR3' für die Verwendung von L3-Ram:

```
1 Program.sectMap[".picturebuff"] = {loadSegment: "DDR3", loadAlign:128};
```

46 Weiterführende Informationen zu Betriebssystem- und Plattformkonfiguration liefert die Dokumentation von Texas Instruments in [SPR11c, S.131 ff.].

Im Quellcode werden die Speicherbereiche für die Bilddaten über ein DATA_SELECTION Pragma dem so definierten Speicherbereich zugeordnet. Das DATA_ALIGN Pragma definiert dabei ein Alignment auf eine 128-Byte-Grenze. Dieses Alignment ist notwendig für die Ausführung von SIMD-Kommandos[47]:

```
1 #pragma DATA_SECTION(".picturebuff");
2 #pragma DATA_ALIGN(128);
3 uint8_t gpRVecStaticMemory[giRVecStaticMemorySize];
```

Ein einwandfreier Kompromiss in der Speicheraufteilung hat signifikante Auswirkungen auf die Systemperformance. So kann eine deutliche Verschlechterung der Netzwerkbandbreite beobachtet werden, wenn dessen Speicherkonfiguration ungünstig gewählt wird.[48] Auch für die in Unterabschnitt 5.4.3 dargestellten Probleme mit der OpenMP-Implementierung spielt die Speicherkonfiguration eine zentrale Rolle.

Für den L2-Speicher lässt sichdie Höhe des Anteils des unbelegten Speichers für die Verwendung als automatischer L2-Cache einstellen, wobei der automatische L2-Cache genauso gut deaktiviert werden kann. Das hat den Vorteil, dass die L2-Cache-Verwaltung programmgesteuert erfolgen kann. Ist der L2-Cache im automatischen Modus, müssen die einzelnen Rechenkerne vor dem Auslesen einer Speicherstelle im MSCSRAM- oder DDR3-RAM sicherstellen, eine Cache-Invalidierung auszulösen (Kommando 'Cache_inv'). Dies geschieht nicht automatisch und ist vom Applikationsprogrammierer vorzunehmen. Beim Beschreiben einer Speicherstelle muss entsprechend vom Applikationsprogrammierer ein Cache-Write-Back ausgelöst werden (Kommando 'Cache_wb').

Unterteilung in ein Master- und ein Slaveprogramm

Ein C6678-DSP lässt sich unter Verzicht auf ein Dateisystem betreiben. System- und Anwendungssoftware werden in einer großen Binärdatei zusammengefasst, die hier als Firmware-Image bezeichnet werden soll. Beim Booten lädt der DSP das Firmware-Image in den Hauptspeicher und beginnt beim Programmeinsprungspunkt mit der Ausführung. Anders als bei PC-Software arbeitet jeder Rechenkern auf einem eigenen Firmware-Image. Beim Booten wird zunächst der Hauptkern mit einem Firmware-Image versorgt und gestartet. Der Hauptkern lädt daraufhin für jeden Rechenkern ein eigenes Firmware-Image nach.[49]

47 SIMD wurde beschrieben in Unterabschnitt 4.1.1.
48 Der Einbruch der Netzwerkperformance wurde bei der Erstellung der Messsoftware häufig als Fehlerauswirkung tiefer liegender Fehlkonfigurationen beobachtet.
49 [Texd]

Um eine schnelle Programmausführung zu gewährleisten, sollten die gesamten Firmware-Images klein genug sein, um das .text-Segment permanent im L2-Speicher halten zu können. Es bietet sich an, für den ersten beim Booten angesprochenen Rechenkern ein großes Firmware-Image vorzuhalten, das im Folgenden als Master-Image bezeichnet wird. Es enthält neben dem Anwendungsprogramm (hier den Algorithmen für die Bildregistrierung) auch das Netzwerkprotokoll (hier HPRPC), den Code für den Bootvorgang sowie den Großteil des für das Betriebssystem benötigten Codes. Neben dem großen Master-Image werden die sieben weiteren Rechenkerne mit einem kleineren Firmware-Image versorgt, das weitestgehend nur die Anwendungssoftware, also die mathematischen Algorithmen, enthält. Als Bezeichnung wird Slave-Image gewählt. Eine solche Aufteilung reduziert den benötigten Umfang an permanent reserviertem L2-Speicher auf sieben von acht Kernen. Dadurch kann die Cacheauslastung und als Konsequenz davon die Rechenperformance gesteigert werden.

5.4.3 Multicore-Parallelisierung über IPC-Nachrichten

Wie im vorigen Abschnitt dargestellt, verwenden C667-DSPs jeweils einen eigenen Cache pro Rechenkern, der von der Applikation explizit mit dem gemeinsamen Speicher synchronisiert werden muss. OpenMP bietet leider keine Beschreibungsmöglichkeit für derartige Cache-Operationen an. Dies ist bei reiner PC-Software auch nicht notwendig, da dort keine explizite Steuerung des Caches zu erfolgen braucht. Um diesen Mangel zu umgehen, verlangt die OpenMP-Implementierung von Texas Instruments, die in OpenMP verwendeten Datenbereiche in einem Speicherbereich unterzubringen, der nicht vom L2-Cache beeinflusst wird.[50] Bedauerlicherweise ist ein Speicherzugriff ohne Cache deutlich langsamer, was die Rechenperformance insgesamt drosselt. Hinzu kommt, dass die Verwendung von OpenMP über Abhängigkeiten zu mehreren, zusätzlichen Betriebssystemmodulen zu einem so großen Firmware-Image führt, dass eine permanente Reservierung des .text-Segments (also des Applikationscodes) im L2-Ram nicht mehr möglich ist. Das verlangsamt weiterhin die Rechenperformance. Eine Performancemessung mit einer OpenMP basierten Version des Registrierungsalgorithmus ergibt einen Performanceverlust von ca. Faktor 10 im Vergleich zu einer Version, die den L2-Cache optimal nutzt. Aus diesem Grund wird für die DSP-Serversoftware eine alternative Lösung ohne OpenMP verwendet.

Für die Kommunikation mehrerer Rechenkerne untereinander bietet Texas Instruments eine Bibliothek für Interprozesskommunikation (IPC) an, die auf dem Austausch von Nachrichten beruht.[51] Über diese Bibliothek können die Rechenkerne beliebige C-Strukturen als Nachrichten untereinander austauschen. Dabei ist zu beachten, dass in den Nachrichten

50 Anleitung für die Einbindung von OpenMP in das SYS/BIOS Betriebssystem siehe [Texa].
51 [SPR11b]

enthaltene Zeiger Datenbereiche referenzieren können, die im privaten L2-Cache des Rechenkerns gespiegelt sind. Derartige Datenbereiche müssen, wie in Abschnitt 5.4.2 beschrieben, über explizit aufzurufende Kommandos mit den geteilten Speicherbereichen synchronisiert werden. Z.B. wird bei der Bildregistrierung ein Zeiger auf die Bilddaten im L3-Ram als Teil der IPC-Nachrichten benötigt. Der Master-Rechenkern, der die Bilddaten im L3-Ram ablegt, hat nach dem Speichern der Bilddaten ein Cache-Write-Back-Kommando auszuführen. Die Slave-Rechenkerne müssen vor dem ersten Zugriff auf die Bilddaten ein Cache-Invalidierungs-Kommando ausführen. Für weitere Zugriffe der Slave-Rechenkerne auf die unveränderten Bilddaten ist von einem erneuten Cache-Invalidierungs-Kommando abzusehen, um unnötigen Datentransfer zwischen L3-Ram und L2-Cache zu vermeiden.

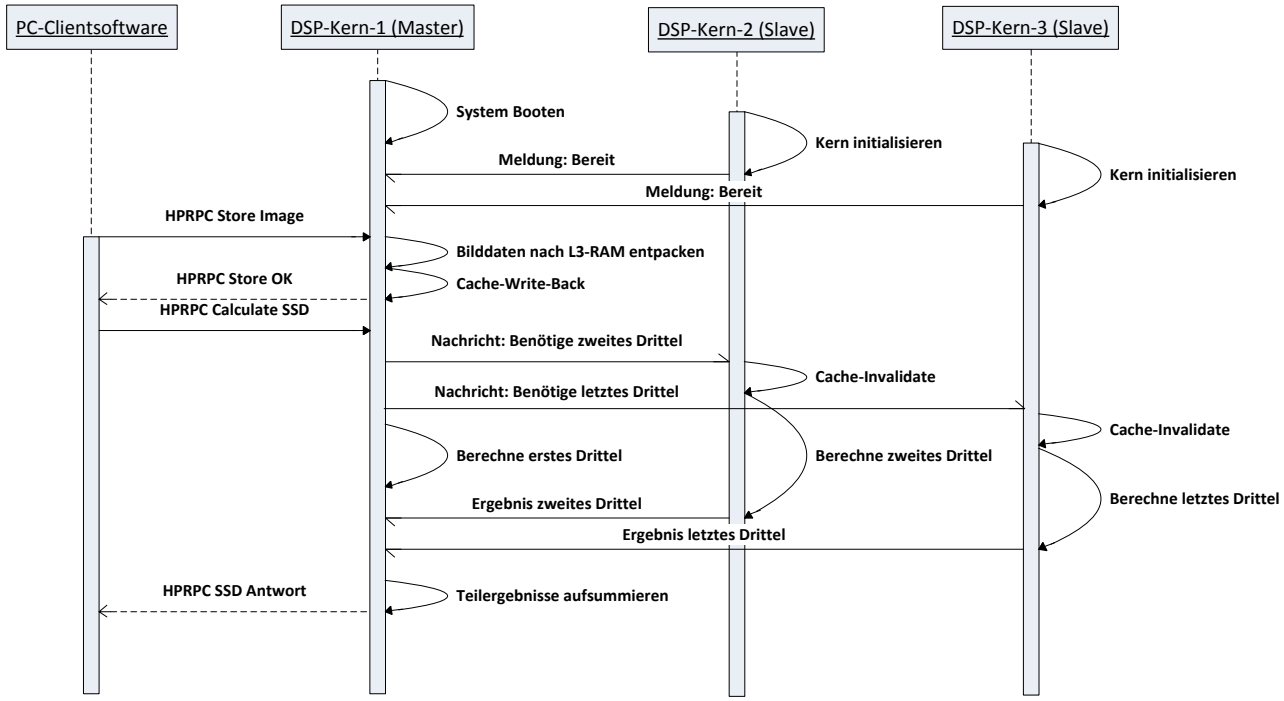


Abbildung 5.11: Sequenzdiagramm zur Verdeutlichung der IPC-Kommunikation

In Abbildung 5.11 ist eine Sequenzdiagramm dargestellt, das die IPC-Kommunikation der Rechenkerne im Gesamtkontext verdeutlicht. Um die wesentlichen Aspekte hervorzuheben, ist die Anzahl der Rechenkerne auf drei reduziert und lediglich der Beginn der Gesamtkommunikation dargestellt. Aus dem Diagramm ist ersichtlich, dass auf dem DSP zunächst das Master-Image auf dem Master-Rechenkern gebootet wird und die Slave-Images auf den Slave-Rechenkernen starten. Der erste IPC-Nachrichtenaustausch ist eine Fertig-Meldung der Slave-Kerne an den Masterkern. Kommt über das HPRPC -Protokoll eine Anfrage zum Speichern von Bilddaten, so bearbeitet der Master-Rechenkern diese Anfrage ohne weitere IPC-Nachrichten an die Slave-Kerne. Bei einer Anfrage, die verteilte Berechnungen erfordert, teilt der Master-Rechenkern jedoch die zu berechnenden

Daten gleichmäßig auf und sendet eine IPC-Nachricht an jeden Slave-Rechenkern darüber, welche Art der Berechnung auszuführen ist und welchen Verantwortungsbereich der Slave-Kern abdeckt (z.B. 'zweites Drittel'). Auch die Rahmenparameter der Berechnung (z.B. die Transformationsparameter w) werden über diese IPC-Nachricht an die Slave-Rechenkerne übergeben.[52] Hat ein Slave-Rechenkern seine Berechnung beendet, sendet er eine IPC-Nachricht mit einer Fertig-Meldung sowie den Berechnungsergebnissen zurück an den Master-Rechenkern. Hat dieser seine eigene Berechnung vollendet und verfügt über die Ergebnisse aller Slave-Rechenkerne, kann er das Gesamtergebnis für den hier verwendeten Algorithmus durch einfaches Aufsummieren der Teilergebnisse berechnen und über eine HPRPC-Antwort an die PC-Clientsoftware versenden.

5.4.4 Datenübertragung

Eine netzwerkfähige Multicore-Anwendung für einen C6678 erfordert eine umfangreiche Auseinandersetzung mit der Konfiguration des SYS/BIOS -Betriebssystems von Texas Instruments. Bei der Untersuchung konnten bereits kleine Unstimmigkeiten zum Kollabieren des Netzwerkstacks von Texas Instruments führen, ein Beispiel dafür findet sich in Unterabschnitt 5.4.3. Um eine C6678-Anwendung netzwerkfähig zu machen, ist die NDK-Bibliothek von Texas Instruments einzubinden und ein Quellcode für die sogenannten OSAL-Funktionen (Operating System Abstraction Layer) zu erstellen. Dabei handelt es sich um eine Hardwareabstraktionsschicht, in welcher Einsprungpunkte für das Betriebssystem vom Systemprogrammierer mit Quellcode für das Management von Systemressourcen, wie Speicher oder Semaphoren, zu füllen sind.[53]

Um große Mengen an Bilddaten schnell empfangen zu können, sind über die in Abschnitt 5.4.2 erläuterten Mechanismen zur Speicheraufteilung optimale Bedingungen für eine schnelle Annahme von Netzwerkpaketen zu schaffen. Für die DSP-Serversoftware kann eine Netzwerkbandbreite von ca. 250 MBit/s Nach Anwendung von zwei Maßnahmen beobachtet werden. Einerseits müssen die Netzwerkpakete im MSMCS-Ram abgelegt werden:

```
1 Program.sectMap[".far:NDK_PACKETMEM"]= {loadSegment: "MSMCSRAM_MASTER",
    loadAlign: 128};
2 Program.sectMap[".far:NDK_OBJMEM"]   = {loadSegment: "MSMCSRAM_MASTER",
    loadAlign: 16};
```

52 Für eine bessere Übersicht wurde auf eine vollständige Darstellung der Ein- und Ausgabeparameter im Sequenzdiagramm verzichtet.

53 [Tex11]

Andererseits ist es notwendig, die TCP-Puffer des Betriebssystems auf 65535 Bytes [54] zu vergrößern:

```
1       size = 65535;
2       CfgAddEntry( hCfg, CFGTAG_IP, CFGITEM_IP_SOCKTCPTXBUF,
3                    CFG_ADDMODE_UNIQUE, sizeof(uint), (uint8_t *)&size, 0 )
                         ;

5       size = 65535;
6       CfgAddEntry( hCfg, CFGTAG_IP, CFGITEM_IP_SOCKTCPRXLIMIT,
7                    CFG_ADDMODE_UNIQUE, sizeof(uint), (uint8_t *)&size, 0 )
                         ;
```

Struct-Alignment

Das HPRPC-Protokoll überträgt direkt Binärdaten. In der PC-Clientsoftware wird mit der Anweisung '#pragma pack(1)' bestimmt, dass Strukturen der Programmiersprache C++ so kompakt wie möglich - also ohne Füllbytes - im Speicher angelegt werden sollen.[55] Dadurch können diese Strukturen direkt über ein HPRPC-Kommando an den Zielrechner übermittelt werden. Die Binärdaten einer so übertragenen Struktur werden beim Datenempfang am DSP direkt im Netzwerkspeicher des Betriebssystems abgelegt. Dabei wird der Speicherbereich 'NDK_PACKETMEM' genutzt, dessen Konfiguration im vorangehenden Abschnitt abgebildet wurde. Die empfangenen Daten in diesem Speicherbereich werden bezüglich des Data-Alignments an einer beliebigen Stelle platziert. Ein C6678-DSP stellt jedoch besondere Anforderungen an das Data-Alignment abhängig vom verwendeten Datentyp.[56] Insbesondere ist es beim C6678-DSP notwendig, dass eine Fließkommazahl an einer 4-Byte-Grenze beginnt. Deswegen ist ein eingehendes Datenpaket, welches Fließkommazahlen enthält vor dem Zugriff durch den DSP an eine Speicherstelle umzukopieren, die an einer 4-Byte-Grenze anfängt. Enthält ein Datenpaket nur einen Bytestrom (wie die Bilddaten), ist ein solches Umkopieren überflüssig. Die Bilddaten können direkt aus dem 'NDK_PACKETMEM' -Speicherbereich ausgelesen und an die Lauflängendekodierung (s. Unterabschnitt 5.2.3) weitergegeben werden.

54 Größtmögliche mit einem uint16_t Datentyp darstellbare Zahl, entspricht knapp 64 Kilobyte.
55 [Micc]
56 „Because the C64x can only issue one non-aligned memory access per cycle, programs should focus on using aligned memory accesses whenever possible“[SPR11e, S.6-37]

5.4.5 Speicherfragmentierung

Für ein eingebettetes System ist es nicht unüblich, dass die eingebettete Software über mehrere Jahre ohne Neustart lauffähig sein muss. Hierfür gibt es zahlreiche Beispiele, vom häuslichen Kühlschrank bis hin zur Steuerung der Notabschaltung eines Atomkraftwerks. Der Ersteller einer eingebetteten Software hat im Softwaredesign zu berücksichtigen, dass es nicht zu einer Heap-Fragmentierung kommen kann, die unter ungünstigen Bedingungen zu einem Systemausfall führt. Dazu kommt es, wenn häufig Speicherbereiche von einem Heap reserviert und wieder freigegeben werden. Im ungünstigen Fall entstehen mit zunehmender Zeit immer mehr Lücken zwischen den reservierten Speicherbereichen, die nicht durch neue Speicherreservierungen belegt werden können, da die Lücken zu klein sind, um der Anforderung eines neuen, zusammenhängenden Speicherbereichs zu genügen. Insbesondere bei Prozessoren ohne Speicherverwaltungseinheit (MMU) - wie dem C6678 - kommt es leicht zu einer Heap-Fragmentierung, weil der Speicher nicht über einen Paging-Mechanismus in virtuelle Speicherbereiche unterteilt wird.[57] Kommt es dazu, dass diese Speicherlücken so viel Speicher belegen, dass eine Anforderung nach neuem Speicher nicht mehr erfüllt werden kann, ist das eingebettete System nicht mehr arbeitsfähig.

Um einer derartigen Speicherfragmentierung vorzubeugen, so dass die DSP -Chips über viele Jahre ohne Neustart ihren Dienst verrichten, werden in der DSP-Serversoftware niemals Speicherbereiche dynamisch von einem Heap angefordert. Stattdessen werden alle Speicherbereiche fest in einer sinnvollen Maximalgröße reserviert, so dass der Speicher beim Systemstart einmal fest zugewiesen und bis zum Ausschalten des Systems nicht mehr freigegeben wird.

5.5 Strukturiertes Vorgehen bei der Implementierung

Die im Rahmen dieser Arbeit entwickelte Softwarelösung ist hochkomplex und für das verfügbare Zeitkontingent von etwa zwei Entwicklermonaten hoch ambitioniert. Für die Implementierung bietet sich ein agiles Vorgehen in kurzen Iterationen an. Die von Robert. C. Martin[58] vorgeschlagene Iterationszeit von zwei Wochen[59] wird dabei wegen der kurzen Projektdauer sowie dem Umstand, dass es ein Einpersonen-Projekt ist, auf eine Woche verkürzt. Dazu werden in wöchentlichen Meilensteinen funktionsfähige Zwischenprodukte entwickelt. Im Gegensatz zu einem monumentalen Vorgehensmodell[60], bei dem sich erst nach einer umfangreichen Planungsphase herausstellt, ob die Software funktionsfähig

57 [Tan06, S.462 ff.]
58 Mitbegründer des "Manifesto for Agile Software Development“, siehe [Mar03, S.4].
59 [Mar03, S.12]
60 [BES08, S.619]

sein kann, wird hier zu einem frühen Zeitpunkt festgestellt werden, ob die eingesetzten Konzepte stabile Zwischenprodukte ergeben, was wiederum eine Prognose über die Funktionsfähigkeit des Endprodukts zulässt. In Tabelle 5.8 werden die wöchentlichen Meilensteine (Iterationen) aufgelistet, sie sind direkt aus den Projektplanungsartefakten entnommen.

Meilenstein	**Termin**	**Zielsetzung**
1	18.05.2012	Matlab: Ermittlung aller Funktionen für das Distanzmaß ($\mathcal{P}, \mathcal{A}, \mathcal{T}, \mathcal{D}_r, \mathcal{D}_s$)
2	25.05.2012	Matlab: Jacobi-Matrizen der Funktionen des vorangegangenen Meilensteins bilden und prüfen, sowie die approximierte Hesse-Matrix bilden
3	01.06.2012	Matlab: Gauß-Newton Algorithmus
4	08.06.2012	Umstellung von matrizenbasierter auf pixelbasierter Berechnung (Ausmultiplizieren der Jacobi-Matrizen). Das so gewonnene Matlab-Programm wird über Matlab-Coder auf eine PC-Software übertragen. Dieser Meilenstein bildet das erste Produkt, es handelt sich um die Software für die Performancemessung einer lokal auf dem PC ausgeführten Bildregistrierung.
5	15.06.2012	In Matlab wird der Algorithmus mit einer simulierten Verteilung auf vier (virtuelle) parallele Recheninstanzen ausgestattet, die sequenziell aufgerufen werden können. Solange sie autark arbeiten, sind sie für die Simulation der späteren Parallelität auf vier DSP-Bausteinen geeignet. In Matlab wird dieser Algorithmus mit einer simulierten Verteilung der Teilbilddaten, wie in Unterabschnitt 3.4.1 beschrieben, ausgestattet. Dies geschieht zunächst noch ohne Lauflängencodierung.

6	22.06.2012	Die PC-Software wird um eine Simulationsschicht erweitert, die den Quellcode der vorangegangenen Meilensteine simuliert. Die Datenübergabe zum Simulator erfolgt über simulierte Socketkommandos. Dabei wird ein Speicherpuffer eingesetzt, dessen Größe der einer MTU (Maximum Transfer Unit, in einer solchen Stückelung erhält die spätere DSP-Software ihre Daten. [ST10, S.54 f.]) ähnelt; beispielsweise 1400 Bytes. Das simuliert den fragmentierten Empfang der Datenpakete auf dem späteren Empfänger. Über diesen Speicherpuffer wird bereits das später zum Einsatz kommende Datenübertragungsprotokoll übermittelt. Um die Simulation zu komplettieren, werden die Lauflängencodierung und -decodierung hinzugefügt. Die Lauflängendecodierung muss dabei kompatibel zu einem fragmentierten, blockweisen Datenempfang sein.
7	29.06.2012	Nachdem die Software im vorangegangenen Meilenstein in der Simulationsumgebung fertiggestellt ist, ist es notwendig, das Betriebssystem des DSP zusammenzustellen. Benötigt wird ein netzwerkfähiges Betriebssystem mit sehr schneller Ausführbarkeit mathematischer Algorithmen sowie der Möglichkeit die Mehrkernarchitektur der DSPs anzusteuern. Für diese Tätigkeit ist neben der Konfiguration auch eine Systemprogrammierung in der Programmiersprache C notwendig. Teil dieses Meilensteins sind ferner der Aufbau und die Konfiguration der Testhardware.
8	06.07.2012	Nachdem simulierter Quellcode und Betriebssystem zur Verfügung stehen, kann Embedded-DSP-Software basierend auf dem Simulations-Quellcode entwickelt werden. Die Notwendigkeit einer Simulation ergibt sich daraus, dass die Entwicklung auf DSP-Hardware deutlich langwieriger ist als bei PC-Software. Alleine der Programmstart auf 32 Rechenkernen kann mehrere Minuten dauern. Durch die PC-Simulation wird somit Zeit gespart.

9	13.07.2012	Auf diese Weise entstehen eine PC- und eine DSP-Software, die zwar auf mehreren DSPs, jedoch nur auf einem Rechenkern rechnen. Abschließend ist es notwendig, sowohl die PC- als auch die DSP-Software so zu erweitern, dass alle Rechenkerne der jeweiligen Hardware in die Berechnung einbezogen werden.

Tabelle 5.8: Planung der wöchentlichen Meilensteine

Vor dem 15.05.2012 gibt es keine Meilensteine wegen der initialen Planungs- und Rechercheaktivitäten. Nach dem 13.07.2012 steht eine Softwarelösung für die messtechnische Untersuchung bereit.

5.5.1 Qualitätssicherung

Die einzelnen Arbeitsschritte lassen erkennen, dass die Entwicklung einer solchen Softwarelösung äußerst komplex ist. Es empfiehlt sich eine permanente Qualitätssicherung zu betreiben, damit das Vorhaben erfolgreich abgeschlossen werden kann. Als probates Mittel haben sich automatisierte Tests erwiesen, die mit bekannten Registrierungsproblemen einen Abgleich von neuen Softwareteilen mit der zuvor erstellten Software ermöglichen. Dabei ist nach dem Verfahren TDD (Test Driven Development) zuerst ein automatisierter Test zu erstellen, dessen Ausführung nicht erfolgreich ist. Nachdem die Software den Test erfolgreich durchlaufen kann, wird der Test bei Änderungen erneut ausgeführt.[61] Werden mit der geänderten Software identische Ergebnisse zur vorherigen Version erzielt, so ist die Wahrscheinlichkeit hoch, dass die vorherigen Änderungen nicht zu einer Regression, also nicht zu einer Verschlechterung in einem vorher funktionsfähigen Programmfeature, geführt haben.

5.6 Wissenschaftliche Vergleichbarkeit der Performance unterschiedlicher Systeme

Die PC- und DSP-Software wurde entwickelt, um die Performance von DSP-Bausteinen mit herkömmlicher PC-Technik zu vergleichen. Eine derartige Messung der DSP-Performance ist vor allem dann aussagekräftig, wenn die Algorithmen auf eine hohe Ausführungseffizienz hin optimiert wurden. Dann ist ein Vergleich mit alternativen Lösungen möglich, die

61 [Mar11b]

auf anderer Technologie basieren, denn es kann davon ausgegangen werden, dass die vergleichbaren Lösungen auch performanceoptimiert sein werden.

Bei dem Vergleich zwischen der Performance einer lokalen Berechnung auf einem PC und der Performance einer entfernten Berechnung auf DSP-Chips (incl. Overhead für den Daten Hin- und Rücktransport) ist zu beachten, dass ein solcher Vergleich nur dann wissenschaftlich aussagekräftig ist, wenn beide Lösungen annähernd gleichen Voraussetzungen unterzogen werden. Würde beispielsweise die Performance einer unoptimierten PC-Version mit einer hochoptimierten DSP Version verglichen werden, so wäre die Aussagekraft des Vergleiches nur eingeschränkt, da sich die Nähe zur maximal möglichen Systemauslastung zwischen beiden Lösungen unterscheiden würde.

Um eine hohe Vergleichbarkeit zu erreichen, wird vorgeschlagen, sowohl den Code der PC-Version als auch den Code der DSP-Version aus identischen Algorithmen (z.B. Matlab m-Dateien) zu generieren. Ferner wird vorgeschlagen, beide Lösungen auf der maximalen Anzahl an verfügbaren Rechenkernen zu parallelisieren. Werden Chipabhängige Optimierungen vorgenommen, so erreicht man eine hohe Vergleichbarkeit, wenn diese Optimierungen auf beiden Plattformen in vergleichbarer Qualität vorgenommen werden. Nachdem beispielsweise eine SIMD-Optimierung des DSP-Quellcodes durchgeführt wird, ist eine gleichwertige Optimierung im PC-Quellcode (z.B. SSE von Intel[62]) durchzuführen.

Findet sich eine Optimierungstechnik, welche nur auf einer Plattform (also entweder nur PC oder nur DSP) verfügbar ist, so wird sie als spezifischer Vorteil dieser Plattform gewertet und nicht etwa deswegen ausgelassen, weil sie auf der jeweils anderen Plattform nicht zur Verfügung steht. Als Beispiel seien hier die in Abschnitt 5.4.2 dargestellten Möglichkeiten der Speicherkonfiguration beim C6678-DSP genannt.

5.6.1 Rechengenauigkeit

Bei der Wahl der Rechengenauigkeit ist eine Abwägung zwischen der Konvergenzrate des Registrierungsalgorithmus und der Rechengeschwindigkeit vorzunehmen. Sowohl auf einem C6678-DSP als auch auf einem Intel-basierten PC werden 32-Bit- sowie 64-Bit-Fließkommazahlen unterstützt.[63] Eine Berechnung mit lediglich 32-Bit-Genauigkeit kann bewirken, dass sich die Anzahl notwendiger Iterationen erhöht, was den Algorithmus verlangsamt. Bei einer Berechnung mit 64-Bit-Fließkommazahlen dauert allerdings jede Iteration für sich länger. Für die DSP-Technologie ist es von Vorteil, 32-Bit-Fließkommazahlen

62 Streaming SIMD Extensions [Tan06, S.56]

63 Wie in Abschnitt 5.2.1 belegt, unterstützt ein Intel-basierter PC den IEEE 754 Standard. Dieser definiert sowohl 32- als auch 64-Bit Fließkommazahlen.

zu verwenden: Bei 32-Bit-Genauigkeit kann auf C6678-DSPs stärker von der Parallelisierung in Bezug auf VLIW und SIMD profitiert werden, sie rechnen überproportional schneller.[64] Diese Entscheidung hat zur Folge, dass die Stärken der DSP-Bausteine besser zur Geltung kommen.

Nachdem Mathematik, Algorithmik, Hard- und Software des hier angewendeten Verfahrens zur Bildregistrierung erläutert wurde, soll im Anschluss mit der Analyse der gemessenen Ausführungsgeschwindigkeit fortgefahren werden.

64 „delivering 160 single-precision GFLOPS and 60 double-precision GFLOPS“ [SPR11g]

6 Messtechnische Untersuchung

In den vorangegangenen Kapiteln ist dargestellt, wie ein Registrierungsalgorithmus mathematisch hergeleitet und in folgenden Systemaufbauten betrieben werden kann:

1. **Lokal** auf einem PC im Mehrkernbetrieb mittels OpenMP
2. **Verteilt** auf vier an einen PC angeschlossene Mehrkern-DSPs

Eine zentrales Kriterium zur Bewertung einer DSP-basierten Lösung ist für das Fraunhofer-Institut der Performanceunterschied zwischen diesen beiden Systemvarianten. Dafür ist ein Messverfahren zu entwerfen.

6.1 Messverfahren

Die im vorigen Kapitel beschriebenen Programme enthalten fest definierte Zeitpunkte für eine Zeitmessung. Bei einer verteilten Registrierung sind dies die Messpunkte gemäß Tabelle 6.1, bei einer lokalen gemäß Tabelle 6.2.

Messpunkt Nummer	**Zeitpunkt**
MP1	Zum Programmstart vor Beginn des TCP-Verbindungsaufbaus.
MP2	Nach Empfang des letzten HPRPC-Anwortpakets zur Quittierung des Erhalts der Bilddaten.
MP3	Nach Beendigung des Registrierungsalgorithmus, sobald das Endergebnis berechnet wurde.

Tabelle 6.1: Messpunkte für eine verteilte Bildregistrierung

Messpunkt Nummer	**Zeitpunkt**
MP2	Unmittelbar vor Beginn der Bildregistrierung
MP3	Nach Beendigung des Registrierungsalgorithmus, sobald das Endergebnis berechnet wurde.

Tabelle 6.2: Messpunkte für eine lokale Bildregistrierung

Die Differenz zwischen MP2 und MP1 kennzeichnet den Overhead für die Komprimierung, Übertragung und Dekompression der Bilddaten an die vier entfernten DSPs. Da bei

einer lokalen Registrierung keine Bilddaten übertragen werden müssen, existiert für die lokale Registrierung weder ein Messpunkt MP1 noch eine Bewertung des Overheads. Die Differenz zwischen MP3 und MP2 hingegen lässt sich in beiden Systemaufbauten berechnen: Es ist die Berechnungszeit für eine komplette Bildregistrierung. Bei diesem Wert ist zu beachten, dass die Anzahl der durchgeführten Iterationen eine wesentliche Einflussgröße ist.

Es stehen für die Messung vier Stück C6678-DSP-Evaluierungsmodule von Texas Instruments zur Verfügung. Der verteilte Systemaufbau ist algorithmisch bedingt mit einem sowie mit vier DSPs messtechnisch bestimmbar. Eine Messung mit zwei oder drei DSPs ist dagegen nicht möglich, weil der in Unterabschnitt 3.4.1 vorgestellte Algorithmus nur eine Chipanzahl unterstützt, deren Quadratwurzel ganzzahlig und entweder 1 oder eine gerade Zahl ist (also 1, 4, 16, 36, 64 ...). Über das in Abbildung 5.2 vorgestellte Kommandozeileninterface kann die zu verwendende Anzahl der DSP-Rechenkerne bestimmt werden. Damit kommt man auf die in Tabelle 6.3 dargestellte Messreihe, die als Kommandozeilenscript ausgeführt werden kann.

Messung Nr.	Anzahl DSPs	Rechenkerne pro DSP	Rechenkerne insgesamt
1	1	1	1
2	1	2	2
3	1	3	3
4	1	4	4
5	1	5	5
6	1	6	6
7	1	7	7
8	1	8	8
9	4	1	4
10	4	2	8
11	4	3	12
12	4	4	16
13	4	5	20
14	4	6	24
15	4	7	28
16	4	8	32

Tabelle 6.3: Messpunkte für eine verteilte Bildregistrierung

Für den Vergleich mit einer lokalen Registrierung wird eine ähnliche Messreihe gemäß Tabelle 6.4 durchgeführt. Hierbei werden unterschiedliche PCs zur Messung herangezogen und es wird jeweils mit der maximal verfügbaren Anzahl an Rechenkernen gerechnet. Dadurch wird berücksichtigt, dass ein PC mit wenigen Rechenkernen aus insgesamt weniger leistungsfähigen Komponenten als ein Hochleistungsrechner mit einer zweistelligen Anzahl an Rechenkernen besteht.

Gerät Nr.	Prozessor	Anzahl CPUs	Rechenkerne (insgesamt)	Taktfrequenz
PC-1	Intel Xeon Dualcore 3065	1	2	2,33 GHz
PC-2	Intel Corei5 2410M	1	4	2,3 GHz
PC-3	Intel Xeon E5620	2	8	2,4 GHz
PC-4	Intel Xeon E5645	2	12	2,46 GHz

Tabelle 6.4: Messpunkte für eine lokale Bildregistrierung

Die Messreihen werden mit den in Tabelle 6.5 aufgeführten Bilddaten ausgeführt. Es handelt sich um Bilddaten unterschiedlicher Größenordnungen, damit der Einfluss der Bilddatenmenge und des Overheads für die Übertragung der Bilddaten untersucht werden kann. Es ist zu beachten, dass für die Bildregistrierung immer zwei Bilder verwendet werden, das Referenz- und das Templatebild. Die insgesamt verarbeitete Menge an Bilddaten beträgt deshalb einen doppelt so hohen Wert, wie in Tabelle 6.5 angegeben.

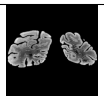
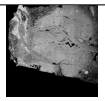
Dimensionen	Größe in Byte	Motiv	Referenzbild
512x512 Pixel	256 KByte	Gehirn	
3000x3000 Pixel	8,6 MByte	Tumor	

Tabelle 6.5: Messpunkte für eine lokale Bildregistrierung

6.2 Messergebnisse

Die im vorangegangenen Abschnitt beschriebenen Messungen beanspruchen bei mehrmaliger Ausführung und Mittelung der Werte eine bestimmte Zeitspanne, die in Abbildung 6.1 sowie Abbildung 6.2 aufgeführt ist. Die erste der beiden Abbildungen zeigt eine Messung für die Bildgröße 512x512 Pixel, die zweite Darstellung gilt für 3000x3000 Pixel. Die Abbildungen enthalten jeweils drei Graphen. Links findet sich die Messung der rein lokalen Registrierung, rechts daneben ist die verteilte Registrierung auf einem bzw. vier DSPs dargestellt. Die Graphen fallen nicht immer gleichmäßig mit steigender Anzahl an Rechenkernen ab, dies liegt am starken Einfluss der Rundungsfehler bei 32-Bit-Fließkommagenauigkeit. In unterschiedlichen Szenarien kommt es zu einem jeweils anderen Wert an aufkumulierten Rundungsfehlern, was dazu führt, dass die Bildregistrierung in manchen Fällen früher abgeschlossen ist. In Abbildung 6.3 ist die Rechengeschwindigkeit einer Bildregistrierung visualisiert. Dabei wurde darauf geachtet, dass die zuvor beschriebene

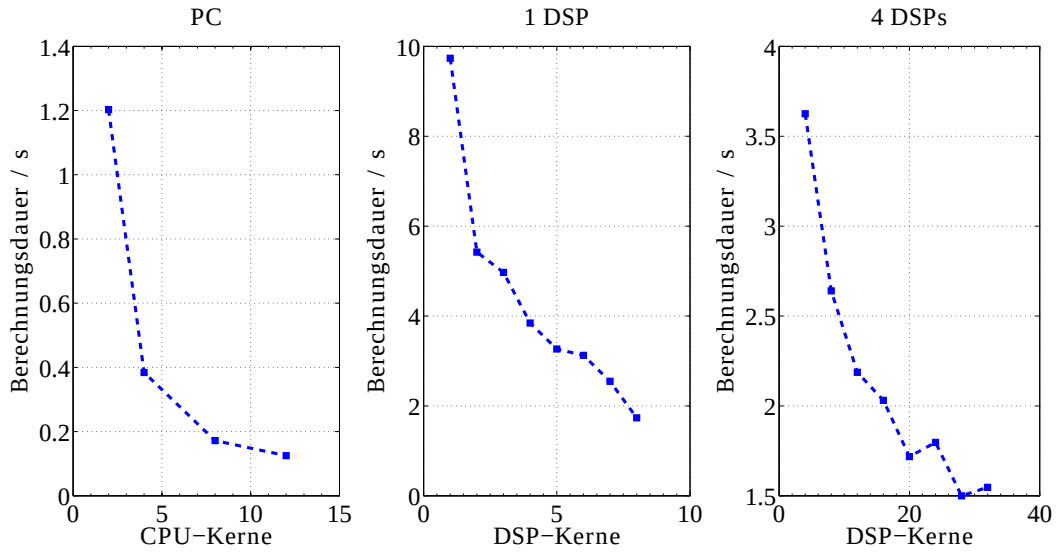


Abbildung 6.1: Berechnungsauer bei 512x512 Pixel Bildgröße

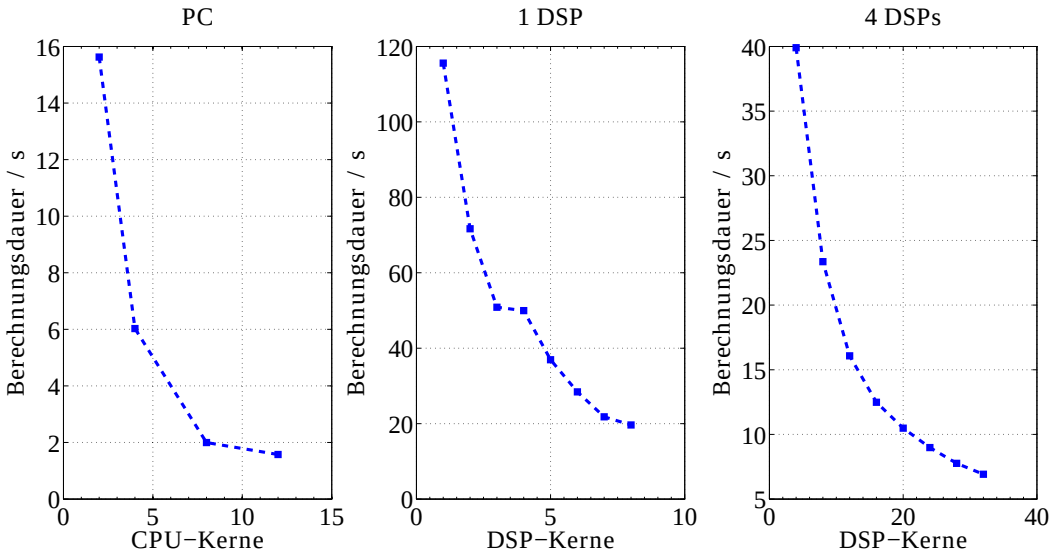


Abbildung 6.2: Berechnungsdauer bei 3000x3000 Pixel Bildgröße

Variation in der Iterationsanzah die Kernaussage nicht verfälscht. Die Iterationsanzahl ist durch eine Anpassung der Stop-Sensitivität für den Gauß-Newton-Algorithmus von Ausreißern bereinigt. Eine derartige Anpassung ist auf dem Kommandozeileninterface der PC-Clientsoftware möglich (s. Abbildung 5.2).

Die blaue Linie zeigt den Geschwindigkeitszuwachs je schneller der Mess-PC ist und je mehr Rechenkerne eingesetzt werden. Zugrunde gelegt sind Messwerte der in Tabelle 6.4 aufgelisteten Rechner. Es wird deutlich, dass schnelle PCs eine bessere Rechenperformance erreichen als die im Testaufbau verwendeten vier DSPs. In Abbildung 6.4 wird die Geschwindigkeit der PCs der der DSPs gegenübergestellt. Als Vergleich dafür dienen zwei Bilder der Größe 512x512 (dunkelblau) sowie 3000x3000 (hellbau) Pixel, die sowohl von einem PC mit den in Tabelle 6.4 definierten Konfigurationen als auch von den vier

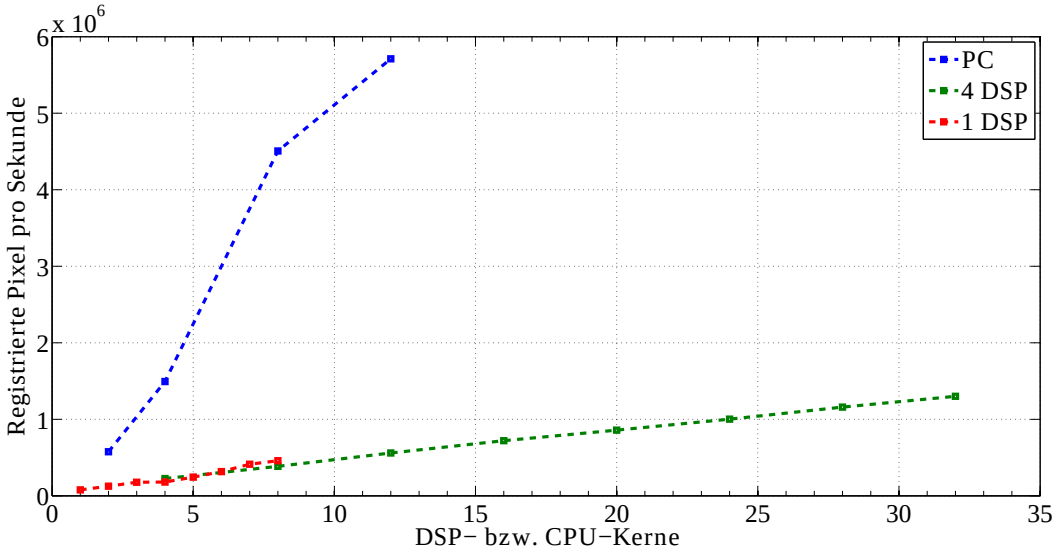


Abbildung 6.3: Registrierungsgeschwindigkeit abhängig von der Anzahl Rechenkerne

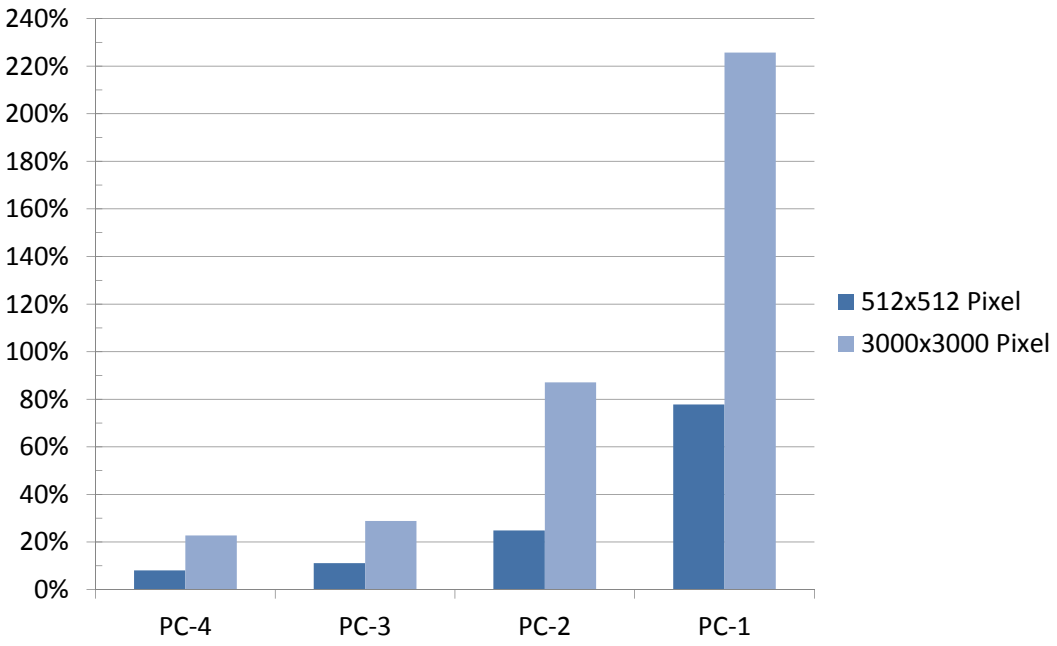
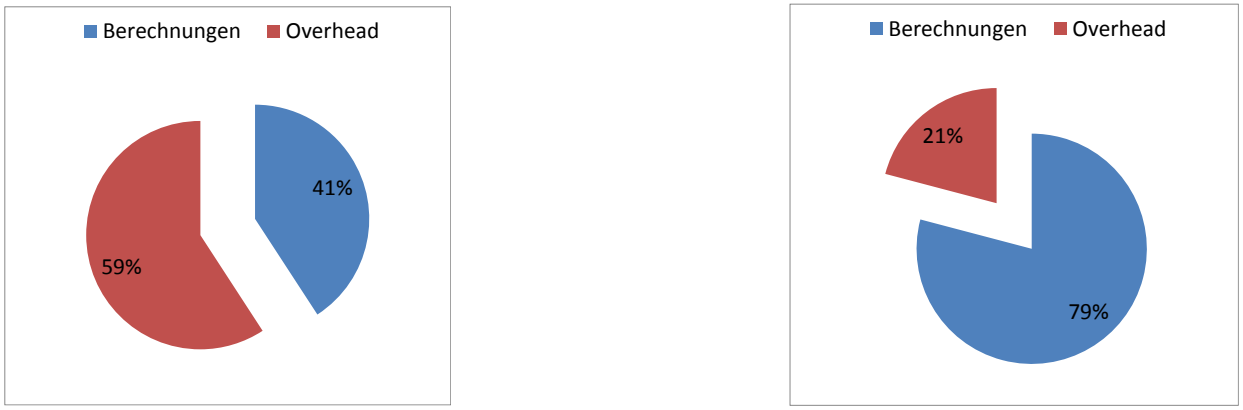


Abbildung 6.4: Relative Geschwindigkeit von vier DSPs in Relation zu verschiedenen PCs

DSPs registriert werden müssen. Ein Wert über 100% weist daraufhin, dass die vier DSPs mit ihren 32 Rechenkernen schneller sind als der Vergleichs-PC. Sieben von acht Messwerten liegen unter 100%, was beweist, dass die DSPs hier langsamer rechnen als der Vergleichs-PC.



(a) 512x512 Pixel (b) 3000x3000 Pixel

Abbildung 6.5: Overhead bei unterschiedlicher Größe der Bilddaten

6.2.1 Overhead durch Datenübertragung

In Abbildung 6.4 fällt auf, dass die dunkelblauen Balken niedriger sind als die hellblauen. Daran zeigt sich, dass die Geschwindigkeit der DSPs für kleinere Bilddaten niedriger ausfällt als beim Vergleichs-PC. Dies liegt darin begründet, dass der relative Overhead der Bildübertragung bei kleinen Bilddaten vergleichsweise größer ist. Die Zeit der Datenübertragung ist hier im Verhältnis zu der Zeit, die für die Berechnung der Bildregistrierung benötigt wird, höher. Die genaue Relation zwischen Overhead und Berechnungszeit für eine Vier-DSP-Lösung ist in Abbildung 6.5 aufgeschlüsselt.

6.2.2 Effizienz der Parallelisierung

Die Effizienz eines parallelen Programms wird durch die Formel

$$E_p(n) = \frac{T^*(n)}{p \cdot T_p(n)} \tag{6.1}$$

berechnet.[1] Dabei bezeichnet $T^*(n)$ die Ausführungszeit des (besten) nicht-parallelisierten und $T_p(n)$ die des parallelisierten Algorithmus. Über p geht die Anzahl der Rechenkerne in die Gleichung mit ein. Ein Ergebnis von 1 steht für eine ideale Effizienz. In Tabelle 6.6 ist die Effizienz der DSP-basierten Parallelisierung bezogen auf einen DSP mit einem Rechenkern dargestellt. Dabei fällt auf, dass die Effizienz durch das Verwenden von vier DSP-Bausteinen deutlich geschwächt wird. In diesem Umstand kann Potenzial für weitere Verbesserungen der DSP-basierten Lösung gesehen werden. In der aktuellen Lösung warten die DSPs so lange, bis der letzte (der vier) mit seinen Berechnungen fertig ist, bevor die Teil-Ergebnisse aufsummiert werden und mit der Berechnung fortgefahren wird. Die deutlich geringere Effizienz beim Einsatz von mehreren DSPs kann ein

1 [RR07, S.171]

Indiz dafür sein, dass derartige Wartezeiten die Gesamtperformance negativ beeinflussen.

DSPs	Rechenkerne	Bildgröße	Effizienz
1	8	512x512 Pixel	0,70
1	8	3000x3000 Pixel	0,74
4	32	512x512 Pixel	0,2
4	32	3000x3000 Pixel	0,52

Tabelle 6.6: Effizienz der Parallelisierung

6.3 Analyse

Bei der Interpretation der gewonnenen Daten spielt die Rechengeschwindigkeit der DPS als neuem im Gegensatz zu PCs als konventionellem Ansatz eine zentrale Rolle.

Für kleinere Bilddaten, wie die hier getesteten 512x512 Pixel, ist das Verhältnis von Overhead zu Berechnungszeit ungünstig. Es wird mehr Zeit für die Datenübertragung als für die eigentliche Berechnung aufgewendet. Da die Berechnung dagegen auf den drei schnellsten der vier Vergleichs-PCs im Bruchteil einer Sekunde abgeschlossen ist, lohnt sich hier der Aufwand für die Datenübertragung zu entfernten DSPs nicht, vorausgesetzt, ein schneller PC lässt sich in das jeweilige zur Bildregistrierung bestimmte Medizingerät integrieren. Kann aus Gründen der Energieeffizienz oder aus Platzmangel jedoch kein hochperformanter PC verbaut werden erweist sich der Einsatz von DSPs als sinnvolle Alternative: Vier DSP-Bauteilen können für kleine Bilddaten gut 20% der Performance eines Vier-Kern-Mittelklasse-PCs erreichen und eine Registrierung in 1,55 Sekunden (im Beispielbild mit 512x512 Pixeln) durchführen.

Bei größeren Bilddaten wendet sich das Blatt: Bei den getesteten neun Megapixeln (3000x3000) wird der relative Overhead so gering, dass die Geschwindigkeit der DSPs nun mit der eines schnellen PCs konkurrieren kann. Zwar übertrifft die Geschwindigkeit der DSP-Lösung nur in einem gemessenen Fall die der PC-Technik, was jedoch für ein energie- und raumsparendes, eingebettetes System eine hohe Leistung ist, die der PC-Technik immerhin nahekommt. Das Beispielbild mit den Ausmaßen 3000x3000 Pixel wird in nur sieben Sekunden registriert, was für viele medizinische Anwendungsfälle schon ausreichend ist.

6.3.1 Skalierbarkeit

Die in Tabelle Abbildung 6.3 dargestellten Parameter lassen Schlüsse über die weitere Skalierbarkeit der DSP-basierten Lösung zu: Zunächst fällt auf, dass die grüne Linie mit steigender Anzahl an DSP-Kernen linear verläuft ohne abzuflachen. Würde mit mehr als vier DSPs gerechnet, steht zu vermuten, dass sich diese Linie zunächst linear fortsetzt. Da die DSP-Bausteine vollkommen autark arbeiten, mit eigenem Speicher und eigener Ethernetverbindung, ist mit einer plötzlichen Abflachung dieser Kurve bei einer geringen Anzahl an DSP-Bausteinen nicht zu rechnen. Auch hinsichtlich der Auslastung von Netzwerk und CPU des Client-PCs, der die DSP-Serversoftware anspricht, gibt es keine Argumente gegen diese Vermutung.[2] Weiterführende Versuche mit einer höheren Anzahl an DSP-Bausteinen würden den finanziellen Rahmen dieser Arbeit übersteigen, deswegen wird eine mathematische Formel zur Prognose der Auswirkung von zumindest wenigen zusätzlichen DSP-Bausteinen hergeleitet.

Wegen der parallelen Struktur aus dedizierten Ethernetverbindungen und der parallelen Socketprogrammierung über Nonblocking-IO ist bei einer Erhöhung von wenigen DSP-Bausteinen kein höherer Overhead bei der Bildübertragung zu befürchten. Im Gegenteil, da sich der Verantwortungsbereich eines DSP bezogen auf das Gesamtbild verkleinert, kommt es auf jeder parallelen Verbindung sogar zu einem geringeren Datenaufkommen. Zur Vereinfachung soll hier angenommen werden, der Overhead bliebe bei geringfügiger Erhöhung der Anzahl der DSPs gleich.

Zur Berechnung einer Prognose der Performance mit mehr als vier DSP-Bausteinen wird eine Formel erstellt, welche berechnet, wie viele DSP-Bausteine notwendig wären, um die Geschwindigkeit eines der Vergleichs-PCs zu erreichen. Die Netto-Geschwindigkeit einer Bildregistrierung auf einem DSP wird anhand der gemessenen Geschwindigkeit auf vier DSPs durch

$$v_{DSP} = \frac{c}{4 \cdot d_4 - h} \tag{6.2}$$

ermittelt. Dabei soll mit d_4 die gemessene Registrierungsdauer auf einem Vier-DSP-System und mit c die Anzahl der Bildpunkte von Referenz- und Templatebild bezeichnet sein. Die Zeitdauer des Datentransfers der Bilddaten wird hier als Overhead h bezeichnet. Für v_{DSP} ergibt sich die Einheit Pixel pro Sekunde. Aus v_{DSP} kann auf die Dauer desselben Registrierungsproblems beim Einsatz von n DSPs über

$$d_n = h + \frac{c}{n \cdot v_{DSP}} \tag{6.3}$$

geschlossen werden. Dabei ist zu beachten, dass dies lediglich eine Näherung ist, die nur

2 Im Windows Taskmanager wird eine vernachlässigbar geringe Systemlast angezeigt.

für Werte von n angewandt werden soll, die nahe an der Berechnungsgrundlage $n = 4$ liegen. Wird diese Gleichung nach

$$n = \left\lceil \frac{c}{v_{DSP} \cdot (d_n - h)} \right\rceil \tag{6.4}$$

aufgelöst und für d_n die Berechnungszeit eines Vergleichs-PCs eingesetzt, kann eine Prognose für die benötigte Anzahl von DSPs berechnet werden, die höchstens in derselben Zeit eine Registrierung wie der Vergleichs-PC durchführen. Nach dem Einsetzen von $d_n = d_{PC}$ sowie Gleichung 6.2 in Gleichung 6.4 wird die Berechnungsvorschrift

$$n = \left\lceil \frac{4 \cdot d_4 - h}{d_n - h} \right\rceil \tag{6.5}$$

gewonnen, die auf die Messdaten der vier Vergleichs-PCs aus Tabelle 6.4 angewendet wird und zur Prognose in Abbildung 6.6 führt. PC-4 wird in dieser Darstellung weggelassen, da zwischen der Berechnungsdauer bei PC-4 und dem Durchschnittsoverhead der DSP-Lösung nur 129 ms liegen. Da dieser Wert kleiner ist als die Schwankungsbreite des Overheads, kann PC-4 nicht durch über Ethernet angebundene DSPs gleichwertig ersetzt werden.

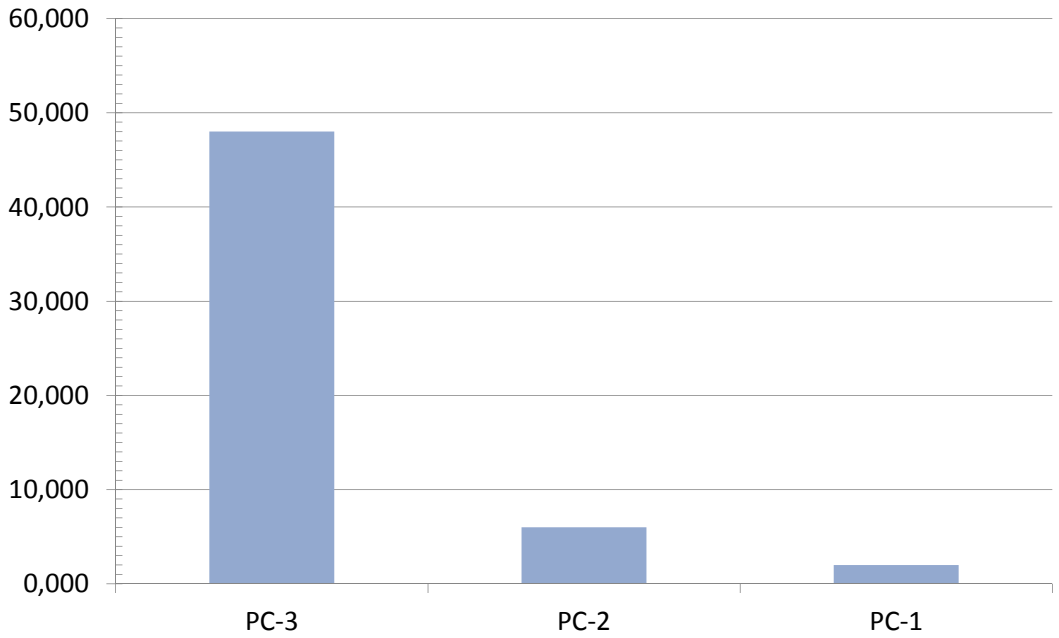


Abbildung 6.6: Prognose für die notwendige Anzahl an DSP-Bausteinen, um die gleiche Leistung einer PC-Lösung zu erzielen

Diese Hochrechnung ist rein theoretisch und muss noch mit entsprechendem Hardwareaufwand bestätigt werden.[3] Die Abbildung 6.6 liefert ausschließlich eine Prognose für die

3 Dabei ist zu beachten, dass der aktuelle Algorithmus zur Reduktion der Bilddaten nur Aufbauten aus 1,

Bildgröße 3000x3000 Pixel. Für 512x512 Pixel große Bilder ist eine derartige Prognose nicht durchführbar, da der gemessene, absolute Overhead bereits mehr Zeit erfordert als die lokale Berechnung auf drei der vier Vergleichs-PCs.

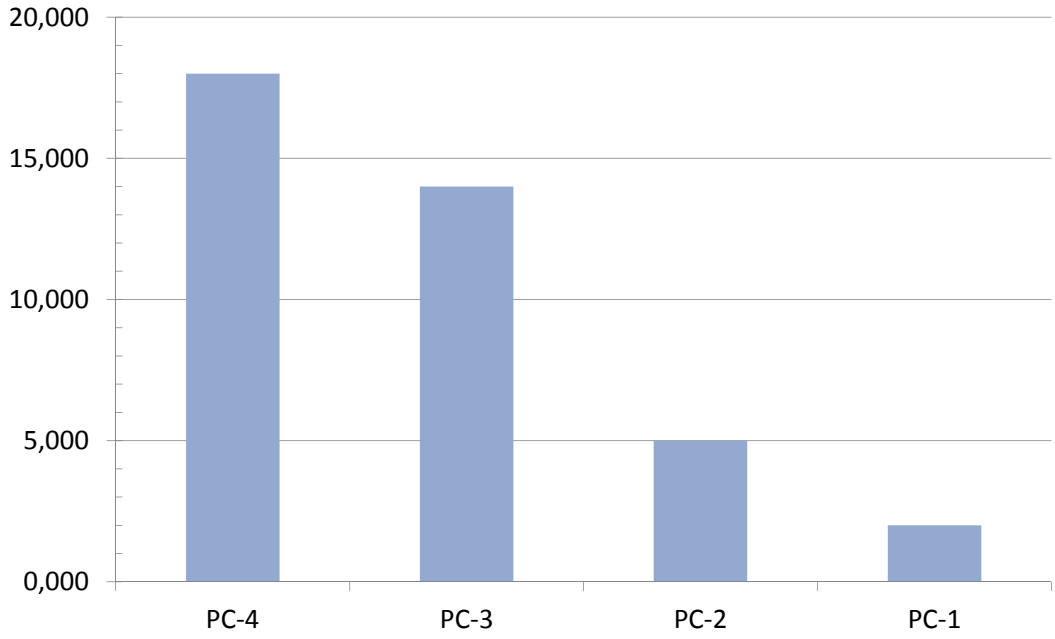


Abbildung 6.7: Prognose für die notwendige Anzahl an DSP-Bausteinen ohne Overhead für die Bilddatenübertragung

6.3.2 Overhead

Um die Ursachen für die Überlegenheit der PC-Lösung zu erforschen, wird der Overhead der Bilddatenübertragung untersucht. Wird in Gleichung 6.5 der Overhead h auf 0 gesetzt, erhält man ein Modell für eine DSP-Lösung, die keine Zeit für die Übertragung von Bilddaten verliert. In Abbildung 6.7 ist dargestellt, wie viele DSP-Bausteine notwendig wären, um einen der Vergleichs-PCs zu ersetzen, wenn kein Overhead für die Bildübertragung anfallen würde. Damit kann davon ausgegangen werden, dass bei einer Overheadfreien Lösung eine Platine ähnlich dem in Abbildung 4.2 vorgestellten Modell DSPA-8901, das über 20 DSP-Chips verfügt, eine der PC-Technik ebenbürtige Rechenperformance erreicht werden kann. Technisch kann der Overhead für die Bildübertragung auf mehrere Weisen eliminiert (oder auf vernachlässigbare Werte reduziert) werden.[4] Dafür ist die Entwicklung einer maßgeschneiderten Platine notwendig, die eines der in Tabelle 6.7 vorgeschlagenen Konzepte verwirklicht:

4, 16, 36, 64 etc. DSPs unterstützt. Eine Unterstützung davon abweichender Konfigurationen wäre für eine entsprechende Skalierung noch zu entwerfen.

4 Dies ist eine rein theoretische Überlegung bzw. eine Prognose, deren wissenschaftliche Überprüfung noch durchzuführen ist.

Technologie	Erläuterung
DMA	Bei einer direkten DMA (Direct Memory Access)[1]Anbindung der DSP-Bausteine an den RAM-Speicher des Mikrocontrollers fällt kein Overhead zur Bildübertragung mehr an, da alle Chips auf dieselben Speicherzellen zugreifen.
Sensoranbindung	Der C6678-DSP ist in der Lage, direkt Sensordaten in Empfang zu nehmen, zu verarbeiten und an den Mikrocontroller die bereits fertig verarbeiteten Daten weiterzugeben. Dafür besitzt er eine Vielzahl an Feldbus-Schnittstellen.[2]Hier wird der Overhead des Einlesens der Sensordaten, der sowohl bei der PC-Technik als auch bei DSPs anfällt, durch den Overhead für das Einlesen der bereits registrierten Bilder vom DSP ersetzt.
Hyperlink	Nicht overheadfrei, aber mit stark reduziertem zeitlichen Overhead, ist eine Anbindung der DSP-Bausteine über die Hyperlink-Schnittstelle. Hyperlink ist ein Hochgeschwindigkeitsbus, der ein 3000x3000 Pixel großes Bild theoretisch im Mikrosekundenbereich übertragen könnte.[3]

Tabelle 6.7: Hardwarelösungen für die Vermeidung/Reduzierung von Overhead

[1] Direkter Speicherzugriff ohne Mitwirkung des Prozessors nach [Tan06, S.120].

[2] [SPR11f, Kapitel 7]

[3] C6678 unterstützt „Hyperlink mit bis zu 50 Gbaud“ [SPR11f, S.11]

6.4 Eignung für die Medizintechnik

In Tabelle 6.7 werden DSP-basierte Hardwarekonzepte vorgeschlagen, die nach theoretischer Überlegung dazu geeignet sind, die Rechenperformance eines PC zu erreichen. In der Medizintechnik ist es jedoch nicht immer notwendig, höchste Rechenperformance anzustreben, da bei einem medizintechnischen Gerät auch andere Komponenten wie Energieeffizienz oder Gerätegröße einkalkuliert werden müssen. Hierin liegt ein großer Vorteil der DSP-Bausteine, die wie in Unterabschnitt 4.1.3 beschrieben weder viel Raum noch Energie benötigen. Die vorgestellte Bildregistrierung über Ethernet kann im Testaufbau trotz des hohen Netzwerkoverheads eine Bildregistrierung innerhalb weniger Sekunden durchführen, wie die Tabelle 6.8 mit Bezug auf die Beispielbilder zeigt.

Ist diese Geschwindigkeit für den Anwendungsfall ausreichend, kann ein medizintechnisches Produkt mit vier über Ethernet angebundenen C6678-DSPs eine Bildregistrierung durchführen, ohne dass ein Industrie-PC mit seinem hohen Bedarf an Energie, Raum und Abwärme verbaut werden muss.

Bilddimension	Größe in Byte	Berechnungsdauer
512x512 Pixel	256 KByte	1,5 Sekunden
3000x3000 Pixel	8,6 MByte	7 Sekunden

Tabelle 6.8: Dauer einer verteilten Bildregistrierung auf vier DSPs

Die Fragestellung der Arbeit ob durch Mehrkern-Signalprozessoren eine Leistungssteigerung in der medizinischen Bildregistrierung erzielt werden kann, kann also nur unter bestimmten Bedingungen bejaht werden. Abschließend sollen die Ergebnisse auch in Hinblick auf weitere wissenschaftliche Untersuchungen zusammengefasst werden.

7 Zusammenfassung

In der Forschungsarbeit wurde ein auf DSP-Chips zugeschnittener Algorithmus zur Bildregistrierung zweidimensionaler Bilddaten entworfen. Der Algorithmenentwurf berücksichtigte zwei Verteilungshierarchiestufen und beinhaltete die Entwicklung eines optimierten Datenübertragungsprotokolls. Die Implementierung des Algorithmus auf einer Hardwarearchitektur aus vier C6678-DSP mit je acht Rechenkernen von Texas Instruments wurde zum Performancevergleich mit einer PC-basierten Implementierung herangezogen. Es zeigt sich, dass beim Einsatz von DSPs eine zur PC-Technik konkurrenzfähige Rechenleistung nur erreichbar ist, wenn der Overhead zur Bildübertragung bereits im Hardwaredesign reduziert wird. Dafür sind die DSPs hinsichtlich des Energie- und Raumbedarfs der PC-Technik überlegen.

Die Erkenntnisse aus der hier durchgeführten Studie lassen sich folgendermaßen subsumieren:

- Für eine schnelle sowie parallele Ausführung einer Bildregistrierung auf Rechnern mit getrenntem Speicher lässt sich eine Formel entwickeln, die pro Pixel ausführbar und dadurch optimal parallelisierbar ist. Dafür eignet sich ein Algorithmus aus bilinearer Interpolation, einem Fehlerquadrat-Distanzmaß sowie dem Gauß-Newton-Verfahren.
- Die Bildregistrierung kann durch analytisches Ausmultiplizieren der Jacobi-Matrizen (statt Berechnung zur Laufzeit) beschleunigt werden.
- Dabei können Pipelinehemmnisse durch Hinzufügen eines schwarzen Bildrands vermieden werden.
- Durch Kompression sowie intelligentes Beschneiden der Bilder kann der Overhead zur Bildübertragung auf etwa 1/6 reduziert werden.
- Es kann ein RPC-Protokoll entworfen werden, das mit lediglich acht Byte Protokolloverhead pro Paket auskommt.
- Die Bildregistrierung auf C6678-DSPs kann nur durch Reduzierung des Overheads so schnell werden wie auf einem modernen PC.
- Gegen den Overhead wurden drei Hardwarekonzepte für die weitere wissenschaftliche Überprüfung vorgeschlagen: DMA-Anbindung des DSP zum RAM des Mikrocontrollers, direkte Verbindung von Bildsensor und DSP sowie Hyperlink.

Da herausgefunden wurde, dass eine Bildregistrierung theoretisch pro Pixel parallelisierbar ist, wäre eine weiterführende Untersuchung aufschlussreich, in der anstelle digitaler Signalprozessoren der Einsatz von FPGAs (Field Programmable Gate Arrays)[1] erprobt wird. Denn auf einem FPGA können logische wie arithmetische Schaltungen mit echter Parallelität ausgeführt werden. Es wäre zu analysieren, ob eine pixelparallele Berechnung, wie sie in Unterabschnitt 3.3.1 theoretisch erarbeitet wurde, auf FPGAs möglich ist, um herauszufinden, ob diese sowohl den hier vorgestellten DSPs als auch der PC-Technik überlegen sind.

1 Programmierbare Schaltung, siehe [Tie10, S.707-715]

Literaturverzeichnis

[ABRW11] Angermann, Anne; Beuschel, Michael; Rau, Martin; Wohlfarth, Ulrich: *MATLAB - Simulink - Stateflow - Grundlagen, Toolboxen, Beispiele*. aktualisierte Auflage. München : Oldenbourg Verlag, 2011. – ISBN 978–3–486–70585–0

[Adv11a] Advantech (Hrsg.): *DSPC-8681 - Half-length PCI Express Card with 4 TMS320C6678 DSPs*. 380 Fairview Way, Milpitas, CA 95035, USA: Advantech, November 2011

[Adv11b] Advantech (Hrsg.): *DSPC-8901 - AdvancedTCA® DSP Blade Provides Industry's Most Powerful DSP Farm for Voice and Video Processing*. 380 Fairview Way, Milpitas, CA 95035, USA: Advantech, Juni 2011

[Bä10] Bähring, Helmut: *Anwendungsorientierte Mikroprozessoren - Mikrocontroller Und Digitale Signalprozessoren*. 4. vollst. überarb. Aufl. Wiesbaden : Gabler Wissenschaftsverlage, 2010. – ISBN 978–3–642–12291–0

[BD] Beman Dawes, David Abrahams Rene R. Daniel Frey F. Daniel Frey (Hrsg.): *Boost Software License*. `http://www.boost.org/users/license.html`, Abruf: 29. Juli. 2002

[BDH05] Brechmann, Gerhard; Dzieia, Michael; Hörnemann, Ernst: *Elektronik Tabellen Betriebs- und Automatisierungstechnik*. 1. A. unveränderter Nachdruck 2007. Braunschweit : Westermann, 2005. – ISBN 978–3–142–35035–6

[BES08] Balzert, Helmut; Ebert, C.; Spindler, Spindler: *Lehrbuch Der Softwaretechnik: Softwaremanagement*. 2. Aufl. Heidelberg : Spektrum Akademischer Verlag, 2008. – ISBN 978–3–827–41161–7

[Bor98] Borowka, Petra: *Internetworking - Wege zum strukturierten Netzwerk*. 2. aktual. u. erw. A. Bonn : Internat. Thomson Publ., 1998. – ISBN 978–3–826–64027–8

[Cat05] Catsoulis, John: *Designing Embedded Hardware*. 2nd ed. Sebastopol, CA : O'Reilly Media, Inc., 2005. – ISBN 978–0–596–00755–3

[Cha] Changbo (Hrsg.): *cvplot - Matlab style plot functions for OpenCV*. `http://code.google.com/p/cvplot`, Abruf: 29. Juli. 2002

[Com] CompuLab Ltd. (Hrsg.): *fit-PC3 - Tiny. Fanless. Extendable.* `http://www.fit-pc.com/web/fit-pc/fit-pc3-info`, Abruf: 29. Juli. 2002

[DS98] Dowd, Kevin; Severance, Charles: *High performance computing* -. 2. A. Sebastopol : O'Reilly, 1998. – ISBN 978—1–5–65–92–3

[Ecl] Eclipse Foundation (Hrsg.): *The Eclipse Foundation open source community website*. `http://www.eclipse.org`, Abruf: 29. Juli. 2002

[Fis01] Fischer, Gerd: *Analytische Geometrie - Eine Einführung für Studienanfänger*. 7. Aufl. Düsseldorf : Vieweg Teubner Verlag, 2001. – ISBN 978—3–5–28–67–2

[FM08] Fischer, Bernd; Modersitzki, Jan: Ill-posed medicine—an introduction to image registration. In: *Inverse Problems* 24 (2008), Nr. 3, 034008. `http://stacks.iop.org/0266-5611/24/i=3/a=034008`

[GBBK09] Grechenig, Thomas; Bernhart, Mario; Breiteneder, Roland; Kappel, Karin: *Softwaretechnik - Mit Fallbeispielen aus realen Entwicklungsprojekten*. 1. Aufl. München : Pearson Deutschland GmbH, 2009. – ISBN 978–3–868–94007–7

[GGL96] Golub, Gene H.; Golub, Golub; Loan, Charles F. V.: *Matrix Computations* -. Third Edition,. London : JHU Press, 1996. – ISBN 978–0–801–85414–9

[Gil81] Gill, Philip E.: *Practical optimization* -. Amsterdam, Boston : Academic Press, 1981. – ISBN 978–0–122–83950–4

[Gri11] Grimm, Rainer: *C++11 - Der Leitfaden für Programmierer zum neuen Standard*. 1. Aufl. München : Addison Wesley Verlag, 2011. – ISBN 978–3–827–33088–8

[GS11] Gleim, Urs; Schüle, Tobias: *Multicore-Software - Grundlagen, Architektur und Implementierung in C/C++, Java und C*. 1. Auflage. Heidelberg : Dpunkt.Verlag GmbH, 2011. – ISBN 978–3–898–64758–8

[Hei] Heise Verlag (Hrsg.): *Tesla M2090: 512 Kerne nun auch fürs Supercomputing*. `http://www.heise.de/ix/meldung/Tesla-M2090-512-Kerne-nun-auch-fuers-Supercomputing-1245209.html`, Abruf: 29. Juli. 2002

[HL09] Hoffmann, Simon; Lienhart, Rainer: *OpenMP - Eine Einführung in die parallele Programmierung mit C/C++*. 1. Aufl. 2008. Korr. Nachdruck. Berlin : Springer DE, 2009. – ISBN 978–3–540–73122–1

[Hof09] Hoffmann, Dirk W.: *Theoretische Informatik*. München, Wien : Hanser Verlag, 2009. – ISBN 978–3–446–41511–9

[HR02] Hruschka, Peter; Rupp, Chris: *Agile Softwareentwicklung für Embedded Real-Time Systems mit der UML* -. München : Hanser, 2002. – ISBN 978–3–446–21997–7

[Huf52] Huffman, D.A.: A method for the construction of minimum-redundancy codes. In: *Proceedings of the IRE* 40 (1952), Nr. 9, S. 1098–1101

[Int] Intel Corporation (Hrsg.): *Intel Xeon Processor E5645*. http://ark.intel.com/products/48768/Intel-Xeon-Processor-E5645-(12M-Cache-2_40-GHz-5_86-GTs-Intel-QPI), Abruf: 29. Juli. 2002

[its] itseez (Hrsg.): *OpenCV*. http://opencv.org, Abruf: 29. Juli. 2002

[Jä93] Jähne, Bernd: *Digitale Bildverarbeitung* -. 6. überarb. u. erw. Aufl. Berlin : Springer Berlin Heidelberg, 1993. – ISBN 978–3–540–24999–3

[Kar06] Karlsson, Björn: *Beyond the C++ standard library - an introduction to Boost*. Amsterdam : Addison-Wesley, 2006. – ISBN 978–0–321–13354–0

[Kon12] Kontron AG (Hrsg.): *KTC5520-EATX*. Oskar-von-Miller-Str. 1, 85386 Eching/Munich, Germany: Kontron AG, 2012. http://de.kontron.com/_etc/scripts/download/getdownload.php?downloadId=OTEzNg==

[KS08] Korff, Andreas; Schacher, Markus: *Modellierung Von Eingebetteten Systemen Mit Uml Und Sysml* -. 1. Aufl. Heidelberg : Spektrum Akademischer Verlag, 2008. – ISBN 978–3–827–41690–2

[Leh08] Lehmann, Arne: *Signaltransformationen auf SIMD-DSPs - Algebra, Parallelisierung und Codegenerierung*. Saarbrücken : VDM Verlag, 2008. – ISBN 978–3–639–00561–5

[Mal10] Malepati, Hazarathaiah: *Digital Media Processing - DSP Algorithms Using C*. Pap/Psc. London : Newnes, 2010. – ISBN 978–1–856–17678–1

[Mar03] Martin, Robert C.: *Agile software development - principles, patterns, and practices*. London : Prentice Hall, 2003. – ISBN 978–0–135–97444–5

[Mar11a] Marques, Oge: *Practical Image and Video Processing Using MATLAB* -. Hoboken : John Wiley, 2011. – ISBN 978—1–1–18–09–3

[Mar11b] Martin, Robert C.: *Clean Coder - Verhaltensregeln für professionelle Programmierer*. 1. Aufl. München : Addison Wesley Verlag, 2011. – ISBN 978–3–827–33104–5

[Mica] Microsoft Corporation (Hrsg.): *ioctlsocket function*. http://msdn.microsoft.com/en-us/library/windows/desktop/ms738573.aspx, Abruf: 29. Juli. 2002

[Micb] Microsoft Corporation (Hrsg.): *Microsoft Visual Studio*. http://www.microsoft.com/germany/visualstudio, Abruf: 29. Juli. 2002

[Micc] Microsoft Corporation (Hrsg.): *Working with Packing Structures*. http://msdn.microsoft.com/en-us/library/ms253935.aspx, Abruf: 29. Juli. 2002

[Mod09] Modersitzki, Jan: *Fair - Flexible Algorithms for Image Registration*. Philadelphia : SIAM, 2009. – ISBN 978–0–898–71690–0

[MRS97] Monien, Burkhard v.; Röttger, Markus; Schroeder, Ulf-Peter: *Einführung in parallele Algorithmen und Architekturen - Gitter, Bäume und Hypercubes*. Bonn : Internat. Thomson Publ., 1997. – ISBN 978–3–826–60248–1

[Noe05] Noergaard, Tammy: *Embedded Systems Architecture - A Comprehensive Guide For Engineers And Programmers*. Har/Cdr. Amsterdam : Elsevier, 2005. – ISBN 978–0–750–67792–9

[NVI] NVIDIA Corporation (Hrsg.): *Was ist CUDA?* `http://www.nvidia.de/object/what_is_cuda_new_de.html`, Abruf: 29. Juli. 2002

[NVI11] NVIDIA Corporation (Hrsg.): *TESLA M-Class GPU Computing Modules Accelerating Science*. 2701 San Tomas Expressway Santa Clara, CA 95050, USA: NVIDIA Corporation, 2011. `http://www.nvidia.com/docs/IO/105880/DS-Tesla-M-Class-Aug11.pdf`

[NW06] Nocedal, Jorge; Wright, Stephen: *Numerical optimization*. 2nd ed. Berlin, Heidelberg : Springer, 2006. – ISBN 978–0–387–30303–1

[OB09] Oestereich, Bernd; Bremer, Stefan: *Analyse und Design mit UML 2.3 - Objektorientierte Softwareentwicklung*. aktualisierte und erweiterte Auflage. München : Oldenbourg Verlag, 2009. – ISBN 978–3–486–58855–2

[Ope] Open Source Initiative (Hrsg.): *Open Source Initiative OSI - The BSD License: Licensing*. `http://opensource.org/licenses/bsd-license.php`, Abruf: 29. Juli. 2002

[Ope11] OpenMP Architecture Review Board: OpenMP Application Program Interface. Version: 2011. `http://www.openmp.org/mp-documents/OpenMP3.1.pdf`. 2011. – Specification

[PFTV89] Press, William H.; Flannery, Brian P.; Teukolsky, Saul A.; Vetterling, William T.: *Numerical Recipes in Pascal - The Art of Scientific Computing*. Rev Sub. Cambridge : Cambridge University Press, 1989. – ISBN 978–0–521–37516–0

[PIC04] PICMG (Hrsg.): *PICMG AMC.0 - Advanced Mezzanine Card Short Form Specification*. PICMG c/o Virtual, Inc., 401 Edgewater Place, Suite 600, Wakefield, MA 01880, USA: PICMG, June 2004. `http://www.picmg.org/pdf/AMC_D0.9_Short_spec.pdf`

[Pic10] Pichler, Roman: *Agile Product Management With Scrum - Creating Products that Customers Love*. 1. Aufl. Amsterdam : Addison-Wesley, 2010. – ISBN 978–0–321–60578–8

[Poh11] Pohl, Melanie: *Bildregistrierung mit linearer und nicht-linearer Elastizität*, Universität zu Lübeck, Masterarbeit, 2011

[Qur05] Qureshi, Shehrzad: *Embedded Image Processing on the TMS320C6000 DSP : Examples in Code Composer Studio and MATLAB* -. 1st ed. 2005. Corr. 2nd printing. Berlin, Heidelberg : Springer, 2005. – ISBN 978–0–387–25280–3

[RR07] Rauber, Thomas; Rünger, Gudula: *Parallele Programmierung* -. 2. neu bearb. u. erw. Aufl. Berlin : Springer DE, 2007. – ISBN 978–3–540–46549–2

[Sab11] Saban, Nissim: MultiCore DSP vs GPUs / Texas Instruments. Post Office Box 655303, Dallas, Texas 75265, USA, August 2011. – Forschungsbericht

[SBD+10] Schatten, Alexander; Biffl, Stefan; Demolsky, Markus; Gostischa-Franta, Erik; Östreicher, Thomas; Winkler, Dietmar: *Best Practice Software-Engineering - Eine Praxiserprobte Zusammenstellung von Komponentenorientierten Konzepten, Methoden und Werkzeugen*. 1st Edition. Berlin : Springer DE, 2010. – ISBN 978–3–827–42486–0

[Sch95] Schmitt, O.: *Die multimodale Architektonik des menschlichen Gehirns*. 1995. – Habilitationsschrift, Medizinische Universität Lübeck, 1995, 190 Seiten

[Sev98] Severance, C.: IEEE 754: An Interview with William Kahan. In: *Computer* 31 (1998), Nr. 3, S. 114–115

[SIA] SIAM Society for Industrial and Applied Mathematics (Hrsg.): *FAIR: Flexible Algorithms for Image Registration - Software and Apps*. `http://www.siam.org/books/fa06`, Abruf: 29. Juli. 2002

[SNV12] Texas Instruments (Hrsg.): *PMP7256 Schematic*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, 2012. `http://www.ti.com/litv/pdf/snvr131`

[Som12] Sommerville, Ian: *Software Engineering* -. 9. München : Pearson Deutschland GmbH, 2012. – ISBN 978—3–8–6894–0

[SPR10a] Texas Instruments (Hrsg.): *Optimizing Loops on the C66x DSP High-Performance and Multicore Processors*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, November 2010. `http://www.ti.com/lit/an/sprabg7/sprabg7.pdf`

[SPR10b] Texas Instruments (Hrsg.): *TMS320C6000 Optimizing Compiler V7.0 User's Guide*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, Februar 2010. `http://www.ti.com/litv/pdf/spru187q`

[SPR11a] Texas Instruments (Hrsg.): *KeyStone Architecture - DDR3 Memory Controller - User Guide*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, November 2011. `http://www.ti.com/lit/ug/sprugv8c/sprugv8c.pdf`

[SPR11b] Texas Instruments (Hrsg.): *SYS/BIOS Inter-Processor Communication (IPC) and I/O User's Guide*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, September 2011. `http://www.ti.com/litv/pdf/sprugo6d`

[SPR11c] Texas Instruments (Hrsg.): *TI SYS/BIOS v6.33 Real-time Operating System*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, Dezember 2011. `http://www.ti.com/litv/pdf/spruex3k`

[SPR11d] Texas Instruments (Hrsg.): *TMDXEVM6678L EVM - Technical Reference Manual - Version 2.0*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, November 2011. `http://wfcache.advantech.com/support/TMDXEVM6678L_Technical_Reference_Manual_2V00.pdf`

[SPR11e] Texas Instruments (Hrsg.): *TMS320C6000 Programmer's Guide*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, Juli 2011. `http://www.ti.com/litv/pdf/spru198k`

[SPR11f] Texas Instruments (Hrsg.): *TMS320C6678 - Multicore Fixed and Floating-Point Digital Signal Processor - Data Manual*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, August 2011. `http://www.ti.com/lit/ds/sprs691c/sprs691c.pdf`

[SPR11g] Texas Instruments (Hrsg.): *TMS320C66x multicore DSPs for high-performance computing*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, 2011. `http://www.ti.com/litv/pdf/sprt619`

[SPR12a] Texas Instruments (Hrsg.): *KeyStone Architecture Bootloader User Guide*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, Juni 2012. `http://www.ti.com/litv/pdf/sprugy5b`

[SPR12b] Texas Instruments (Hrsg.): *Throughput Performance Guide for C66x KeyStone Devices*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, Juli 2012. `http://www.ti.com/litv/pdf/sprabk5a`

[SPR12c] Texas Instruments (Hrsg.): *TI Network Developer's Kit (NDK) v2.21 API Reference Guide*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, Mai 2012. `http://www.ti.com/litv/pdf/spru524h`

[ST10] Stevens, W. R.; Travis, Ian: *TCP/IP - Der Klassiker. Protokollanalyse. Aufgaben und Lösungen*. 1. Auflage. Berlin, Offenbach : Vde Verlag GmbH, 2010. – ISBN 978–3–800–73223–4

[Sta05] Stallings, William: *Betriebssysteme/ Bafög-Ausgabe - Prinzipien und Umsetzung*. 4. überarbeitete Auflage. München : Pearson Studium, 2005. – ISBN 978–3–827–37185–0

[Sti12] Stiller, Andreas: Höhenflüge - International Supercomputing Conference 2012 und die 39. Top500-Liste der Super Computer. In: *c't Magazin für Computertechnik* 15 (2012), Juli, S. 68–71

[Str98] Stroustrup, Bjarne: *Die C++-Programmiersprache*. 3. aktualis. u. erw. A. Amsterdam : Addison-Wesley, 1998. – ISBN 978–3–827–31296–9

[Str09] Strutz, Tilo: *Bilddatenkompression - Grundlagen, Codierung, Wavelets, JPEG, MPEG*. 4, überarb. u. erg. Aufl. 2009. Wiesbaden : Vieweg Teubner Verlag, 2009. – ISBN 978—3–8–34–80–4

[Sun] Sundance Digital Signal Processing Inc. (Hrsg.): *GDD9000-CBLAS*. `http://www.sundancedsp.com/libraries/gdd9000`, Abruf: 29. Juli. 2002

[Tan06] Tanenbaum, Andrew S.: *Computerarchitektur - Strukturen, Konzepte, Grundlagen*. 5. überarb. A. München : Pearson Studium, 2006. – ISBN 978–3–827–37151–5

[Texa] Texas Instruments (Hrsg.): *BIOS Multicore Software Development Kit - Version 2.1.0 - Addendum*. `http://processors.wiki.ti.com/index.php/BIOS-MCSDK_2.1_Addendum`, Abruf: 29. Juli. 2002

[Texb] Texas Instruments (Hrsg.): *Code Composer Studio (CCStudio) Integrated Development Environment (IDE) v5*. `http://www.ti.com/tool/ccstudio`, Abruf: 29. Juli. 2002

[Texc] Texas Instruments (Hrsg.): *GEL*. `http://processors.wiki.ti.com/index.php/GEL`, Abruf: 29. Juli. 2002

[Texd] Texas Instruments (Hrsg.): *Multicore Application Deployment (MAD) Utilities*. `http://processors.wiki.ti.com/index.php/MAD_Utils_User_Guide`, Abruf: 29. Juli. 2002

[Texe] Texas Instruments (Hrsg.): *TMS320C6678*. `http://www.ti.com/product/tms320c6678`, Abruf: 29. Juli. 2002

[Texf] Texas Instruments (Hrsg.): *XDS100*. `http://processors.wiki.ti.com/index.php/XDS100#Q:_Can_I_change_the_serial_number_on_my_XDS100v2.3F`, Abruf: 29. Juli. 2002

[Tex11] Texas Instruments (Hrsg.): *OSAL and HAL*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, September 2011. `http://processors.wiki.ti.com/images/a/a9/06_-_OSAL_and_HAL.pdf`

[Tex12] Texas Instruments (Hrsg.): *C6678 Power Consumption Model (Rev. B)*. Post Office Box 655303, Dallas, Texas 75265, USA: Texas Instruments, April 2012. `http://www.ti.com/litv/zip/sprm545b`

[Thea] The Apache Software Foundation (Hrsg.): *Apache License, Version 2.0*. http://www.apache.org/licenses/LICENSE-2.0, Abruf: 29. Juli. 2002

[Theb] The MathWorks Inc. (Hrsg.): *Embedded Coder - Generierung von optimiertem C- und C++-Code für Embedded Systeme*. http://www.mathworks.de/products/embedded-coder, Abruf: 29. Juli. 2002

[Thec] The MathWorks Inc. (Hrsg.): *MATLAB Coder - Generieren von C- und C++-Code aus MATLAB*. http://www.mathworks.de/products/matlab-coder, Abruf: 29. Juli. 2002

[Tie10] Tietze, Schenk: *Halbleiter-Schaltungstechnik*. 13., neu bearb. Aufl. Berlin : Springer, 2010. – ISBN 978–3–540–42849–7

[Unb] Unbekannte Entwickler der quelloffenen Software (Hrsg.): *WINE HQ*. http://www.winehq.org, Abruf: 29. Juli. 2002

[Uta05] Utas, Greg: *Robust Communications Software - Extreme Availability, Reliability And Scalability For Carrier-Grade Systems*. 1. Auflage. West Sussex, England : John Wiley, 2005. – ISBN 0–470–85434–0

[Wen04] Wenckebach: *Volumetrische Registrierung zur medizinischen Bildanalyse*, Humboldt-Universität zu Berlin, Diplomarbeit, 2004

[ZF03] Zitová, Barbara; Flusser, Jan: Image registration methods: a survey. In: *Image and Vision Computing* 21 (2003), Nr. 11, 977 - 1000. http://dx.doi.org/10.1016/S0262-8856(03)00137-9. – DOI 10.1016/S0262–8856(03)00137–9. – ISSN 0262–8856

[ZG10] Zöller-Greer, Peter: *Multi-Media-Systeme - Grundlagen und Anwendungen; mit einer Einführung in die Projektplanung, Datenkompression und Datenformate, digitale Bildverarbeitung, Audio- und Video-Verarbeitung, eLearning, Internet, Computer-Games und Animationen*. 2. überarbeitete Auflage. Norderstedt : BoD – Books on Demand, 2010. – ISBN 978–3–981–16392–6

A Anhang

Es folgen ausgewählte Projektartefakte als Anhang.

A.1 Pflichtenheft

Für die hier verwendete Hard- sowie Softwarelösung wurde zum Projektstart ein Pflichtenheft erstellt. Da es sich wie in Abschnitt 5.5 beschrieben um ein Projekt mit agilem Vorgehen handelt, wird das Pflichtenheft bewusst auf einem hohen Abstraktionslevel gehalten und folgt nicht der formalen Struktur eines Pflichtenheftes für monumentale Vorgehensmodelle. Es folgt eine Kopie des Inhalts (ohne Deckblatt).

A.1.1 Projektziel

Es wird eine Hard- und Softwarelösung entwickelt, die sich für die messtechnische Erforschung von Mehrkern DSP-Bausteinen in der medizinischen Bildregistrierung eignet. Dabei wird eine Hardwareumgebung für die spätere messtechnische Untersuchung der Rechenperformance erstellt sowie eine verteilte Software entwickelt, die eine Bildregistrierung hochperformant ausführt und die Berechnungsgeschwindigkeit für Analysezwecke ausgibt.

A.1.2 Funktionale Anforderungen

Es folgen die funktionale Anforderungen an Hard- und Software.

Anforderungen an die Hardware

Der Hardwareaufbau wird folgenden Anforderungen genügen:

10. Es werden vier Stück C6678-DSP von Texas Instruments eingesetzt.
20. Zentral gesteuert werden die Bausteine von einem Client-PC (MS Windows basiert).
30. Die Vernetzung erfolgt in (echter) Sterntopologie über Ethernet mit Cat-6a-Kabeln.

Anforderungen an die PC-Clientsoftware

Die PC-Clientsoftware unter Microsoft Windows hat folgende Anforderungen:

110. Laden der zu registrierenden Bilddaten vom Dateisystem in gängigen Bildformaten.
120. Senden der zu registrierenden Bilddaten (Referenz- und Templatebild) an die DSP-Bausteine.
130. Anfordern einer Berechnung des SSD-Distanzmaßes von einem oder vier DSPs im Netzwerk über das HPRPC Protokoll (TCP).
140. Anfordern einer Berechnung von SSD-Distanzmaß, Jacobi- sowie Hesse-Matrix von einem oder vier DSPs über das HPRPC Protokoll (TCP).
150. Lokale Berechnung des SSD-Distanzmaßes.
160. Lokale Berechnung von SSD-Distanzmaß, Jacobi- sowie Hesse-Matrix.
170. Ausführen des Gauß-Newton-Algorithmus unter Einbezug der entfernten DSP-Bausteine.
180. Ausführen des Gauß-Newton-Algorithmus mit rein lokaler Berechnung.
190. Messen der Ausführungsdauer.
200. Aufzeichnen des SSD-Distanzmaßes über die Registrierungsiterationen hinweg.
210. Lauflängencodierung zur Kompression der Bilddaten.
220. Beschneiden der Bilddaten für deinen DSP auf die prognostiziert minimal notwendigen (sowie neue Übermittlung bei Überschreitung der Prognose).
230. Einbezug aller Rechenkerne bei einer lokalen Berechnung von SSD, Jacobi- und/oder Hesse-Matrix.

Anforderungen an die DSP-Serversoftware

Für den C6678-DSPs von Texas Instruments gelten folgende Anforderungen:

310. Erstellen eines Firmware-Images mit Mehrkern- sowie Ethernetfunktionalität.
320. Empfang der zu registrierenden Bilddaten (Referenz- und Templatebild).
330. Lauflängendecodierung der Bilddaten.
340. Berechnung des SSD-Distanzmaßes über beide Bilder.
350. Berechnung von SSD-Distanzmaß, Jacobi-Matrix und approximierter Hesse-Matrix.

360. Es wird ein TCP-Server angeboten, der das HPRPC-Protokoll unterstützt. HPRPC mit den notwendigen Kommandos sind dem Dokument 'HPRPC-Protokoll-für-Bildregistrierung' zu entnehmen.

370. Einbezug der per HPRPC definierten Anzahl an Rechenkernen für die Berechnung.

Mathematisch-Algorithmische Anforderungen

Für die Mathematik und Algorithmik gilt:

410. Es wird eine rigide Bildregistrierung nach Modersitzki mit linearer Interpolarion, SSD-Distanzmaß und Gauss-Newton Algorithmus implementiert.

420. Auf die Verwendung von Matrizen wird verzichtet, es wird eine Formel entwickelt, die pro-Pixel ausführbar ist.

430. Der Applikationsübergreifende Datentyp für Realzahlen ist an zentraler Stelle zur Compilezeit einstellbar (z.B per typedef), er ist per Default wegen der eingesetzten Hardware float (32 Bit Fließkomma).

440. Der pro-Pixel ausgeführte Bereich wird performanceoptimiert. Auf Sprungbefehle (If-Anweisungen) wird verzichtet, die Bilder werden dazu beim Empfang mit einem Randbereich umgeben.

Anforderungen an die Benutzerschnittstelle

Als Benutzerschnittstelle kommt ein Kommandozeileninterface als Teil der PC-Clientsoftware zum Einsatz.

510. Es gibt eine Hilfebeschreibung. Auf Anfrage oder bei Parsing-Fehlern wird sie dargestellt.

520. Pflichtparameter: Referenzbild (Dateiname).

530. Pflichtparameter: Templatebild (Dateiname).

540. Optionaler Parameter: IP-Adressen der DSPs (wenn keine definiert, lokale Bildregistrierung auf allen Rechenkernen des PC).

550. Optionaler Parameter: Anzahl der Rechenkerne auf den DSPs (Default: alle).

560. Optionaler Parameter: Parameter für die spekulative Bildbeschneidung (Rotation und Translation).

570. Optionaler Parameter: Stop-Sensitivität (1=früher Stop, kleiner=späterer Stop) für den Gauß-Newton Algorithmus

580. Optionaler Parameter: Möglichkeit, schon bei einer Stagnation des SSD-Distanzmaßes den Gauß-Newton Algorithmus zu stoppen (für Registrierungsprobleme ohne lokale Minima).

590. Visualle Darstellung von Referenz-, Template- sowie Differenzbild vor und Differenzbild nach der Registrierung.

600. Darstellen der Ausführungsdauer.

610. Visuelles ausgeben des Verlaufs des SSD-Distanzmaßes über die Registrierungsiterationen hinweg.

620. Visualle Darstellung von Referenz-, Template- sowie Differenzbild vor und Differenzbild nach der Registrierung.

Nichtfunktionale Anforderungen

Es handelt sich um eine intern verwendete Software für eine Performancemessung. Die Produktqualität braucht nur einem Proof-Of-Concept-Level zu genügen, es ist kein Industrieprodukt. Daraus geben sich folgende Anforderungen:

810. Die safety-Anforderungen sind gering.

820. Für die Messergebnisse ist Präzision sowie Reproduzierbarkeit gewünscht.

A.2 Auszüge aus dem Softwareentwurf

Der Softwareentwurf basiert auf UML. In der Arbeit wurde bereits ein Sequenzdiagramm in Abbildung 5.11 dargestellt, hier folgt eine Teilauswahl weiterer UML-Diagramme.

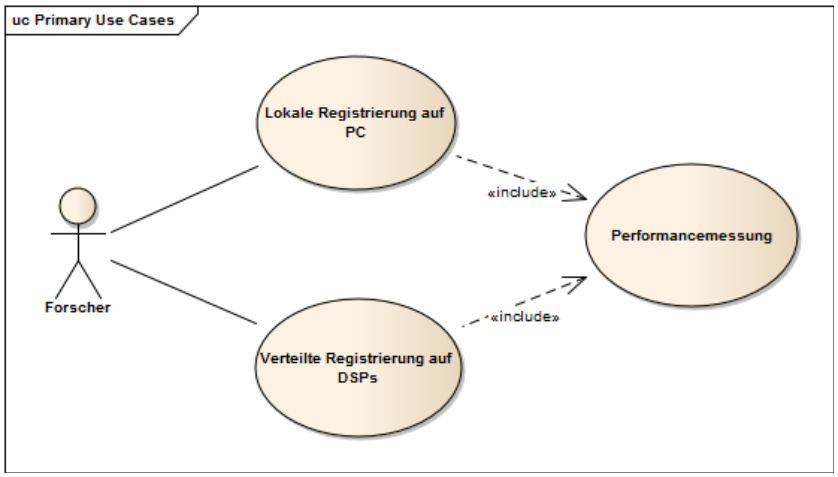


Abbildung A.1: Use-Case-Diagramm der Gesamtanwendung

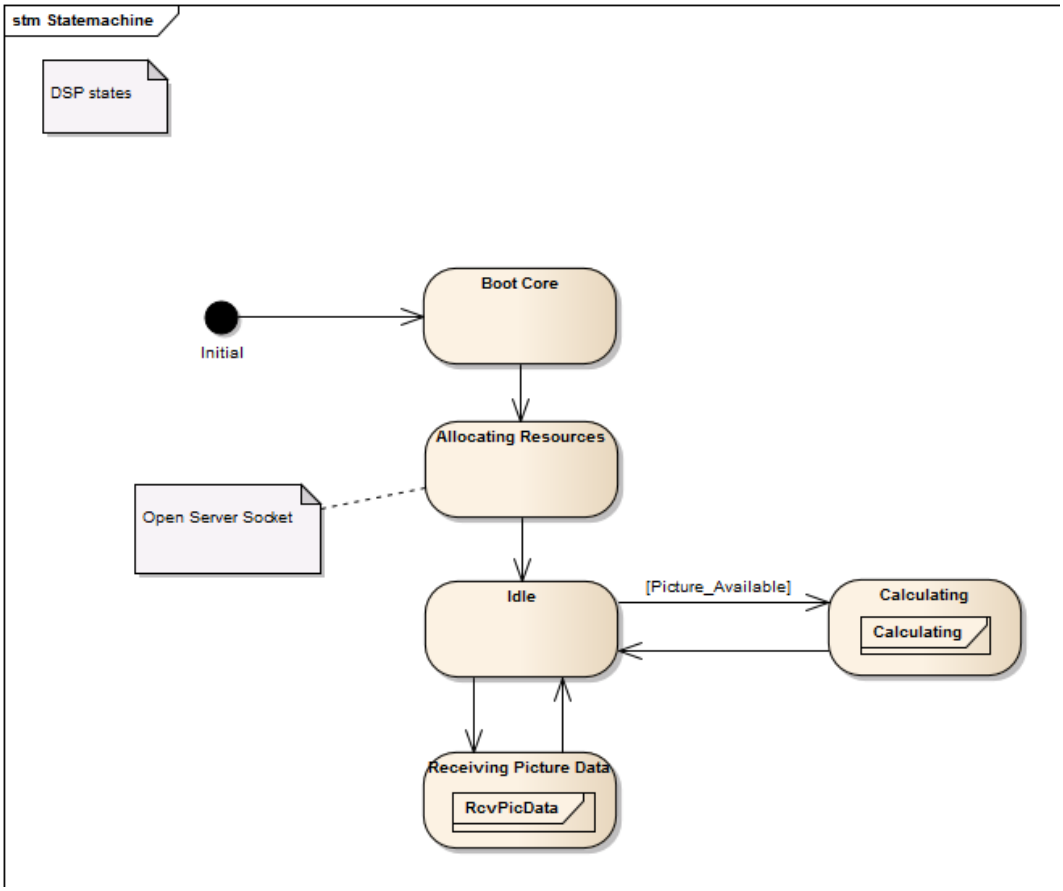


Abbildung A.2: Statusdiagramm für den DSP-Rechenkern

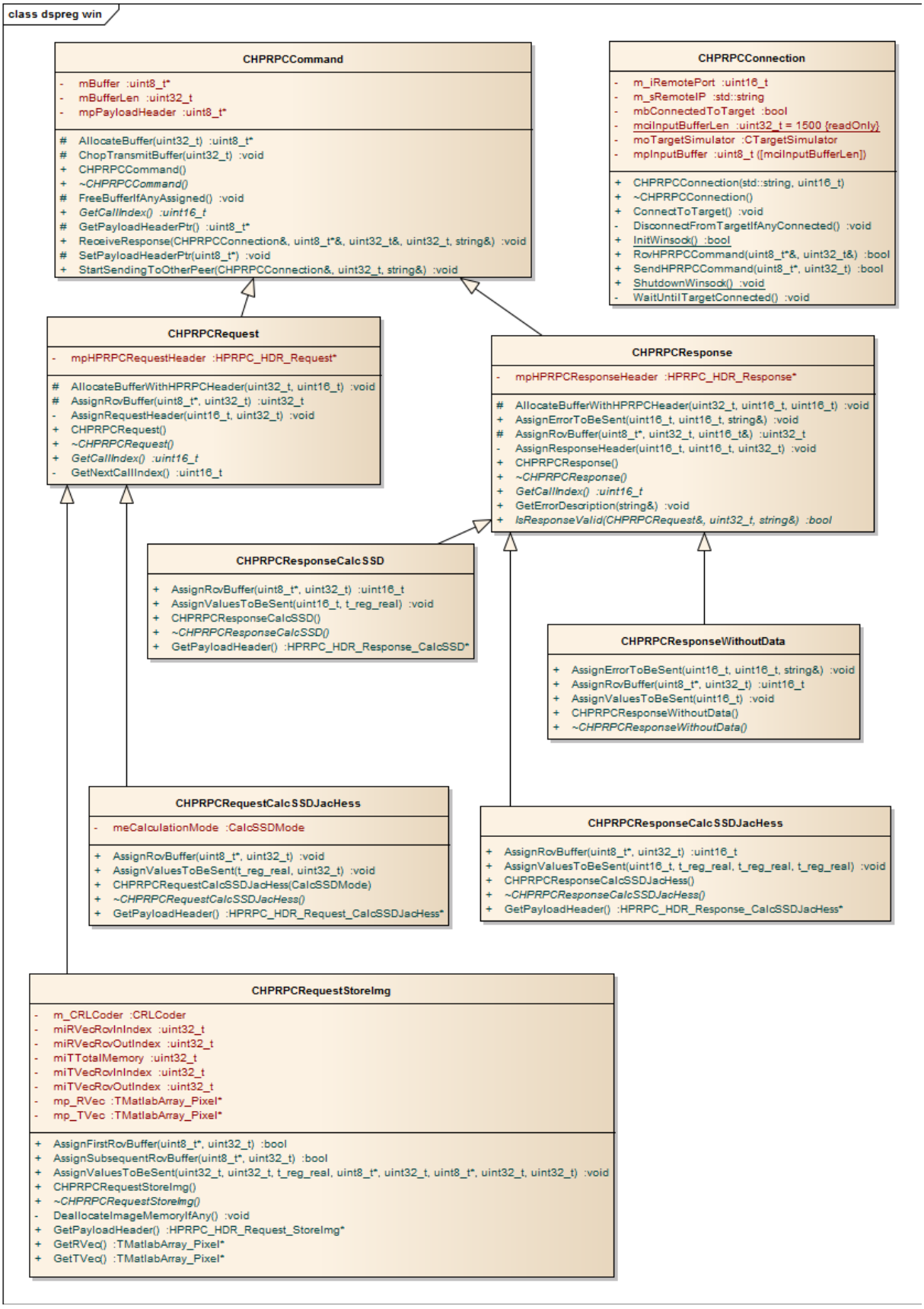


Abbildung A.3: Klassendiagramm des HPRPC-Protokolls

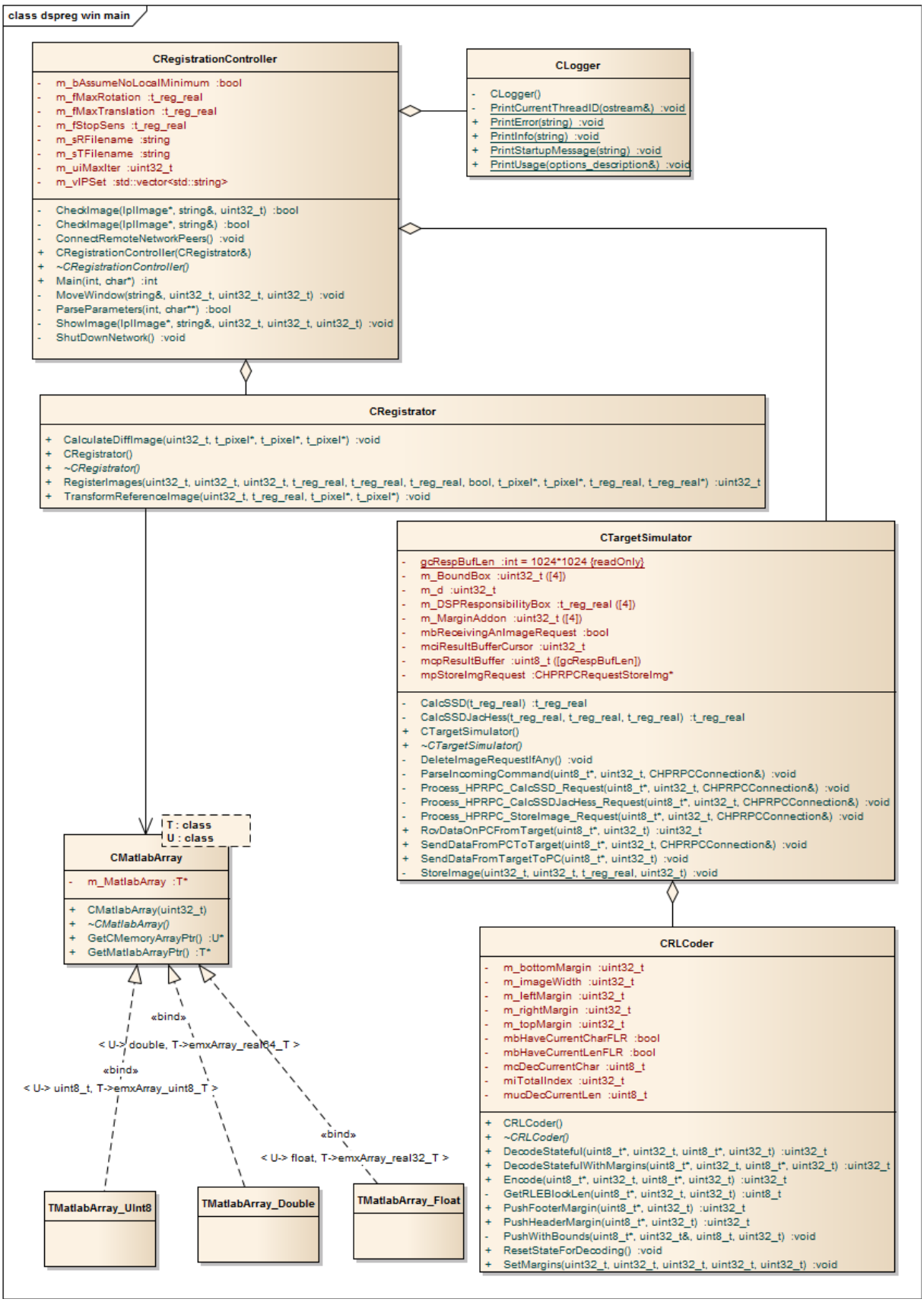


Abbildung A.4: Steuerungsklassen der PC-Clientsoftware

A.3 Konfiguration des SYS/BIOS Betriebssystems

Die Betriebssystemkonfiguration erfolgt in einem proprietären Format von Texas Instruments. Die folgende Darstellung basiert auf einer modifizierten Beispielkonfiguration von Texas Instruments.

```
1  /*
2   *   @file  dspreg_evm6678l_master.cfg
3   *
4   *   @brief
5   *      Memory Map and Program initializations for the registration
        application.
6   *
7   */

9  /*************************************************************************************
10 *  Specify all needed RTSC Moudles and configure them.
                                                    *
11 *************************************************************************************


13 var Memory    = xdc.useModule('xdc.runtime.Memory');
14 var Log       = xdc.useModule('xdc.runtime.Log');
15 var Error     = xdc.useModule('xdc.runtime.Error');
16 var Diags     = xdc.useModule('xdc.runtime.Diags');
17 var Timestamp = xdc.useModule('xdc.runtime.Timestamp');
18 var Startup   = xdc.useModule('xdc.runtime.Startup');
19 var System    = xdc.useModule('xdc.runtime.System');
20 var SysStd    = xdc.useModule('xdc.runtime.SysStd');

22 System.SupportProxy = SysStd;

24 /* Load and configure SYSBIOS packages */
25 var BIOS      = xdc.useModule('ti.sysbios.BIOS');
26 var Task      = xdc.useModule('ti.sysbios.knl.Task');
27 var Clock     = xdc.useModule('ti.sysbios.knl.Clock');
28 var Sem       = xdc.useModule('ti.sysbios.knl.Semaphore');
29 var Hwi       = xdc.useModule('ti.sysbios.hal.Hwi');
30 var Ecm       = xdc.useModule('ti.sysbios.family.c64p.EventCombiner');
31 var BiosCache = xdc.useModule('ti.sysbios.hal.Cache');
32 var HeapBuf   = xdc.useModule('ti.sysbios.heaps.HeapBuf');
33 var HeapMem   = xdc.useModule('ti.sysbios.heaps.HeapMem');
34 var Exc       = xdc.useModule('ti.sysbios.family.c64p.Exception');
35 var Cache     = xdc.useModule('ti.sysbios.family.c66.Cache');
```

```
37 BIOS.taskEnabled = true;
38 Task.common$.namedInstance = true;

40 /*
41  * Enable Event Groups here and registering of ISR for specific GEM INTC is done
42  * using EventCombiner_dispatchPlug() and Hwi_eventMap() APIs
43  */

45 Ecm.eventGroupHwiNum[0] = 7;
46 Ecm.eventGroupHwiNum[1] = 8;
47 Ecm.eventGroupHwiNum[2] = 9;
48 Ecm.eventGroupHwiNum[3] = 10;

50 /* Create a Heap. */
51 var heapMemParams = new HeapMem.Params();
52 heapMemParams.size = 0x8000000;
53 heapMemParams.sectionName = "systemHeapMaster";
54 Program.global.heap0 = HeapMem.create(heapMemParams);
55 Memory.defaultHeapInstance = Program.global.heap0;

57 /* Load and configure NDK */
58 var Global = xdc.useModule('ti.ndk.config.Global');

60 /*
61 ** This allows the heart beat (poll function) to be created but does not generate the stack threads
62 **
63 ** Look in the cdoc (help files) to see what CfgAddEntry items can be configured. We tell it NOT
64 ** to create any stack threads (services) as we configure those ourselves in our Main Task
65 ** thread.
66 */
67 Global.enableCodeGeneration = false;

69 /* Load the PDK packages */
70 var Csl  = xdc.useModule('ti.csl.Settings');
71 var Pa   = xdc.useModule('ti.drv.pa.Settings');
72 var Cppi = xdc.loadPackage('ti.drv.cppi');
73 var Qmss = xdc.loadPackage('ti.drv.qmss');

75 /* Load the Platform/NDK Transport packages */
76 var PlatformLib  = xdc.loadPackage('ti.platform.evmc6678l');
77 var NdkTransport = xdc.loadPackage('ti.transport.ndk');

79 /* Load and configure the IPC packages */
80 var MessageQ     = xdc.useModule('ti.sdo.ipc.MessageQ');
```

```
81 var Ipc          = xdc.useModule('ti.sdo.ipc.Ipc');
82 var HeapBufMP    = xdc.useModule('ti.sdo.ipc.heaps.HeapBufMP');
83 var SharedRegion = xdc.useModule('ti.sdo.ipc.SharedRegion');
84 var MultiProc    = xdc.useModule('ti.sdo.utils.MultiProc');

86 MultiProc.setConfig(null, ["CORE0", "CORE1", "CORE2", "CORE3", "CORE4",
       "CORE5", "CORE6", "CORE7"]);

88 /* Synchronize all processors (this will be done in Ipc_start) */
89 Ipc.procSync = Ipc.ProcSync_ALL;

91 /* Shared Memory base address and length */
92 var SHAREDMEM           = 0x0c200000;
93 var SHAREDMEMSIZE       = 0x00200000;

95 SharedRegion.setEntryMeta(0,
96     { base: SHAREDMEM,
97       len:  SHAREDMEMSIZE,
98       ownerProcId: 0,
99       isValid: true,
100      name: "MSMCSRAM_IPC",
101    });

103 /* ================ Logger configuration ================ */
104 var LoggerCircBuf = xdc.useModule('ti.uia.runtime.LoggerCircBuf');
105 var Diags         = xdc.useModule('xdc.runtime.Diags');
106 var Defaults      = xdc.useModule('xdc.runtime.Defaults');
107 var Main          = xdc.useModule('xdc.runtime.Main');
108 var Load          = xdc.useModule('ti.sysbios.utils.Load');

110 Load.windowInMs = 50;
111 CpuTimestamp     = xdc.useModule('ti.uia.family.c66.TimestampC66XLocal')
       ;
112 GlobalTimestamp = xdc.useModule('ti.uia.family.c66.TimestampC66XGlobal')
       ;
113 LogSync       = xdc.useModule('ti.uia.runtime.LogSync');
114 LogSync.GlobalTimestampProxy = GlobalTimestamp;
115 LogSync.CpuTimestampProxy = CpuTimestamp;

117 CpuTimestamp.maxTimerClockFreq = {lo:1000000000,hi:0};
118 CpuTimestamp.maxBusClockFreq = {lo:1000000000,hi:0};
119 GlobalTimestamp.maxTimerClockFreq = {lo:250000000,hi:0};
120 GlobalTimestamp.maxBusClockFreq = {lo:1000000000,hi:0};
121 GlobalTimestamp.cpuCyclesPerTick = 4;

123 Exc.common$.logger = Main.common$.logger;
124 Exc.enablePrint = true;
```

```
126 var LoggingSetup = xdc.useModule('ti.uia.sysbios.LoggingSetup');
127 /* Increase the sysbios logger and turn Hwi and Swi logging off */
128 LoggingSetup.sysbiosLoggerSize = 32768;
129 LoggingSetup.mainLoggerSize = 8*1024;
130 LoggingSetup.loadLoggerSize = 32768;
131 LoggingSetup.loadLogging = true;
132 LoggingSetup.sysbiosTaskLogging = false;
133 LoggingSetup.sysbiosSwiLogging = false;
134 LoggingSetup.sysbiosHwiLogging = false;
135 LoggingSetup.eventUploadMode = LoggingSetup.UploadMode_NONJTAGTRANSPORT;

137 /* ================ UIA configuration ================ */
138 /*
139  *  This example is a multi-core example, so UIA's ServiceMgr
140  *  must be configured to collect events from multiple cores.
141  */
142 var ServiceMgr = xdc.useModule('ti.uia.runtime.ServiceMgr');
143 ServiceMgr.topology = ServiceMgr.Topology_MULTICORE;
144 ServiceMgr.masterProcId = 0;

146 /* The application is using the UIA benchmark events. */
147 var UIABenchmark  = xdc.useModule('ti.uia.events.UIABenchmark');

149 /*
150 ** Create a Heap.
151 */
152 var Memory      = xdc.useModule('xdc.runtime.Memory');
153 Memory.defaultHeapSize = 0x10000;
154 Program.heap = 0x10000;
155 Program.sectMap[".vecs"]             = {loadSegment: "MSMCSRAM_MASTER", loadAlign:1024};
156 Program.sectMap[".switch"]           = {loadSegment: "MSMCSRAM_MASTER", loadAlign:8};
157 Program.sectMap[".cio"]              = {loadSegment: "L2SRAM", loadAlign:8};
158 Program.sectMap[".args"]             = {loadSegment: "L2SRAM", loadAlign:8};
159 Program.sectMap[".cppi"]             = {loadSegment: "L2SRAM", loadAlign:16};
160 Program.sectMap[".qmss"]             = {loadSegment: "L2SRAM", loadAlign:16};
161 Program.sectMap[".nimu_eth_ll2"]     = {loadSegment: "L2SRAM", loadAlign:16};
162 Program.sectMap[".far:NDK_PACKETMEM"]= {loadSegment: "MSMCSRAM_MASTER", loadAlign: 128};
163 Program.sectMap[".far:NDK_OBJMEM"]   = {loadSegment: "MSMCSRAM_MASTER", loadAlign: 16};
```

```
164 /*Program.sectMap[".far:WEBDATA"]       = {loadSegment: "DDR3", loadAlign
        : 8};*/
165 Program.sectMap[".resmgr_memregion"] = {loadSegment: "L2SRAM", loadAlign
        :128};
166 Program.sectMap[".resmgr_handles"]    = {loadSegment: "L2SRAM", loadAlign
        :16};
167 Program.sectMap[".resmgr_pa"]         = {loadSegment: "L2SRAM", loadAlign
        :8};

169 Program.sectMap[".picturebuff"] = {loadSegment: "DDR3", loadAlign:128};
        /*todo: try out MSMCRAM for bigger images ...*/
170 Program.sectMap["systemHeapMaster"]      = "DDR3";
171 Program.sectMap[".cinit"]                = "MSMCSRAM_MASTER";
172 Program.sectMap[".const"]                = "MSMCSRAM_MASTER";
173 Program.sectMap[".text"]                 = "MSMCSRAM_MASTER";
174 Program.sectMap[".far"]                  = "L2SRAM";
175 Program.sectMap[".bss"]                  = "L2SRAM";
176 Program.sectMap[".rodata"]               = "L2SRAM";
177 Program.sectMap[".neardata"]             = "L2SRAM";
178 Program.sectMap[".code"]                 = "L2SRAM";
179 Program.sectMap[".data"]                 = "L2SRAM";
180 Program.sectMap[".sysmem"]               = "L2SRAM";
181 Program.sectMap[".defaultStackSection"] = "L2SRAM";
182 Program.sectMap[".stack"]                = "L2SRAM";
183 Program.sectMap[".plt"]                  = "L2SRAM";
184 Program.sectMap["platform_lib"]          = "L2SRAM";

186 /* Add init function */
187 Startup.lastFxns.$add('&EVM_init');

189 /* Create the stack Thread Task for our application.  */
190 var tskMainThread        =  Task.create("&master_main");
191 tskMainThread.stackSize  = 0x2000;
192 tskMainThread.priority   = 0x5;
193 tskMainThread.instance.name = "master_main";

195 var tskSlaveThread       = Task.create("&slave_main");
196 tskSlaveThread.stackSize = 0x2000;
197 tskSlaveThread.priority  = 0x5;
198 tskSlaveThread.instance.name = "slave_main";
```

A.4 Matlab-Quellcode

Zur Demonstration der in Unterabschnitt 3.3.1 vorgeschlagenen matrixfreien Berechnung pro Pixel ist hier ein entsprechender Codeauszug aus Matlab aufgeführt, der pro Pixel die Gleichungen 3.1, 3.2 sowie 3.3 berechnet.

```
1  %Written by Roelof Berg at Fraunhofer Mevis Germany as part of a Bachelor Thesis for the university of applied sciences
2  %'Willhelm Buechner Hochschule'.
3  %
4  % Copyright 2012, Roelof Berg, http://www.berg-solutions.de. Licensed under the Apache License, Version 2.0 (the "License"
5  % ); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.a
6  % pache.org/licenses/LICENSE-2.0. Unless required by applicable law or agreed to in writing, software distributed under th
7  % e License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. S
8  % ee the License for the specific language governing permissions and limitations under the License.

10 %Berechnung von SSD, Jacobi-Matrix sowie Hesse-Matrix
11 function [SSD, JD, JD2] = jacobian(w, BoundBox, DSPRange, Tvec, Rvec, d)
12 %#codegen

14 s=single(single(d)/2+0.5);   %Shifting, um negative Koordinaten in Matrixbereich zu bringen

16 width = BoundBox(2)-BoundBox(1)+1;  %+1 because last entry marks last INCLUDED pixel
17 %height = uint32(BoundBox(4)-BoundBox(3)+1);

19 %Registrierungsparameter
20 FP = [cos(w(1)); -sin(w(1)); w(2); sin(w(1)); cos(w(1)); w(3)];

22 %sin/cos vorberechnen
23 msin_w1=single(-sin(w(1)));
24 mcos_w1=single(-cos(w(1)));
25 cos_w1=single(cos(w(1)));

27 SSD=single(0);
28 JD = zeros(1,3,'single');
29 JD2 = zeros(3,3,'single');
30 %Folgende beiden For-Loops laufen 1..mn mal durch das pixelmittige Koordinatengitter
31 %Dabei zeigt i auf die Position von 1..mn
```

```
32 i=uint32(1);
33 for X_mni=DSPRange(3):1:DSPRange(4)
34     for X_i=DSPRange(1):1:DSPRange(2)

36         FA_i = FP(1)*X_i + FP(2)*X_mni + FP(3) + s;
37         FA_mni = FP(4)*X_i + FP(5)*X_mni + FP(6) + s;
38         %+s weil Drehpunkt in Bildmitte

40         Ax=uint32(floor(FA_i));
41         Ay=uint32(floor(FA_mni));

43         %Fetch picture values (cache-optimized by black bounding box)
44         %Zunaechst Dimensionen des Teilbilds beruecksichtigen
45         yOffTop=uint32((Ay-BoundBox(3))*width);
46         yOffBottom=uint32((Ay-BoundBox(3)+1)*width);
47         xOffLeft=uint32(Ax-BoundBox(1)+1);
48         xOffRight=uint32(xOffLeft+1);
49         %Luminanzwerte auslesen
50         k11=single(Tvec( yOffTop + xOffLeft)) / single(255);
51         k12=single(Tvec( yOffTop + xOffRight)) / single(255);
52         k21=single(Tvec( yOffBottom + xOffLeft)) / single(255);
53         k22=single(Tvec( yOffBottom + xOffRight)) / single(255);

55         %Interpolation
56         xi=single(mod(FA_i, 1));
57         yi=single(mod(FA_mni, 1));
58         FT_i = single(k11*(1-xi)*(1-yi)+k12*xi*(1-yi)+k21*(1-xi)*yi+k22*
               xi*yi);

60         %Mathematisch abgeleitet (nach Xi)
61         JT_i = single(k22*yi - k21*yi + k11*(yi - 1) - k12*(yi - 1));
62         JT_mni = single(k22*xi - k12*xi + k11*(xi - 1) - k21*(xi - 1));

64         %Residuum
65         rdiff=single(FT_i - single(Rvec(i))/single(255));
66         FDr_i = single((rdiff*rdiff)/2);
67         JDr_i = single(rdiff);
68         SSD = single(SSD + FDr_i);

70         %Jdrta hat 6 Spalten (bei affiner Registrierung).
71         rDx=single(JDr_i*JT_i);
72         rDy=single(JDr_i*JT_mni);
73         sA = single(rDx*X_i);
74         sB = single(rDx*X_mni);
75         sC = single(rDx);
76         sD = single(rDy*X_i);
77         sE = single(rDy*X_mni);
78         sF = single(rDy);
```

```
80          %Alle Spalten von JDtap einzeln aufsummieren
81          %entspricht der Operation JDs*JDrtap
82          %(Bei Affin 6 Spalten, statt Spalte angle)
83          JD(1)=JD(1)+angle;
84          JD(2)=JD(2)+sC;
85          JD(3)=JD(3)+sF;

87          %JD2 = JDrtap' * JDrtap;    %Approximation der Hesse-Matrix s.
               Nocedal
88          JD2(1,1)= JD2(1,1) + angle*angle;
89          JD2(2,1)= JD2(2,1) + angle*sC;
90          JD2(3,1)= JD2(3,1) + angle*sF;
91          JD2(1,2)= JD2(1,2) + sC*angle;
92          JD2(2,2)= JD2(2,2) + sC*sC;
93          JD2(3,2)= JD2(3,2) + sC*sF;
94          JD2(1,3)= JD2(1,3) + sF*angle;
95          JD2(2,3)= JD2(2,3) + sF*sC;
96          JD2(3,3)= JD2(3,3) + sF*sF;

98          i=i+1;
99      end
100 end
```

A.5 Messdaten

Es folgen die Rohdaten der Performancemessungen. Damit lassen sich die zuvor getroffenen Aussagen nachvollziehen.

PC	Rechenkerne	Rechenzeit 512x512 Pixel	Rechenzeit 3000x3000 Pixel
PC-1	2	1,203 s	15,625 s
PC-2	4	0,384 s	6,027 s
PC-3	8	0,172 s	1,997 s
PC-4	12	0,125 s	1,576 s

Tabelle A.1: Messdaten PC-basierte Registrierung

DSPs	Rechenkerne	Rechenzeit 512x512 Pixel	Rechenzeit 3000x3000 Pixel
1	1	9,734 s	115,59 s
1	2	5,422 s	71,671 s
1	3	4,969 s	50,859 s
1	4	3,844 s	49,969 s
1	5	3,266 s	36,953 s
1	6	3,125 s	28,421 s
1	7	2,547 s	21,812 s
1	8	1,734 s	19,640 s
4	1	3,625 s	39,906 s
4	2	2,640 s	23,359 s
4	3	2,187 s	16,078 s
4	4	2,031 s	12,484 s
4	5	1,718 s	10,484 s
4	6	1,797 s	8,984 s
4	7	1,500 s	7,766 s
4	8	1,547 s	6,922 s

Tabelle A.2: Messdaten DSP-basierte Registrierung

Bildgröße	Transferzeit
512x512 Pixel	0,916 s
3000x3000 Pixel	1,447 s

Tabelle A.3: Messdaten Bildtransfer

A.6 HPRPC-Protokoll

Die für die vorliegende Arbeit definierten HPRPC-Netzwerkkommandos, Operations- sowie Fehlercodes sind hier aufgelistet. Der Datentyp t_reg_real ist einstellbar auf float (32-Bit) oder double (64-Bit). PC und DSP müssen identisch eingestellt sein, Default ist float. Die Protokollheader der HPRPC-Pakete sind bereits in Unterabschnitt 5.2.2 dargestellt.

A.6.1 Opcodes

Opcode:	0
Request:	HPRPC_OP_STORE_IMG
uint32_t	Image Dimensions
uint32_t[4]	BoundingBox Coordinates
uint32_t[4]	MarginAddon Sizes (to prevent pipeline hazard)
t_reg_real[4]	DSPResponsibilityBox (0 is the overall picture center)
uint32_t	Size of the uncompressed template image
uint32_t	Size of the template image after RLE compression
uint32_t	Size of the uncompressed reference image
uint32_t	Size of the reference image after RLE compression
uint8[...]	Raw data of the compressed template image
uint8[...]	Raw data of the compressed reference image
Response:	Returncode only, no data

(Image data contains 1 byte luminance information per pixel, RLE marker-char:60)

Opcode:	1
Request:	HPRPC_OP_CALC_SSD_JAC_HESS
uint32_t	Number of DSP cores to use
t_reg_real[w]	Current transformation parameters (angle in rad, x-shift, y-shift)
Response:	
t_reg_real	SSD distance measure
t_reg_real[3]	Jacobian
t_reg_real[9]	Hessian (aprx.)

Opcode:	2
Request:	HPRPC_OP_CALC_SSD
uint32_t	Number of DSP cores to use
t_reg_real[3]	Current transformation parameters (angle in rad, x-shift, y-shift)
Response:	
t_reg_real	SSD distance measure

A.6.2 Fehlercodes

0=ok (kein Fehler)
1=Unknown error
2=Invalid Argument
3=Parse Error
4=Wrong Endian
5=OutOfMemory

A.7 CD-ROM

Auf der beiliegenden CD-ROM befinden sich die in der Bibliographie gelisteten Forschungsartikel, Präsentationen sowie Datenblätter (sofern eine Genehmigung zur Veröffentlichung eingeholt werden konnte).