IMPLEMENTIERUNG EINES DICHTE-KORRESPONDENZFINDUNGS-ALGORITHMUS AUF EINEM FPGA

Diplomarbeit
vorgelegt von Torge Rabsch

unter Betreuung von Dr. C. Perwass
bei Prof. Dr. G. Sommer,
Lehrstuhl für kognitive Systeme
am Institut für Informatik der
Christian-Albrechts-Universität zu Kiel

Kiel, 25. April 2003

Zusammenfassung

Diese Arbeit beschäftigt sich mit einem wichtigen Aspekt der digitalen Bildverarbeitung. Es wird ein vorhandenes Verfahren zum Ermitteln von Bildpunktkorrespondenzen untersucht, um dessen Implementierung in einer Hardwareschaltung zu realisieren.
Das Verfahren gründet sich auf der Annahme, daß die Disparitäten korrespondierender Bildpunkte lokal einer statistischen Verteilung genügen. In einem iterativen Prozeß werden lokale Wahrscheinlichkeiten von globalen Bedingungen beeinflußt. Es wird für jeden Bildpunkt ein am besten passender Korrespondenzpartner gefunden. Dazu wird keine Kenntnis der Kamerageometrie vorausgesetzt, so daß das Verfahren sowohl für Stereo-Sehen als auch für optischen Fluß angewendet werden kann.
Den Hauptteil der Arbeit bildet die teilweise parallele, einem neuronalen Netz entsprechende Realisierung des Verfahrens in einer Hardwareschaltung, die auf einem FPGA implementiert und angewendet wird. Dazu wird ein Koprozessor entworfen, der ohne Steuerung von außen arbeitet. So wird zum einen die Verarbeitung beschleunigt und zum anderen die CPU entlastet. In Experimenten zeigt sich, daß das Einsatzgebiet des System zwar beschränkt ist, innerhalb der Rahmenbedingungen aber gute Ergebnisse erzielt werden.

Schlüsselwörter: Korrespondenzfindung, Neuronal, FPGA, Hardware, Implementierung

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Arbeit beschäftigt sich mit der Implementierung eines Korrespondenzfindungsalgorithmus auf einem FPGA. Das Korrespondenzfindungsproblem ist ein wichtiger Aspekt in der digitalen Bildverarbeitung. Es geht darum, Zusammengehörigkeiten von Bildpunkten aus zwei Kamerabildern zu bestimmen. Daraus können dann Informationen wie Bewegungen oder die dreidimensionale Tiefe innerhalb der abgebildeten Szene gewonnen werden. Grundidee zur Bestimmung des optischen Flusses und des Stereosehens ist, daß Kamerabilder eine Projektion einer realen Szene darstellen. Unterscheiden sich die Positionen zweier Kameras oder der Zeitpunkt der Bildaufnahme, so haben nach B. Horn und B. Schunck [HS81] korrespondierende Bildpunkte gleichbleibende Eigenschaften. Diese Eigenschaften sind beispielsweise Farbe und die Struktur der Nachbarschaft. Es gibt verschiedene Ansätze, die Invarianz der Eigenschaften, aber auch die Eigenschaften selbst zu beschreiben. Merkmalbasierte Methoden beschränken sich darauf, einzelne, besondere Merkmale aus den Bildern zu extrahieren und diese untereinander in Beziehung zu setzen. Dies hat zur Folge, daß Korrespondenzen nur zwischen diesen Merkmalen gewonnen werden, aber keine Aussagen über Orte gemacht werden können, an denen keine besonderen Merkmale extrahiert wurden.
In der Anwendung werden häufig aber dichte Korrespondenzen gefordert, also muß eine Aussage für eine große Anzahl von Bildpunkten getroffen werden. Eine Möglichkeit ist es, von einzelnen leicht extrahierbaren und in Beziehung setzbaren Merkmalen, sogenannter *Saat*-Punkte, auszugehen. Darauf wendet M. Lhuillier in seinem Bereichserweiterungsverfahren [Lhu98] eine Wachstumsmethode an, um bekannte Regionen zu vergrößern. Dazu können nach Lhuillier und L. Quan [LQ99] zunächst lokale und darauf folgend globale geometrische Bedingungen angewendet werden, die Korrespondenzen untereinander erfüllen müssen. So werden Nachbarschaften korrelationsbasiert untersucht und es fließen Informationen von sicheren Positionen in unsichere Bereiche hinein. Ein anderes Verfahren, welches sich von vornherein mit der Bestimmung dichter Korrespondenzen beschäftigt, ist von A. Zuloaga *et al.* [ZEMB] entwickelt. Es basiert auf dem Ansatz von Horn und Schunck [HS81], die den räumlichen und zeitlichen Gradienten der Disparitäten betrachten, wobei Zuloaga zusätzlich die Glattheit des Gradienten fordert. Falls die genaue Kamerageometrie bekannt ist, kann wie bei M. Männer [MMMH01] mit Hilfe der Epipolargeometrie für das Stereoproblem der Suchraum auf eine Linie

eingeschränkt und somit die Korrespondenzfindung erleichtert werden.
Ein anderer Ansatz ist es, der Wahrscheinlichkeitstheorie von Bayes folgend Wahrscheinlichkeiten für Korrespondenzen zu verwenden und daraus eine diskrete Wahrscheinlichkeitsverteilungsfunktion (*pdf*) zu bilden. Dieser Ansatz ist von S. Scharstein und R. Szeliski in ihrem Verfahren zur Stereo-Korrespondenzfindung mit nichtlinearer Diffusion [SS98] aufgegriffen worden. In dieser Arbeit wird das von C. Perwass und G. Sommer aus [PS01] weiterentwickelte Verfahren zur dichten Korrespondenzfindung [PS02] angewendet. Es wird angenommen, daß Korrespondenzen lokal durch eine partielle *pdf* beschrieben werden können. Dabei folgen richtige Korrespondenzen der angenommenen Verteilung, während falsche Korrespondenzen gleichverteilt sind. In einem iterativen Prozeß wirken sich lokale Bedingungen auf eine zunehmend größere Nachbarschaft aus, so daß diese Bedingungen globalen Einfluß bekommen. Der Ansatz des in dieser Arbeit verwendeten Verfahrens ist in Anlehnung an Perwass und Sommer [PS02] in **Kapitel 2** vorgestellt.
Die Realisierung, welche den Hauptteil dieser Arbeit bildet, wird in **Kapitel 3** diskutiert. Es wird die Parallelisierbarkeit des Verfahrens genutzt und dem neuronalen Ansatz folgend ein Rechenwerk entwickelt, welches in einer Hardwareschaltung implementiert wird. Zum Aufbau der Schaltung wird ähnlich der Hardware-Beschreibungssprache VHDL [Gau] eine Hochsprache auf Basis von C++ verwendet. Die von K. Kornmesser bereitgestellte Entwicklungsumgebung CHDL [Kor02] ist eine C++ Klassenbibliothek, in der jedes Schaltelement in einer Klasse repräsentiert wird. Komplexe Schaltelemente können von den Grundelementen abgeleitet werden. Dies bietet ein auf höchster Ebene übersichtlichen Aufbau des Designs und ermöglicht es, eigene Bauteile in einer Bibliothek anzulegen und daraus zu verwenden. Die Implementierung erfolgt in einem FPGA (*Field Programmable Gate Array*), dessen grundlegende Funktion von M. Wannemacher in seinem FPGA-Kochbuch [Wan97] erläutert ist. Der FPGA wird mittels PCI-Bus in einen Rechner integriert und wird als selbst programmierter Koprozessor genutzt. Ergebnisse der Implementierung sind in **Kapitel 4** anhand einiger Beispiele in verschiedenen Experimenten dargestellt und erläutert. Den Abschluß bildet eine Schlußfolgerung in **Kapitel 5**, die sich besonders auf die vorgestellte Realisierung bezieht.

1.1 Hardware

Die verwendete Hardware ist das von der Firma Silicon Software GmbH[Sil] entwickelte System *microEnable* [Nof99], welches neben einem FPGA der Familie $XC4000$ von Xilinx [Xil] ein $2\mathsf{MB}$ RAM beinhaltet und über den PCI-Bus mit dem Hauptrechner kommunizieren kann. Weitere Implementierungen auf dieser Plattform sind die Echtzeit-Eckendetektion von K. Köser [Kös01] und die schnelle Berechnung von FIR-Filteroperationen von M. Männer und S. Hezel[MH99]. Der FPGA selbst besteht aus einer Anzahl von 3136 programmierbaren Logikblöcken (*CLB*), die in einem zweidimensionalen Feld angeordnet sind. Die Logikblöcke selbst bestehen aus jeweils zwei frei programmierbaren $4bit$ Wertetabellen (*Lookup-Table*), zwei Zustandsspeichern (*FlipFlop*)

und weiterer spezieller Logik zur Behandlung des Übertrags bei Additionen. Die Logikblöcke sind nicht fest miteinander verbunden, sondern werden in einer Initialisierungsphase nach Bedarf verschaltet, ebenso wie die Wertetabellen mit Funktionswerten belegt werden. Das ermöglicht es, komplexe Schaltwerke mit individueller Funktion zu realisieren. Die Initialisierung kann direkt vor der Anwendung erfolgen und ist reversibel, was den Unterschied zu ASICs (*Application-Spezific-Integrated-Circuits*) bildet. Diese Eigenschaft bringt nach S.M. Scalera [SML98] zusätzliche Vorteile, da die Hardware in einem Prozeß mehrfach genutzt werden kann.
Anders als in herkömmlichen programmierbaren logischen Feldern (*PLA*), die zur Realisierung individueller binärer Schaltnetze verwendet werden, sind durch die Zustandsspeicher und die frei schaltbaren Verbindungen unter den Logikblöcken Schaltwerke möglich, deren Datenfluß keiner vorgegebenen Struktur folgen muß. So können neben einfachen arithmetischen Funktionen, die I.V. Klotchkov und S. Pedersen in ihrer Codedesign-Studie [KP] realisieren, auch komplexere Aufgaben bearbeitet werden, wie es die Arbeitsgruppe um B. Draper in allgemeiner Bildverarbeitung [DNB$^+$] und A. Zuloaga im speziellen Problem des optischen Flusses [ZEM] gezeigt haben. Durch die Anordnung des FPGAs auf einer PCI-Bus Karte mit eigenem Speicherelement und eigener Taktung kann dieser wie bei Männer und Hezel [MH99] als frei programmierbarer Koprozessor verwendet werden. So ist es möglich, selbständig und unabhängig von außen arbeitende Prozessoren zu konstruieren, deren Aufbau und Funktion dem jeweiligen Anwendungsgebiet angepaßt sind. Die Vorteile sind, daß anders als in herkömmlichen Prozessoren, die nach dem SISD-Prinzip (*Single-Instruction-Single-Data*) arbeiten und deren Architektur fest vorgegeben ist, neben der eigentlichen Funktion auch die Architektur selbst bestimmt werden kann. So ist es möglich, mehrere Rechenwerke auf die jeweilige Funktion optimiert zu implementieren, deren Steuerung unabhängig voneinander erfolgen kann. Daraus kann sich eine für Anwendungen wie die von J. Woodfill entwickelte Stereo-Detektion [WHZ] sehr effiziente Architektur nach dem MIMD-Prinzip (*Multiple-Instruction-Multiple-Data*) ergeben, die Woodfill in dem PARTS-System [WH97] realisiert hat. Besonders in der digitalen Bildverarbeitung, wie B. Jähne sie in seinem Buch [Jäh97] darüber beschreibt und mit der sich auch die vorliegende Arbeit beschäftigt, sind häufig viele Daten von diskreten Punkten eines Bildes vorhanden, auf die dieselbe Operation angewendet werden soll. Für diese Aufgabe eignet sich besonders eine Architektur nach dem SIMD-Prinzip (*Single-Instruction-Multiple-Data*), wie auch Männer sie in seinem Korrespondenzfindungsverfahren [MMMH01] vorschlägt, die mit nur einem Steuerwerk und mehreren parallel angeordneten Rechenwerken die gewünschte Funktion erfüllt. Dies führt im Vergleich zu sequentiell arbeitenden herkömmlichen Prozessoren zu einem Geschwindigkeitsvorteil, der sowohl auf der Optimierung der Rechenwerke auf die jeweilige Funktion beruht, als auch durch deren Parallelität begründet ist.
In dieser Arbeit werden genau diese Vorteile genutzt, um einen speziellen Prozessor zu entwerfen, der nach dem SIMD-Prinzip das Verfahren von Perwass und Sommer [PS02] zur dichten Korrespondenzfindung realisiert. Die Architektur wird nach dem *Top-Down*-Prinzip von höchster Ebene in Schritten hin zu Elementen, welche sich von Basiselementen ableiten lassen, entwickelt. Hierzu werden die einzelnen Verschaltungen der Elemente graphisch dargestellt, um deren Funktion und deren Parallelität zu verdeutlichen. Die Umsetzung der entwickelten Elemente in Form von CHDL-Klassen ist in die Bibliothek KCC

(*Kiel CHDL Component-library*) [PGR01] integriert. Sie stehen somit als neue Basiselemente für weitere Entwicklungen zur Verfügung. Die Einbettung des Prozessors in einen Rechner wird mit der nötigen Kommunikation und möglicher Vor- und Nachverarbeitung vorgestellt.

Kapitel 2

Vorstellung des Korrespondenzfindungs-Algorithmus

Bildpunktkorrespondenzen zu finden ist eine wichtige Aufgabe in der Bildverarbeitung. Es ist kompliziert, alle Aspekte eines realen Systems zu modellieren. Daher werden von C. Perwass in der Herleitung eines Verfahrens zur dichten Korrespondenzfindung [PS02], einige Annahmen gemacht, um sich an die Realität anzunähern. Es wird einen Ansatz der Wahrscheinlichkeitstheorie nach Bayes verfolgt, um alle einschränkenden Annahmen explizit machen zu können. So wird ein Verfahren entwickelt, welches in dem gewählten Zusammenhang nachweislich korrekt ist.
Perwass geht von zwei Bildern aus, deren Bildpunkte insofern zusammengehörig sind, als daß sie dieselbe Szene abbilden. Dabei unterscheiden sich die beiden Bilder entweder durch einen unterschiedlichen Kamerablickwinkel oder durch einen unterschiedlichen Zeitpunkt der Aufnahme, wobei sich die Szene in der Zwischenzeit leicht verändert haben kann. Im ersten Fall handelt es sich um *Stereo-Sehen* und im zweiten Fall um den *optischen Fluß*. Beide Themengebiete beinhalten das Problem, herauszufinden, welche Zusammengehörigkeiten zwischen den Bildpunkten der beiden Bilder bestehen. Somit wird mit dem so genannten Korrespondenzfindungs-Problem ein wichtiger Teil im *Computer Vision*, dem künstlichen Sehen erschlossen.
In dem Ansatz wird keine Kenntnis der Kamerageometrie vorausgesetzt. Die einzigen einschränkenden Annahmen, die gemacht werden, berufen sich auf Ähnlichkeiten der Bildpunkte sowie deren Ordnung untereinander.
Es wird angenommen, daß richtige Korrespondenzen einer bestimmten statistischen Verteilung genügen, während falsche angenommene Korrespondenzen gleichbedeutend mit Rauschen sind und somit gleichverteilt sind. Das Verfahren verstärkt schrittweise die Wahrscheinlichkeiten von den Korrespondenzen, die der entsprechenden Verteilung genügen, und alle anderen werden abgeschwächt.

2.1 Rahmenbedingungen

Zunächst werden die Rahmenbedinungen für das Korrespondnsfindungsproblem entwickelt. Seien A und B zwei mehrkanälige, diskrete, zweidimensionale Aufnahmegeräte (Kameras), wobei jedes diskrete Element *Pixel* genannt wird. Mathematisch kann man $\{A, B\}$ als Abtastraum mit dem folgenden Teilbereich betrachten.

$$\{A_i^v \in \mathcal{S},\ B_i^v \in \mathcal{S} :\ v \in \mathcal{M},\ i \in \mathcal{I}\} \tag{2.1}$$

A_i^v ist dabei eine Variable, die einem Pixel in Bild A an Position i in Kanal v entspricht. Dabei werden alle Elemente in dem Teilbereich des Abtastraumes als statistisch unabhängig angenommen. Daraus folgt die Annahme, die Punkt-Ausbreitungs-Funktion der Kamera sei ein Dirac-Impuls.
Der Bereich $\mathcal{M}$ ist definiert als $\mathcal{M} := \{1, \ldots, M\}$, wobei $M \in \mathbb{N}$ die Anzahl der (Farb-) Kanäle eines Bildes angibt. Für ein RGB Farbbild würde $M = 3$ sein. Die Menge $S \subset \mathbb{Q}$ ist eine gleichmäßige Unterteilung des Intervalls $[0, 1]$.

$$\mathcal{S} := \{(r/(S-1)) \in \mathbb{Q} : r \in \{0, \ldots, S-1\}, S \in \mathbb{N}, S > 1\} \tag{2.2}$$

S gibt dabei die Anzahl der diskreten Werte an, die ein Pixel in einem bestimmten Kanal annehmen kann. Die Menge $\mathcal{I}$ ist die Menge der Positionen im Bild, wobei $\mathsf{Img_x}$ die Anzahl der horizontalen und $\mathsf{Img_y}$ die Anzahl der vertikalen Pixel angibt.

$$\mathcal{I} := \{(x, y) : x \in \{1, \ldots, \mathsf{Img_x}\}, y \in \{1, \ldots, \mathsf{Img_y}\}\} \tag{2.3}$$

Der Algorithmus läßt sich in zwei grundlegende Schritte unterteilen. Zuerst wird die Wahrscheinlichkeitsverteilung der Ähnlichkeiten für jedes Pixel in Bild A in einem zugehörigen Suchraum in Bild B berechnet. Weiter ist die Annahme, daß richtige Korrespondenzen einer speziellen örtlichen statistischen Verteilung entsprechen. In einem zweiten Schritt wird die Ordnung der möglichen Korrespondenzen betrachtet und daraus eine neue Wahrscheinlichkeitsverteilung ermittelt. Im folgenden wird Zunächst gezeigt, wie die Wahrscheinlichkeiten basierend auf den Ähnlichkeiten ermittelt wird. Dann wird näher auf die Extraktion der richtigen Korrespondenzen aufgrund ihrer Ordnung eingegangen.

2.2 Pixel-Ähnlichkeiten

Die erste Annahme ist, daß ein spezielles Merkmal einer Szene, projeziert auf zwei verschiedene Bildebenen, ähnliche Pixel-Farbwerte erzeugt. Daher brauchen wir ein Maß für die Ähnlichkeit von Pixelwerten. So ein Maß kann auch als Wahrscheinlichkeit angesehen werden, daß zwei Pixel vom selben Merkmal der Szene stammen. Es wird angenommen, daß die Farbe des reflektierten Lichtes der Merkmale der Szene sich nur sehr wenig in Abhängigkeit vom Betrachtungswinkel ändert. Dabei ist mit leichten Störungen durch unterschiedliche Wege durch die Luft und durch Rauschen der Kamera zu rechnen.

Die gemessenen Pixel-Farbwerte der Projektion eines Merkmals der Szene in Bezug auf den Betrachtungswinkel wird als Gauß-Verteilung angenommen. Das bedeutet, mit dem Mittelwert C^v_{Ai} und der Standardabweichung σ^v des Bildaufnahmeprozeß der Kamera A in Bezug auf ein Pixel an Position i in Farbkanal v können wir die Wahrscheinlichkeitsfunktion für A^v_i angeben als

$$P(A^v_i = a | C^v_{Ai} = c) = g(a, c, \sigma), \tag{2.4}$$

Die Funktion g ist ein Gauß mit Mittelwert c, Standardabweichung σ^v und einem konstanten Normalisierungsfaktor ρ definiert als

$$g(x, y, \sigma) := \rho \ \exp\left(-\frac{(x-y)^2}{2\sigma^2}\right). \tag{2.5}$$

Anders ausgedrückt kann man auch sagen, daß es einen wahren Pixel-Wert c gibt, von dem der gemessene Wert a durch Rauschen abweicht. Wenn nun zwei Pixel A_i und B_j vom selben Merkmal der Szene stammen, dann ist deren "wahrer" Wert derselbe. Somit ergibt sich für Pixel B_j in Anlehnung an Gleichung (2.4)

$$P(B^v_j = b | C^v_{Bj} = c) = g(b, c, \sigma). \tag{2.6}$$

Wir sind weiter an der Ähnlichkeit zweier Pixel A^v_i und B^v_j interessiert, deren wahre Pixel-Werte C^v_{Ai} und C^v_{Bj} gleich sind, da sie von dem selben Merkmal der Szene stammen. Mit Hilfe der Bayes'schen Gesetze schreibt Perwass die Wahrscheinlichkeit, daß C^v ein gemeinsamer wahrer Pixel-Wert ist, wenn A^v_i und B^v_j gegeben sind, als

$$P(C^v | A^v_i, B^v_j) \simeq P(A^v_i, B^v_j | C^v)\ P(C^v). \tag{2.7}$$

Dabei soll $\simeq$ die Gleichheit bis auf einen skalaren Faktor bedeuten. Weil es keinen Grund zur Annahme von bevorzugten Werten für C^v gibt, wird hier eine Gleichverteilung angenommen. Weiter wird angenommen, daß A^v_i und B^v_j nicht a priori korreliert sind.
Also ist die kombinierte Wahrscheinlichkeitsverteilungsfunktion für A^v_i und B^v_j bei gegebenem gleichen C^v das einfache Produkt der einzelnen Wahrscheinlichkeitsfunktionen, geschrieben als

$$P(C^v | A^v_i, B^v_j) \simeq P(A^v_i | C^v)\ P(B^v_j | C^v). \tag{2.8}$$

Wir sind an der maximalen Wahrscheinlichkeit interessiert, daß zwei Pixel auf dem selben wahren Pixel basieren. Dazu wird die Pixelähnlichkeitsfunktion als

$$s^v(a, b) := \max_{c \in [0,1]} P(C^v = c | A^v_i = a,\ B^v_j = b), \quad mit \ \ a, b \in \mathcal{S} \tag{2.9}$$

definiert. Durch Ableiten nach c findet man das Maximum bei $c = \frac{1}{2}(a + b)$. Somit kann Gleichung (2.9) als

$$s^v(a, b) = g(a, b, \sqrt{2}\sigma^v) \tag{2.10}$$

geschreiben werden. Die gesamte Ähnlichkeit von zwei Pixeln ist dann durch das Produkt der Ähnlichkeiten aller (Farb-)Kanäle definiert durch:

$$s(a,b) := \prod_{v \in \mathcal{M}} s^v(a^v, b^v) \tag{2.11}$$

Wobei hier $a = (a^1, \ldots, a^{\mathcal{M}})$ und $b = (b^1, \ldots, b^{\mathcal{M}})$ ist. Letztendlich ist dies also die Exponentialfunktion der negativen Summe der quadrierten Differenzen[1] der Pixel-Farben. Dieses Pixelähnlichkeitsmaß ist das Gleiche wie das in [Bel96] entwickelte. Bei der Wahl der Farbkanäle v verwenden wir wie bei der Repräsentation im RGB-Format die drei Farben Rot, Grün und Blau. In [SH98] sind weitere denkbare Repräsentationen diskutiert. Außer den Farbkanälen sind in Gleichung (2.11) auch andere Eigenschaften, die Korrespondenzen repräsentieren, denkbar.
Mit Gleichung (2.11 kann nun anhand der Farbwerte die Wahrscheinlichkeit berechnen werden, daß ein Pixelpaar (A_i, B_j) vom selben Merkmal der Szene stammt. Diese Bedingung ist notwendig, aber nicht ausreichend für eine Korrespondenz, da in realen Bildern häufig viele Pixel die ähnliche Farbe haben. Daher werden von Perwass nun zwei weitere Bedingungen vorgestellt. Erstens wird ein bestimmter Bereich im Bild B begrenzt, wo eine korrekte Korrespondenz zu einem Pixel A_i zu erwarten ist. Weiter wird eine Ordnungsbedingung aufgestellt, die korrekte Korrespondenzen untereinander erfüllen müssen.

2.3 Suchraum

Sei $d \in \mathbb{Z}^2$ der durchschnittliche Abstand der Pixelpositionen von korrekten Korrespondenzen zwischen den Bildern A und B. Dann wird angenommen, daß für alle Positionen $i \in \mathcal{I}$ die korrekte Korrespondenz zu A_i in kleinem Abstand zu B_{i+d} liegt. Man sucht also die Korrespondenz nur in einem begrenzten Bereich um B_{i+d}. Das Berechnen der Gleichung (2.11) wird also nicht für alle möglichen Pixel-Kombinationen durchgeführt, sondern beschränkt sich somit auf einen kleinen "Suchraum".
Diese Bedingung wird nun mathematisch beschrieben. Seien $X_A, X_B \in \mathcal{I}$ zwei Zufallsvariablen, welche die Pixelpositionen eines Korrespondenzpaares in den Bildern A und B angeben. Dann ist a priori ihre gemeinsame Wahrscheinlichkeitsverteilungsfunktion gegeben durch

$$\begin{aligned} P(X_A = x_A, X_B = x_B) &:= U_{\mathcal{T}}(x_B - x_A - d), \\ U_{\mathcal{T}} &:= \begin{cases} 1/|\mathcal{T}| & : \ x \in \mathcal{T} \\ 0 & \text{sonst} \end{cases}, \\ \mathcal{T} &:= \{(x,y) \in \mathbb{Z}^2 : -\mathsf{Test_x} \le x \le \mathsf{Test_x}, -\mathsf{Test_y} \le y \le \mathsf{Test_y}\}, \end{aligned} \tag{2.12}$$

wobei $\mathsf{Test_x}$, $\mathsf{Test_y} \in \mathbb{N}$ die Größe des Suchraumes angeben, welcher hier als Rechteck angenommen wird. Dabei ist $U_{\mathcal{T}}(x)$ eine Gleichverteilung über die Elemente der Menge $\mathcal{T}$. Die so definierten Wahrscheinlichkeitswerte innerhalb des Suchraumes können also auch als diskrete Wahrscheinlichkeitsverteilung für die Korrespondenzen von Pixeln in Bild A zu Pixeln in Bild B angesehen werden. Abbildung 2.1 zeigt die Größe und die Position

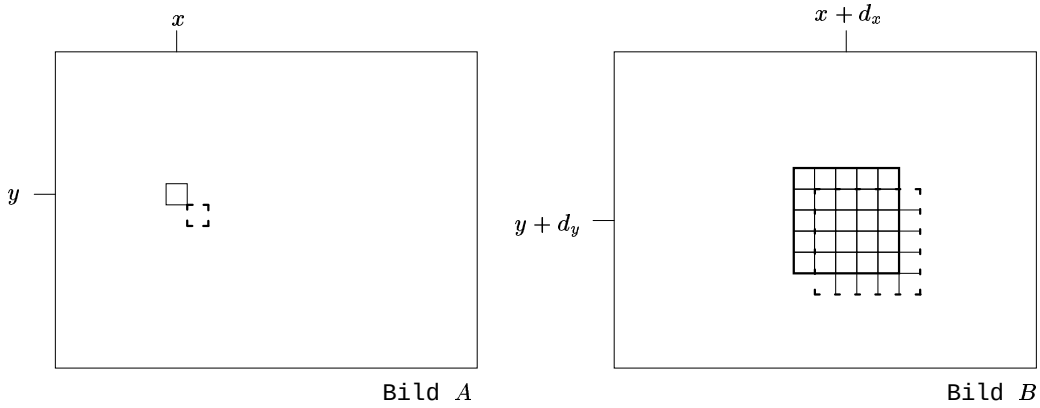


Abbildung 2.1: Größe und Position der Suchräume im Bild

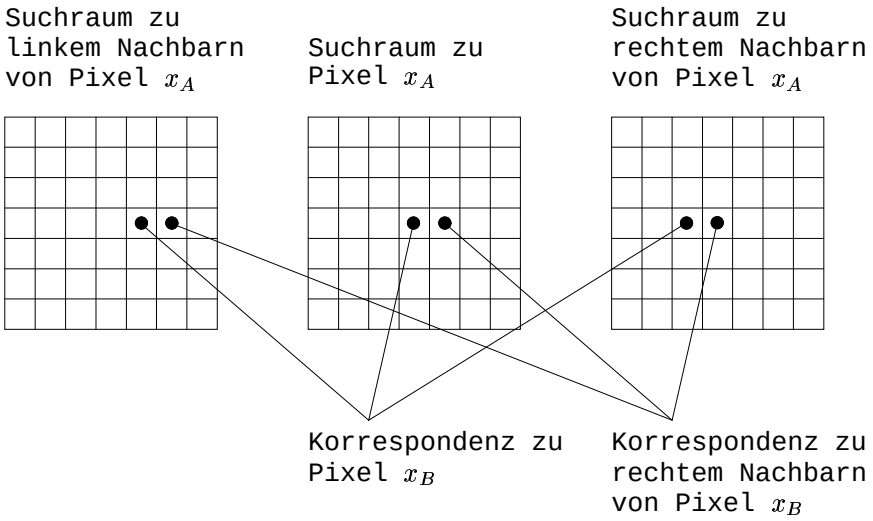


Abbildung 2.2: Suchraum-Einträge und Korrespondenz-Partner bei festem x_A und x_B

der Suchräume in Bild B für zwei benachbarte Pixel aus Bild A. Die Größe ist hier durch $\mathsf{Test_x} = \mathsf{Test_y} = 5$ festgelegt. Die Position ergibt sich durch Verschiebung um den Abstand d. Der fett umrandete Bereich in Bild B, zentriert an Position $(x+d_x, y+d_y)$ ist der Suchraum zum Pixel an Position (x, y) in Bild A. Der benachbarte, gestrichelt umrandete Pixel an Position $(x+1, y+1)$ in Bild A hat als Suchraum für mögliche Korrespondenzen den gestrichelt umrandeten Bereich zentriert an Position $(x + d_x + 1, y + d_y + 1)$ in Bild B. Dieser überlappt teilweise mit dem Suchraum seines Nachbarn.
Abbildung 2.2 zeigt die Zuordnung von Einträgen in Suchräumen zu den Korrespondenzen von Pixel-Paaren. Jeder Eintrag entspricht einer Korrespondenzwahrscheinlichkeit. Zum Beispiel enthält das mittlere Feld im Suchraum zu Pixel x_A die Wahrscheinlichkeit der Korrespondenz des Pixels x_A zu Pixel x_B. Weitere Beispiele von Korrespondenzen sind entsprechend gekennzeichnet.
Im folgenden wird die Suchraum-Wahrscheinlichkeitsverteilung mit einer Ordnungsbedingung kombiniert, um Korrespondenzen zu begünstigen, die eine hohe Pixelähnlichkeit haben und in Bezug auf ihre Position der Ordnungsbedingung genügen.

2.4 Ordnungsbedingung

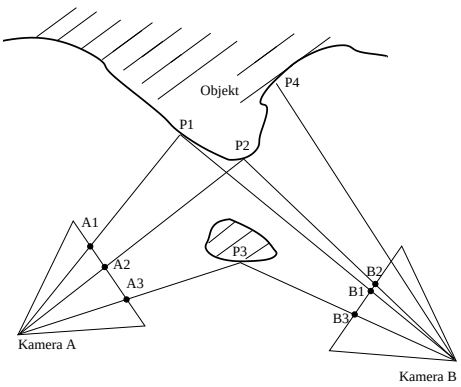


Abbildung 2.3: Ordnungs-Szenerie in der Draufsicht

Die von Perwass gewählte Ordnungsbedingung basiert auf folgender Annahme. Abbildung 2.3 zeigt schematisch eine Szene mit zwei Kameras in der Draufsicht. Darin sind drei Objekt-Punkte $P1, P2, P3$ und deren Projektionen $A1, A2, A3$ und $B1, B2, B3$ in die Bildebenen der Kameras eingezeichnet. Die Punkte $P1$ und $P2$ sind so angeordnet, daß sie in beiden Projektionen die gleiche Reihenfolge haben. In der Draufsicht ist $P1$ links von $P2$. In Bild A ist $A1$ links von $A2$, und ebenso ist in Bild B $B1$ links von $B2$. Diese Ordnungsübereinstimmung wird zerstört, wenn die Szene bzw. das darin betrachtete Objekt eine Unterbrechung in der Tiefe aufweist. Dieser Fall ist in der Abbildung durch den Objektteil mit Punkt $P3$ gegeben. Die Ordnung von $P3$ im Vergleich zu den Punkten $P1$ und $P2$ ist in den beiden Bildern unterschiedlich. Bei diesem Aufbau der

[1] Vergleiche engl. “Sum of Squared Differences (SSD)” in [Bel96]

Szene und dieser durch die Kameras gewählten Betrachtungsweise ist also keine einheitliche Ordnung festzustellen. In realen Szenen ist relativ häufig mit solchen Sprüngen in Bezug auf die Tiefe zu rechnen. Diese treten besonders dann auf, wenn sich mehrere Objekte in der Szene befinden. Störungen der Ordnung sind dann an den Objektkanten zu finden, während innerhalb von Objekten die Ordnung meist wieder einheitlich ist.
Es nun angenommen, daß im allgemeinen zumindest lokal die Ordnung gemäß $P1$ und $P2$ erfüllt ist und nur bei Unterbrechungen ebenfalls lokal gestört ist. Lokale Störungen dieser Annahme werden zu ebenfalls lokalen Fehlern in den Korrespondenzen führen, die in in Kauf genommen werden.
Ebenfalls kann es durch die verschiedenen Betrachtungswinkel der beiden Kameras dazu kommen, daß durch Tiefensprünge in Abhängigkeit vom Betrachtungswinkel einige Punkte der Szene zwar in dem einen Bild erscheinen, aber im anderen Bild von einem Objekt verdeckt sind. Dies ist in der Abbildung 2.3 mit Punkt $P4$ der Fall. Dieser ist in Bild B ganz rechts zu sehen, in Bild A allerdings wegen Verdeckung nicht zu zu sehen. Solche halb verdeckten Pixel können nach [SS98] gesondert behandelt werden. Unbeachtet dessen definiert Perwass nun die Ordnungsbedingung wie folgt lokal.
Sei (x_A, x_B) ein Paar von Pixelpositionen mit korrekter Korrespondenz zwischen Bild A und Bild B. Weiterhin sei in Bild A eine andere Pixelposition y_A in der $8er$ Nachbarschaft von x_A, was bedeutet $y_A - x_A = z \in \mathcal{N}$ mit

$$\mathcal{N} := \{(u, v) : u \in \{-1, 0, 1\}, v \in \{-1, 0, 1\}, (u, v) \neq (0, 0)\}. \tag{2.13}$$

Die Ordnungsbedingung sagt nun, wenn $y_A - x_A = z$ ist, dann werden die Korrespondenzpartner $y_B - x_B \approx z$ erfüllen. Paare von direkt benachbarten Pixel sollen also Paare von Korrespondenzpartnern haben, deren Nachbarschaft in Bezug auf Richtung und Entfernung sehr ähnlich ist. Diese Ähnlichkeit ist in Abbildung 2.4 durch den Differenzvek-

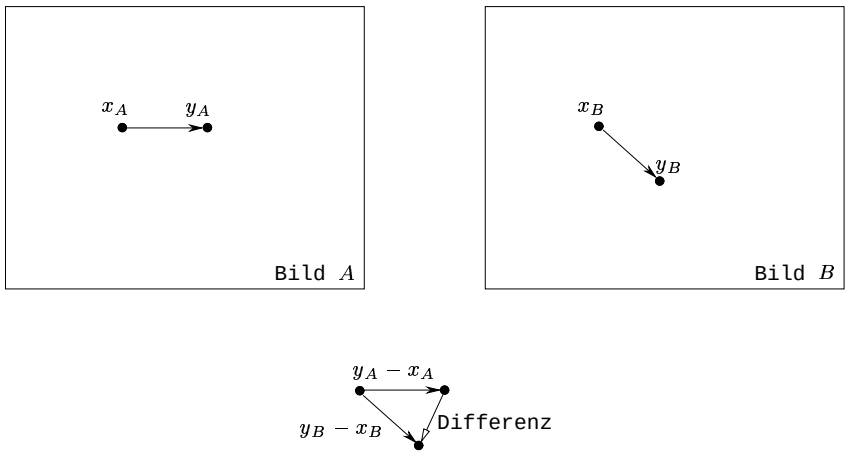


Abbildung 2.4: Differenz der Abstandsvektoren als Kriterium für Ordnung der Korrespondenz-Paare (x_A, x_B) und (y_A, y_B).

tor veranschaulicht. Um diese Bedingung zu implementieren, muß eine Wahrscheinlichkeitsverteilungsfunktion $h(x_A, x_B, y_A, y_B)$ definiert werden, welche das Maximum bei

$y_A - x_A = y_B - x_B$ hat. Die Form von h stellt die Erwartung über die Verteilung von korrekten Korrespondenzen dar. Eine mögliche Wahl ist

$$h(x_A, x_B, y_A, y_B) := g(y_B - x_B, y_A - x_A, \sigma_h). \tag{2.14}$$

Daraus leitet Perwass nun ein Wahrscheinlichkeitsmaß ab, welches die Ordnungsbedingung beschreibt.
Sei $\mathcal{C} := \{X_A^l, X_B^l\}$ die Menge der Pixel-Korrespondenzen zwischen den Bildern A und B. Dann ist $P(A, B|\mathcal{C})$ die Wahrscheinlichkeitsverteilungsfunktion für die Bilder A, B bei gegebener Menge der Pixel-Korrespondenzen. Da einzelne Pixel in den Bildern als statistisch unabhängig angenommen werden, kann man dies schreiben als:

$$P(A, B|\mathcal{C}) = \prod_l P(A_{x_A^l}, B_{x_B^l}|X_A^l, X_B^l). \tag{2.15}$$

Für ein einzelnes Pixelpaar wird als Wahrscheinlichkeitsverteilungsfunktion die Pixelähnlichkeitsfunktion (2.11) aus dem Abschnitt 2.2 angenommen. Damit ergibt sich

$$P(A_{x_A} = a, B_{x_B} = b|X_A^l = x_A, X_B^l = x_B) = s(a, b). \tag{2.16}$$

Nun sind wir natürlich andersherum an $P(C|A, B)$ interessiert und damit an der Menge der Korrespondenzen bei gegebenen Bildern. Wie wir oben gesehen haben, sind die unterschiedlichen Pixel-Korrespondenzen generell statistisch unabhängig von den gegebenen Bildern. Grundlegender kann man die Bedingung für Korrespondenzen folgendermaßen beschreiben. Man stellt sich einen blickdichten, festen Körper vor, der von leicht verschiedenen Positionen von den Kameras A und B betrachtet wird. Man nimmt ein Pixel A_i und seine acht Nachbarn $\{A_i, A_{i+x} : x \in \mathcal{N}\}$, die alle zu einem bestimmten Bereich des betrachteten Objektes gehören. Wenn dieser Oberflächenbereich ebenso in einer Nachbarschaft von neun Pixeln in Kamera B abgebildet wird, dann wird die Ordnung der Pixel beibehalten. Wenn also $(A_i, B_j), (A_{i+x_1}, B_{j+y_1}), (A_{i+x_2}, B_{j+y_2})$ mit $x_1, x_2, y_1, y_2 \in \mathcal{N}$ korrekte Korrespondenzen sind, dann sind $\{A_i, A_{i+x_1}, A_{i+x_2}\}$ untereinander Nachbarn genauso wie es $\{B_i, B_{i+x_1}, B_{i+x_2}\}$ sind. Solch ein Szenario ist in der später vorgestellten Abbildung 2.5 dargestellt.
Im folgenden wird angenommen, daß jedes Pixel in Bild A einen korrekten Korrespondenzpartner in B hat und daß alle Pixel-Korrespondenzen der oben genannten Ordnungsbedingung genügen. Bei realen Daten ist dies typischerweise aus in Zusammenhang mit Abbildung 2.3 genannten Gründen nicht der Fall. Trotzdem erfüllen normalerweise die Mehrzahl der Pixel diese These. Die anderen repräsentieren fehlerhafte Einschränkungen in diesem Korrespondenzfindungs-Verfahren. Ihre Effekte sollten nach Perwass im Vergleich zu der großen Anzahl der Korrespondenzen, die korrekt gefunden werden können, klein sein. Es wird also erlaubt, daß Pixel, die nicht zugeordnet werden können, falsche Korrespondenpartner bekommen. Dabei wird angenommen, daß dies nur einen kleinen Einfluß auf die Pixel hat, die korrekt zugeordnet werden können. Daraus folgt allerdings, daß man nicht in der Lage ist, korrekte von unkorrekten Korrespondenzen zu unterscheiden.
Perwass legt die Ordnungsbedingung für benachbarte Pixel nun wie folgt fest. Seien

(X_A, X_B) und (Y_A, Y_B) die Zufallsvariablen zu zwei Pixel-Korrespondenzen. Dann ist ihre gemeinsame Wahrscheinlichkeitsverteilung gegeben durch

$$P(X_A = x_A, X_B = x_B, Y_A = y_A, Y_B = y_B) = U_{\mathcal{I}}(x_A)\; U_{\mathcal{N}}(y_A - x_A) \\ U_{\mathcal{T}}(x_A - x_A - d)\; U_{\mathcal{T}}(y_B - y_A - d)\; h(x_A, x_B, y_A, y_B), \qquad (2.17)$$

wobei $h(x_A, x_B, y_A, y_B)$ die zuvor angenommene Ordnungsbedingung realisiert. $U_{\mathcal{I}}(x_A)$ drückt dabei aus, daß es keine Präferenz gibt, welches Pixel aus dem Bild A ausgewählt wird. $U_{\mathcal{N}}(y_A - x_A)$ sagt, daß es keine Präferenz für einen speziellen Nachbarn von x_A gibt. Weiter beschreiben $U_{\mathcal{T}}(x_A - x_A - d)$ und $U_{\mathcal{T}}(y_B - y_A - d)$ die Bedingung, daß man nur nach Pixeln im Suchraum sucht, dort aber auch keine Präferenzen besteht.
Wenn nun also die Einschränkungen $x_B - x_A - d \in \mathcal{T}$, $y_B - y_A - d \in \mathcal{T}$, $y_A - x_A \in \mathcal{N}$ und $x_A \in \mathcal{I}$ gelten, dann folgt aus Gleichung (2.17)

$$P(X_B = x_B, Y_B = y_B \mid X_A = x_A, Y_A = y_A) = \frac{1}{|\mathcal{T}|}\, h(x_A, x_B, y_A, y_B). \qquad (2.18)$$

Dies bedeutet, daß bei gegebenen Pixelpositionen x_A und y_A in Bild A, die Wahrscheinlichkeit, daß x_B und y_B die zugehörigen Positionen der korrespondierenden Pixel in Bild B sind, proportional zu $h(x_A, x_B, y_A, y_B)$ ist.

2.5 Wahrscheinlichkeitsverteilung

Aus den vorigen Bedingungen wurde jeweils ein Wahrscheinlichkeitsmaß für Pixel-Korrespondenzen entwickelt. In Abschnitt 2.2 haben wir uns dazu mit Pixelähnlichkeiten basierend auf deren Farbwerten beschäftigt, und in den darauf folgenden Abschnitten 2.3 und 2.4 haben wir den jeweiligen Ort und seine Nachbarschaft betrachtet. Nun kombiniert Perwass die beiden Wahrscheinlichkeiten, um eine Wahrscheinlichkeitsverteilung für Korrespondenzen basierend auf Ähnlichkeit und Nachbarschaft zusammen zu erhalten. Wie zuvor seien (X_A, X_B) und (Y_A, Y_B) die Zufallsvariablen von zwei benachbarten Pixel-Korrespondenzen mit $(X_A - Y_A) \in \mathcal{N}$. Nach Bayes kann man nun für gegebene Bilder A und B schreiben:

$$P(X_A, Y_A, X_B, Y_B \mid A, B) \simeq P(A, B \mid X_A, Y_A, X_B, Y_B)\;\; P(X_A, Y_A, X_B, Y_B) \qquad (2.19)$$

Für drei allgemeine Zufallsvariablen X, Y, Z kann man zeigen, daß $P(X, Y|Z) = P(X|Y, Z)P(Y)$ ist, genau dann wenn $P(Y, Z) = P(Y)P(Z)$ ist. Da X_A und Y_A allein statistisch unabhängig von den Bildern A, B sind, kann man somit

$$\begin{aligned} & P(X_B, Y_B|A, B, X_A, Y_A) \\ & \simeq \; P(A, B|X_B, Y_B, X_A, Y_A)P(X_B, Y_B|X_A, Y_A) \\ & = \; P(A_{X_A}, B_{X_B}|X_A, X_B)P(A_{Y_A}, B_{Y_B})P(X_B, Y_B|X_A, Y_A) \end{aligned} \qquad (2.20)$$

schreiben. Durch Einsetzen von Gleichung (2.16) und (2.18) erhalten wir

$$\begin{aligned} & P(X_B, Y_B|A, B, X_A, Y_A) \\ & \simeq \; s(A_{X_A}, B_{X_B})\; s(A_{Y_A}, B_{Y_B})\; h(X_B, Y_B, X_A, Y_A). \end{aligned} \qquad (2.21)$$

Die Funktion h nach Gleichung (2.14) definiert die anfängliche Verteilung der korrekten Korrespondenzen von benachbarten Paaren, die angenommen wird. Die Werte der Funktion s ergeben sich aus den aufgenommenen Bilddaten nach Gleichung (2.11).
Man will herausfinden, welches Paar benachbarter Korrespondenzen die angenommene Verteilung am besten erfüllt. Dazu betrachtet Perwass das folgende Verhältnis:

$$\hat{P}(X_B, |A, B, X_A, Y_A) := \rho \, \frac{\max_y P(X_B, Y_B = y|A, B, X_A, Y_A)}{\max_y P(X_B, Y_B = y|X_A, Y_A)} \tag{2.22}$$

Stellen wir uns eine korrekte Korrespondenz $(X_A = x_A, X_B = x_B)$ und zusätzlich ein Pixel $Y_A = y_A$ in der Nachbarschaft von $X_A = x_A$ vor. Der Zähler der Gleichung 2.22 wird genau dann sein Maximum an derselben Stelle y annehmen wie der Nenner, wenn die Bilddaten die angenommene Verteilung der Pixel-Korrespondenzen bestätigen. Also gibt der Quotient in Gleichung (2.22) ein Maß an, wie gut die Bilddaten der angenommenen Verteilung entsprechen, wobei ρ einen Normalisierungsfaktor darstellt.
Gleichung (2.22) könnte für alle Nachbarn von X_A berechnet werden. Man will aber nicht, daß dieses Maß von einem bestimmten Nachbarn Y_A abhängt, obwohl es für jedes Y_A einen anderes Ergebnis der Gleichung gibt. Die endgültige Wahrscheinlichkeit, daß ein einzelner Wert von X_B ein korrekter Korrespondenzpartner für ein X_A ist, nimmt Perwass daher als Erwartungswert der Gleichung (2.22) über alle Y_A bei gegebenem X_A an:

$$\hat{P}(X_B|A, B, X_A) := \rho \, E_{Y_A}[\hat{P}(X_B|A, B, X_A, Y_A) \mid X_A], \tag{2.23}$$

Dabei ist

$$\begin{aligned} & E_{Y_A}[\hat{P}(X_B|A, B, X_A, Y_A) \mid X_A = x] \\ &= \sum_{y:(y-x)\in\mathcal{N}} \hat{P}(X_B|A, B, X_A = x, Y_A = y) \; P(Y_A = y|X_A = x) \\ &= \frac{1}{|\mathcal{N}|} \sum_{y:(y-x)\in\mathcal{N}} \hat{P}(X_B|A, B, X_A = x, Y_A = y) \end{aligned} \tag{2.24}$$

mit $P(Y_A = y|X_A = x) = U_{\mathcal{N}}(x - y) = 1/|\mathcal{N}|$ für $(x - y) \in \mathcal{N}$. Durch Verwendung der Funktionen s und h gemäß Gleichung (2.21) kann man nun auch

$$\begin{aligned} & \hat{P}(X_B = x_B|A, B, X_A = x_A) \\ &\simeq \; s(A_{x_A}, B_{x_B}) \, \frac{1}{|\mathcal{N}|} \sum_{y_A:(y_A-x_A)\in\mathcal{N}} \max_{y_B} s(A_{y_A}, B_{y_B}) \, \hat{h}(x_A, x_B, y_A, y_B) \end{aligned} \tag{2.25}$$

scheiben, wobei $\hat{h}$ definiert ist als:

$$\hat{h}(x_A, x_B, y_A, y_B) := \frac{h(x_A, x_B, y_A, y_B)}{\max_y h(x_A, x_B, y_A, y)}. \tag{2.26}$$

Gleichung (2.25) ist in Verbindung mit Abbildung 2.2 zu sehen. Es wird die Wahrscheinlichkeit berechnet, daß ein Pixelpaar (x_A, x_B) eine korrekte Korrespondenz ist. Jedes Paar

von Pixeln aus Bild A und B entspricht einem Feld im Suchraum. Jedes Feld des Suchraumes enthält den zugehörigen Ähnlichkeitswert der beiden Pixel nach Gleichung (2.11). Die Vorgehensweise des Korrespondenzfindungs-Verfahrens von Perwass [PS02] ist nun folgende. Um den Wahrscheinlichkeitswert für die Korrespondenz eines Paares zu berechnen, werden in Gleichung (2.25) zuerst die Korrespondenzwahrscheinlichkeiten für die am besten passenden Pixel zu den acht Nachbarn in den jeweiligen Suchräumen ermittelt. Dann wird der Erwartungswert dieser Wahrscheinlichkeiten mit dem Wert der Pixelähnlichkeit des Paares multipliziert. Somit wird also eine Korrespondenz als sehr wahrscheinlich angesehen, wenn die Pixelähnlichkeit groß ist und wenn zusammen mit benachbarten Korrespondenzen die lokale Ordnungsbedingung erfüllt ist.
Die aufgestellten Bedingungen sollen nun noch einmal zusammengefaßt werden und in direkten Bezug zu den später verwendeten Gleichungen gesetzt werden. Zur Erinnerung ist die Farbähnlichkeitsfunktion s in Gleichung (2.11) definiert. Sie ist abhängig von den Farbwerten zweier Pixel. Die Farbdifferenzen bilden den Parameter einer Gaußfunktion. Diese Gaußkurve hat ihr Maximum bei Null, was also bei exakt gleichen Farbwerten der Fall wäre. Durch den Normalisierungsfaktor wird der Wertebereich in das Intervall $[0, 1]$ gelegt. Nun ist der Farbähnlichkeitswert zweier Pixel als Wahrscheinlichkeitswert aufzufassen, der ebenfalls im Intervall $[0, 1]$ liegt. Es handelt sich hierbei um die Wahrscheinlichkeit, daß die beiden Pixel vom gleichen Merkmal der Szene stammen und somit möglicherweise miteinander korrespondieren. Zu beachten ist, daß sich diese Wahrscheinlichkeit zunächst nur auf die Ähnlichkeit der Farbwerte bezieht. Die Position im Bild ist bis hierhin noch nicht betrachtet. Folglich kommen bei dieser Betrachtung durchaus mehrere Pixel als Korrespondenzpartner zu einem ausgewählten Ausgangspixel A_{x_A} in Betracht, nämlich alle diejenigen, deren Farbwerte gleich oder sehr ähnlich denen von A_{x_A} sind. Wie groß die Ähnlichkeit sein muß, bzw. wie sehr die Differenzen Auswirkung auf die zugehörige Wahrscheinlichkeit haben, ist durch die Glockenform der Gaußkurve bestimmt. Der zusätzliche Parameter σ erlaubt eine Einstellung des Bereichs, in dem kleine Differenzen noch als annähernd gleich angesehen werden, bzw. größere Differenzen bereits eine erhebliche Verminderung der Korrespondenzwahrscheinlichkeit zur Folge haben. Diese Betrachtungsart soll den linken (ersten) Teil der Berechnung der erweiterten Korrespondenzwahrscheinlichkeit nach Gleichung (2.25) verdeutlichen.
Im rechten Teil des Produktes wird nun die Ordnungsbedingung betrachtet. Die hier verwendete Funktion $\hat{h}$ aus Gleichung (2.26) basiert auf der vorher eingeführten Gleichung (2.14) und stellt eine normalisierte Form hiervon dar. Die Normalisierung wird in Bezug auf den maximal möglichen Wert durchgeführt, was zu einem Wertebereich von Null bis Eins führt und somit wieder als Wahrscheinlichkeitswert aufgefaßt werden kann. Diese Funktion soll ein Maß für die lokale Ordnung ausdrücken. Betrachten wir die Definition in Gleichung (2.14), die zu den Pixelpositionen von zwei Korrespondenz-Paaren einen Ordnungswert liefert. Hier wird wieder die Gauß-Funktion verwendet, wobei als Laufvariable die Differenz der Abstandsvektoren eingesetzt wird. Dazu wird der Abstandsvektor $(y_A - x_A)$ der Positionen zweier Pixel in Bild A verglichen mit dem Abstandsvektor $(y_B - x_B)$ der beiden zugehörigen Korrespondenzpartner in Bild B. Eine Veranschaulichung eines Beispiel-Szenarios ist in Abbildung 2.4 dargestellt. Die Differenz beider Abstandsvektoren ist bei erfüllter Ordnungsbedingung möglichst gering. Dies würde bedeuten, daß zum Beispiel der rechte Nachbar y_A eines Pixels x_A seinen Korre-

spondenzpartner y_B auch als rechten Nachbarn von x_B, dem Korrespondenzpartner zu x_A, hat.
Im rechten Teil der Gleichung (2.25) wird diese Ordnungsbedingung angewendet, um einen Wahrscheinlichkeitswert zu erhalten, der im Produkt mit dem Wahrscheinlichkeitswert basierend auf der Farbähnlichkeit die Verbundwahrscheinlichkeit für die gesamte Korrespondenz liefert.
Es werden nun alle Nachbarn des gewählten Pixels an der Position x_A betrachtet. Genauer werden zu allen acht Nachbarn $y_A : (y_A - x_A) \in \mathcal{N}$ die am besten passenden Pixel in Bild B gesucht. Für jeden Nachbarn y_A werden alle möglichen Korrespondenzpartner y_B aus dem entsprechenden Suchraum untersucht. Zum einen muß nun wie oben die Farbähnlichkeit von A_{y_A} und B_{y_B} groß sein und zum anderen müssen die Positionen der nun angenommenen beiden Pixelpaare einen möglichst großen Wert in der Ordnungsbedingung, ausgedrückt durch die Funktion $\hat{h}$, ergeben. Hier soll nur dasjenige Pixel y_B ausgewählt werden, welches das Produkt aus den Funktionen s und $\hat{h}$ maximiert. Das Produkt hat hierbei die Bedeutung, daß beide Bedingungen zusammengenommen maximal erfüllt sein sollen. Beide Funktionen liefern hierzu einen Wahrscheinlichkeitswert im Intervall $[0, 1]$, deren Verbundwahrscheinlichkeit durch die Multiplikation wieder im selben Interwall liegen wird.
Anschließend wird der Erwartungswert dieser Verbundwahrscheinlichkeiten der besten Nachbarkorrespondenzen nach Gleichung (2.24) durch Summation und Teilen durch deren Anzahl gewonnen. Dieser Erwartungswert symbolisiert nun die Wahrscheinlichkeit, daß die acht Nachbarn von x_A die Korrespondenz des gewählten Pixel-Paares (x_A, x_B) unterstützen. Durch Multiplikation dieses Wertes mit dem Ähnlichkeitswert des gewählten Paares erhalten wir letztendlich die Verbundwahrscheinlichkeit von Farbähnlichkeit und Erfüllen der Ordnungsbedingung für ein Korrespondenz-Pixel x_B in Bild B zu einem gewählten Pixel x_A aus Bild A, bei gegebenen Bildern.

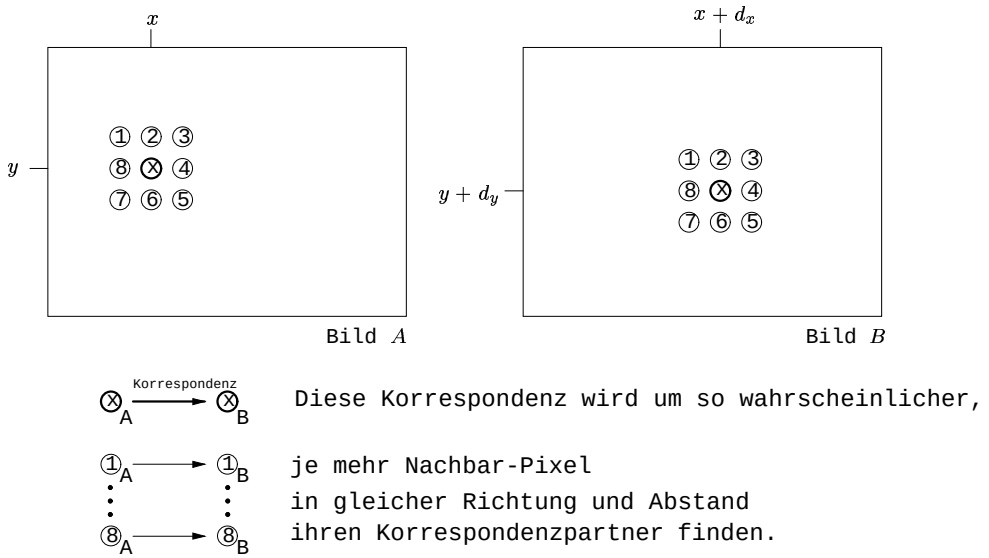


Abbildung 2.5: Nachbarkorrespondenzen unterstützen angenommene Korrespondenz von x_A zu x_B.

Abbildung 2.5 zeigt ein Szenario, in dem alle acht Nachbarn von x_A ihren Korrespondenzpartner unter Beibehaltung der Ordnung, also in gleicher Nachbarschaft zu x_B finden. In diesem Fall wird die Wahrscheinlichkeit der angenommenen Korrespondenz $\hat{P}(X_B = x_B|X_A = x_A)$ vergrößert. Das Finden anderer Korrespondenzpartner insbesondere zu den Nachbarn von x_A wird in dieser Abbildung nicht dargestellt, sondern wird basierend auf den Farbähnlichkeiten als bekannt angenommen. Das hier dargestellte Szenario stellt einen Idealfall dar. In der Realität wird es immer Ausreißer geben, die einige Bedingungen nicht oder nur teilweise unterstützen. Da wir nicht mit harten Entscheidungen, sondern mit Wahrscheinlichkeitswerten arbeiten, können wir dieses vernachlässigen.

2.6 Verbreitung von lokalen Bedingungen

Im vorigen Abschnitt haben wir die Herleitung von einem Maß für die Wahrscheinlichkeit, daß ein Pixel-Paar ein korrektes Korrespondenzpaar ist, gesehen. Dieses Maß ist allein aus lokalen Bedingungen hervorgegangen. Um nun Korrespondenzen in Bildern zu finden, reicht dieser erste Schritt aber oft nicht aus. Daher benutzt Perwass ein iteratives Verfahren, um lokale Bedingungen in einem Bild zu verteilen. In diesem Abschnitt wird nun diese Iteration vorgestellt.

2.6.1 Der iterative Prozeß

Zunächst ist für ein Pixelpaar A_{X_A}, B_{X_B} eine Korrespondenzwahrscheinlichkeit aufgrund der Farbähnlichkeit und der angenommenen Verteilung von Korrespondenzpaaren in der Nachbarschaft durch Gleichung (2.25) gegeben. Dabei ist die angenommene Verteilung von Korrespondenzen eine wichtige Bedingung. Die betrachtete Farbähnlichkeit führt in realen Bildern allerdings zu Problemen. Häufig gibt es innerhalb des Suchraumes sehr viele Pixel, zu denen die Farbähnlichkeit groß ist, so daß dann diese Bedingung nicht sehr aussagekräftig sein kann. Dazu kommt, daß besonders in großen homogenen Bereichen der Bilder auch noch die Ordnungsbedingung an Bedeutung verliert, da sich hier viele Pixel gleichermaßen anbieten. Insbesondere bieten sich dann für alle Nachbarpixel, die eine Korrespondenz bestätigen sollen, so viele Korrespondenzpartner an, daß sie auch jede andere Korrespondenz bestätigen könnten und somit hieraus keine Information gewonnen werden kann. Da Gleichung (2.25) nur jeweils die acht direkten Nachbarn betrachtet, wird das Ergebnis nur korrekt sein, wenn die Korrespondenzwahrscheinlichkeit allein durch diese acht Nachbarn bedingt ist. In der Regel ist dies aber nicht der Fall. Es muß eine größere Nachbarschaft für jedes Pixel in Betracht gezogen werden. Nun aber die Ordnungsbedingung für eine größere Nachbarschaft zu definieren, wäre nach Perwass keine gute Lösung, da dies eine sehr starre Annahme der Szene bedeuten würde. Angenommen, wir haben durch Berechnung von $\hat{P}(X_B|A, B, X_A = x_A)$ für jedes Pixel $X_B = x_B$ innerhalb unseres Suchraumes die Wahrscheinlichkeit, zu x_A zu korrespondieren. Dann enthält diese Wahrscheinlichkeitsverteilung alle Beschränkungen, hervorgehend aus der direkten Nachbarschaft von x_A. Wenn wir nun darauf wiederholt die Gleichung (2.25) anwenden,

indem wir die zuvor erhaltene Wahrscheinlichkeitsverteilung anstatt der vorher als Grundinformation angenommenen Werte der s-Funktion verwenden, dann berücksichtigen wir mit derselben Gleichung implizit eine weitere Stufe der Nachbarschaft. Da der Wahrscheinlichkeitswert der benachbarten Pixel jeweils zuvor von seinen Nachbarn bedingt wurde, trägt er dessen Information zusammen mit seinen eigenen nun wiederum an seine Nachbarn weiter. Daher werden so nach dem wiederholten Anwenden der Gleichung Bedingungen der nun um einen Schritt vergrößerten Nachbarschaft in Betracht gezogen. Dies bedeutet aber nicht, daß nun diese größere Nachbarschaft im Ganzen die Ordnungsbedingung erfüllen muß. Lediglich die s-Funktion in Gleichung (2.25), welche anfangs eine initiale Korrespondenzwahrscheinlichkeit aufgrund der Farbähnlichkeit ausdrückte, wird bei der folgenden Iteration durch die vorigen Ergebnisse ersetzt. Das hat den Effekt, daß nun nicht mehr die Verteilung der Farbähnlichkeit, sondern die Verteilung der nach Gleichung (2.25) gewonnenen Korrespondenzen erneut mit der gleichen Ordnungsbedingung $\hat{h}$ ausgewertet wird, um zu einer neuen Verteilung von Korrespondenzwahrscheinlichkeiten zu gelangen. Jede Korrespondenz wird nun also nicht mehr von allein acht Nachbarn, sondern auch zusätzlich von deren Nachbarn bedingt, da diese im vorigen Schritt betrachtet wurden.
Dieser Zusammenhang wird von Perwass mathematisch beschrieben. Dazu nennt er $f^t(x, y)$ allgemein die Korrespondenzwahrscheinlichkeit von zwei Pixeln nach dem Iterationsschritt t. Für ein gegebenes x wird die Funktion $f^t(x, y)$ für alle y im Suchraum zu x ausgewertet. Dabei sind die initialen Werte von f^t für $t = 0$ gegeben durch die Farbähnlichkeiten der Pixel A_x und B_y, also $f^0(x, y) = s(A_x, B_y)$. Sei $\mathcal{F}^t$ die Menge der Werte von $f^t(x, y)$, definiert als

$$\mathcal{F}^t := \{f^t(x, y) \ : \ x \in \mathcal{I}, \ (y - x - d) \in \mathcal{T}\} \tag{2.27}$$

Dann kann man Gleichung (2.25) nun erweitern zu

$$\begin{aligned} &\hat{P}(X_B = x_B|\mathcal{F}^t, X_A = x_A) \\ &\simeq \ f^t(x_A, x_B) \ \frac{1}{|\mathcal{N}|} \sum_{y_A:(y_A - x_A)\in\mathcal{N}} max_{y_B} \ f^t(y_A, y_B) \ \hat{h}(x_A, x_B, y_A, y_B). \end{aligned} \tag{2.28}$$

Für $t = 0$ entspricht dies genau Gleichung (2.25). Allgemein erhält man für jeden folgenden Iterationsschritt

$$f^{t+1}(x_A, y_A) = \hat{P}(X_B = x_B|\mathcal{F}^t, X_A = x_A). \tag{2.29}$$

Dies definiert Perwass als eine *inhomogene* Markov-Kette, da der Übergang von f^t zu f^{t+1} von f^t und dies von t abhängt. Nun kann man Gleichung (2.29) auch als rückgekoppeltes neuronales Netz interpretieren. Diese Betrachtungsweise soll nun im folgenden diskutiert werden.

2.6.2 Neuronaler Ansatz

Künstliche neuronale Netze sind in Aufbau und Funktion dem menschlichen Gehirn nachempfunden. Sie bestehen aus Neuronen, den Knoten des Netzes und aus Verbindungen,

die Datenfluß zwischen den Neuronen darstellen. Die Basis dieser Betrachtungsweise wurde 1943 von McCulloch und Pitts [MP43] entwickelt. Jedes Neuron hat eine Funktion, die zu einer Menge von Eingangsdaten einen Ausgangswert liefert. Abbildung 2.6 zeigt eine schematische Darstellung eines Neurons mit zugehöriger interner Funktion. Hier wird zunächst als Propagierungsfunktion die gewichtete Summe aus den Eingängen $x_1, \cdots, x_n$ mit den zugehörigen Gewichten $g_1, \cdots, g_n$ gebildet und darauf die Aktivierungsfunktion f angewendet, deren Funktionswert den Ausgang bildet. Dabei wird als Aktivierungsunktion häufig eine Stufen- bzw. Sigmoid-Funktion verwendet, so daß das Ergebnis eine Klassifikationsaussage bildet. Verbindet man eine Menge von Neuronen untereinander, so ergibt sich daraus ein neuronales Netz. Mehrere Neuronen, die parallel arbeiten und die gleiche Funktion für ihre jeweiligen Eingänge erfüllen, faßt man zu einer Schicht zusammen. Somit kann man das Netzwerk in verschiedene Schichten einteilen, wobei die Neuronen einer Schicht jeweils ihre Eingangsdaten von der vorigen Schicht erhalten und ihre Ausgangsdaten an die nachfolgende Schicht weitergeben.

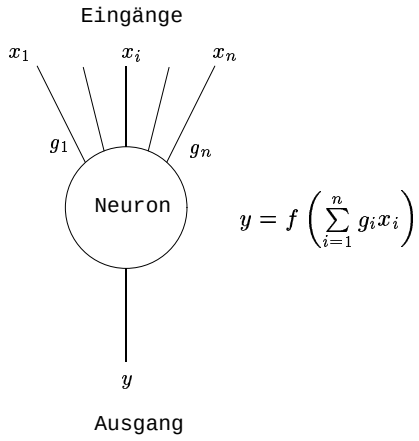


Abbildung 2.6: Schematische Darstellung eines Neurons

In Abbildung 2.7 ist links ein Netz aus Neuronen dargestellt, welches aus zwei jeweils voll verbundenen Schichten besteht. Die erste Schicht enthält die Neuronen N_i^1 mit $i \in \{1..m\}$ mit jeweils n Eingängen, welche direkt mit den Eingangsdaten I_1 bis I_n verbunden sind. Die nachfolgende Schicht besteht aus den Neuronen N_j^2 mit $j \in \{1..m\}$ mit jeweils m Eingängen. Hier sind die Eingänge jeweils mit den Ausgängen der Neuronen der vorigen Schicht verbunden.
Wenn im Datenfluß durch die Schichten ein Zyklus besteht, handelt es sich um ein rückgekoppeltes neuronales Netz[2]. Ein Beispiel für diesen Fall ist in Abbildung 2.7 rechts dargestellt. Hier haben die Neuronen N_1^1 und N_2^1 einen Eingang, der vom Ausgang des Neurons der folgenden Schicht abhängt. Somit fließen hier Ergebnisse aus vorangegangenen Berechnungen in die aktuelle Berechnung mit ein. Dieser Zyklus im Datenfluß stellt eine Rückkoppelung dar.
In Bezug auf Gleichung (2.28) und (2.29) stellen wir uns nun vor, daß alle einzelnen Berechnungsschritte durch Neuronen realisiert werden können. Die dazu verwendeten

[2] *vgl. engl. recurrent neural network*

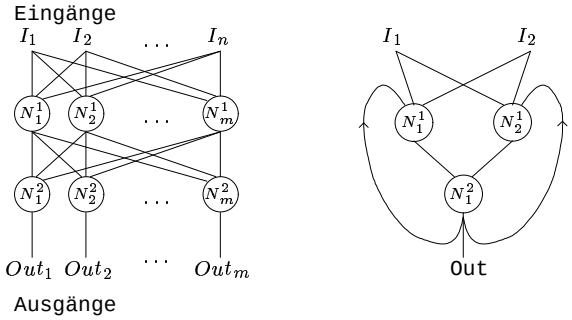


Abbildung 2.7: Netze aus Neuronen

Neuronen sollen Funktionen wie Gewichtung durch Multiplikation, gewichtete Addition und die Auswahl des Maximums leisten. Abbildung 2.8 stellt die einzelnen Berechnungschritte der Gleichung (2.28) graphisch dar. Zunächst werden die Eingangsdaten aus der

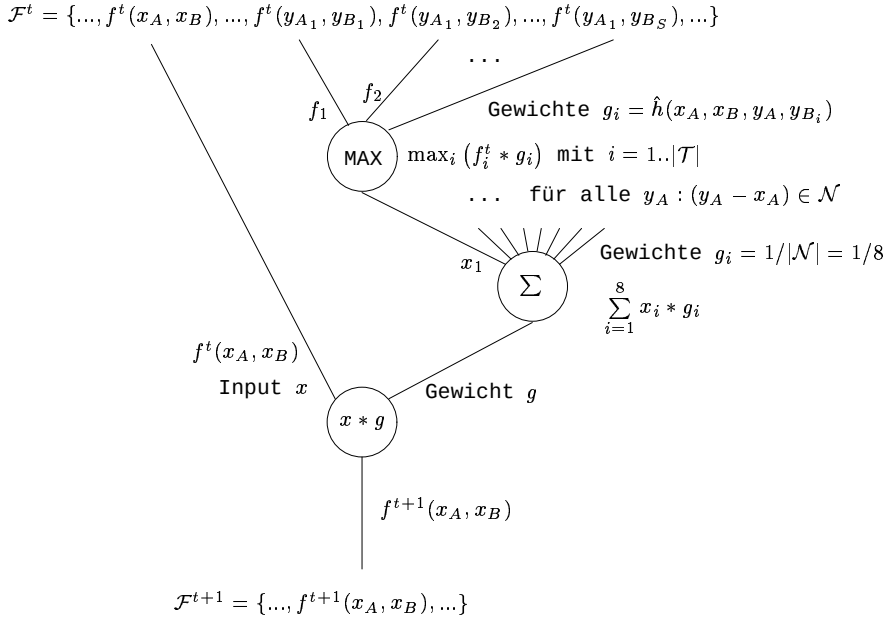


Abbildung 2.8: Gleichung (2.28) dargestellt in Baumstruktur

gegebenen Wahrscheinlichkeitsverteilung mit den der Pixelposition entsprechenden Gewichten gemäß der Funktion $\hat{h}$ multipliziert. Dann wird daraus das Maximum ausgewählt. Diese beiden Schritte wollen wir im folgenden zusammenfassen und als ein mehrschichtiges Neuron betrachten. Wie in der Abbildung 2.8 angedeutet, sind entsprechend der gewählten Nachbarschaft mehrere dieser Neuronen parallel angeordnet. Sie führen alle die gleiche Funkton aus und unterscheiden sich lediglich in den verwendeten Eingangsdaten aus der gegebenen Wahrscheinlichkeitsverteilung und den verwendeten Gewichten. Die Gewichte resultieren aus den unterschiedlichen Positionen der betrachteten Pixel unter Anwendung der Funktion $\hat{h}$. Die darauf folgende gewichtete Addition der parallel ge-

wonnenen Werte kann durch ein einfaches Neuron wie in Abbildung 2.6 realisiert werden. Hierbei sind alle Gewichte $g_i = 1/|\mathcal{N}|$, sodaß sich als Ergebnis direkt das arithmetische Mittel der Eingangswerte ergibt. Im nächsten Schritt wird eine einfache Multiplikation durchgeführt, die somit den gegebenen Wahrscheinlichkeitswert $f^t(x_A, x_B)$ mit dem Erwartungswert resultierend aus der Betrachtung der Nachbarkorrespondenzen gewichtet. Dies ergibt den neuen Wahrscheinlichkeitswert $f^{t+1}(x_A, x_B)$ für die Korrespondenz des gewählten Pixelpaares.
So können alle Berechnungschritte für die Korrespondenzwahrscheinlichkeit eines gegebenen Pixelpaares gemäß Gleichung (2.28) durch Neuronen realisiert werden. Diese können zusammengefaßt werden zu einem mehrschichtigen Netz. Dieses gesamte Netz möchten wir nun wiederum als interne Funktion eines Neurons höherer Ordnung[3] betrachten. Da die Funktionswerte der $\hat{h}$-Funktion in Gleichung (2.28) für alle zu betrachtenden Pixelpaare lediglich von deren Positionen bzw. relativen Positionsunterschieden zueinander abhängen, somit unabhängig von den anderen Berechnungen sind und für alle Berechnungen konstant bleiben, können diese den entsprechenden Neuronen als konstante Gewichte $g_i = \hat{h}(x_A, x_B, y_A, y_{B_i})$ zugeordnet werden.
Die Berechnung der Wahrscheinlichkeitsverteilung für alle möglichen Korrespondenzpaare kann durch ein Netz mit einer Schicht eben dieser Neuronen realisiert werden. Wobei jedes Neuron dieser Schicht dann die Berechnung der Korrespondenzwahrscheinlichkeit eines Pixelpaares durchführt. Alle diese Neuronen können parallel und unabhängig voneinander arbeiten. Wenn nun eine Schicht eines neuronalen Netzwerks aus einer ausreichenden Anzahl dieser Neuronen besteht, deren Eingänge mit den entsprechenden Werten belegt sind, dann ergeben ihre Ausgänge die neue Wahrscheinlichkeitsverteilung nach Gleichung (2.29). Jedes Neuron einer Schicht hat Eingangsdaten aus der Wahrscheinlichkeitsverteilung $\mathcal{F}^t$ der vorigen Schicht und liefert daraus als Ausgang einen Wert einer neuen Wahrscheinlichkeitsverteilung $\mathcal{F}^{(t+1)}$. Zusammen liefern alle Neuronen dieser Schicht die gesamte neue Wahrscheinlichkeitsverteilung. Die Anzahl der dafür benötigten Neuronen ergibt sich aus der Anzahl der Pixel in den Bildern und der zugehörigen Suchraumgröße. Jede Schicht entspricht also in Bezug auf Gleichung (2.29) einem Iterationsschritt vom Index t zu $t + 1$. Die Ausgangsdaten jeder Schicht entsprechen dann der Wahrscheinlichkeitsverteilung der Korrespondenzen nach dem jeweiligen Iterationsschritt. Nun könnte man sich ein Netz aus mehreren dieser Schichten vorstellen, welches entsprechend viele Iterationsschritte berechnet. Dabei sind alle Schichten von ihrer Funktion her gleich. Da jede Schicht ihre Eingangsdaten ausschließlich aus der vorhergehenden Schicht erhält und an die nachfolgende Schicht weitergibt und alle Schichten gleiche Funktionalität haben, kann man hier auch ein rekurrentes Netz verwenden. Dieses Netz würde aus nur einer einzelnen Schicht der Neuronen bestehen, welche vom Datenfluß wiederholt durchlaufen wird. Nach einmaliger Initialisierung mit der Korrespondenzwahrscheinlichkeits-Verteilung $\mathcal{F}^0$ nach Gleichung (2.11) wird nach dem ersten Durchlauf die Verteilung $\mathcal{F}^1$ gewonnen. Alle folgenden Durchläufe der Neuronenschicht basieren dann auf den zwischengespeicherten Ergebnissen des vorigen Durchlaufes. Somit entspricht die Anzahl der Durchläufe dieser Schicht den Iterationschritten t nach Gleichung (2.29).

[3] *engl. higher order neuron*

Zur Veranschaulichung des Rechenvorgangs und des zugehörigen Netzwerks betrachten wir nun ein Beispiel mit einem sehr begrenzten Suchraum und begrenzten Nachbarschaften. Wir wählen eine vereinfachte lineare Problemstellung, deren Berechnung ein Mindestmaß an Komplexität erfordert, wobei aber die Struktur des Rechenvorgangs trotzdem dem allgemeinen Fall entspricht. Wir beschränken uns auf einen Suchraum der Größe 3×1 und eine Nachbarschaft, die nur aus linkem und rechtem Nachbarn besteht. Somit sind nur lineare Verschiebungen in den Korrespondenzen möglich, die auch nur durch die direkten Nachbarn derselben Ausrichtung bedingt werden. Zum Vergleich haben wir bisher einen beliebig großen Suchraum und eine $8er$ Nachbarschaft angenommen. Zur Berechnung des Erwartungswertes der Nachbarn werden also zunächst die einzelnen Werte der Wahrscheinlichkeitsverteilung in deren Suchräumen mit den entsprechenden Funktionswerten der $\hat{h}$-Funktion gewichtet. Zur Bestimmung der Gewichte betrachten wir die Gleichung (2.26). Diese wurde aus Gleichung (2.14) durch Normierung gewonnen, welche wiederum einem Gauß entspricht, der durch die Differenzen von Pixelpositionen parametrisiert ist. Dazu werden die Pixelpositionen der zu betrachtenden Korrespondenzpartner und alle Positionen aus dem Suchraum der Nachbarpixel in Gleichung (2.26) eingesetzt. Dies ergibt eine Gaußverteilung der Gewichte, die entsprechend der Positionen der betrachteten Korrespondenzpartner verschoben ist. In unserem einfachen Fall mit dem Suchraum, der nur aus 3 Positionen besteht, ergeben sich also 3×3 Gewichtswerte. Jeweils 3 Funktionswerte der entsprechend verschobenen Gaußverteilung bilden die Gewichte g_1, g_2, g_3 für die 3 Einträge x_1, x_2, x_3 der Suchräume.

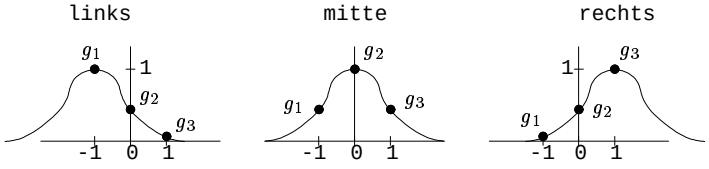


Abbildung 2.9: Gewichte gemäß Funktion $\hat{h}$ für linearen Suchraum der Größe (3×1)

Abbildung 2.9 zeigt die verschobenen Gaußverteilungen und die daraus resultierenden Gewichte für die drei zu untersuchenden Suchraumeinträge eines Pixels an Position x_A. Die Überschriften `links`, `mitte` und `rechts` unterscheiden dabei die drei möglichen Korrespondenzpartner im Suchraum des Ausgangspixels. Die Gewichte g_1, g_2, g_3 unter `links` sind also zur Berechnung des Erwartungswertes der Nachbarn bei angenommener Verschiebung des Ausgangspixels nach *links* bestimmt. Der daraus gebildete Erwartungswert wird wiederum zur Gewichtung des ursprünglichen Wertes im entsprechend *linken* Suchraumeintrag des Ausgangspixels verwendet. Für andere angenommene Verschiebungen, die den anderen beiden Suchraumeinträgen des Ausgangspixels entsprechen, werden die Erwartungswerte der Nachbarn durch die Gewichtung der Nachbarsuchräume mit den entsprechenden Werten der verschobenen Gaußkurven `mitte` und `rechts` gebildet. Abbildung 2.10 zeigt die schrittweise Berechnung des linken Suchraumeintrags x_1 für ein Pixel an der Position $x_A = (x, y)$ mit einzelnen Neuronen. Die Berechnung der anderen beiden Suchraumeinträge findet parallel dazu und mit den gleichen Neuronen statt, wobei diese in der Darstellung zu übergeordneten Neuronen zusammengefaßt sind. Einziger Unterschied ist, daß jeweils die der Verschiebung entsprechenden Gewichte eingesetzt

werden. Das gesamte in Abbildung 2.10 dargestellte Netz ermöglicht also eine parallele Berechnung aller Suchraumeinträge und damit die gesamte Wahrscheinlichkeitsverteilung der möglichen Korrespondenzpartner für das Pixel an Position $x_A = (x, y)$. Dabei ist für jeden Suchraumeintrag eines der zusammengefaßten Neuronen nötig, die alle unabhängig voneinander arbeiten und jeweils einen Wahrscheinlichkeitswert liefern. Somit wird ausgehend von einem Pixel an einer Position x_A für jeden möglichen Kandidaten der Wahrscheinlichkeitswert für eine Korrespondenz berechnet. Dabei wird jeweils der gesamte Suchraum der beiden Nachbarn $y_{A_1} = (x-1, y)$ und $y_{A_2} = (x+1, y)$ entsprechend der Position des Kandidaten $\{x_1, x_2, x_3\}$ gewichtet und fließt in die folgende Berechnung ein. Das in Abbildung 2.10 dargestellte Berechnungsnetz läßt sich nun auch für alle anderen Pixel anwenden, indem eine entsprechende Anzahl dieser Neuronen mit den jeweiligen Eingangsdaten der entsprechenden Pixel und deren Nachbarn verbunden werden. Da jedes Neuron unabhängig von allen anderen arbeitet, können alle diese Neuronen parallel angeordnet werden, sodaß sich daraus ein Netzwerk ergibt, welches aus einer Schicht gleicher Neuronen besteht. Somit lassen sich dann nicht nur alle Einträge des Suchraumes eines Ausgangspixels, sondern die gesamte neue Wahrscheinlichkeitsverteilung für die Korrespondenzen aller Pixel durch einen Durchlauf dieser Schicht berechnen. Mehrere dieser Schichten im Netz, oder ein rekurrentes Netz, in dem diese Schicht wiederholt durchlaufen wird, ermöglicht dann also das Iterieren der Gleichung (2.29).

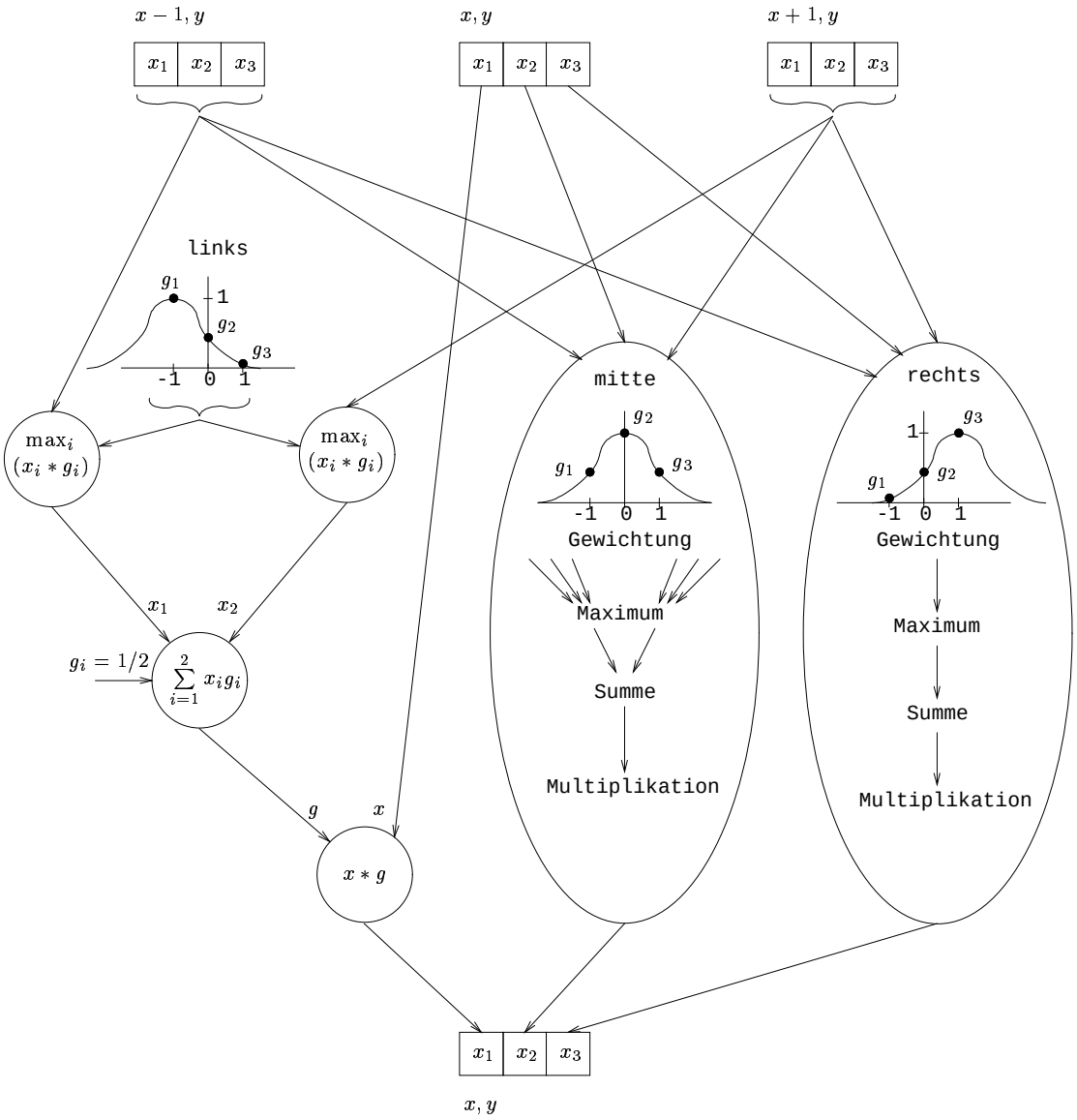


Abbildung 2.10: Berechnung der Suchraumeinträge durch Neuronen: *links* detailliert, *mitte* und *rechts* zusammengefaßt zu übergeordnetem Neuron

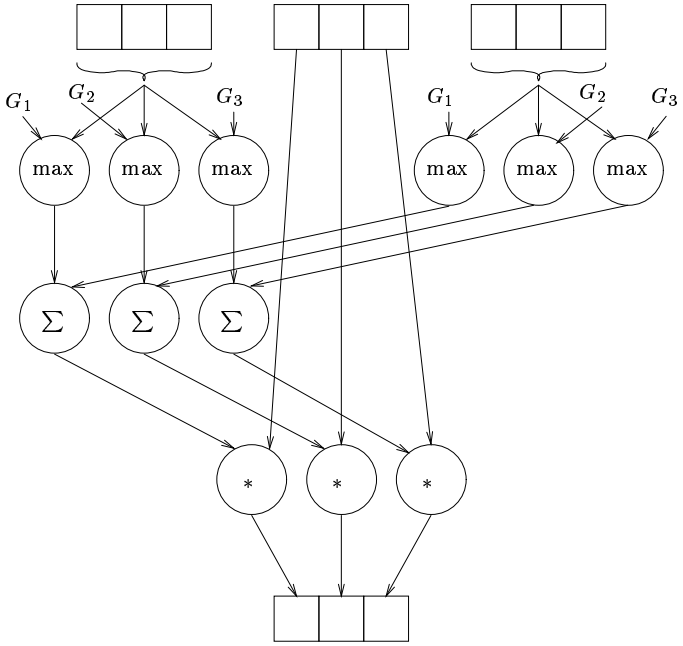


Abbildung 2.11: lineares Netz

Kapitel 3

Realisierung

Dieses Kapitel beschäftigt sich mit der Realisierung des im vorigen Kapitel vorgestellten Verfahrens zur dichten Korrespondenzfindung von Perwass und Sommer [PS02]. Dabei konzentrieren wir uns zunächst auf die Realisierung der einzelnen Rechenschritte, die zu einem Rechenwerk[1] zusammengefaßt werden. Dann gehen wir näher auf die Datenorganisation, den Datenfluß und die nötige Steuerung ein, welche durch ein Steuerwerk[2] realisiert wird. Es wird gezeigt, wie ein System entwickelt werden kann, welches als Koprozessor den iterativen Teil des Algorithmus selbständig ausführt. Die Architektur des Prozessors entspricht dem SIMD-Prinzip, welches aus einem Steuerwerk und mehreren parallel angeordneten Rechenwerken besteht. Abbildung 3.1 zeigt den schematischen Aufbau des Prozessors, dessen einzelne Elemente in den folgenden Anschnitten erläutert werden. Voran geht ein Abschnitt über die Vorverarbeitung und abschließend folgt ein Abschnitt über eine mögliche Nachverarbeitung.

3.1 Vorverarbeitung der Bilder

Der erste Schritt in der Berechnung der Wahrscheinlichkeitsverteilung von Korrespondenzen ist die Betrachtung der Farbähnlichkeiten von Pixelpaaren. Diese führt zu einer initialen Annahme von Korrespondenzen, die allein durch die Farbähnlichkeit bedingt ist. In Abschnitt 2.2 ist die Herleitung dieser Bedingung vorgestellt und erläutert. Sie begründet sich auf der Annahme, daß zwei Bildpunkte, die dasselbe Merkmal der Szene abbilden, dieselbe Farbe aufweisen. Durch zusätzliches Rauschen bei der Bildaufnahme können die Farbwerte der Pixel allerdings vom wahren Farbwert des Merkmals der Szene abweichen, so daß die Wahrscheinlichkeit für die Korrespondenz zweier Pixel durch die Differenz ihrer Farbwerte bedingt ist. Dies ist durch Gleichung (2.11) beschrieben, welche zur Erinnerung zusammengefaßt als

$$P(A_i = a, B_j = b) := \prod_{v \in \mathcal{M}} exp\left(-\frac{(a^v - b^v)^2}{2\sigma^2}\right) \tag{3.1}$$

[1] *engl. Operational Unit*
[2] *engl. Control Unit*

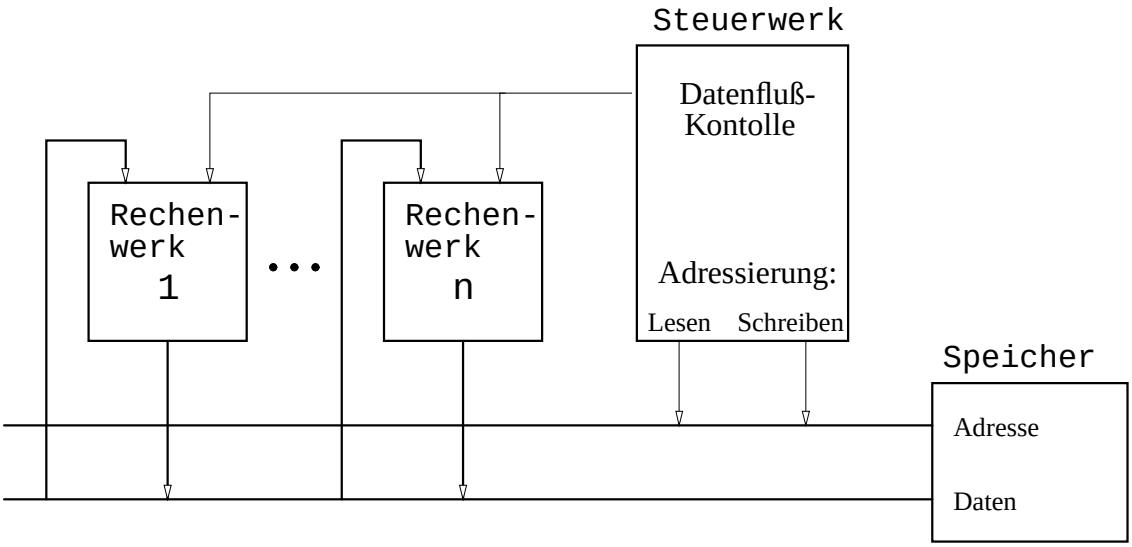


Abbildung 3.1: Aufbau des Prozessors nach dem SIMD-Prinzip

geschrieben werden kann. Dabei sind $v \in \mathcal{M}$ die betrachteten Farbkanäle. Entsprechend sind a^v und b^v die Farbwerte der Bildpunkte im jeweiligen Farbkanal. Der Parameter σ ist variabel, in der Praxis verwenden wir einen Wert von $\sigma = 0.16$. Zum Erhalten der initialen Wahrscheinlichkeitsverteilung der Korrespondenzen werden zu jedem Ausgangspixel aus Bild A alle Pixel aus dem zugehörigen Suchraum in Bild B auf Farbähnlichkeit geprüft. Der jeweils aus Gleichung (3.1) gewonnene Wahrscheinlichkeitswert bildet somit einen Wert der Wahrscheinlichkeitsverteilung. Für die gesamte Wahrscheinlichkeitsverteilung muß Gleichung (3.1) also für alle Ausgangspixel aus Bild A und jeweils alle möglichen Korrespondenzpartner im zugehörigen Suchraum in Bild B ausgewertet werden. Der Aufwand dieser Berechnung ist abhängig von der Bildgröße und der verwendeten Suchraumgröße. Da dieser Berechnungsschritt aber nur einmal zum Erhalten der initialen Wahrscheinlichkeitsverteilung durchgeführt wird, wollen wir diese Berechnung nicht in der FPGA-Hardware realisieren, sondern diesen in einem Vorverarbeitungsschritt auf dem Hauptrechner durchführen. Hier ist im Vergleich zu den späteren Iterationsschritten, die auf dem FPGA implementiert werden, eine höhere Rechengenauigkeit und die Verwendung der Exponentialfunktion leichter zu realisieren. Außerdem wird dieser Berechnungsschritt vom Datenfluß nur einmal durchlaufen, so daß bei einer Hardwarerealisierung die verwendeten Ressourcen für den wesentlich rechenaufwendigeren zweiten Teil des Algorithmus brach liegen würden. Die Iteration der Gleichung (2.29) wollen wir hingegen in der FPGA-Hardware implementieren, da hier die nötigen Ressourcen mehrfach genutzt werden können. Im Hinblick auf die begrenzten Ressourcen des verwendeten FPGA ist diese Aufteilung ein Kompromiß, der sich darauf konzentriert, den Teil des Algorithmus zu beschleunigen, der auf herkömmlicher, seriell arbeitender Hardware sehr zeitaufwendig wäre. Denkbar wäre natürlich auch eine Realisierung der Vorverarbeitung mittels parallel arbeitender Hardware, welche auch Geschwindigkeitsvorteile bieten würde, aber zu einer ineffizienteren Nutzung der gesamten Hardware führen würde, bzw.

sich mit den uns zur Verfügung stehenden Ressourcen nicht zusammen mit der folgenden Implementation realisieren läßt.

3.2 Operational Unit

In diesem Abschnitt beschäftigen wir uns nun mit der Realisierung des iterativen Teils des Algorithmus. Wir haben durch die Vorverarbeitung eine initiale Wahrscheinlichkeitsverteilung $\mathcal{F}^0$ gegeben und wollen darauf Gleichung (2.29) anwenden. Dieser Rechenschritt, angewendet auf die Wahrscheinlichkeitswerte f^0 aller möglichen Korrespondenzpaare (x_A, x_B), ergibt eine neue Wahrscheinlichkeitsverteilung $\mathcal{F}^1$. Zur Erinnerung schreiben wir die angewendete Gleichung zusammengefaßt als:

$$\begin{aligned} & f^{t+1}(x_A, x_B) \\ &= \hat{P}(X_B = x_B | \mathcal{F}^t, X_A = x_A) \\ &= f^t(x_A, x_B) \ \tfrac{1}{|\mathcal{N}|} \sum_{y_A : (y_A - x_A) \in \mathcal{N}} \max_{y_B} \ f^t(y_A, y_B) \ \hat{h}(x_A, x_B, y_A, y_B) \end{aligned} \tag{3.2}$$

Abbildung 3.2 zeigt den schematischen Aufbau der einzelnen Rechenschritte, die zur Auswertung der Gleichung für ein Korrespondenzpaar (x_A, x_B) nötig sind. Dabei sind die Eingangsdaten des Suchraumes vom Ausgangspixel $x_A = (x, y)$ detailliert und die Eingangsdaten der benachbarten Suchräume zusammengefaßt angegeben. Der untersuchte Korrepondenzpartner x_B hat in diesem Fall die Position links über dem Ausgangspixel. Die Elemente mit der Beschriftung `h` werten die Funktion $\hat{h}$ für alle Positionen eines Suchraumes aus, gewichten damit den jeweiligen Suchraumeintrag und bestimmen das Maximum aus den gewichteten Werten. Da zur Auswertung der Ordnungsbedingung $\hat{h}$ keine festen Positionen, sondern nur deren relativer Abstand bekannt sein muß, unterscheiden sich die Elemente `h` genau um diesen Parameter, der in jedem Element festgelegt ist. Das Element `E` bildet den Erwartungswert aus seinen acht Eingangswerten und das Element `M` führt eine Multiplikation aus. Diese Abbildung bietet einen Überblick, um den Zusammenhang der folgenden Abschnitte und die Aufgabe der entwickelten Schaltelemente im Rechenwerk zu verdeutlichen. Es folgt eine Erläuterung zur Codierung der verarbeiteten Daten und die Realisierung der einzelnen Elemente. Anschließend wird eine Normalisierung der Werte diskutiert.

3.2.1 Datencodierung

Die zentrale Datenmenge, mit der wir arbeiten, ist die Wahrscheinlichkeitsverteilung der Korrespondenzen. Diese wird in initialer Form von der Vorverarbeitung geliefert. Zum Weiterverarbeiten müssen diese Daten in geeigneter Form zwischengespeichert werden. Alle einzelnen Werte der Verteilung sind Wahrscheinlichkeitswerte zwischen Null und Eins. In der Vorverarbeitung verwenden wir üblicherweise die Genauigkeit der $32bit$

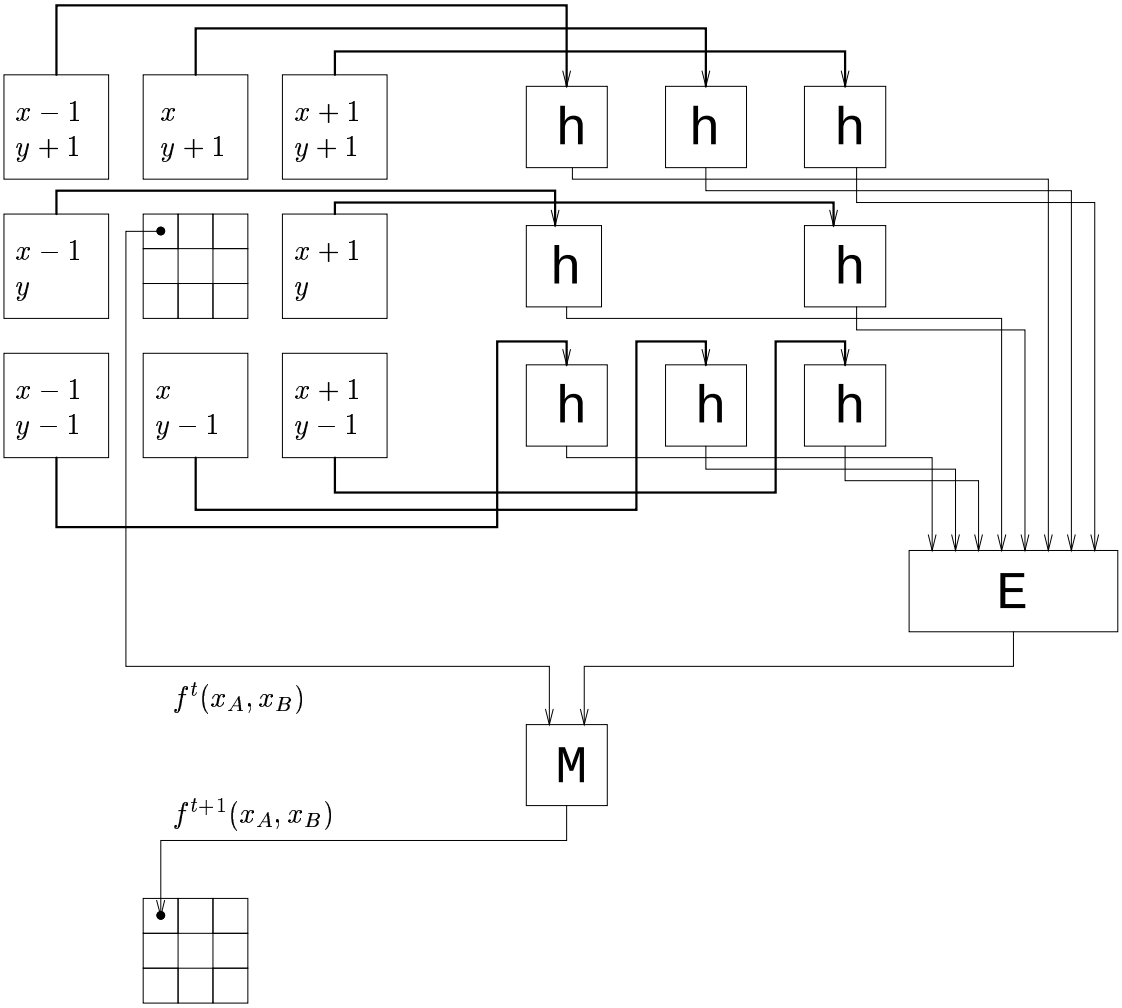


Abbildung 3.2: Schaltung zu Gleichung (3.2) mit $x_A = (x, y)$ und $x_B = (x-1, y+1)$

Fließkommadarstellung[3], in der Vorzeichen, Zahlenwert und Größenordnung getrennt angegeben sind. Der Zahlenwert ist so normiert, daß in der angegebenden Größenordnung keine Vorkommastellen außer einer 1 gibt. Die Größenordnung ist als Potenz der Basis 2 angegeben. Die Mantisse *MANT* beinhaltet die Nachkommastellen und ist mit $24bit$ codiert. Der Exponent der Größenordnung *EXP* ist mit $7bit$ *biased*[4] codiert. Zusätzlich wird ein Vorzeichenbit *VZ* verwendet. Dies ergibt die Darstellung einer Zahl nach folgender Gleichung:

$$Z_{FLOAT} = (-1)^{VZ} * (1.MANT_2) * 2^{(EXP_2 - 127_{10})} \tag{3.3}$$

Zur weiteren Verarbeitung auf dem FPGA ist dieses Format nur bedingt geeignet. Hier sind Fließkommaoperationen sehr aufwendig zu realisieren, da zu jeder Operation der Exponent angepaßt werden muß. Daher wollen wir zu einer Festkommadarstellung übergehen. Zusätzlich begrenzen wir die Genauigkeit auf $7bit$. Dies hat, wie wir in Abschnitt 3.3 sehen werden, in unserem Fall spezielle Vorteile in der Datenorganisation. Somit ist der Wertebereich zwischen Null und Eins in $2^7 = 128$ diskrete Werte eingeteilt. Wir codieren einen Wahrscheinlichkeitswert Z dargestellt durch die ersten 7 binären Nachkommastellen mit den Bits $< x_6, x_5, x_4, x_3, x_2, x_1, x_0 >$, so daß gilt:

$$Z_{7bit-FESTKOMMA} = x_0 * 2^{-7} + x_1 * 2^{-6} + \cdots + x_6 * 2^{-1} \tag{3.4}$$

Diese Darstellung entspricht einem Stellenwertsystem, bei dem jedem Bit entsprechend seiner Stelle eine Wertigkeit als Potenz der Basis zugeordnet ist. Allgemein gilt für ein Stellenwertsystem

$$Z = \sum_{i=n}^{N} \alpha_i * b^i, \tag{3.5}$$

wobei b die Basis ist und α_i die Koeffizienten der Stellen sind. In unserem Fall ist die Basis $b = 2$, die höchste Stelle $N = -1$ und die niedrigste Stelle $n = -7$. Die Koeffizienten α_i sind durch die Bits x_j gegeben, wobei $i = j - 7$ gilt. Zu beachten ist, daß bei dieser Codierung die Darstellung der 1 nicht möglich ist. Die größte darstellbare Zahl bestehend aus sieben Einsen ist $Z_{max} = 2^{-7} + 2^{-6} + 2^{-5} + 2^{-4} + 2^{-3} + 2^{-2} + 2^{-1} = 0,9921875$, was um $2^{-7} = 0,0078125$ kleiner ist als 1. Alternativ könnte man den gesamten Wertebereich um 2^{-7} verschieben, so daß die exakte 1 darstellbar ist, dann würde aber die Darstellung der 0 verlorengehen. Außerdem würde diese Darstellung zum Verlust des Stellenwertsystems führen, womit auch die bekannten allgemeinen Rechenregeln nicht mehr angewendet werden könnten. Wir wollen also in der Realisierung unseres Rechenwerkes eine feste Datenwortlänge von $7bit$ mit einer Darstellung nach Gleichung (3.4) verwenden.

3.2.2 Rechenschritte

In diesem Abschnitt werden die einzelnen Schritte zur Berechnung der zentralen Gleichung (3.2) vorgestellt. Dabei werden verschiedene Arten der Realisierung diskutiert, um

[3] nach IEEE 754

[4] um den halben Wertebereich verschoben, so daß auch negative Exponenten möglich sind

einen Kompromiß aus Rechengenauigkeit, Rechenzeit und Ressourcenbedarf zu finden. Wir unterteilen die Berechnung in Auswertung der Ordnungsbedingung, Bestimmung des Erwartungswertes und Gewichtung durch Multiplikation.

Ordnungsbedingung

In Abschnitt 2.4 haben wir eine Bedingung aufgestellt, die benachbarte Pixel zusammen mit ihren Korrespondenzpartnern erfüllen sollen. Dies war die Annahme über die Verteilung von Korrespondenzen innerhalb der Bilder. Wir haben in Gleichung (2.26) ein Maß $\hat{h}$ definiert. Dieses Maß drückt die Ordnung zweier Pixelpaare in einem Wahrscheinlichkeitswert aus. Ausgehend von einem einem Pixelpaar bestehend aus einem Ausgangspixel und einem seiner Nachbarn werden die Positionen ihrer Korrespondenzpartner verglichen. Sei x_A ein Ausgangspixel in Bild A und x_B sein angenommener Korrespondenzpartner in Bild B. Weiter sei y_A ein Nachbarpixel von x_A und y_B ein möglicher Korrespondenzpartner zu y_A in Bild B. Dann ergibt sich aus Gleichung (2.26) ein Wert, der aussagt, wie genau die Ordnung von x_A und y_A mit der von x_B und y_B übereinstimmt. Zur Erinnerung war die Funktion $\hat{h}$ die Normierung der Funktion h aus Gleichung (2.14), die über eine Gauß-Funktion wie folgt definiert war:

$$h(x_A, x_B, y_A, y_B) = \exp\left(-\frac{\left((x_A - y_A) - (x_B - y_B)\right)^2}{\sigma_h^2}\right) \tag{3.6}$$

Dabei ist σ_h ein Parameter, der die Strenge der Ordnungsbedingung bestimmt. Bei großem σ_h wird die Gaußkurve breiter, so daß geringe Abweichungen in der Ordnung noch als fast gleich gewertet werden, während kleinere σ_h die Ordnungsbedingung zunehmend verschärfen. Die Gaußkurve wird dann schmaler und es fallen somit Abweichungen stärker ins Gewicht. Wir möchten nun zur Realisierung dieser Bedingung den Implementationsaufwand der Exponentialfunktion vermeiden. Wir wählen ein sehr kleines σ_h, so daß nur exakt gleiche Ordnungen einen großen Funktionswert liefern, während alle davon abweichenden Pixelpositionen zu einem sehr kleinen Funktionswert führen. Somit definieren wir die Ordnungsbedingung sehr streng. Durch die diskreten Positionen der Pixel in den Bildern ergibt sich bei einer Abweichung der Positionen von nur einem Pixel bereits ein Ordnungsmaß von Null. Abbildung 3.3 zeigt diesen Übergang von einer großzügigen zu einer strengen Ordnungsbedingung. Beide Graphen zeigen das Ordnungs-

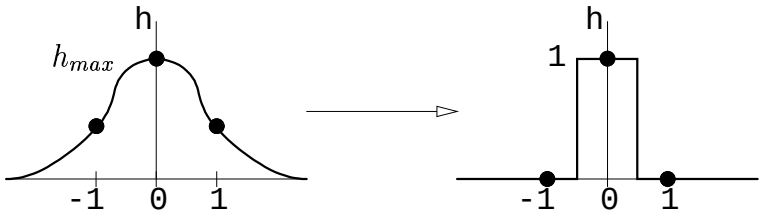


Abbildung 3.3: Ordnungsbedingungen: links großzügig, rechts streng

maß in Abhängigkeit von der Differenz der Abstandsvektoren der Korrespondenzpaare.

Im linken Graphen ist das σ_h so gewählt, daß die diskreten Abstandsdifferenzen von 1 und -1 noch einen in etwa halb so großen Ordnungswert erhalten, wie die Abstandsdifferenz 0 bei exakter Ordnung. Rechts hingegen ist die Ordnungsbedingung so scharf gewählt, daß bereits diese und alle größeren Abstandsdifferenzen zu einem Ordnungswert von 0 führen, während nur die Abstandsdifferenz 0 einen maximalen Ordnungswert von 1 ergibt. Durch die diskreten Positionen der Pixel in den Bildern sind die zu betrachtenden Differenzvektoren ebenfalls diskret in ihrem Betrag. Somit kann die Auswertung der Funktion durch eine *Lookup*-Tabelle realisiert werden, in der den diskreten Werten aus dem Definitionsbereich jeweils ein zuvor berechneter Funktionswert zugeordnet wird. In unserem Fall der strengen Ordnungsbedingung läßt sich diese *Lookup*-Tabelle weiter einschränken, da es nur zwei Funktionswerte gibt, die sich aus einer Fallunterscheidung ergeben. Dies hat zur Folge, daß wir die Funktion h nun wesentlich einfacher definieren können.

$$\tilde{h}(x_A, y_A, x_B, y_B) = \begin{cases} 1 : \text{für} \quad (x_A - y_A) - (x_B - y_B) = 0 \\ 0 : \text{sonst} \end{cases} \tag{3.7}$$

Gleichung (3.7) zeigt unsere vereinfachte Definition der strengen Ordnungsbedingung $\tilde{h}$. Durch die Wahl der Funktionswerte $\{0, 1\}$ erübrigt sich auch die nötige Normalisierung auf 1, so daß wir nun die Funktion $\tilde{h}$ direkt anstatt von $\hat{h}$ verwenden können. Weiter ist durch diese Wahl der Funktionswerte die Weiterverarbeitung der Werte mittels Multiplikation wesentlich vereinfacht möglich. Eine Multiplikation eines Wertes mit 1 und aller anderen mit 0 ist also die Auswahl eines Wertes aus einer gegebenen Menge, was der Funktion eines Multiplexers entspricht. Ein Multiplexer ist eine digitale Schaltung, die mit einfachen UND- und $ODER$-Operationen einen Wert oder ein Signal aus einer Menge von Eingängen auswählt und am Ausgang zur Verfügung stellt. In unserem Fall sind die Parameter der Funktion schon im Voraus durch die Pixelpositionen bekannt, so daß die Auswahl der Werte durch eine feste Verdrahtung der Datenleitungen möglich ist. Im Hinblick auf Gleichung (3.2) vereinfacht sich weiter die Maximumsuche, da die einzelnen Wahrscheinlichkeitswerte bereits durch die modifizierte Ordnungsfunktion gefiltert wurden, so daß nur ein Wert in Frage kommt. Somit können wir Gleichung (3.2) durch Anwendung der Funktion $\tilde{h}$ aus Gleichung (3.7) zu Gleichung (3.8) vereinfachen.

$$f^{t+1}(x_A, x_B) = f^t(x_A, x_B) \, \frac{1}{|\mathcal{N}|} \sum_{y_A : (y_A - x_A) \in \mathcal{N}} f^t(y_A, y_A - x_A + x_B) \tag{3.8}$$

Hier ist $y_B = (y_A - x_A) + x_B$ gesetzt, was durch die Verwendung von Gleichung (3.7) folgt. Desweiteren entfällt in diesem Zusammenhang die Maximumsuche wie oben beschrieben, da hier jeweils nur noch ein Wert betrachtet wird.

Erwartungswert

Betrachten wir nun weiter die Gleichung (3.8) und beschäftigen uns mit dem Erwartungswert im rechten Teil der Gleichung. Dieser besteht aus der Summe der Wahrscheinlichkeiten der ausgewählten Nachbarkorrespondenzen und der Division durch deren Anzahl.

Die Nachbarschaft $\mathcal{N}$ des Pixels x_A besteht, wie in Abschnitt 2.4 besprochen, aus den acht direkt benachbarten Pixeln y_A. Bei der Realisierung dieser Berechnung sind verschiedene Rechenwege möglich, die sich durch die Reihenfolge der Einzeloperationen unterscheiden. Verbunden damit sind Unterschiede in Bezug auf die benötigten Ressourcen, die Rechenzeit und den zu erwartenden Rundungsfehler. Im trivialen Fall wird erst die Summe aller Werte gebildet und diese dann durch die Anzahl der Werte geteilt. Dies führt dazu, daß die Zwischenergebnisse bis zur Summe in einem im Vergleich zu den Einzelwerten und dem Endergebnis vergrößerten Wertebereich liegen. In diesem Zusammenhang werden entsprechend große Addierwerke und Speicherplätze benötigt. Ausgehend von der Verwendung eines Addierwerkes, welches angepaßt an den Wertebereich der Summanden implementiert ist und jeweils zwei Werte addiert, vergrößert sich der Wertebereich dieser Summe um das Bit des eventuell entstehenden Übertrags. Wird so zu der Zwischensumme schrittweise jeweils einer der acht Werte hinzuaddiert, so benötigt man dazu insgesamt sieben Addierer, deren Wortlänge sich nach jeder Verdoppelung der bereits verarbeiteten Summanden um ein Bit erhöht. Nach Summation aller acht Summanden hat sich der Wertebereich verachtfacht, was einer Erhöhung der Wortlänge um drei Bit entspricht. Dieses Verfahren läßt sich auf beliebig viele Summanden anwenden. Da die Anzahl der Summanden in unserem Fall durch die Nachbarschaftsbeziehungen der Pixel auf acht festgelegt ist, betrachten wir nur diesen Umfang. Abbildung 3.4 zeigt diesen Rechenweg mit Addierern `ADD` [5] [6], die jeweils zwei Werte addieren und eine auf den Wertebereich angepaßte Wortlänge von `7bit`, `8bit` und `9bit` haben. Im Vergleich zu dieser rein sequentiell arbeitenden Implementation ist in Abbildung 3.5 eine Implementation dargestellt, die teilweise parallel arbeitet, somit weniger Rechenzeit und zudem noch weniger Ressourcen benötigt. Hier wird ausgenutzt, daß die Anzahl der Summanden

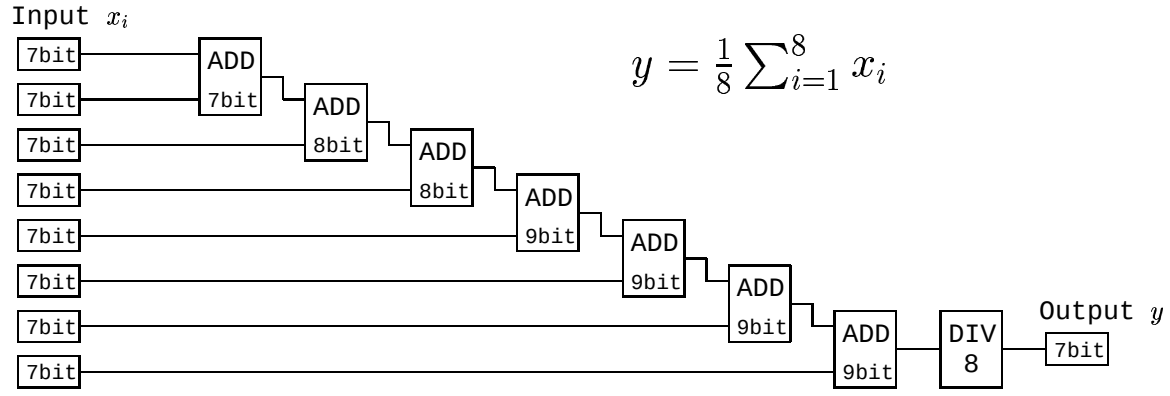
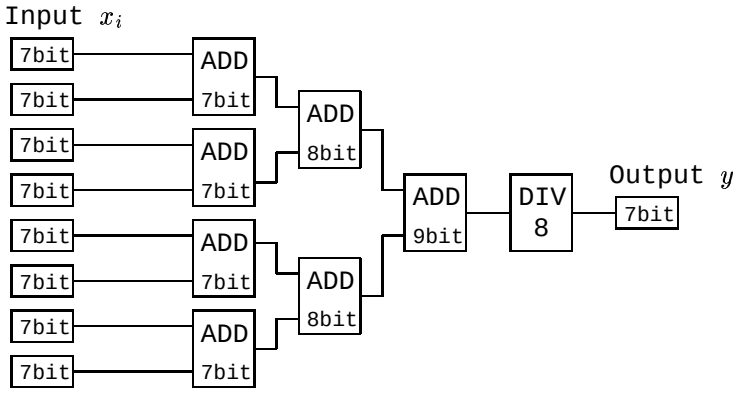


Abbildung 3.4: Mittelwertbildung sequentiell

festgelegt ist und in unserem Fall eine Potenz von 2 ist. Dadurch können wir paarweise parallel arbeiten, indem wir die acht Summanden zunächst in vier Paaren addieren und deren Summen wiederum paarweise addieren. So können im ersten Schritt vier Operationen

[5] Dieses sind Basiselemente, die als vorhanden vorausgesetzt werden. Erläuterung zu allen Basiselementen ist in [Kor02] zu finden.

[6] `Text in dieser Schriftart ist direkt in der Abbildung wiederzufinden.`



$$y = \frac{1}{8}\left([(x_1 + x_2) + (x_3 + x_4)] + [(x_5 + x_6) + (x_7 + x_8)]\right)$$

Abbildung 3.5: Mittelwertbildung kaskadiert

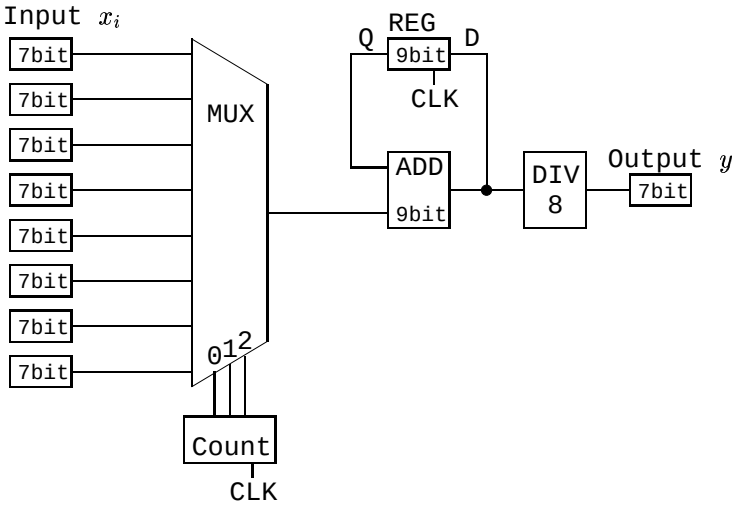


Abbildung 3.6: Mittelwertbildung Schleife

und im zweiten Schritt zwei Operationen nebeneinander ausgeführt werden. Dies ergibt eine Gesamtdurchlaufzeit von drei Stufen, die jeweils der Durchlaufzeit eines Addierwerkes entsprechen. Im Vergleich zu dem in Abbildung 3.4 dargestellten Verfahren, welches eine Durchlaufzeit proportional zu n benötigt, erreichen wir mit diesem Verfahren eine Abhängigkeit, die logarithmisch zu n ist. Betrachten wir die benötigten Ressourcen zur Realisierung dieser beiden Verfahren, so bestehen sie zunächst einmal jeweils aus sieben Addierwerken `ADD` und einem Divisionswerk `DIV` [7]. Der Unterschied liegt in der Größe der Addierer, denn diese sind jeweils an die Wertebereiche der zu erwartenden Zwischenergebnisse angepaßt. So erhöht sich, ausgehend von $7bit$ Wortbreite in der ersten Stufe, nach jeder Verdoppelung der betrachteten Einzelwerte die Wortbreite der Zwischenergebnisse und damit auch die der folgenden Addierer um ein Bit. Im ersten Verfahren, dargestellt in Abbildung 3.4, beträgt die Gesamtwortbreite aller sieben Addierer zusammen $59bit$. Hier steigt die Wortbreite in Abhängigkeit des Zwischenergebnisses aus den vorigen Berechnungen, während der neue Summand jeweils weiterhin nur im Wertebereich von $7bit$ liegt. Dies führt dazu, daß zunehmend größere Addierer verwendet werden müssen, weil einer der Summanden wächst. Im zweiten Verfahren aus Abbildung 3.5 wächst auch die Wortbreite der Addierer mit denen der Zwischenergebnisse, doch hier sind die zu addierenden Summanden jeweils aus dem gleichen Wertebereich. Hier wird die Größe eines Addierers immer ganz ausgenutzt und nicht aufgrund nur eines Summanden vergrößert. Somit ergibt sich eine gesamte Wortbreite der Addierer von insgesamt $53bit$. Im Vergleich zu dem vorigen Verfahren können wir also neben der verringerten Durchlaufzeit auch eine Ersparnis der Ressourcen erreichen. Eine in Bezug auf die benötigten Ressourcen ebenso vorteilhafte Realisierung ist in Abbildung 3.6 dargestellt. Ebenso wie im ersten Verfahren werden hier zunächst sequentiell die einzelnen Werte zu einer Zwischensumme addiert, die anschließend durch deren Anzahl geteilt wird. Die Zwischensumme wird in einem Register `Reg` [8] zwischengespeichert und dient der folgenden Addition als einer der beiden Summanden. Jedoch wird hier nur ein einziges Addierwerk benötigt, welches wiederholt verwendet wird. Durch den vorgeschalteten Multiplexer `MUX` [8] in Zusammenhang mit einem Zählwerk `Count` [8] wird nacheinander jeweils einer der Inputwerte als zweiter Summand ausgewählt. Die Elemente `Reg` und `Count` sind getaktete Bauelemente, die durch ein zusätzliches Taktsignal `CLK` gesteuert werden. Die gewählte Taktfrequenz muß dabei an die Durchlaufzeit des Addierers `ADD` und die des Multiplexers angepaßt sein. Das langsamere dieser beiden Elemente bestimmt die minimale Zeit zwischen zwei Berechnungen und somit die Zeit zwischen zwei Taktsignalen. Aufgrund der Komplexität wird hier der Addierer das langsamere Element sein und die Taktfrequenz begrenzen. Die gesamte Verarbeitungszeit dieser Implementation benötigt also wie auch die des ersten Verfahrens mindestens die siebenfache Durchlaufzeit eines Addierers und zusätzlich die Durchlaufzeit der Division. Der Vorteil dieser Implementation ist die Einsparung der weiteren Addierwerke. Da hier nur ein Addierer verwendet wird, ist die Ausnutzung dieser Ressource verbessert. Jedoch muß nun der Datenfluß dieses Element wiederholt durchlaufen. Dadurch ist zusätzlicher Aufwand nötig, um den Datenfluß zu steuern. Die Steuerung des Datenflusses wird durch die zusätzlichen Elemente Register und Multiplexer zusammen mit dem Taktsignal realisiert.

[7] Wird hier als Basiselement angesehen und später erläutert.

[8] Wird als Basiselement vorausgesetzt.

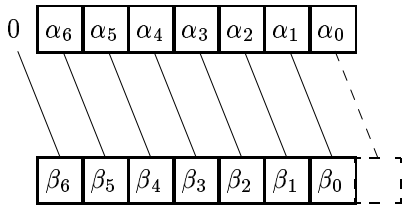
Deren Bedarf an Ressourcen ist im Vergleich zu den eingesparten Addierern geringer, so daß, auch obwohl nun von vornherein ein $9bit$ Addierer verwendet wird und die kleineren Zwischenergebnisse in einem großen $9bit$ Register gespeichert werden, in der Summe ein Teil der Ressourcen eingespart werden kann.
Alle drei vorgestellten Verfahren haben gemeinsam, daß zunächst die Summe aller Werte gebildet und darauf folgend die Division durchgeführt wird. Dabei wurden verschiedene Rechenwege der Summation und verschiedene Implementierungen dieser in Bezug auf Rechenzeit und Ressourcenbedarf verglichen. Die Division wurde jeweils am Ende durchgeführt, deren Implementation aber noch nicht erläutert. Da wir uns auf die Anzahl der Werte als Potenz von 2 festgelegt haben und die Daten in binärer Form nach Gleichung (3.4) vorliegen, ist in unserem Fall die Division sehr einfach zu realisieren. Wir können ausnutzen, daß in einem Stellenwertsystem die Division durch die Basis einer Verschiebung der Wertigkeit der Stellen entspricht. Allgemein ist bei dem Stellenwertsystem aus Gleichung (3.5) eine Division durch die Basis b und ebenso eine Multiplikation mit der Basis b durch eine Verschiebung der Wertigkeiten nach Gleichung (3.9) um eine Stelle nach rechts bzw. links zu realisieren.

$$b * \sum_{i=n}^{N} \alpha_i * b^{i-1} \quad = \quad \sum_{i=n}^{N} \alpha_i * b^i \quad = \quad \frac{1}{b} \sum_{i=n}^{N} \alpha_i * b^{i+1} \tag{3.9}$$

Ebenso können wir, anstatt die Wertigkeiten zu verschieben, diese auch konstant lassen und entsprechend die Koeffizienten in die entgegengesetzte Richtung verschieben. So kann also eine Division durch 2 entsprechend auch durch eine Verschiebung der Koeffizienten um eine Stelle nach rechts realisiert werden. Im Fall der Datenrepräsentation nach Gleichung (3.4) ergibt sich somit eine Verschiebung gemäß Gleichung (3.10), welche mit einem Rundungsfehler im letzten Bit aufgrund der begrenzten Genauigkeit behaftet ist.

$$\begin{aligned} Z &= < \alpha_6, \alpha_5, \alpha_4, \alpha_3, \alpha_2, \alpha_1, \alpha_0 > \\ \frac{1}{2}Z &\approx < 0\ , \alpha_6, \alpha_5, \alpha_4, \alpha_3, \alpha_2, \alpha_1 > \end{aligned} \tag{3.10}$$

Hier wird jedes Bit um eine Stelle nach rechts geschoben. Dabei fällt das zuvor niederwertigste Bit α_0 weg und das nun höchstwertige Bit wird durch eine Null aufgefüllt. Eine Division durch eine beliebige Potenz von 2 ist dann durch wiederholte Anwendung der Gleichung (3.10) möglich. Die in unserem Fall nötige Division durch 8 ist also durch dreimalige Anwendung der Gleichung zu realisieren, was einer Verschiebung der Koeffizienten um drei Stellen nach rechts entspricht. So ist die Division von Zahlen in einem Stellenwertsystem durch deren Basis bzw. Potenzen davon keine aufwendige Operation, sondern lediglich eine andere Interpretation der einzelnen Stellen. Abbildung 3.7 zeigt die nötigen Verbindungen, um $y = x/2$ mittels Shift-Operation zu realisieren. Hier sind x und y im Format nach Gleichung (3.4) angegeben. Die begrenzte Genauigkeit der Festkommadarstellung mit $7bit$ führt dazu, daß das letzte Bit α_0 nicht beachtet wird und sich daraus ein Rundungsfehler von maximal 2^{-8} ergibt. Der entscheidende Vorteil dieser Operation ist, daß zur Realisierung keine Ressourcen, außer den immer anfallenden Verbindungen zwischen den Elementen, benötigt werden. Wir können also ohne Zeitverlust und ohne Verwendung zusätzlicher Ressourcen eine Division realisieren. Diese



$$x = \sum_{i=0}^{6} \alpha_i 2^{i-7}$$

$$y = \sum_{i=0}^{6} \beta_i 2^{i-7} = \sum_{i=0}^{5} \alpha_{i+1} 2^{i-7}$$

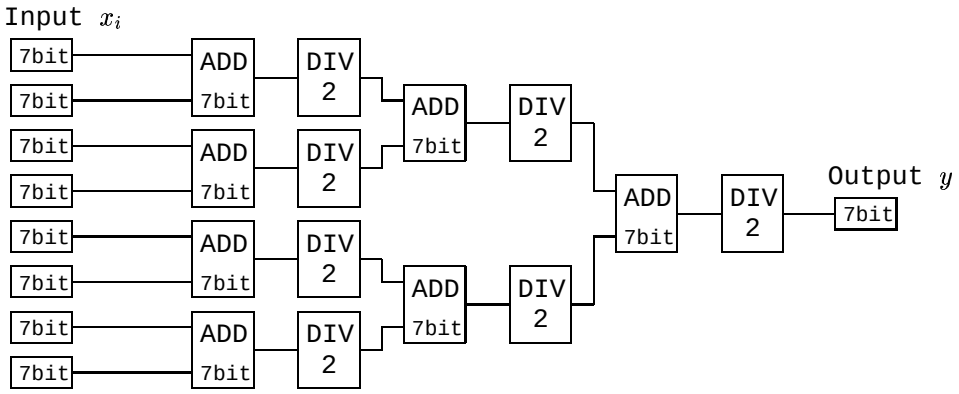
Abbildung 3.7: Division als Shift-Operation der Bits

Implementation läßt sich in allen drei Verfahren aus den Abbildungen 3.4, 3.5 und 3.6 anwenden. Weiter können wir nun das bisher in Bezug auf die Durchlaufzeit beste Verfahren aus Abbildung 3.5 bezüglich des Bedarfs an Ressourcen weiter verbessern. Wir haben festgestellt, daß die kaskadierte Berechnung der Summe bereits ressourcensparend im Vergleich zu der sequentiellen Summation aus Abbildung 3.4 ist. Beide Verfahren haben jedoch gemeinsam, daß die Zwischenergebnisse der Summation und damit auch die nötige Wortbreite der Addierwerke sich mit jeder Additionsstufe vergrößern. Die anschließende Division, realisiert durch einen dreifachen Rechtsshift, führt in beiden Fällen dazu, daß sich das Endergebnis wieder im Wertebereich von 7bit befindet. Alle zusätzlich berechneten weiteren Nachkommastellen fallen bei der Division dem Rundungsfehler des durch die Darstellung begrenzten Wertebereichs zum Opfer. Wir wollen nun den entstehenden Rundungsfehler dahingehend ausnutzen, daß wir die wegfallenden Stellen erst gar nicht berechnen, sondern auch mit den Zwischenergebnissen stets im selben Wertebereich bleiben. Zur Realisierung der Berechnung des Mittelwerts wollen wir ein rekursives Verfahren anwenden, in dem die Werte paarweise addiert werden und anschließend das Ergebnis durch 2 geteilt wird. Mit den so gewonnenen Zwischenergebnissen wird ebenso verfahren, bis sich ein einzelnes Endergebnis ergibt. Gleichung (3.11) zeigt einen möglichen rekursiven Aufbau der Berechnung, falls n eine Potenz von 2 ist.

$$\frac{1}{n}\sum_{i=1}^{n} x_i = \begin{cases} x_i & : \text{falls } n = 1 \\ \frac{1}{2}\left(\frac{1}{n/2}\sum_{i=0}^{n/2} x_i + \frac{1}{n/2}\sum_{i=(n/2)+1}^{n} x_i\right) & : \text{sonst} \end{cases} \tag{3.11}$$

Hier wird die Menge der Werte zunächst in zwei gleich große Untermengen eingeteilt. Falls die Mittelwerte der Untermengen bekannt sind, werden diese addiert und ihre Summe wird mit dem Faktor $1/2$ multipliziert, um wiederum den Mittelwert daraus zu erhalten. Andernfalls wird mit den entstehenden Untermengen das Verfahren wiederholt, bis diese einelementig sind und deren Mittelwert somit direkt bekannt ist. Die eigentliche Berechnung erfolgt rückwärts von den Einzelwerten über Zwischenergebnisse hin zum Endergebnis. In unserem Fall werden zunächst aus den acht Werten vier Paare gebildet. Durch paarweise Addition und Halbieren des Ergebnisses erhalten wir vier Zwischenergebnisse, mit denen ebenso verfahren wird. Sie werden in zwei Paare unterteilt, paarweise addiert und die Summen halbiert. Darauf folgt ein weiterer Schritt, in dem die letzten beiden Zwischenergebnisse addiert werden und deren Summe wiederum halbiert wird. Daraus resultiert das Endergebnis, welches bis auf Rundungsfehler dem Mittelwert der Eingangswerte entspricht. Dieses Verfahren läßt sich nur anwenden, falls die Anzahl der Werte, aus denen der Mittelwert gebildet werden soll, eine Potenz von 2 ist. Nur so ist sichergestellt, daß sich die Werte und alle Zwischenergebnisse einer Stufe jeweils in Paare

einteilen lassen und jeweils die Division durch eine Shift-Operation nach Abbildung 3.7 und Gleichung (3.10) realisiert werden kann. In unserem Fall ist die Anzahl der Werte $8 = 2^3$, somit sind drei der rekursiven Stufen nötig. Dieses Verfahren ist in Abbildung 3.8



$$y = \frac{1}{2}\left(\frac{1}{2}\left[\frac{x_1+x_2}{2} + \frac{x_3+x_4}{2}\right] + \frac{1}{2}\left[\frac{x_5+x_6}{2} + \frac{x_7+x_8}{2}\right]\right)$$

Abbildung 3.8: Mittelwert rekursiv

dargestellt. Hier sehen wir, daß auf jede Addition `ADD` eine Division `DIV` folgt. So wird der Wertebereich der Zwischenergebnisse, die am Addiererausgang durch den Übertrag um ein Bit erhöht sind, wieder um ein Bit erniedrigt und liegt wieder im Wertebereich der Eingangswerte. Alle Addierer benötigen hier eine Wortbreite von $7bit$, so daß insgesamt $49bit$ Addiererbreite benötigt wird. Im Vergleich zu dem kaskadierten Verfahren aus Abbildung 3.5 mit nur einer Division am Ende ist das eine Ersparnis von $4bit$. Die Durchlaufzeit der Berechnung bleibt durch die verteilte Division jedoch unverändert, da diese in beiden Fällen nach Abbildung 3.7 durchgeführt wird. Zu beachten ist, daß der nun bei jeder Division auftretende Rundungsfehler von maximal einem Bit nicht mehr ins Gewicht fällt, als der am Ende mögliche Rundungsfehler von drei Bits. Dieser Rundungsfehler ist also der gleiche wie der bei den anderen Verfahren mit der Division durch 8 am Ende und betrifft bei der Darstellung nach Gleichung (3.4) jeweils nur das letzte Bit des Ergebnisses, welches immer abgerundet wird. So ist das rekursive Verfahren aus Abbildung 3.8 vorteilhaft im Vergleich zu allen anderen vorgestellten Verfahren und wir wählen dieses aus, um es in unserem Rechenwerk zu implementieren. Wir können so durch Verbinden der Eingänge x_i mit den entsprechenden Werten von $f^t(y_A, y_A - x_A + x_B)$ nach der benötigten Durchlaufzeit am Ausgang y den Erwartungswert E_{x_A,x_B} der Nachbarkorrespondenzen von (x_A, x_B) weiterverarbeiten.

$$E_{x_A,x_B} = \frac{1}{|\mathcal{N}|} \sum_{y_A:(y_A-x_A)\in\mathcal{N}} f^t(y_A, y_A - x_A + x_B) \tag{3.12}$$

Multiplikation

Im vorigen Abschnitt haben wir die Realisierung der Berechnung des Erwartungswertes, den rechten Teil der Gleichung (3.8), behandelt. Wir können Gleichung (3.8) nun also durch Einsetzen von Gleichung (3.12) übersichtlicher schreiben.

$$f^{t+1}(x_A, x_B) = f^t(x_A, x_B)\ E_{x_A, x_B} \tag{3.13}$$

Betrachten wir nun das Produkt des Wahrscheinlichkeitswertes der Ausgangskorrespondenz $f^t(x_A, x_B)$ mit dem zuvor berechneten Erwartungswert E_{x_A,x_B} der Nachbarkorrespondenzen in Gleichung (3.13). Diese Multiplikation wollen wir nun ebenso wie die Berechnung des Erwartungswertes in einer Schaltung realisieren, die wir in die Hardware des FPGA implementieren wollen. Anstatt einer Implementation als *Lookup*-Tabelle, in der alle möglichen Ergebnisse zu allen möglichen Eingabewerten abgespeichert sind, suchen wir nach einer Möglichkeit, Ressourcen zu sparen. Denkbar wäre die Entwicklung eines Schaltnetzes, basierend auf der Betrachtung der Schaltfunktionen für jedes Bit des Ergebnisses. Hierzu erstellt man zunächst in Analogie zu der *Lookup-Tabelle* eine Wahrheitstabelle aller möglichen Eingangswerte, in der jedes Bit des Ausgangswertes einen Wahrheitswert *true* oder *false* zugeordnet bekommt. Für jedes Ausgangsbit ergibt sich daraus eine Schaltfunktion in disjunktiver Normalform (DNF) durch die Eingangswerte, die zu einem Wahrheitswert des jeweiligen Ausgangsbits von *true* führen. Jede dieser Schaltfunktionen kann nun durch das Verfahren von Quine und McCluskey [QM55] minimiert werden, woraus sich jeweils die minimale DNF der einzelnen Schaltfunktionen ergibt. Diese Schaltfunktionen bestehen dann nur aus UND- und $ODER$-Operationen, welche sich entsprechend in der Hardware implementieren lassen.
Wir wollen nun hingegen ein Verfahren entwickeln, welches die Multiplikation durch wiederholte Addition realisiert. Dazu betrachten wir das Beispiel der schriftlichen Multiplikation aus Abbildung 3.9.

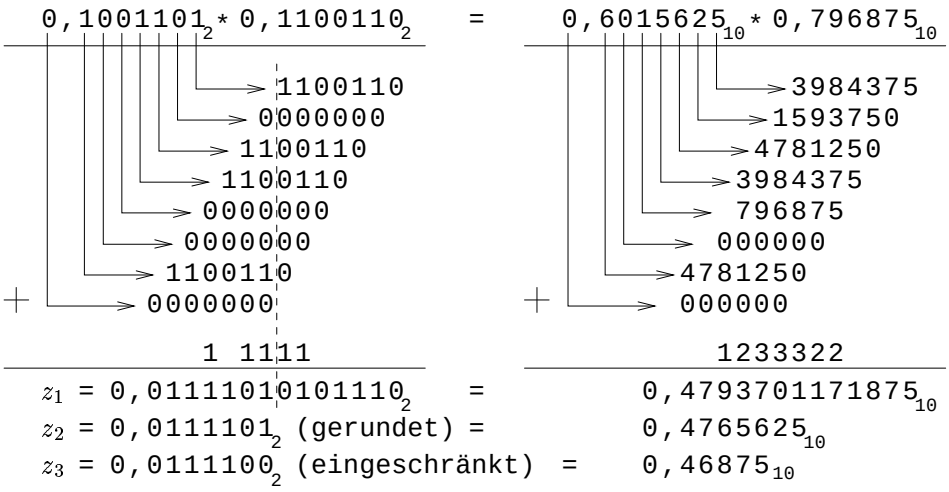


Abbildung 3.9: Beispiel: Schriftliches Multiplizieren (links im Binär- und rechts im Dezimalsystem)

$$\begin{aligned} x &= x_1 + x_2 \\ z &= x * y \\ &= (x_1 + x_2) * y \\ &= x_1 * y \ + \ x_2 * y \quad \text{(Distributiv-Gesetz)} \end{aligned} \tag{3.14}$$

Durch Anwendung des Distributivgesetzes aus Gleichung (3.14) können wir die Multiplikation in mehrere Einzelprodukte aufteilen, die anschließend addiert werden. Dies erlaubt in Verbindung mit der Darstellung der Werte im Stellenwertsystem nach Gleichung (3.5) das stellenweise Betrachten der zu multiplizierenden Werte. Speziell im Binärsystem vereinfacht sich so die Bildung der Einzelprodukte, da nur entweder mit 1 oder 0 multipliziert werden muß. Es muß also nur entschieden werden, ob ein Einzelprodukt ein Summand des Endergebnisses ist oder nicht. Die eigentliche Rechnung erfolgt dann erst bei dem Aufsummieren der Einzelprodukte. Durch die Zusammensetzung des Wertes x aus den Koeffizienten α_i der einzelnen Stellen, gewichtet mit der entsprechenden Potenz der Basis b^i, ergibt sich eine Aufteilung gemäß der Definition des Stellenwertsystems. Dabei nutzen wir den Vorteil, daß die Einzelprodukte aus der Multiplikation von dem gesamten Wert y mit einem der Koeffizienten α_i bestehen. Die jeweilige Potenz der Basis b^i bleibt als Faktor auch der Einzelprodukte erhalten. Wie wir im vorigen Abschnitt festgestellt haben, ist die Multiplikation einer Zahl im Stellenwertsystem mit ihrer Basis nach Gleichung (3.9) lediglich eine Verschiebung der Stellen. Eine Multiplikation mit einer Potenz der Basis ist durch wiederholte Anwendung der Gleichung zu realisieren und entspricht folglich einer Verschiebung um eine Anzahl von Stellen, die sich aus dem Exponenten der Basis ergibt. So ist auch hier die Multiplikation der Einzelprodukte mit dem Faktor b^i durch eine Shift-Operation wie in Abbildung 3.7 zu realisieren.
Dieses Verfahren ist allgemein als *schriftliches Multiplizieren* bekannt. Es läßt sich auf beliebige Zahlen in Stellenwertsystemen mit gleicher Basis anwenden. In Abbildung 3.9 wurden beispielhaft das Binär- und das Dezimalsystem gewählt und gezeigt, wie zwei Werte mit diesem Verfahren schrittweise multipliziert werden können. Links sind zwei Werte im Binärsystem mit sieben Nachkommastellen gemäß Gleichung (3.4) gegeben. Die Zwischenergebnisse ergeben sich aus dem Produkt jeweils eines Koeffizienten einer Stelle des ersten Wertes mit dem geamten zweiten Wert und werden in Zeilen untereinander geschrieben. Die Pfeile deuten an, mit welchen Koeffizienten jeweils multipliziert wurde. Die Verschiebung der Zwischenergebnisse hängt von der jeweiligen Wertigkeit des Koeffizienten ab und entspricht jeweils dem Exponenten der Basis. Der letzte Koeffizient des ersten Wertes hat z.B. die Wertigkeit 2^{-7}, entsprechend wird das Zwischenergebnis im Vergleich zum Endergebnis um sieben Stellen nach rechts verschoben. Sind so alle Zwischenergebnisse gebildet und verschoben, werden sie addiert und deren Summe ergibt das Endergebnis, das Produkt der beiden Faktoren. Allgemein ist die Anzahl der Nachkommastellen des Produktes die Summe der Nachkommastellen der beiden Faktoren. In diesem Fall haben beide Faktoren sieben Nachkommastellen, folglich hat das Produkt insgesamt 14 Stellen hinter dem Komma. Die Vorkommastellen ergeben sich aus der Summation, in unserem Fall ist es eine Null. Im rechten Teil der Abbildung wurden im Vergleich dieselben Werte im Dezimalsystem dargestellt und die Rechnung entsprechend durchgeführt. Es ergibt sich folglich dasselbe Ergebnis z_1 zur jeweiligen Basis.
Abbildung 3.10 zeigt eine Schaltung, die dieses Verfahren realisiert. Hier ist eine Dar-

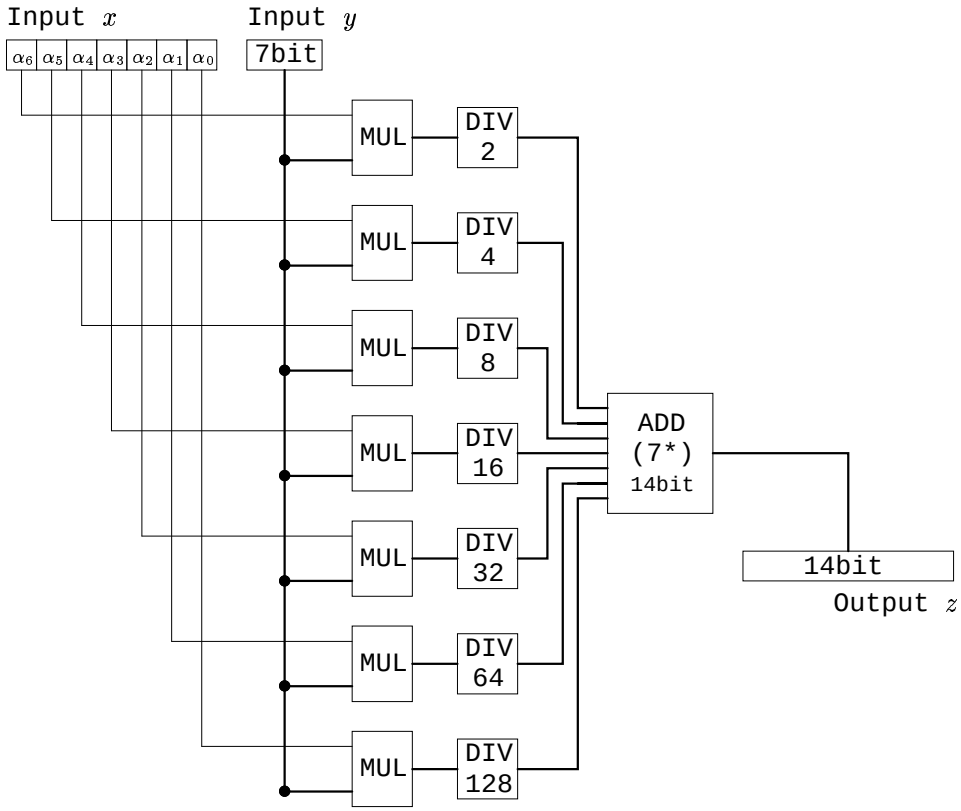


Abbildung 3.10: Multiplikations-Schaltung

stellung der Eingangswerte x und y nach Gleichung (3.4), welche auch im Beispiel aus Abbildung 3.9 verwendet wurde, vorausgesetzt. Der Wert y wird jeweils mit einem Bit α_i des Wertes x multipliziert und anschließend durch eine Division entsprechend der Wertigkeit des Koeffizienten α_i nach rechts verschoben. Alle so erzeugten Zwischenergebnisse werden zum Endergebnis addiert. Durch die Wortlänge der Zwischenergebnisse von $7bit$ und deren Verschiebung um maximal sieben Stellen ergibt sich die Breite des Addierers und die des Ergebnisses von $14bit$. Die hier verwendeten Multiplizierer `MUL` multiplizieren jeweils einen $7bit$-Wert mit einem $1bit$-Koeffizienten. Diese Operation läßt sich durch eine *UND*-Verknüpfung der einzelnen Bits des Wertes y jeweils mit dem Koeffizienten α_i implementieren. Die folgende Division `DIV` entspricht wie in Abbildung 3.7 und im Beispiel in Abbildung 3.9 lediglich einer Verschiebung der einzelnen Bits, da jeweils durch eine Potenz der Basis 2 geteilt wird. Da hier aber die Wortbreite der Zwischenergebnisse entsprechend groß gewählt ist, fallen die niederwertigen Bits beim Rechts-Shift nicht weg, sondern bleiben erhalten. Diese Verschiebung benötigt, wie im Zusammenhang mit Abbildung 3.7 im vorigen Abschnitt erläutert, keine zusätzlichen Ressourcen. Der Addierer `ADD` ist hier so konstruiert, daß er sieben Summanden gleicher Wortlänge addieren kann. Dies läßt sich auch durch wiederholte Anwendung eines einfachen Addierers für zwei Summanden ähnlich der Berechnung des Mittelwertes im vorigen Anschnitt kaskadiert implementieren. Hierbei gibt es wieder verschiedene Möglichkeiten Vorteile in Bezug auf Ressourcenbedarf und Rechenzeit auszunutzen.
Zunächst versuchen wir den Rechenaufwand schon im Voraus zu reduzieren. Da nicht alle 14 Nachkommastellen wie im Beispiel in Abbildung 3.9 zur Weiterverarbeitung interessant sind, sondern wir auch das Ergebnis im Format nach Gleichung (3.4) mit sieben binären Nachkommastellen speichern wollen, können wir die Berechnung auf diese sieben Bit einschränken. Gerundet[9] auf diese Genauigkeit ergibt sich aus dem Ergebnis z_1 der Wert z_2. Im Vergleich zu dem exakten Wert z_1 ist auch der auf sieben binäre Nachkommastellen gerundete Wert z_2 im Dezimalsystem dargestellt. Ein korrektes Runden fordert aber trotzdem die Berechnung aller Nachkommastellen, so daß dies keine Vereinfachung der Berechnung darstellt. Die Idee ist nun, die weitergehenden Nachkommastellen gar nicht erst zu berechnen und die Berechnung somit auf die interessanten Stellen zu beschränken. Das bedeutet, daß nur die Anteile der Zwischenergebnisse gebildet werden, die nach deren Verschiebung im Bereich der ersten sieben Bit des Ergebnisses liegen. Diese Grenze ist durch die senkrechte gestrichelte Linie im linken Beispiel aus Abbildung 3.9 angedeutet. Werden nur die Stellen der Zwischenergebnisse links von dieser Linie betrachtet, so ergibt sich in deren Summe z_3. Hierin sind dann natürlich auch eventuelle Überträge aus den hinteren Stellen nicht enthalten. Es ergibt sich also nicht das exakte Ergebnis der Multiplikation, sondern eine Annäherung, die etwas kleiner ist. Der Rechenfehler umfaßt bei dieser Einschränkung alle Stellen, die rechts der gestrichelten Linie stehen, einschließlich eventuell entstehender Überträge. In diesem Beispiel beträgt der Fehler im Vergleich zum exakten Ergebnis nur 2^{-7}, was hier etwa 2% entspricht. In der Summe kann der Fehler maximal $6 * 2^{-7}$ betragen. Dies ist der Fall, wenn alle Bits der beiden Zahlen 1 sind, da dann rechts der Linie ein großer Anteil der Summanden wegfällt. Der entstehende Fehler liegt auch dann mit 4.7% in einem für unsere Zwecke vertretbaren

[9] binäres Runden durch Abschneiden der weiteren Stellen

Rahmen. Diesen Nachteil des Rechenfehlers nehmen wir für den Vorteil der Einsparung des Rechenaufwands in Kauf, da die hinteren Nachkommastellen zur Weiterverarbeitung nur wenig von Bedeutung sind.
Die Schaltung in Abbildung 3.11 stellt eine Implementierung dieses modifizierten Verfahrens dar. Hier werden von vornherein nur jeweils die Bits des Wertes y betrachtet,

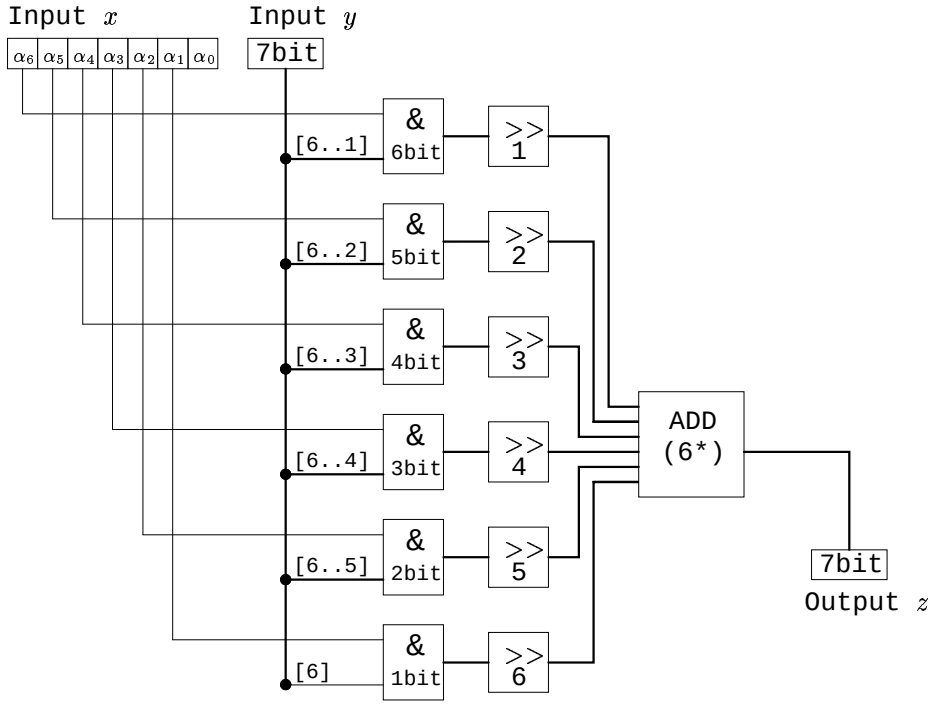


Abbildung 3.11: Multiplikation modifiziert

die nach der Verschiebung im Bereich der ersten sieben Nachkommastellen liegen. Die Auswahl ist durch die Angabe der Bits in eckigen Klammern an der jeweiligen Leitung angegeben. Im ersten Fall steht dort z.B. `[6..1]`, was bedeutet, daß alle Bits außer dem letzten betrachtet werden und im letzten Fall wird durch die Einschränkung `[6]` nur das höchstwertige Bit betrachtet. Die allgemeinen Elemente `MUL` aus der vorigen Schaltung sind wie angesprochen durch die bitweise &-Verknüpfung der eingeschränkten Werte mit den Koeffizienten α_i ersetzt. Es ist jeweils angegeben, welche Wortbreite diese Elemente haben. Die Divisionen `DIV` sind durch den Rechts-Shift `[>>]` um die angegebenen Stellen ersetzt, wobei nun die jeweils hinteren Bits wegfallen und nicht mehr betrachtet werden sollen.
Auf die Addition wird nun genauer eingegangen, da hier weitere Einsparungen der Ressourcen möglich sind. Die Anzahl der Summanden ist um den aus α_0 gebildeten verringert, da dieser durch die Verschiebung komplett im nicht betrachteten Bereich liegt. Alle Summanden haben nach dem Rechts-Shift nur noch eine verkleinerte Wortbreite, da zumindest das niederwertigste Bit wegfällt. Wollen wir nun wie im vorigen Abschnitt Addierer für zwei Summanden gleicher Wortbreite verwenden, so bietet es sich an, wieder kaskadiert vorzugehen, um teilweise parallel arbeiten zu können und desweiteren die Paare so zu bilden, daß jeweils möglichst Werte mit gleicher Wortbreite zusammen addiert

werden. Dies hat den Vorteil, daß die gesamte Wortbreite aller Addierer verkleinert wird und die Durchlaufzeit verringert wird. Wieder gehen wir einen Kompromis aus minimaler Durchlaufzeit und den dafür verwendeten Ressourcen ein. Wir addieren zuerst paarweise jeweils zwei der sechs Summanden mit möglichst gleicher Wortlänge parallel. Daraus ergebenen sich drei Zwischenergebnisse mit einer jeweils um das Bit des eventuell entstehenden Übertrages vergrößerten Wortlänge. Diese werden mit zwei weiteren Addierern zusammengefaßt, indem erst die beiden kleineren addiert werden und abschließend dazu das größte addiert wird. So benötigen wir eine gesamte Wortbreite aller Addierer von $24bit$ und kommen auf eine Druchlaufzeit von drei Addiererstufen.

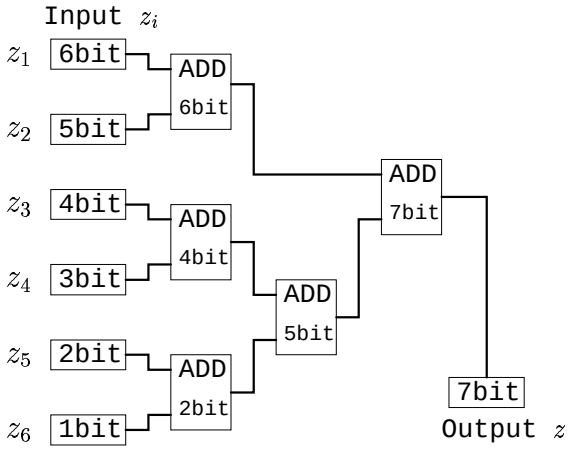


Abbildung 3.12: Addition von Summanden unterschiedlicher Wortlänge

Diese Vorgehensweise ist in Abbildung 3.12 dargestellt, wobei jeweils die Wortbreite der Werte z_i und der Addierer ADD angegeben ist. Da sich die nötige Größe eines Addierers immer nach der Wortlänge des größeren beider Summanden richtet, werden die Paare so gebildet, daß die Summanden möglichst gleiche Wortlänge haben. Bei ungerader Anzahl der Summanden, wie es hier bei den Zwischenergebnissen nach der ersten Addiererstufe der Fall ist, wird mit den kleineren Summanden begonnen und der größte bleibt bis zur nächsten Stufe übrig. So wird verhindert, daß die Zwischenergebnisse und damit die Wortbreite der folgenden Addierer unnötig groß werden. Nach dem letzten Addierer mit der Wortbreite von $7bit$ ist in unserer Anwendung kein Übertrag zu erwarten, da das Ergebnis der Multiplikation im selben Wertebereich wie die Faktoren liegen wird. Diese sind wie zuvor besprochen im Format nach Gleichung (3.4) angegeben und liegen damit zwischen Null und Eins. Also wird auch das Ergebnis in diesem Intervall liegen und - nach oben besprochenen Einschränkungen - im selben Format mit sieben Bit dargestellt. Wir können also die Berechnung der Gleichung (3.13) mit der Schaltung in Abbildung 3.11 und 3.12 implementieren. Zusammen mit der Einschränkung der Ordnungsbedingung, der daraus folgenden Berechnung des Erwartungswertes realisiert durch die Schaltung in Abbildung 3.8 und der Schaltung zur Multiplikation können wir nun die zentrale Gleichung (3.2) für gegebene Eingangswerte auswerten.

3.2.3 Normalisierung

Durch die Wahl der Darstellung der Wahrscheinlichkeitswerte mit fester Wortlänge nach Gleichung (3.4) ist der Wertebereich diskret. Wahrscheinlichkeiten für Korrespondenzen können also nur diskrete Werte mit sieben binären Nachkommastellen annehmen. Durch die Auswertung der Gleichung (3.2), die aus der Multiplikation von Wahrscheinlichkeitswerten besteht, ergeben sich neue Wahrscheinlichkeitswerte, die im allgemeinen kleiner sind als die, von denen ausgegangen wird. Wird die Gleichung mehrfach iteriert, so sinken die Werte mit jedem Iterationsschritt durch die Multiplikation, da das Produkt von zwei Werten aus dem Intervall $[0, 1]$ nach Gleichung (3.15) immer kleiner oder gleich dem kleineren der beiden Faktoren ist.

$$p * q \leq \min\{p, q\} \quad \text{für } p, q \in [\ 0..1\ ] \tag{3.15}$$

Da speziell der exakte Wert 1, der bei der Multiplikation zum Erhalt des anderen Faktors führen würde, in unserem Format nicht darstellbar ist, werden die Wahrscheinlichkeitswerte im allgemeinen durch die Multiplikation nach Gleichung (3.16) bei jedem Iterationsschritt kleiner.

$$p * q < \min\{p, q\} \quad \text{für } p, q \in [\ 0..1[\tag{3.16}$$

Zusammen mit der begrenzten Genauigkeit unseres Formates bedeutet das Sinken der Wahrscheinlichkeitswerte ein Verlust an Information, da viele Werte gerundet werden müssen. Dies ist kein allgemeiner Nachteil des vorgestellten Verfahrens zur Korrespondenzfindung, da letztendlich die Verhältnisse der Wahrscheinlichkeiten zueinander interessant sind. Durch unsere Darstellung der Werte mit begrenzter Genauigkeit ergibt sich aber das Problem, daß Werte, die eng beieinander liegen, auf denselben Wert gerundet werden und somit nicht mehr unterschieden werden können, oder daß zunehmend kleinere Werte nach unten aus dem Wertebereich heraus wandern und auf Null gerundet werden. Um dieses Problem zu umgehen, wollen wir nach jedem Iterationsschritt eine Normalisierung der Werte durchführen. Die Normalisierung soll nicht über die gesamte Wahrscheinlichkeitsverteilung der Korrespondenzen durchgeführt werden, sondern einzeln für jedes Pixel. Zu jedem Pixel aus dem Ausgangsbild existiert ein Suchraum mit einer Verteilung von Korrespondenzwahrscheinlichkeiten zu einer begrenzten Anzahl von Pixeln aus dem Zielbild. Durch die Annahme der im Kapitel 2 vorgestellten Bedingungen an die abgebildete Szene gehen wir davon aus, daß zu jedem Ausgangspixel x_A ein korrespondierendes Pixel x_B innerhalb des jeweiligen Suchraumes existiert. Daher normalisieren wir jeden Suchraum auf die maximale Wahrscheinlichkeit von Eins. Das heißt wir suchen zunächst innerhalb jedes Suchraumes das Maximum der Wahrscheinlichkeitswerte. Dann multiplizieren wir alle Wahrscheinlichkeitswerte des Suchraumes mit dem Kehrwert des Maximums. So werden alle Werte in Abhängigkeit des Maximums erhöht, so daß sich als neues Maximum der Wert Eins ergibt. Dadurch nutzen wir den zur Verfügung stehenden Wertebereich wieder ganz aus und können weiter mit der maximalen Genauigkeit rechnen.

Maximumsuche

Die Bestimmung des Maximums aus einer Menge von Werten erfordert im allgemeinen eine Anzahl von sequentiellen Vergleichen, die der Anzahl der Werte entspricht. Wir wollen ein Verfahren vorstellen, welches teilweise parallel arbeitet, die Darstellung der Werte im binären Stellenwertsystem ausnutzt und stellenweise das Maximum bestimmt. Nehmen wir zunächst die Einschränkung an, daß alle Werte nur eine Wortlänge von $2bit$ haben. Dann gibt es nur vier Fälle, die unterschieden werden müssen, um das Maximum zu bestimmen. Beginnend mit einer *ODER*-Verknüpfung der höchstwertigen Bits aller Werte wird ermittelt, ob dieses Bit in einem oder mehreren der Werte vorkommt. Ist dies der Fall, so wird auch das Maximum dieses Bit enthalten und wir können das höchstwertige Bit im Ergebnis setzten. Dann wird weiter jeweils das nächst niedrigere Bit aller Werte betrachtet und mit *ODER* verknüpft. Ist dieses nicht vorhanden, so wird auch im Ergebnis dieses Bit nicht gesetzt. Ist das höchstwertige Bit nicht, aber das nächst niedrigere in einem oder mehreren Werten vorhanden, so können wir dieses auch im Ergebnis setzten. Nun bleibt noch der Fall zu prüfen, in dem das höchstwertige Bit vorhanden ist, und das nächst niedrigere auch in einem oder mehreren Werten vorkommt. Dann muß zusätzlich geprüft werden, ob beide Bits im selben Wert auftreten, denn nur dann ist dieser Wert das Maximum, und es wird auch das Ergebnis diese beiden Bit enthalten. Wir wollen mit der Genauigkeit nicht weiter in die Tiefe gehen, sondern die Komplexität der Maximumsuche auf diese vier Fälle begrenzen. Zur Erinnerung sollte die Maximumsuche nur zum Normalisieren dienen, damit die Werte durch die wiederholte Multiplikation nicht den Wertereich verlassen. Betrachten wir nur die obersten beiden Bit unserer aus sieben Bit bestehenden Wahrscheinlichkeitswerte, so läßt sich dieses Verfahren anwenden, um eine grobe Abschätzung des Maximums zu erhalten. Durch die vier zu unterscheidenden Fälle

$< m_1, m_0 >$	Intervall				
`0 0`	0	$\leq$	$\max\{x_i\}$	$<$	0.25
`0 1`	0.25	$\leq$	$\max\{x_i\}$	$<$	0.5
`1 0`	0.5	$\leq$	$\max\{x_i\}$	$<$	0.75
`1 1`	0.75	$\leq$	$\max\{x_i\}$	$<$	1

Tabelle 3.1: Zuordnung der Intervalle

wird der Wertebereich in vier gleich große Intervalle eingeteilt und wir können mittels Tabelle 3.1 die Aussage treffen, in welchem dieser Intervalle das Maximum der Werte liegt.
Dieses Verfahren läßt sich mit der Schaltung aus Abbildung 3.13 realisieren. Hier wird das Maximum aus drei Werten mit jeweils $2bit$ Wortlänge durch die oben genannten Vergleiche ermittelt. Dabei wird für jedes Ergebnisbit getrennt vorgegangen, indem die Inputwerte auf die vier Fälle hin überprüft werden. Wegen der Übersichtlichkeit sind hier nur drei Inputwerte {`A,B,C`} bearbeitet, eine Erweiterung auf eine größere Anzahl von Werten ist jedoch nach gleichem Schema möglich. Die Komplexität wächst dabei in die Breite, da alle zusätzlichen Vergleiche parallel ausgeführt werden können. Die Durchlaufzeit hängt von den drei nacheinander durchlaufenden Stufen ab, wobei die

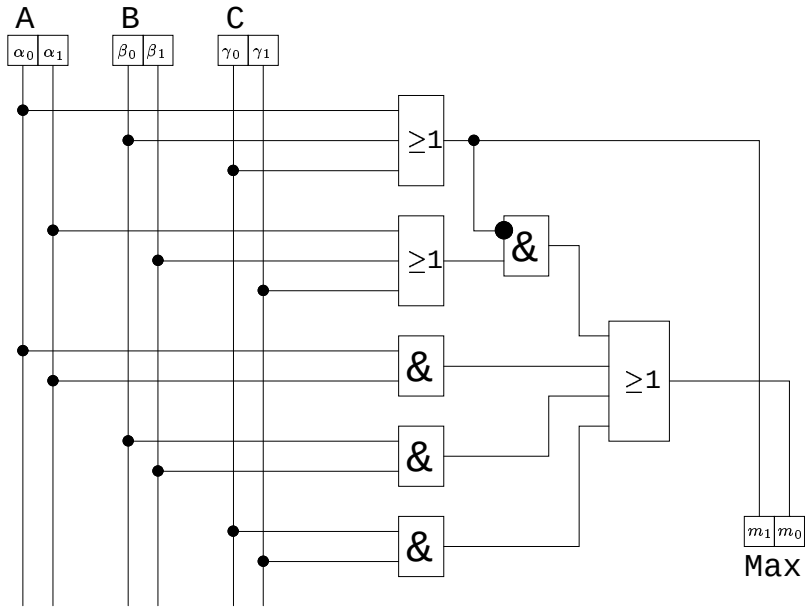


Abbildung 3.13: Maximumsuche aus drei Werten mit je 2bit

ODER-Operationen $[\geq 1]$ durch eine größere Anzahl von Eingängen sowohl zeit- als auch ressourcenaufwendiger werden. In unserer Anwendung hängt die Anzahl der Werte von der Suchraumgröße ab. Im Zusammenhang mit den im Abschnitt 3.3 folgenden Überlegungen zur Datenorganisation und -Verwaltung werden wir uns in der vorgestellten Realisierung auf einen Suchraum der Größe 3×3 festlegen. Daraus ergeben sich zu jedem Ausgangspixel Suchräume mit Wahrscheinlichkeitswerten für Korrespondenzen zu neun Pixelpositionen. Die Maximumsuche wird also auf diese neun Werte angewendet.

Anpassen der Werte

Basierend auf der Aussage, in welchem Intervall das Maximum der Wahrscheinlichkeitswerte eines Suchraumes liegt, sollen nun alle Werte neu auf den gesamten Wertebereich gestreckt werden. Die Multiplikation mit dem Kehrwert des Maximums wollen wir, angepaßt an die zuvor vorgestellte Maximumsuche, entsprechend kostengünstig realisieren. Wir unterscheiden die vier Fälle und ersetzen die vollständige Multiplikation durch eine einzige Addition in Verbindung mit Shift-Operationen zur Bildung der Summanden. Dabei ersetzen wir, wie in Tabelle 3.2 angegeben, die allgemein nötige Multiplikation der zweiten Spalte durch die Addition rechts daneben. Zur Aufspaltung der Faktoren benutzen wir wieder das Distributiv-Gesetz aus Gleichung (3.14). Dabei versuchen wir die neuen Faktoren so zu wählen, daß sie Potenzen der Basis 2 sind, damit die Multiplikation durch eine Shift-Operation wie in Gleichung (3.9) und Abbildung 3.7 realisiert werden kann. Im dritten Fall ist der Faktor $1\frac{1}{3}$ leider nicht durch eine endliche Folge zur Basis 2 darstellbar. Daher wählen wir die etwas kleinere Approximation durch die Faktoren 1 und $1/4$. Werden die Summanden entsprechend der rechten Spalte der Tabelle 3.2 gebildet, so

Max $< m_1, m_0 >$	Multiplikation (allgemein)	Addition (und Shift)
0 0	$x_i * 4$	$x_i * 2 + x_i * 2$
0 1	$x_i * 2$	$x_i * 2 + x_i * 0$
1 0	$x_i * 1\frac{1}{3}$	$x_i * 1 + x_i * 1/4$
1 1	$x_i * 1$	$x_i * 1 + x_i * 0$

Tabelle 3.2: Zuordnung der Operationen

ist nur eine einzige Addition nötig, um einen Wert an ein durch zwei Bits gegebenes Maximum anzupassen. Abbildung 3.14 zeigt eine Schaltung, welche die Shift-Operationen und die Addition realisiert. Hier werden zunächst die beiden Summanden in Abhängigkeit

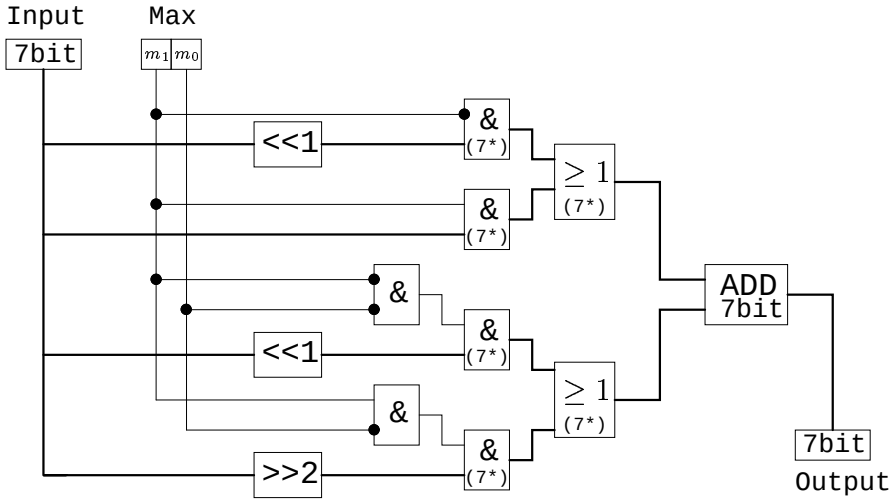


Abbildung 3.14: Normalisierung

des Maximums `Max` aus dem Eingangswert `Input` gebildet. Das Bit m_1 entscheidet, ob sich der erste Summand direkt aus dem Eingangswert ergibt, oder ob dieser vorher durch einen Links-Shift `[<<1]` verdoppelt wird. Für den zweiten Summanden wird der Eingangswert entweder ebenfalls verdoppelt, oder durch einen Rechts-Shift um zwei Stellen `[>>2]` geviertelt. In den übrigen Fällen mit $(m_1, m_0) \in \{(0,1),(1,1)\}$ bleibt der zweite Summand Null. Nach der anschließenden Addition ergibt sich der entsprechend gebildete Wert am Ausgang. Diese Normalisierung kann anschließend an die Maximumsuche für jeden der Werte im Suchraum parallel durchgeführt werden.

3.2.4 Gesamtes Rechenwerk

In den vorhergehenden Abschnitten haben wir die Realisierung der einzelnen Rechenschritte diskutiert, die wir nun anwenden wollen, um das gesamte Rechenwerk angeben zu können. Wir verwenden schematische Darstellungen der Schaltungen, wobei es

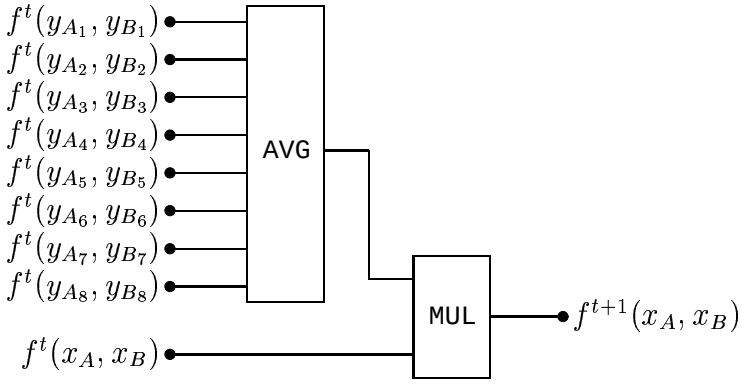


Abbildung 3.15: Schaltnetz zur Gleichung (3.2)

nun um die Verknüpfung der Rechenschritte untereinander gehen soll. Abbildung 3.15 zeigt die Verbindung der Berechnung des Erwartungswertes aus Gleichung (3.12) mit der Schaltung zur Multiplikation in Gleichung (3.13). Dabei sind y_{A_i} die Positionen der acht direkten Nachbarn von x_A. Die dazu betrachteten Korrespondenzpartner ergeben sich durch Anwendung der Gleichung (3.7) zu $y_{B_i} = y_{A_i} - x_A + x_B$. Diese Schaltung beinhaltet die eigentliche Iteration eines Eintrags der Wahrscheinlichkeitsverteilung f^t. Im folgenden werden wir dies zusammengefaßt zu dem Schaltsymbol AVG+MUL verwenden. Sind alle Werte des Suchraumes zu x_A berechnet, so kann darauf die Maximumsuche aus Abbildung 3.13 und die Normalisierung aus Abbildung 3.14 folgen. Da die Maximumsuche erst durchgeführt werden kann, wenn alle Werte im Suchraum berechnet sind, deren Berechnung aber unabhängig voneinander ist, bietet es sich an, diese parallel zu berechnen. Verwenden wir die Schaltung AVG+MUL mehrfach parallel entsprechend der Größe des Suchraumes, so benötigen wir zwar die mehrfache Menge an Ressourcen, nicht aber mehr Bearbeitungszeit, wie es bei der Implementation auf einem sequentiell arbeitenden Rechner der Fall wäre. Dieser Grad der Parallelität bietet sich an, da so alle zur Weiterverarbeitung nötigen Werte zur gleichen Zeit bereitgestellt werden. Die Anordnung und Verschaltung der parallelen Elemente AVG+MUL, der zentralen Maximumsuche MAX und der anschließend wieder parallelen Normalisierung NORM ist in Abbildung 3.16 dargestellt. Diese gesamte Schaltung bildet die *Operational Unit*. Die Durchlaufzeit beträgt bei der Hardwarerealisierung dieser Schaltung ca. 2 Nanosekunden, was eine Taktfrequenz von bis zu $40\mathsf{MHz}$ erlaubt. Wir bearbeiten damit einen Suchraum der Größe 3×3 und berechnen somit neun Werte parallel. Die Eingangswerte sind alle Elemente der Verteilung f^t, wobei eine Position x_A fest gewählt ist. Daraus ergeben sich die neun möglichen Korrespondenzpartner x_B mit einem durchnumerierten Index. Die Werte der acht Nachbarkorrespondenzen y_{A_i}, y_{B_i} sind wie in Abbildung 3.15 auszuwählen und sind jeweils zusammengefaßt angegeben. Es ergibt sich eine Menge von $(3 \times 3) \times 9 = 81$ Eingangswerten, die zum einen aus dem Suchraum zu x_A und zum anderen aus den Suchräumen zu allen acht Nachbarn y_{A_i} resultieren. Daraus werden die neun Werte des Suchraumes zu x_A berechnet, die einen Teil der Wahrscheinlichkeitsverteilung $\mathcal{F}^{t+1}$ bilden. Wird dieses Rechenwerk auf alle Ausgangspixel x_A im Bild $\mathcal{A}$ angewendet, so entspricht dies einem Iterationsschritt der in Abschnitt 2.6 angesprochenen Verteilung der lokalen Bedingun-

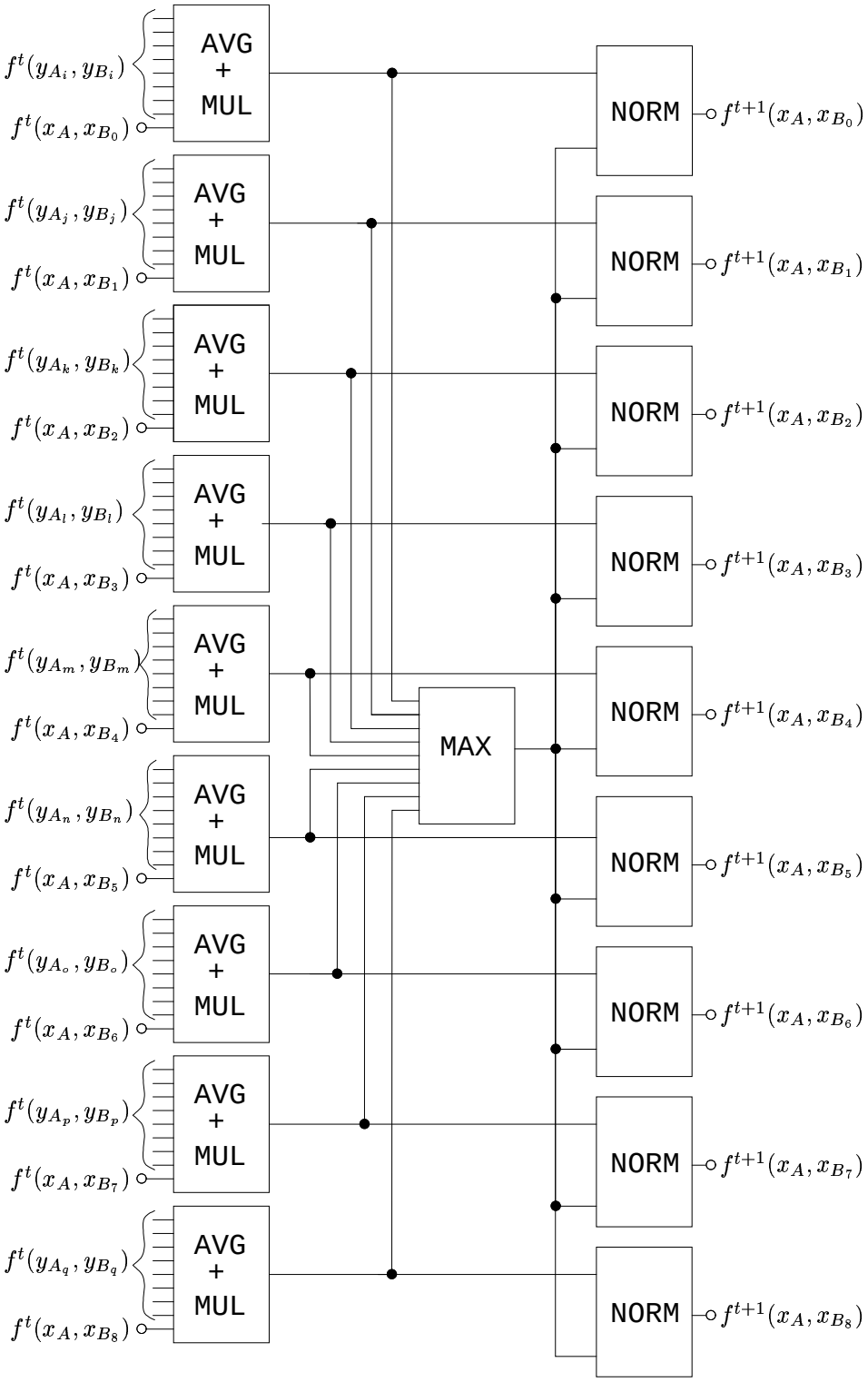


Abbildung 3.16: Operational Unit

gen. Denkbar wäre eine weitere Parallelisierung der Berechnung durch weitere wiederholte Implementierung des Rechenwerkes. Dies würde entsprechend mehr Ressourcen der Hardware benötigen, welche in unserem Fall nicht zur Verfügung stehen. Weiter ist zu beachten, daß der Datentransport nicht beliebig parallelisierbar ist, da Speicherzugriffe und Datenaustausch zwischen verschiedenen Hardwarekomponenten im allgemeinen seriell erfolgen. Das bedeutet in unserem Fall, daß wesentlich mehr Zeit benötigt wird, um Eingangsdaten bereitzustellen und Ausgangsdaten abzuspeichern, als diese zu verarbeiten. Daher beschränken wir uns auf den oben vorgestellten Grad der Parallelität von neun Berechnungen und diskutieren im folgenden Abschnitt die Verwaltung der Daten und Steuerung des Rechenwerkes. Diese Aufgabe soll in einer *Control Unit* zusammengefaßt und implementiert werden.

3.3 Control Unit

Hier wird die zentrale Kontrolle des Datenflusses diskutiert. Es wird der Datentransfer zwischen FPGA und Hauptrechner besprochen und eine Implementation des internen Datenflusses im FPGA vorgestellt. Die Control Unit übernimmt die Steuerung des Rechenwerkes, Speicherzugriffe auf den FPGA-internen Speicher und den Datenaustausch mittels PCI-Bus.

3.3.1 Speicherzugriffe

Der von uns verwendete FPGA verfügt über ein eigenes SRAM[10] mit einer Kapazität von 2MB. Die Kommunikation mit dem RAM übernimmt ein Schaltelement, welches auf dem FPGA implementiert wird. Ausgehend von Schaltelement `S2RAM`[11] zur Ansteuerung des RAM, das seriell Lese- oder Schreibzugriffe erlaubt, soll nun eine erweiterte Implementierung vorgestellt werden, die es ermöglicht, parallel zu lesen und zu schreiben. Die Funktionalität des sogenannten *Dual-Port–RAM*[12] wird erreicht, indem das vorhandene RAM intern doppelt so schnell getaktet wird und somit in einem externen Takt zwei interne Takte realisiert. Dadurch sind zwei Zugriffe möglich, die aus externer Sicht parallel abgearbeitet werden. Die beiden internen Takte teilen sich auf in eine Leseoperation und eine Schreiboperation. Das Schreiben ist zusätzlich mit einer Bedingung, dem Zustand des Signals `WE` verknüpft, welches aktiv sein muß, um die Daten zu übernehmen. So wird verhindert, daß bei vorzeitigem Anlegen von Adresse oder Daten ungewollt Speicherplätze überschrieben werden. Das SRAM verfügt über eine $21bit$ Adressleitung, mit der Speicherplätze ausgewählt werden können, die jeweils $36bit$ Wörter umfassen. Wir verwenden allerdings bedingt durch das Datenformat, welches auch über den nur $32bit$ breiten PCI-Bus transportiert werden soll, allgemein eine Wortbreite von $32bit$. Von außen gesehen besitzt das Dual-Port-RAM zwei Adressleitungen und ebenso zwei Daten-

[10] SRAM = statisches RAM

[11] Ist als Basiselement vorhanden.

[12] vergleiche engl. dual = zwei, port = Anschluß

leitungen. Hinzu kommt das Write-Enable-Signal[13] WE und die von außen generierten Taktsignale, welche aus einem Haupttakt CLK zum externen Ansteuern und einem dazu phasensynchronen Takt CLK_fast mit doppelter Frequenz zum internen Ansteuern des RAM bestehen. Um nach außen hin unabhängig die Eingangsdaten DI verarbeiten zu können, werden diese intern zunächst in einem Register[14] RegDI zwischengespeichert. Ebenso werden die gelesenen Ausgangsdaten DO in einem Register RegDO zwischengespeichert, damit sie nach außen dem Haupttakt entsprechend lange verfügbar sind. Die

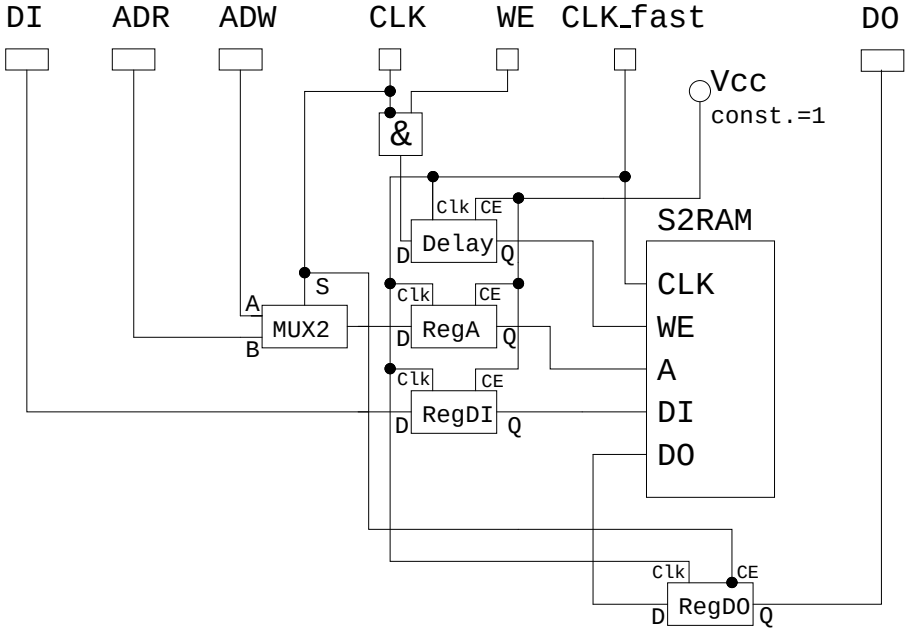


Abbildung 3.17: Dual-Port-RAM

entscheidende Erweiterung ist ein Multiplexer, der von außen zwei Adressleitungen aufnimmt und diese abwechselnd an das SRAM weitergibt. Die Umschaltung wird durch den Pegel des Haupttakts gesteuert, wodurch immer einen halben Takt lang die Leseadresse ADR und die andere Hälfte die Schreibadresse ADW ausgewählt wird. Die jeweilige Adresse wird zwischen Multiplexer MUX2 und RAM in einem Register RegA zwischengespeichert, damit sie jeweils genügend lange Zeit konstant am RAM anliegt. Dies ist nötig, damit unabhängig von Verzögerungen des Multiplexers bei jeder aufsteigenden Flanke des Arbeitstakts am RAM eindeutige Daten zur Verfügung stehen. Das Signal WE zum Schreiben der anliegenden Daten in den Speicher wird nur bei negiertem Haupttakt beachtet und durch ein Verzögerungsglied Delay geleitet, damit es nicht die Leseoperation behindert und erst beim internen Schreibtakt wirksam wird. Dies ermöglicht auch bei von außen konstant anliegendem Schreibsignal die Unterteilung der Lese- und Schreiboperation. Mit der in Abbildung 3.17 dargestellten Schaltung läßt sich diese Funktiona-

[13] vergleiche engl. write = schreiben, enable = ermöglichen

[14] Register: Speicherelement für eine Reihe boolescher Zustände, die bei aufsteigender Taktflanke CLK und aktivem Signal CE parallel am Eingang D übernommen und kurz darauf am Ausgang D bereitgestellt werden. Vergleiche [TS02]

lität implementieren. Wir verwenden dieses Element mit einer Haupttaktfrequenz von bis zu $20\mathsf{Mhz}$, da das zuverlässige Funktionieren des verwendeten SRAM nur bis zu einer Taktrate von $40\mathsf{Mhz}$ gewährleistet ist und durch die Verdoppelung des Haupttaktes diese Grenze somit erreicht ist.

3.3.2 Datenfluß

Die zuvor erläuterte Schnittstelle für Speicherzugriffe soll nun genutzt werden, um das Rechenwerk mit Eingangsdaten zu versorgen und die Ergebnisse aufzunehmen. Der Datenfluß soll einen Kreislauf ausführen, in dem ständig neue Werte aus dem Speicher ins Rechenwerk transportiert werden und gleichzeitig die Ergebnisse der vorigen Berechnung wieder abgespeichert werden. Diese Art von Kreislauf wird *Pipeline*[15] genannt und ermöglicht eine Optimierung der Verarbeitungsgeschwindigkeit, wenn alle beteiligten Elemente unabhängig voneinander arbeiten können und jeweils dieselbe Zeit benötigen, um einen Datensatz zu verarbeiten. Der Vorteil besteht darin, daß jedes Element in jedem Arbeitstakt aktiv ist und sich immer mehrere Datensätze in der Pipeline befinden. Die Verarbeitungszeit eines einzelnen Datensatzes ergibt sich dabei aus der Länge der Kette der durchlaufenden Elemente. Werden mehrere Datensätze hintereinander durch diese Kette verarbeitet, so erhöht sich die Verarbeitungszeit nicht um ein vielfaches der Einzellaufzeit, sondern steigt, nachdem die Kette vollständig gefüllt ist, mit jedem weiteren Datensatz nur noch um die Laufzeit eines Elementes. Wichtig dabei ist die Möglichkeit, den Speicher im selben Zeittakt sowohl lesend als auch schreibend zu benutzen. Ansonsten müßten diese beiden Operationen nacheinander ausgeführt werden, was den Rest der Kette unnötig abbremsen würde. Sind die Verarbeitungszeiten der Elemente unterschiedlich, so richtet sich die Geschwindigkeit der gesamten Kette nach dem langsamsten Element. In unserem Fall gibt es eine Diskrepanz zwischen der Eingangs- uns Ausgangsdatenmenge und somit zwischen den Zeiten, die für die Speicherzugriffe benötigt werden. Unser Rechenwerk aus Abbildung 3.16 benötigt 81 Eingangswerte und liefert daraus 9 Ausgangswerte. Das langsamste Element der Pipeline, *von Neumann-Flaschenhals* genannt, ist also das Lesen und Bereitstellen der Eingangswerte. Darauf werden wir später noch genauer eingehen und durch geeignete Anordnung der Werte im Speicher und geschickte Vorgehensweise bei der Berechnung dieses Problem einschränken.
Betrachten wir zunächst die Anordnung der Daten im Speicher. Wie schon im vorigen Abschnitt erwähnt, wird der Speicher in Worten zu je $32bit$ adressiert. Da unsere Werte jedoch nur eine Wortlänge von $7bit$ haben und wir immer Datensätze von je 9 Werten zusammenhängend verarbeiten, bietet es sich an, diese zusammenzufassen. Wir codieren einen Datensatz mit $9 \times 7bit = 63bit$ in zwei Speicherwörtern, die zusammen $64bit$ umfassen. Die einzelnen Werte werden dabei hintereinander gehängt, wobei jedes Bit eines Wertes einem Bit der so entstandenen Doppelwörter entspricht. Abbildung 3.18 zeigt die Anordnung dieser Codierung. Die gestrichelte Linie ist die Trennung zwischen oberem und unterem Speicherwort. Der Wert p_4 liegt auf dieser Grenze, so daß seine unteren vier Bits im unteren Wort und die restlichen drei Bits im nächsten Wort liegen. In Abbildung 3.18 ist unten die Numerierung der Bits innerhalb der Werte und oben deren Stelle

[15] vergleiche *Fließband-Prinzip*

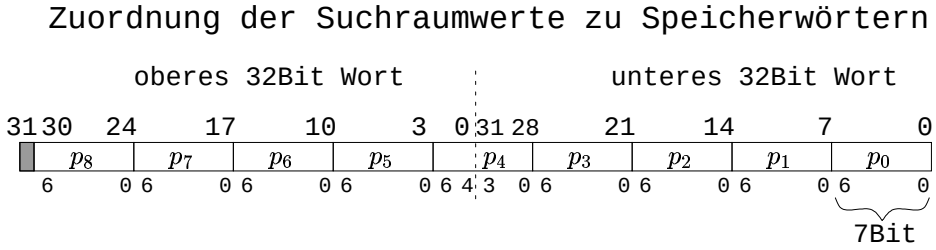


Abbildung 3.18: Codierung im Speicher

im Speicherwort angegeben. Das oberste Bit des oberen Speicherwortes bleibt bei dieser Codierung unbenutzt. Die zugehörige Zuordnung der Werte p_i eines Datensatzes zu den

p_0	p_1	p_2
p_3	p_4	p_5
p_6	p_7	p_8

Abbildung 3.19: Indizierung der Werte im 3×3 Suchraum

Feldern im 3×3 Suchraum ist in Abbildung 3.19 dargestellt. So können in einem Doppelwort die Daten eines kompletten Suchraumes gespeichert werden.
Zu jeder neuen Berechnung eines Datensatzes werden zusätzlich zu dem Datensatz selbst die Datensätze in seiner $8er$ Nachbarschaft benötigt. Um das Rechenwerk mit den nötigen 9 Datensätzen starten zu können, sind also 9×2 Speicherzugriffe erforderlich. Das Speichern des Ergebnisdatensatzes erfolgt dann mit zwei Speicherzugriffen. Da diese Speicherzugriffe seriell nacheinander ausgeführt werden, das Rechenwerk die Eingangswerte aber parallel erfordert, müssen die Datensätze in geeigneter Form zwischengespeichert und in die Einzelwerte decodiert werden. Dazu verwenden wir eine spezielle Anordnung von Registern, die ein Speicherelement bilden, welches seriell beschrieben und parallel ausgelesen werden kann. Dieses Speicherelement nennen wir `SWPRM`[16] und realisieren es wie in Abbildung 3.20 dargestellt. Alle Register sind in einer Kette angeordnet, wobei jeweils der Ausgang eines Registers mit dem Eingang des nächsten verbunden ist. Ist das Signal `CE`[17] aktiv, so werden neue Eingangsdaten `Input` beim Taktsignal `CLK` von Register `Reg[0]` übernommen und gleichzeitig werden die vorigen Werte aller Register `Reg[i]` an das in der Kette folgende `Reg[i+1]` weitergegeben. Da ein Register die Werte am Eingang `D` nur bei aufsteigender Flanke des Taktsignals übernimmt und erst kurz danach am Ausgang `Q` zur Verfügung stellt, liest das folgende Register bei derselben Taktflanke noch die alten Daten. So wandern die seriell eingelesenen Werte in einem Takt um eine Stelle in der Kette weiter. Daten, die am letzten Register angekommen sind, werden beim nächsten Lesetakt nicht weitergegeben und fallen aus dem Speicher heraus. Ist das Signal `CE` nicht aktiv, so verbleiben alle Werte im jeweiligen Register. Die Daten aller

[16] *engl.* Serial Write Parallel Read Memory
[17] *engl.* Clock Enable

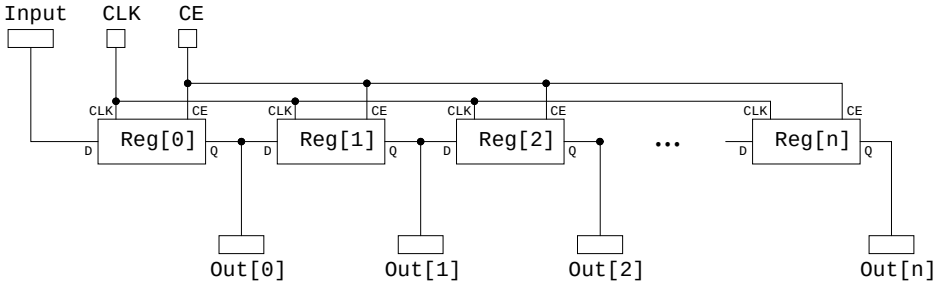


Abbildung 3.20: Serial Write Parallel Read Memory

Register sind jeweils an den Ausgängen `Out[i]` gleichzeitig zum Lesen verfügbar. Die Anzahl der Register in dieser Kette bestimmt die Anzahl der Werte, die aufgenommen werden können und somit auch die Anzahl der Lesetakte, bis die Kette vollständig gefüllt ist und alle parallelen Ausgänge einen Wert liefern. In unserer Anwendung realisieren wir diese Kette mit 18 Registern zu je $32bit$ Wortlänge. So können wir die Datenwörter, die aus dem RAM-Speicher ausgelesen werden, direkt an dieses Speicherelement weiterleiten. Nach der Zeit, die zum Füllen des Speicherelementes benötigt wird, stehen dann alle neun Datensätze zum parallelen Weiterleiten an das Rechenwerk zu Verfügung.
Die Weiterleitung an das Rechenwerk beinhaltet auch die Decodierung der in zwei $32bit$ Registern gespeicherten neun $7bit$ Werten. Damit das Prinzip der Pipeline weiter aufrecht erhalten werden kann, plazieren wir an den Ausgängen des `SWPRM` für jeden darin enthaltenen Wert ein $7bit$ Register, welches nach dem Füllen des gesamten Speicherelementes mit neuen, zur Berechnung bestimmten Daten, einen der 81 Eingangswerte für das Rechenwerk zwischenspeichert. Die Aufteilung der Werte geschieht in Umkehrung des in Abbildung 3.18 vorgestellten Schemas und bildet somit die Decodierung der Doppelwörter. Dieses Zwischenspeichern und Decodieren erfolgt parallel in einem Zeittakt und bildet einen weiteren Schritt der Pipeline. Mit den in den Registern bereitstehenden Werten kann nun das Rechenwerk anfangen zu arbeiten. Gleichzeitig kann das `SWPRM`-Speicherelement mit neuen Werten geladen werden. Die Ergebnisse des Rechenwerkes bilden einen Datensatz aus neun $7bit$ Werten, die wieder in einem Doppelwort nach Abbildung 3.18 codiert werden und anschließend in zwei Speicherzugriffen zurück ins Ram geschrieben werden.
Die gesamte Pipeline besteht also aus fünf Schritten. Die Schritte lassen sich in Lesen, Decodieren, Rechnen, Codieren und Schreiben einteilen. Ein Datenpaket durchläuft diese Kette in fünf gleich langen Zeiteinheiten, die durch das langsamste Glied bestimmt wird. Dabei ist die Nutzung der Ressourcen darauf hin optimiert, daß jedes Element in jedem Zeitschritt der Pipeline aktiv ist. Durch den Leseschritt, welcher im allgemeinen Fall 18 Takte benötigt und damit der langsamste ist, läuft die Pipeline entsprechend in Zeitschritten, die 18 der gemeinsamen Takte `CLK` umfassen. Um die Pipeline zu beschleunigen, kann zum einen die Frequenz des Taktsignals `CLK` erhöht werden, was aber durch die Zugriffsgeschwindigkeit auf das RAM begrenzt ist. Zum anderen läßt sich die Reihenfolge der Berechnungen geschickt wählen, um die Anzahl der Leseoperationen zu verringern.

Letzteres erfolgt durch das *Sliding Window*[18] Prinzip, welches im folgenden Abschnitt erläutert wird.

3.3.3 Sliding Window

Im vorigen Abschnitt stellte sich heraus, daß die Lesezugriffe auf den RAM-Speicher das bremsende Element in der Pipeline sind. Da zu einer Berechnung die benötigten neun Datensätze in jeweils zwei Speicherzugriffen geladen werden, benötigt dieses Glied der Kette 18 Zeittakte. Die Eingangsdaten für das Rechenwerk bestehen aus dem Datensatz zu einem Ausgangspixel und den Datensätzen zu dessen acht direkten Nachbarpixeln. Betrachten wir nun ein benachbartes Pixel in horizontaler oder vertikaler Nachbarschaft zum Ausgangspixel und wiederum dessen acht Nachbarn, so stellen wir fest, daß sich diese beiden Pixelmengen überschneiden. Führen wir die Berechnungen für diese beiden Pixel nacheinander durch, so wird auch die benötigte Datenmenge sich nur teilweise ändern. Dies nutzen wir aus, um die Anzahl der Lesezugriffe zu reduzieren. Nach der Berechnung zu einem Ausgangspixel gehen wir der Anordnung im Bild entsprechend einen Schritt nach rechts und führen die nächste Berechnung für dieses Pixel aus. Dann können $2/3$ der Datensätze im Zwischenspeicher verbleiben, und nur die Datensätze zu den drei rechten Pixeln der $8er$ Nachbarschaft müssen dazu geladen werden. Die Datenmenge wird also durch ein Fenster betrachtet, in dem nur der zur Zeit interessante Bereich der Daten zu sehen ist. Ein Verschieben des Fensters um einen Schritt nach rechts verursacht, daß eine neue Spalte von Daten hinzukommt, eine Spalte alter Daten wegfällt und intern alle Daten in ihrer Position eine Spalte weiterrücken. Dieses Verfahren wird *Sliding Window* genannt und auch von R. Männer [MMMH01] angewendet. Es ermöglicht bei geeignetem Zwischenspeicher die Reduzierung der Leseanforderungen.
Im Zwischenspeicher `SWPRM` wandern beim Hinzufügen neuer Daten alle alten Daten in einer Kette weiter nach hinten. Mit ihrer Position im Zwischenspeicher ändert sich auch deren Bedeutung nach der Decodierung und damit die für das Rechenwerk. Bei einer Reihenfolge der Datensätze im Zwischenspeicher nach Abbildung 3.21 bedeutet dies, daß bei drei neuen Datensätzen, die einer Spalte entsprechen, die alten Datensätze um jeweils eine Spalte nach hinten rücken. Das Laden drei neuer Datensätze entspricht dem Verschieben des Bereiches der betrachteten Daten um einen Schritt nach rechts. Ist der Zwischenspeicher einmal mit Daten gefüllt, so kann die Anzahl der Lesezugriffe neuer Datensätze zur nächsten Berechnung von neun auf drei reduziert werden. Zu Beginn jeder neuen Bildzeile muß allerdings der Zwischenspeicher komplett neu beschrieben werden, da das Datenfenster in diesem Fall keinen Schritt in die ausgewählte Richtung der $4er$ Nachbarschaft macht, was Voraussetzung für diese Vereinfachung war. Durch diese Vorgehensweise läßt sich also die Geschwindigkeit der Pipeline verdreifachen, wobei nach jedem Zeilenwechsel zwei zusätzliche Wartezyklen eingelegt werden müssen. Zwar ist so immer noch das Anfordern der Daten das bremsende Element der Pipeline, aber das Verhältnis von Lesezeit zu Schreibzeit kann von $9:1$ auf $3:1$ verbessert werden.

[18] *engl.* sliding = gleiten, window = Fenster

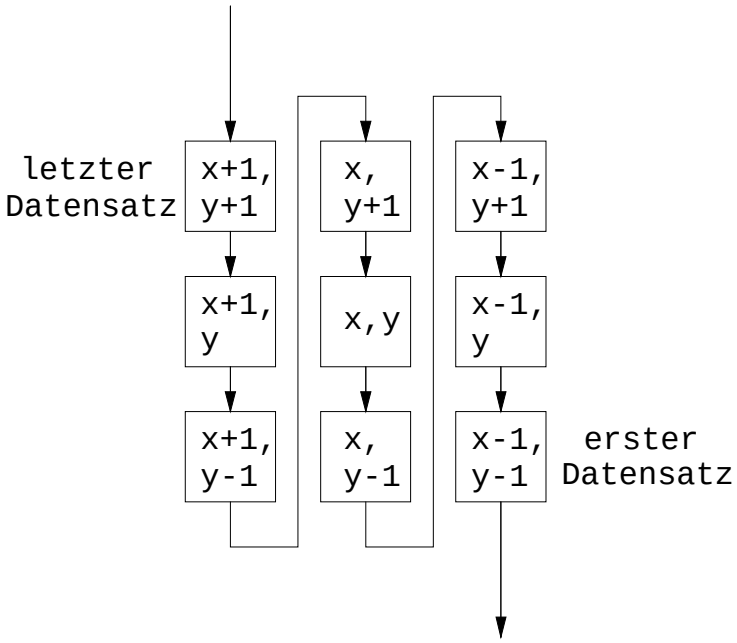


Abbildung 3.21: Reihenfolge der Datensätze zu den angegebenen Pixelpositionen im Zwischenspeicher SWPRM

3.3.4 Adressierung

Die Auswahl der Adressen zum Lesen und Schreiben muß der Position der Datensätze im Speicher entsprechen. Gehen wir davon aus, daß unteres und oberes Teilwort eines Datensatzes aufeinander folgend im Speicher abgelegt sind. Weiter sind die Datensätze den zugehörigen Bildpunkten entsprechend zeilenweise nacheinander angeordnet. Durch Kenntnis der Startadresse S_0 und der Zeilenlänge Z können wir die Anfangsadresse zu jedem Datensatz durch die Gleichung (3.17) bestimmen.

$$adr(x, y) = S_0 + (x + Z * y) * 2 \tag{3.17}$$

Der Datensatz zu dem Bildpunkt an Position (x, y) befindet sich dann in der Speicherstelle an Adresse $adr(x, y)$ und an der darauf folgenden, da ein Datensatz codiert in einem Doppelwort nach Abbildung 3.18 ja immer zwei Speicherstellen umfaßt. Im vorigen Abschnitt haben wir uns auf eine Reihenfolge der Einzelberechnungen festgelegt. Nun geht es darum, die Daten in dieser Reihenfolge aus dem Speicher auszulesen und die Ergebnisse abzuspeichern. Die Adressierung kann durch die fortlaufende Anordnung der Daten entsprechend der Verarbeitungsreihenfolge ebenso fortlaufend erfolgen. Dazu verwenden wir ein Zählwerk, welches initialisiert mit der Startadresse die Leseadresse bei jedem Zugriff um einen Schritt erhöht. Da die benötigten Daten einer Spalte aus drei aufeinander folgenden Zeilen stammen, verwenden wir drei Zählelemente, die nacheinander angesteuert werden. Die Anordnung der Zählelemente und die nötige Steuerung ist in Abbildung 3.22 dargestellt.
Eingangsdaten sind die Startadressen S0,S1,S2 der ersten drei Zeilen, die sich mit Glei-

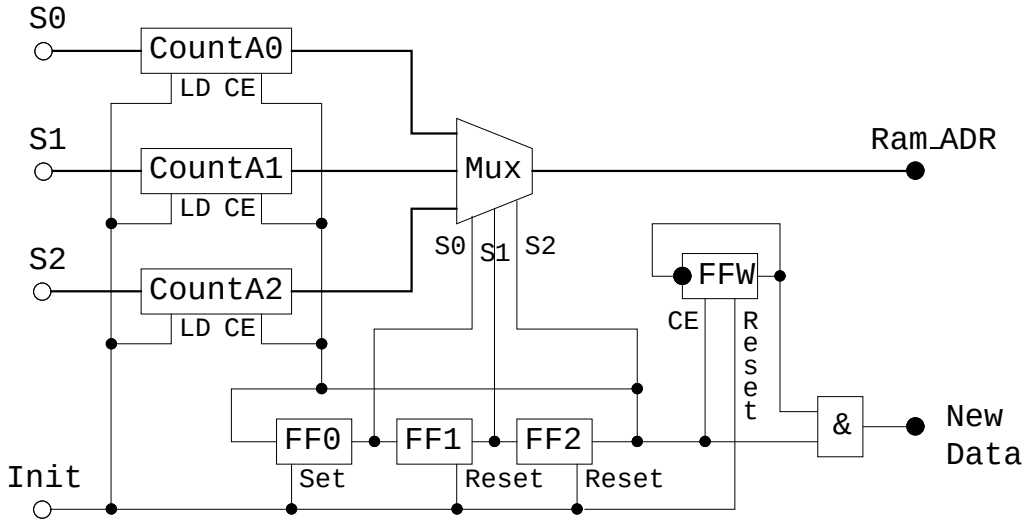


Abbildung 3.22: Auswahl der Leseadressen

chung (3.17) ermitteln lassen und ein Signal `Init` zur Initialisierung des Zählers und deren Steuerung. Bei der Initialisierung werden die Zähler `Count` und die Flipflops[19] `FF` in den Ausgangszustand versetzt. Dabei werden die Startadressen in die jeweiligen Zähler geladen und durch das Signal `Set` eine logische 1 an den Anfang der Flipflop-Kette gesetzt. Alle diese Elemente sind zusätzlich an den Haupttakt `CLK` angeschlossen, welcher hier nicht mit dargestellt ist. Die Flipflops `FF0`, `FF1` und `FF2` sind in einem Ring verbunden, so daß die initiale 1 bei jedem Takt einen Schritt weiter und vom Ende wieder an den Anfang gegeben wird. Der jeweilige Ort der 1 bestimmt die Auswahl des Multiplexers `Mux`, der einen der Zählerstände auswählt und diesen an dem Ausgang `Ram_ADR` bereitstellt, welcher als Leseadresse an das RAM weitergeleitet wird. Bei jedem Durchlauf der Kette werden die Zähler durch deren Signal `CE` einmal aktiviert und laufen einen Schitt in der Adressierung weiter. So wird nacheinander aus drei aufeinanderfolgenden Zeilen jeweils ein Wort gelesen und im Anschluß eine Spalte weiter gezählt. Da in jeder Zeile zwei aufeinanderfolgende Leseoperationen nötig sind, um einen kompletten Datensatz zu erfassen, muß diese Kette zweimal durchlaufen werden. Das Flipflop `FFW` wechselt durch die Negation seines Zustands am Eingang bei jedem Durchlauf durch das Signal `CE` seinen Zustand. Durch die folgende *UND*-Verknüpfung des Zustandes und das Signal zum Wechseln wird bei jedem zweiten Durchlauf der Kette ein Impuls erzeugt, der einen Takt andauert. So zeigt das Signal `New Data` nach außen an, wann in jeder Zeile der aus zwei Speicherwörtern bestehende Datensatz komplett adressiert wurde. Dieses Signal gibt den Arbeitstakt der Pipeline an und steuert somit die Decodierung der gelesenen Datensätze, deren Verarbeitung im Rechenwerk und die anschließende Codierung und Abspeicherung der Ergebnisse. Diese Schaltung fassen wir zusammen und verwenden sie unter dem Namen `ReadAR`.

[19] Flipflop: Zustandsspeicher, übernimmt einen booleschen Wert bei aufsteigender Taktflanke und aktivem Signal `CE` und stellt diesen dauerhaft am Ausgang zur Verfügung. Zusätzliche Signale: `Set` zum Setzen und `Reset` zum Zurücksetzen

Die Auswahl der Adressen zum Schreiben der Ergebnisse erfolgt ebenso mit einem ladbaren Zähler. Dies ist aber nicht so aufwendig, da nur ein einzelner Datensatz geschrieben werden muß. Abbildung 3.23 zeigt diesen Zähler und die nötigen Steuerung. Hier wird

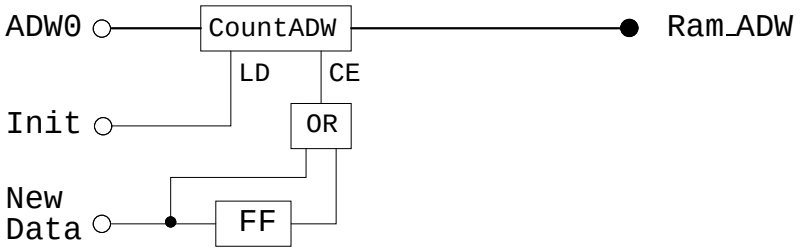


Abbildung 3.23: Auswahl der Schreibadresse

neben der Startadresse `ADW0` und dem Signal `Init` zur Initialisierung das vom Element `ReadADR` generierte Signal `New Data` verwendet, um einen neuen Datensatz zu adressieren. Dazu wird das Signal in einem Flipflop `FF` übernommen und dort gespeichert. Da das Flipflop einen Zeittakt braucht, um das Eingangssignal zu übernehmen und am Ausgang bereitzustellen, wird das einen Takt andauernde Signal `New Data` am Ausgang um einen Takt verzögert angezeigt. Werden nun originales und verzögertes Signal mit einer *ODER*-Operation verknüpft, so dauert das daraus erzeugte Signal zwei Takte an und wird zum Ansteuern des Zählers benutzt. Auf diese Weise bekommt der Zähler zwei Impulse zum Generieren der nächsten beiden Adressen eines Datensatzes, wenn auch die Lesezähler jeweils einen neuen Datensatz adressiert haben.
Zur vollständigen Steuerung der Pipeline muß nun noch entschieden werden, welche Ausgangsdaten des Rechenwerkes gespeichert werden, oder aufgrund eines Zeilenwechsels gemäß des *Sliding Window*-Prinzips nicht nutzbar waren und daher zwei Wartetakte beim Speichern eingelegt werden müssen. Ebenso muß die Unterscheidung der Iterationsschritte und das Stoppen nach der vorbestimmten Anzahl realisiert werden. Zum Erkennen des Zeilenwechsels wird ein weiterer Zähler `CountdnX` verwendet, der anfangs mit dem Wert der Zeilenlänge aus Register `RegPX` geladen wird und von dort aus herunter zählt. Wieder wird das Eingangssignal `New Data` als Zählimpuls verwendet. Ist der Zähler bei 0 angekommen, so wird dieser zum einen neu initialisiert und zum anderen das sonst aktive Ausgangssignal `Valid` für zwei Takte unterbrochen. Zur Unterbrechung wird das Signal des Zeilenendes in einem Flipflop gespeichert und somit um einen Takt verzögert. Dann wird auf das Signal selbst und das verzögerte die `NOR` Operation angewendet, was bedeutet, daß keiner der beiden aktiv sein darf, um einen positiven Ausgangswert zu erhalten. So wird das `Valid`-Signal für den Takt des Zeilenendes und den darauf folgenden Takt ausgesetzt. Zusätzlich dazu verwenden wir noch einen weiteren Zähler `CountdnY`, um die Anzahl der Zeilen zu kontrollieren. Dieser Zähler wird mit dem Wert der Anzahl der Zeilen aus Register `RegPY` initialisiert und bei jedem Zeilenwechsel einen Schritt herunter gezählt. Dessen Ausgang wird mit 0 verglichen und, ist dieser Fall eingetreten, so wird dieser neu initialisiert und das Ausgangssignal `PageSwitch` generiert. Dieses Signal gibt an, daß ein Iterationsschritt für alle Datensätze beendet ist. Es dient einerseits als `Init`-Signal der beiden Schaltungen zur Adressauswahl aus Abbildung 3.22 und 3.23

und wird weiter zur Steuerung im nächsten Abschnitt verwendet. Abbildung 3.24 zeigt diese beiden Kontrollmechanismen zur übergeordneten Steuerung.

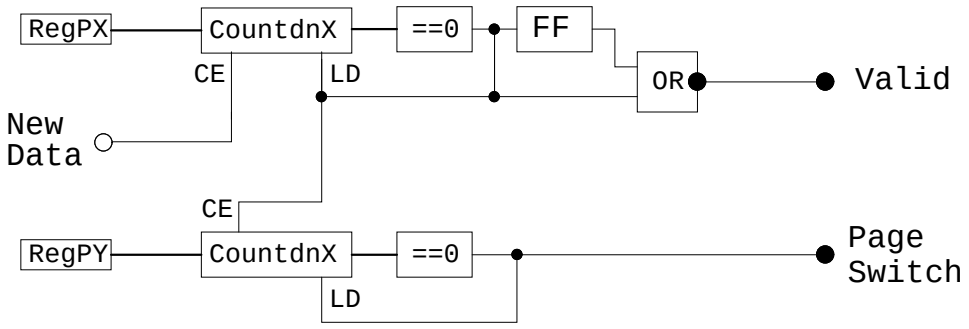


Abbildung 3.24: Steuerung der Zeilen- und Seitenwechsel

3.3.5 Iteration

Ist ein Iterationsschritt für alle Datensätze vollständig abgearbeitet, so kann mit dem nächsten Iterationsschritt begonnen werden. Dazu müssen die zuvor gewonnenen und separat abgespeicherten Ergebnisse ihre Bedeutung ändern und nun wiederum zu den Eingangsdaten der folgenden Berechnung werden. Um dieses zu verwalten, unterteilen wir den zur Verfügung stehenden Ram-Speicher in zwei Bereiche[20]. Einer der Bereiche dient als Eingang und der andere als Ausgang. Um nun die Adressierung für die Lesesoperation und die der Schreiboperation entsprechend zu wählen, verwenden wir Multiplexer, die es ermöglichen, die Startadressen in Abhängigkeit des Signals `PageSwitch` zu vertauschen. Das Signal `PageSwitch` wird benutzt, um eine Zustandsänderung eines Flipflops zu veranlassen. In Registern sind die nötigen Startadressen beider Speicherbereiche für die Lese- und Schreiboperationen bereitgestellt. Multiplexer `Mux` wählen in Abhängigkeit vom Zustand des Flipflops jeweils eine der beiden Startadressen aus und leiten diese an die zuvor vorgestellten Adressierungsschaltungen aus Abbildung 3.22 und 3.23 zu deren Initialisierung weiter. So wird bei gerader Anzahl von bereits durchgeführten Iterationsschritten der eine Speicherbereich und bei ungerader Anzahl der andere Speicherbereich zum Lesen ausgewählt. Die Auswahl der Startadresse zum Schreiben erfolgt jeweils genau entgegengesetzt. Hier wird allerdings erst in der zweiten Zeile begonnen, da genau wie bei den Randbereichen beim Zeilenwechsel diese Positionen aufgrund der unvollständigen Nachbarschaft nicht neu berechnet werden können. Die Anzahl der durchzuführenden Iterationsschritte ist in dem Wert des Registers `RegPS` festgehalten. Dieser wird beim Initialisieren ebenfalls in einen Zähler `CountdnS` geladen und bei jedem Iterationsschritt, angezeigt durch das Signal `PageSwitch`, herabgezählt. Abbildung 3.26 zeigt die beschriebene Schaltung, die von außen das Startsignal `Start` bekommt, welches in dem Flipflop `FFC` mittels `Set`-Operation dauerhaft gespeichert wird und dann als aktives Signal `RUN` nach außen weitergegeben wird. Ist die vorbestimmte Anzahl von

[20] vergleiche engl. page = Seite

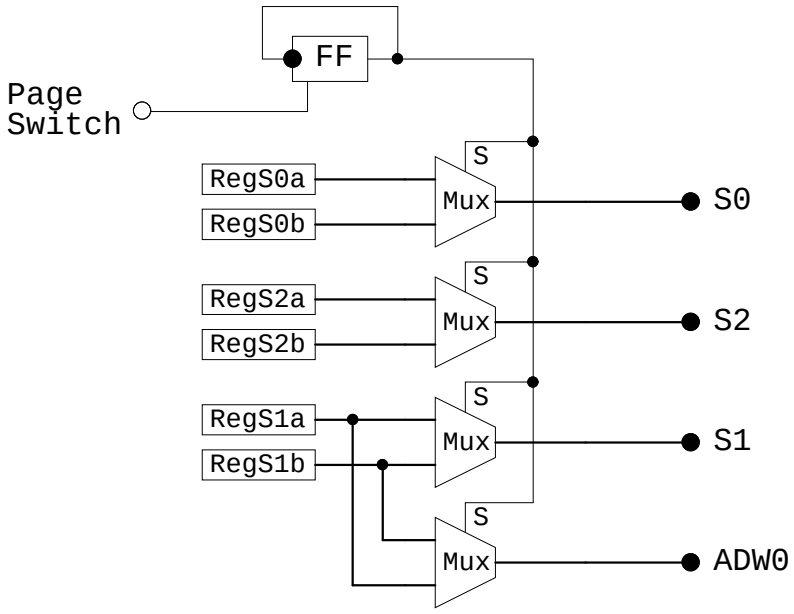


Abbildung 3.25: Auswahl der Startadressen

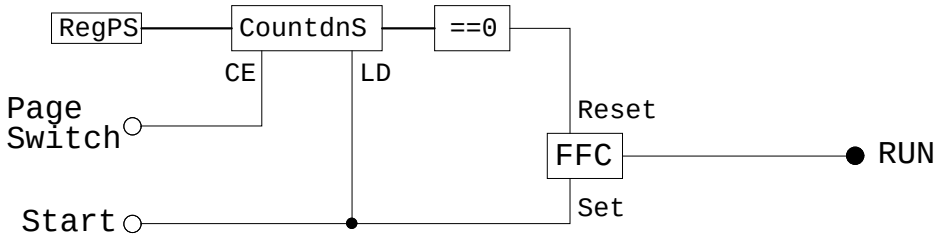


Abbildung 3.26: Steuerung der Anzahl der Iterationsschritte

Iterationen erreicht, so wird dieses Signal zurückgesetzt und der Iterationsprozeß wird angehalten. Alle taktgesteuerten Elemente der Schaltung zur Steuerung der Zeilen- und Spaltenwechsel aus Abbildung 3.24 und der Schaltungen zur Adressauswahl aus den Abbildungen 3.22, 3.23 und 3.25 werden nur bei aktivem RUN-Signal mit dem Haupttakt getaktet. Dies ermöglicht eine übergeordnete Kontrolle für den gesamten Iterationsprozeß, welche durch die Schaltung in Abbildung 3.26 realisiert wird.
Alle in diesem Abschnitt 3.3 vorgestellten Schaltungen werden entsprechend ihrer angegebenen Eingangs- und Ausgangssignale, gekennzeichnet durch einen einfachen bzw. einen gefüllten Kreis, verbunden und bilden zusammengesetzt die *Control Unit*, welche die zentrale Steuerung übernimmt, indem sie den Datenfluß durch die Pipeline steuert und die gewählte Anzahl von Iterationen durchführt.

3.4 Gesamte Architektur

In den vorangegangenen beiden Abschnitten wurde im einzelnen die Realisierung der Rechenschritte und mit deren Verbindung das Rechenwerk sowie die Kontrolle des Datenflusses, zusammengefaßt in der zentralen Steuerung, erläutert. Daraus ergibt sich nun die gesamte Architektur, die nach der Initialisierung selbständig die gewünschten Operationen ausführt und die Ergebnisse im RAM bereitstellt. Abbildung 3.27 zeigt die zusammengefaßten Elemente der zentralen Steuerung und des Rechenwerkes, sowie deren Position in der Pipeline. Weitere Elemente der Pipeline sind neben dem Seriell-Parallelwandler die Codierung und Decodierung der Datensätze. Die zentrale Steuerung beginnt mit der Adressierung der Datensätze, die vom RAM angefordert und dann in die Pipeline weiter gegeben werden. Die Pipeline nimmt diese in den Seriell-Parallel-Wandler auf und gibt sie, sobald genügend neue Daten bereitstehen, weiter an die Decodierung. Die Decodierung teilt die Datensätze in die Einzelwerte auf und gibt diese weiter an das Rechenwerk. Das Rechenwerk liefert daraus die neu iterierten Einträge eines Suchraumes, welche anschließend wieder zu einem Datensatz codiert werden. Dieser Datensatz wird in zwei Speicherwörtern, adressiert durch die zentrale Steuerung, in dem zweiten Speicherbereich abgespeichert. Dieser Kreislauf wird ausgeführt, bis die angegebene Anzahl von Iterationen erreicht ist.
Wird diese Schaltung mit einer Frequenz von $20\mathsf{MHz}$ betrieben, so ergibt sich daraus die Länge eines Taktes durch den Kehrwert zu $50\ ns$ [21]. Da zu jeder Berechnung nach dem Sliding-Window-Prinzip drei neue Datensätze, bestehend aus jeweils zwei RAM-Wörtern, benötigt werden, läuft die Pipeline mit einer Geschwindigkeit, die sich aus $3 \times 2 = 6$ Takten ergibt. In jedem Schritt der Pipeline werden die Daten zugehörig zu einem Ausgangsbildpunkt iteriert. Haben die zugrundeliegenden Bilder eine Dimension von 256×256 Pixeln, so dauert ein Iterationsschritt aller Pixel $6 \times 256^2 = 393216$ Takte, was ca. $20\ ms$ [22] entspricht. In der Praxis wird eine Anzahl von bis zu 15 Iterationsschritten verwendet, was dann ca. $0,3$ Sekunden benötigt. Dies ermöglicht es also, abgesehen vom nötigen Datentransfer, auf den im folgenden Abschnitt eingegangen wird

[21] Einheit [ns]=[Nanosekunden]: $1\ ns = 10^{-9}\ Sekunden$
[22] Einheit [ms]=[Millisekunden]: $1\ ms = 10^{-3}\ Sekunden$

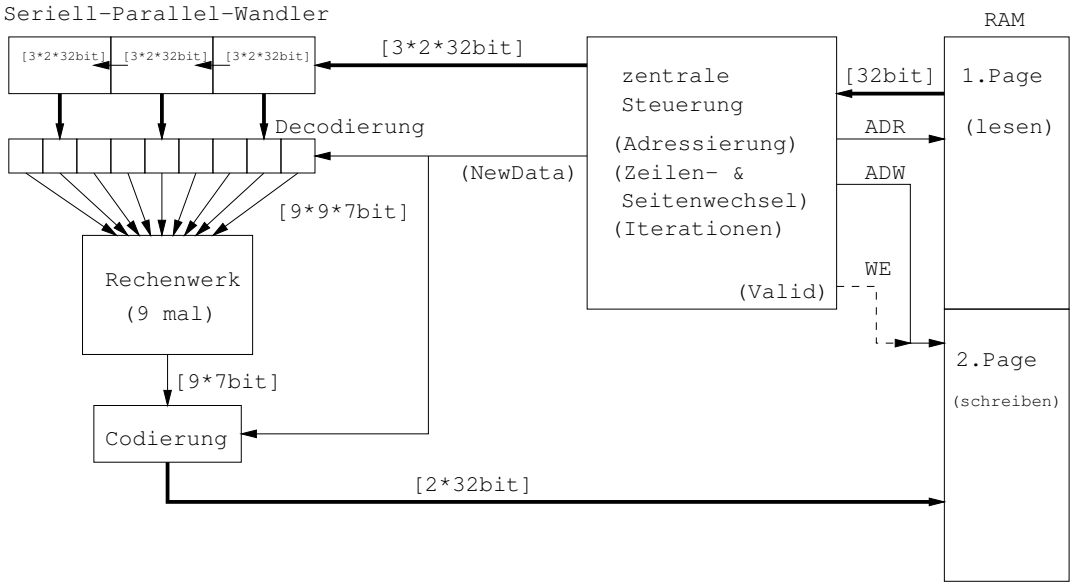


Abbildung 3.27: Gesamt-Architektur

und der Vor- und der Nachverarbeitung, die auf dem Hauptrechner durchgeführt wird, bis zu drei Bilder der angegebenen Größe in der Sekunde zu verarbeiten.
Einzige Schnittstelle nach außen ist die Initialisierung der Register mit den Startadressen und den Bilddimensionen wie im Abschnitt 3.3 beschrieben, die Bereitstellung der aus der Vorverarbeitung gewonnenen initialen Datensätze im RAM und das abschließende Auslesen der iterierten Wahrscheinlichkeitsverteilung. Das Bereitstellen und Auslesen der Daten im RAM wird durch eine Verbindung über den PCI-Bus zum Hauptrechner realisiert. Der Aufbau dieser Verbindung und die nötige Steuerung wird im folgenden Abschnitt erläutert.

3.5 Datentransfer über den PCI-Bus

Einzige Verbindung zwischen Hauptrechner und FPGA ist der PCI-Bus. Diese Schnittstelle läßt sich direkt vom Hauptrechner ansprechen, wobei zusätzlich zu einer Adresse $32bit$ Datenwörter gelesen und geschrieben werden können. Im FPGA stehen die Adresse und die Daten als Signal zusätzlich zu einer Flußkontrolle zur Verfügung.
Das Hauptproblem besteht darin, daß PCI-Bus und FPGA-Design unterschiedlich getaktet werden. Während der PCI-Bus vom Hauptrechner betrieben wird, bildet das FPGA-Design eine dazu unabhängige Komponente. Der Takt des PCI-Bus ist zwar auch innerhalb des FPGA verfügbar, doch wird für das Design im allgemeinen ein schnellerer intern selbstgenerierter Takt verwendet, der auch bei eventuell gleicher Frequenz nicht phasensynchron zum PCI-Takt ist. Da bei den verwendeten getakteten Elementen wie Register und Zähler die Daten und Steuersignale synchron zum Taktsignal anliegen müssen, implementieren wir eine Schnittstelle, die einen Übergang zwischen den beiden Takten bildet.

Dazu verwenden wir spezielle Zwischenspeicher, genannt FIFO[23], die zwischen Lese- und Schreibtakt unterscheiden können und die aufgenommenen Daten bis zum Auslesen in einer Kette speichern, und zusätzlich eine Flußkontrolle für größere Datenmengen bieten. Weiter verwenden wir zum Übernehmen einzelner Daten spezielle Register, die ebenfalls zwischen Lese- und Schreibtakt unterscheiden können, um Daten vom PCI-Bus aufzunehmen und phasensynchron zum FPGA-Takt bereitzustellen. Es soll damit die Möglichkeit geschaffen werden, große zusammenhängende Datenmengen zwischen PCI-Bus und FPGA-RAM zu transportieren.
Abbildung 3.28 zeigt eine etwas vereinfachte Darstellung der Schaltung zum Datentransfer zwischen PCI-Bus und FPGA-RAM. Die vom PCI-Bus übermittelte Adresse `PCI_A` wird verwendet, um zwischen Initialisierung der Verwaltung der RAM-Adressen und den eigentlichen Lese- und Schreiboperationen zu unterscheiden. In Abhängigkeit dieser Adresse werden die vom PCI-Bus bereitgestellten Daten `PCI_DI` zur Initialisierung verwendet, oder als zu speichernde Daten interpretiert und an das `FIFO_in` weitergegeben. Die Initialisierung besteht aus dem Laden der Startadressen für die RAM-Zugriffe in Registern. Dabei wird zwischen der Adresse für Lesezugriffe und der für Schreibzugriffe unterschieden. Hat die angelegte PCI-Adresse den Wert 2, so werden die bereitgestellten Daten als Startadresse für Schreibzugriffe verwendet und in das Register `RegAW` geladen. Bei einem Wert der PCI-Adresse von 3 werden die Daten in das entsprechende Register `RegAR` geladen und als Startadresse für Lesezugriffe interpretiert. Diese beiden Register sind so genannte *Slow-Fast*-Register[24], die in der Lage sind, Daten mit dem PCI-Takt aufzunehmen und dann ein zum schnelleren FPGA-Takt synchrones Signal zum Auslesen zu erzeugen. Dieses Signal, welches in der Abbildung 3.28 nicht mit dargestellt ist, veranlaßt die nachfolgenden Zähler, sich mit den neuen Daten zu initialisieren.
Betrachten wir zunächst den Fall, in dem Daten vom PCI-Bus zum FPGA-RAM transportiert werden sollen. Ist der Zähler der Schreibadresse `CountAW` initialisiert, so kann mit dem Schreiben der Daten begonnen werden. Diese werden bei der PCI-Adresse mit dem Wert 0 von dem Signal `PCI_DI` in das Eingangs-FIFO `FIFO_in` übernommen. Ist das FIFO dann nicht mehr leer, so wird das zuvor aktive Signal `EMPTY` zurückgesetzt und durch dessen Negierung das Ausgangssignal `RAM_WE` aktiv. Das bedeutet, daß das angeschlossene RAM-Element die Daten durch die in Abschnitt 3.3.1 vorgestellte Schnittstelle aus dem FIFO an die durch den Zähler angegebene Adresse übernimmt. Im selben Takt, wie das RAM die Daten aus dem FIFO liest, wird die Adresse für die nächsten Daten erhöht, so daß die Daten in der Reihenfolge im RAM abgelegt werden, in der sie vom PCI-Bus geliefert werden. Dabei werden Datenwörter mit $32bit$ verwendet, was der Wortbreite des PCI-Bus entspricht. Da vom PCI-Bus eventuell durch Verzögerungen auf dem Hauptrechner und durch den langsameren Takt nicht zu jedem FPGA-Takt neue Daten geliefert werden, wird das Signal `WE` und die zu den Daten zugehörige RAM-Adresse nur generiert, wenn das FIFO nicht leer ist. Dies stellt eine Flußkontrolle dar, die bei unterschiedlicher Taktung und eventuellen Verzögerungen seitens des Hauptrechners unerläßlich ist.
Betrachten wir nun die umgekehrte Richtung, in der eine zusammenhängende Datenmenge aus dem FPGA-RAM gelesen werden soll. Wie zuvor erläutert, wird die Startadresse

[23] FiFo: engl. First In First Out
[24] Basiselement zu finden in KCC-Bibliothek [PGR01]

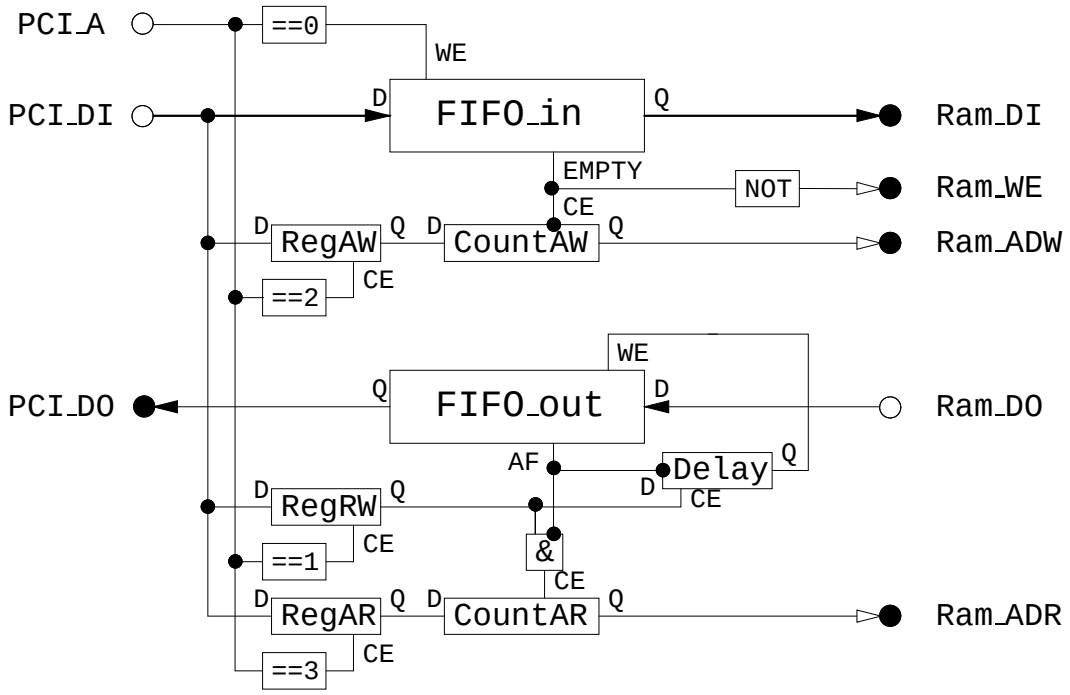


Abbildung 3.28: Verbindung zwischen PCI-Bus und RAM

über das Register RegAR in den Zähler CountAR geladen. Um nun erst nach Beendigung dieser Initialisierung mit dem Füllen des FIFOs zu beginnen, wird zusätzlich der boolesche Wert des Registers RegRW abgefragt. Dieser wird ebenfalls über den PCI-Bus gesetzt und gibt an, ob sich die Schaltung im Lesemodus befindet. Also sollte erst nach der Angabe der Startadresse dieser Wert auf $true$ gesetzt werden. Damit wird das Weiterzählen der RAM-Adresse ermöglicht und die Flußkontrolle gestartet. Die Flußkontrolle ist in diesem Fall etwas aufwendiger, da je nach Implementierung der RAM-Schnittstelle nach Anlegen der Leseadresse RAM_ADR ein oder mehrere Wartetakte eingelegt werden müssen, bis die zugehörigen Daten vom Ram geliefert werden. Ist das Ausgangs-FIFO FIFO_out aufnahmebereit für weitere Daten, was durch die Negation des Signals AF (*engl. Almost Full*) angezeigt wird, so wird eine neue Leseadresse generiert. Das negierte Signal AF wird durch das Element Delay verzögert und dann dazu benutzt, um die Daten des Signals RAM_DO in das FIFO aufzunehmen. So wird versucht, das FIFO immer nahezu mit Daten gefüllt zu halten, wobei die aufsteigend adressierten Daten immer erst nach der Verzögerung übernommen werden. Vom PCI-Bus aus können nun die Daten gemäß ihrer Reihenfolge im Speicher über den Anschluß PCI_DO ausgelesen werden. Da der PCI-Bus langsamer getaktet ist als das FPGA-Design, wird das FIFO immer genügend Daten enthalten, so daß hier keine weitere Flußkontrolle nötig ist. Nach dem Lesen einer gewünschten Datenmenge ist das Register RegRW über die Adresse mit dem Wert 1 wieder auf den booleschen Wert $false$ zu setzen, damit der Lesezähler nicht weiter zählt und das FIFO keine neuen Daten vom RAM übernimmt. Zu Beginn des Lesens einer neuen Datenmenge ist darauf zu achten, daß zunächst das FIFO leergelesen werden muß, da es vom vorangegangenen Lesezyklus noch mit eventuell überflüssigen

Daten gefüllt wurde. Dazu werden so lange Daten aus dem FIFO gelesen, wie dort welche verfügbar sind, was durch die im PCI-Bus integrierte Flußkontrolle angezeigt wird. Dieses Leerlesen erfolgt am besten bei durch den Wert *false* in Register `RegRW` abgeschalteter Flußkontrolle, da sonst bei Beginn eines neuen Lesezyklusses nicht zwischen alten, noch im FIFO befindlichen Daten und neu angeforderten Daten unterschieden werden kann.
Diese Schaltung ermöglicht es also, zusammenhängend adressierte Datenmengen über den PCI-Bus zwischen Hauptrechner und FPGA-RAM zu transportieren. Wir verwenden die Schaltung aus Abbildung 3.28, um die zu iterierenden Datensätze im RAM bereitzustellen und nach der Iteration durch die Architektur aus Abbildung 3.27 die Ergebnisse aus dem RAM auszulesen. Beim Bereitstellen der Datensätze ist darauf zu achten, daß zunächst beide später verwendeten Speicherbereiche mit initialen Daten zu füllen sind, da zumindest die Datensätze der Randpixel intern nicht neu berechnet werden, aber in jedem Iterationsschritt benötigt werden. Die initiale Wahrscheinlichkeitsverteilung der Korrespondenzen, gewonnen aus der Vorverarbeitung (erläutert in Abschnitt 3.1) ist vor dem Übertragen in den FPGA in Datensätze nach Abbildung 3.18 zu kodieren und in RAM-Wörter aufzuteilen. Das Auslesen der Ergebnisse hat aus dem Speicherbereich zu erfolgen, der durch die Iteration zuletzt beschrieben wurde. Dieser läßt sich durch die Anzahl der Iterationen ermitteln. Da nach jeder Iteration die Speicherbereiche vertauscht werden, muß bei gerader Anzahl von Iterationen der erste Bereich und bei ungerader Anzahl der zweite Bereich zum Auslesen ausgewählt werden. Zur weiteren Verwendung der RAM-Daten als Werte der Wahrscheinlichkeitsverteilung von Korrespondenzen sind diese nach Abbildung 3.18 zu dekodieren.

3.6 Nachverarbeitung

Die aus der Iteration gewonnene Wahrscheinlichkeitsverteilung der Korrespondenzen von Bildpunkten aus Bild A zu Bildpunkten im zugehörigen Suchraum in Bild B beinhaltet zu jedem untersuchten Paar einen Wahrscheinlichkeitswert für die Korrespondenz dieser beiden Bildpunkte. Um diese Information weiter zu verarbeiten oder in geeigneter Form darzustellen, empfiehlt es sich, die Werte zu normalisieren. Während der Verarbeitung im FPGA wurde bereits nach jedem Iterationsschritt eine Normalisierung durchgeführt. Dabei wurde, um den Wertebereich konstant zu halten, innerhalb eines Suchraumes auf den Maximalwert 1 normalisiert. Abschließend soll nun zum Ermitteln des Optischen Flusses die Aussage getroffen werden, wo ein Bildpunkt x_A aus Bild A im folgenden Bild B wiederzufinden ist. Diese Aussage wird in einem Verschiebungsvektor ausgehend von der Position x_A angegeben. Dazu werden zunächst die Wahrscheinlichkeitswerte der Pixelkorrespondenzen in dem Suchraum zu x_A so normalisiert, daß deren Summe 1 ergibt. Es wird also, wie in Kapitel 2 als Bedingung gestellt, davon ausgegangen, daß zu jedem Bildpunkt aus Bild A ein Korrespondenzpartner in dem entsprechenden Suchraum in Bild B existiert. Zu erwarten ist, daß die Wahrscheinlichkeitsverteilung innerhalb des Suchraumes ein eindeutiges Maximum beinhaltet und weiter, daß alle anderen Werte Null sind. Durch Rechenungenauigkeiten beim Iterieren, aber auch durch die Ab-

bildung der Szene durch Kameras, die Rauschen beinhalten, und durch die Möglichkeit, daß ein Merkmal der Szene in Bild A genau auf einem Bildpunkt, in Bild B aber eventuell auf der Grenze zwischen zwei Bildpunkten abgebildet wird, betrachten wir nicht nur das Maximum, sondern bilden den Erwartungsvektor. Der Erwartungsvektor setzt sich in diesem Fall zusammen aus den Abstandsvektoren vom Ausgangspixel zu allen untersuchten Korrespondenzpartnern, gewichtet mit den normierten Wahrscheinlichkeitswerten der jeweiligen Korrespondenz. Das Aufsummieren dieser gewichteten Abstandvektoren ergibt einen Verschiebungsvektor, den wir zur Darstellung des optischen Flusses verwenden. Dieser Verschiebungsvektor zeigt dann von dem Ausgangspixel auf seinen als wahrscheinlichsten ermittelten Ort im folgenden Bild, welcher nicht unbedingt mit den diskreten Positionen der Bildpunkte übereinstimmen muß. Sind zur Weiterverarbeitung Korrespondenzen mit diskreten Pixelpositionen erforderlich, so ist nicht die gewichtete Summe der Abstandsvektoren zu bilden, sondern die Pixelposition auszuwählen, zu der die Korrespondenz-Wahrscheinlichkeit am größten ist. Ist kein eindeutiges Supremum zu bestimmen, so ist keine exakte Aussage möglich. Dies kann in großen homogenen Bildbereichen und bei zu niedrig gewählter Anzahl von Iterationen der Fall sein.

Kapitel 4

Experimente

Im vorigen Kapitel wurde ein Schaltungsdesign entwickelt, welches nun in Experimenten mit verschiedenen Bildern angewendet und auf seine Funktion hin untersucht werden soll. Wir verwenden zunächst synthetische Bilder, um die grundlegende Funktion zu kontrollieren und darzustellen. Weiter wird die Auswirkung einzelner Besonderheiten durch manuelle Veränderungen von realen Bildern untersucht. Dabei wird das Verhalten bei Positionierung der Korrespondenzpartner auf Subpixelpositionen untersucht und die Effekte von Rauschen simuliert. Danach wird das Verfahren auf Bilder einer Standardsequenz zur Ermittlung des optischen Flusses angewendet und mit realen Bildern getestet.
Bei den Versuchen soll ermittelt werden, welche Möglichkeiten die vorgestellte Implementierung bietet und welche Geschwindigkeitsvorteile gegenüber konventionell arbeitender Hard- und Software erzielt werden können, aber auch, welche Einschränkungen sich aus dieser speziellen Implementierung ergeben.

4.1 Synthetische Bilder

In diesem Abschnitt wird das Verhalten des Rechenwerkes auf dessen Grundfunktionen hin untersucht. Dazu werden konstruierte Bilder mit bekannten Korrespondenzen verwendet, um die Ergebnisse quantitativ beurteilen zu können.

4.1.1 Punkt-Punkt Korrespondenzen

Betrachten wir zunächst den konstruierten Fall, in dem beide Bilder neben einem homogenen Hintergrund einen einzelnen dazu farblich unterscheidbaren Bildpunkt beinhalten. Die Position dieses Punktes variiert zwischen den beiden Bildern um den diskreten Abstand zweier benachbarter Pixel. Abbildung 4.1 zeigt zwei Bilder der Dimension 20×20 mit weißem Hintergrund und jeweils einem schwarzem Bildpunkt, wobei der schwarze Punkt im rechten Bild gegenüber dem im linken Bild um einen Pixel nach rechts verschoben ist.

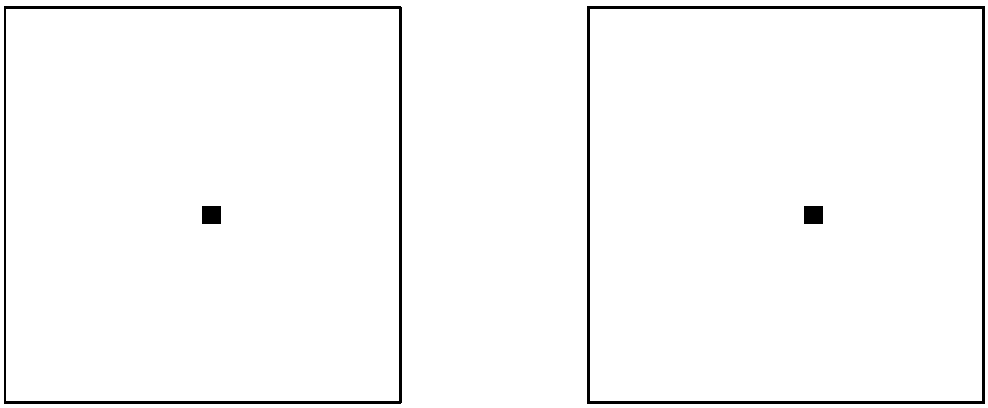


Abbildung 4.1: Zwei konstruierte Bilder der Dimension 20×20

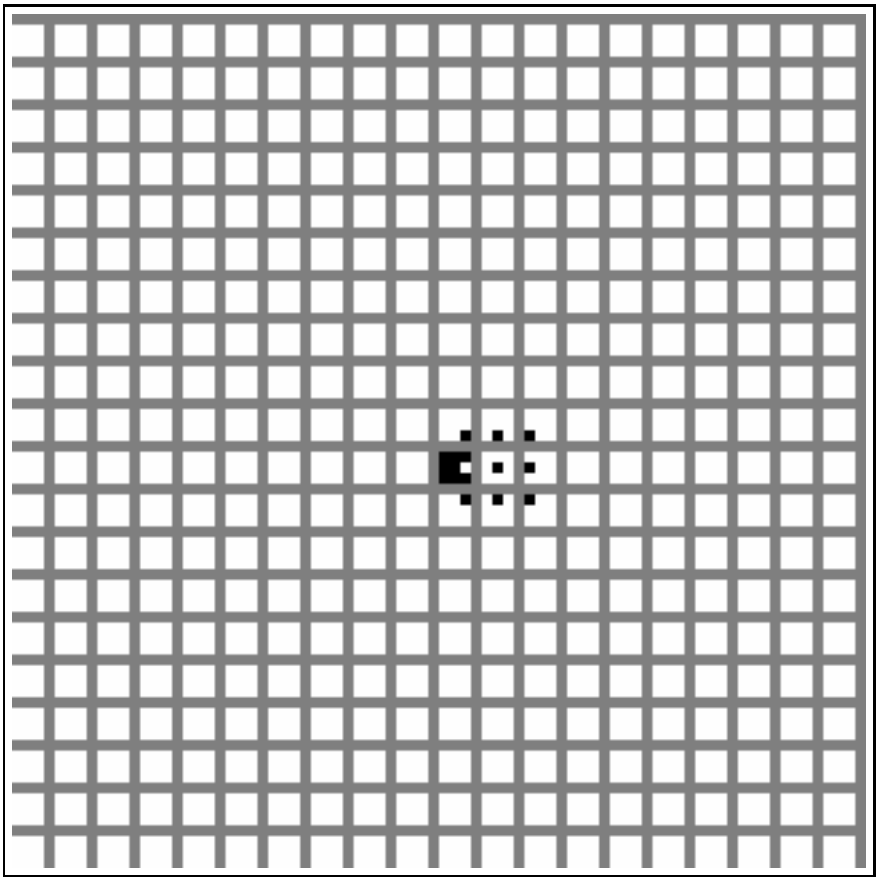


Abbildung 4.2: Darstellung der Farbähnlichkeiten

Die aus der Vorverarbeitung gewonnene Wahrscheinlichkeitsverteilung von Korrespondenzen hat durch die komplementären Farben und die Positionierung der beiden schwarzen Bildpunkte eine ganz besondere Gestalt. In Abbildung 4.2 sind die Wahrscheinlichkeitswerte von allen Bildpunkten des linken Bildes zu allen im jeweiligen Suchraum befindlichen möglichen Korrespondenzpartnern des rechten Bildes dargestellt. Jeder durch das graue Gitter unterteilte Bereich gehört zu einem Ausgangsbildpunkt aus dem linken Bild, und umfaßt den Suchraum zu diesem Pixel. Der Suchraum besteht aus 3×3 Positionen, deren Wahrscheinlichkeitswert hier jeweils durch einen Helligkeitswert dargestellt ist. Dabei ist eine hohe Wahrscheinlichkeit hell und eine niedrige Wahrscheinlichkeit entsprechend dunkel gekennzeichnet. Die Suchräume sind jeweils auf der Position des Ausgangspixels zentriert. Ein komplett weißer Bereich bedeutet also, daß für dieses Ausgangspixel das Pixel im rechten Bild an derselben Position, aber auch alle seine acht direkten Nachbarn als Korrespondenzpartner sehr wahrscheinlich sind. Dies ist in den homogenen Bildbereichen der Fall. Hier kann also zunächst noch keine Aussage für oder gegen einen speziellen Korrespondenzpartner getroffen werden.
Mehr Informationen beinhalten die Bereiche mit vielen schwarzen Positionen, da diese kleine Wahrscheinlichkeitswerte repräsentieren, was bedeutet, daß diese Positionen für Korrespondenzen ausgeschlossen werden können. Eindeutig ist der Korrespondenzpartner für den einen zentralen schwarzen Bildpunkt aus dem linken Bild. Dessen Suchraum ist derjenige, der aus nur einer weißen und sonst schwarzen Positionen besteht. Der schwarze Bildpunkt des linken Bildes hat nur eine hohe Ähnlichkeit zur rechts benachbarten Position im rechten Bild, weil der einzige schwarze Bildpunkt im rechten Bild um ein Pixel nach rechts verschoben zu finden ist. Die Darstellung seines Suchraumes ist für alle Positionen, außer der rechts von ihm, schwarz. Für diesen Bildpunkt ist also nur ein einziger Korrespondenzpartner sehr wahrscheinlich. Hingegen für den Bildpunkt rechts von ihm und alle dessen weiteren sieben Nachbarn, die im linken Bild weiß waren, kann ein Korrespondenzpartner ausgeschlossen werden, nämlich der im rechten Bild schwarze.
Die in Abbildung 4.2 dargestellten Wahrscheinlichkeiten sind die aus der Vorverarbeitung allein durch Betrachtung der Farbähnlichkeiten gewonnenen Informationen und stellen die initiale Wahrscheinlichkeitsverteilung dar. Wenden wir nun darauf unser Rechenwerk an, um diese zu iterieren, dann ist zu erwarten, daß bei jedem Iterationsschritt eine weitere Stufe der Nachbarschaft die durch die bekannte Korrespondenz vorgegebene Ordnung zu erfüllen versucht. Betrachten wir die Ergebnisse der angegebenen Iterationen in Abbildung 4.3, so sehen wir, daß sich in der Darstellung mit zunehmender Anzahl von Iterationsschritten t der zentrale dunkle Bereich vergrößert. Dabei sind unterschiedliche homogene Graufärbungen zu vernachlässigen, da diese aus der implementierten Renormalisierung folgen, die nur in sehr groben Schritten realisiert ist. Ebenso sind besonders bei Iteration $t = 06$ auftretende Artefakte durch Randeffekte in Verbindung mit grober Normalisierung zu erklären, denn alle Pixel am Bildrand behalten ihre initialen exakt normierten Wahrscheinlichkeiten.
Die folgende Abbildung 4.4 zeigt einen Ausschnitt aus einem der dunklen Bereiche. Hier ist zu sehen, daß alle Suchraumeinträge bis auf der an der um ein Pixel nach rechts verschobenen Position schwarz sind, was eine niedrige Wahrscheinlichkeit bedeutet. Es ist also jeweils ein eindeutiges Maximum zu bestimmen, welches daraus folgt, daß alle Bildpunkte der vorgegebenen Ordnung gefolgt sind und ebenfalls ihren Korrespondenzpartner

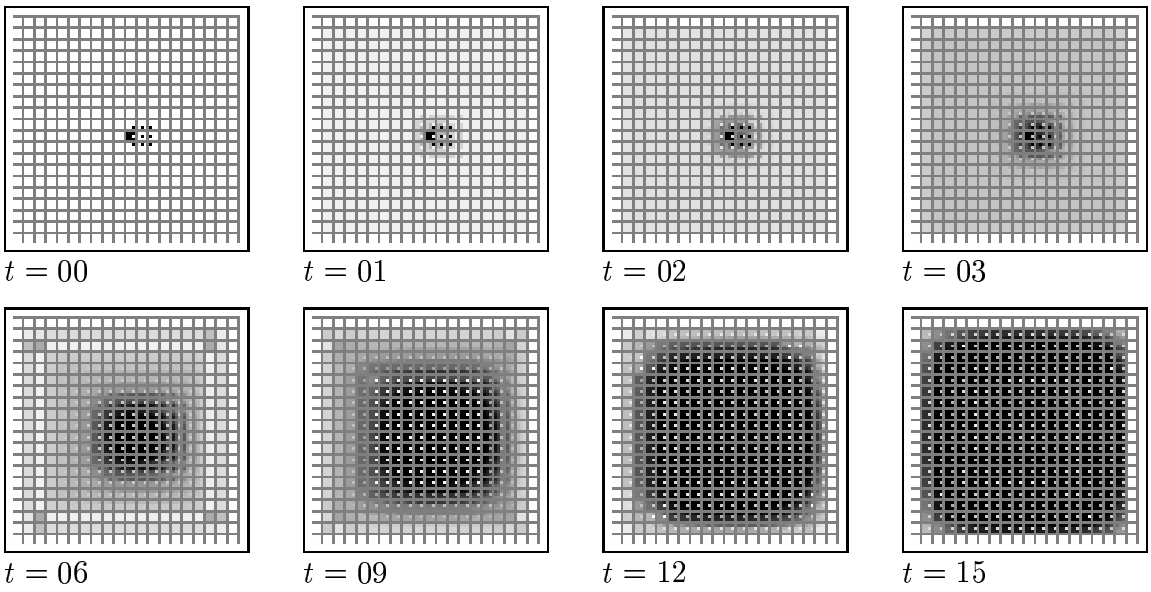


Abbildung 4.3: Darstellungen der iterierten Wahrscheinlichkeitsverteilung

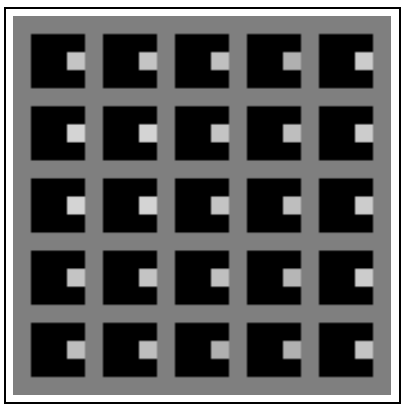


Abbildung 4.4: Vergrößerter Ausschnitt aus Iteration $t = 15$

rechts neben ihrer Position bevorzugen. Zu beachten ist, daß sich auch in den homogenen Bildregionen, wo anfangs alle Korrespondenzpartner gleich gut geeignet erschienen, durch die Weitergabe der Information von stark gewichteten Korrespondenzen, diese Gewichtung durchgesetzt hat.
Die Iteration der Ordnungsbedingung hat also dazu geführt, daß nun auch in homogenen Bereichen eine Aussage über den wahrscheinlichsten Korrespondenzpartner gemacht wird. Gemäß der Annahme über die Ordnungsbedingung aus Kapitel 2, Abschnitt 2.4 wird davon ausgegangen, daß alle Pixel einen Korrespondenzpartner finden und daß diese Korrespondenzen lokal dieselbe Ordnung zueinander aufweisen wie ihre Ursprungspixel. Daraus folgt in unserem Beispiel aus Abbildung 4.1, daß die durch den einen schwarzen Bildpunkt vorgegebene Verschiebung von allen seinen Nachbarn angenommen wird und weiter, daß deren Nachbarn auch diese vorgegebene Ordnung erfüllen. Die Interpretation der Abbildung 4.1 ist also nicht, daß nur der zentrale schwarze Bildpunkt verschoben wurde, sondern auch dessen Nachbarn und wiederum deren Nachbarn, also auch der gesamte homogene Hintergrund. Mit dieser Betrachtungsweise ist das Ergebnis nach der $15ten$ Iteration korrekt, denn es wird die Aussage getroffen, daß alle Pixel um eine Position nach rechts verschoben wurden. Auf den ersten Blick mag diese Aussage zwar übertrieben sein, da der Unterschied beider Bilder doch nur das zentrale schwarze Pixel ist, aber unter Berücksichtigung der Ordnungsbedingung, ist genau dieses Ergebnis zu erwarten. Die Ordnungsbedingung ermöglicht es also, auch in homogenen Bildbereichen, ihren entscheidbaren Rändern gemäß, Aussagen über bevorzugte Korrespondenzen zu treffen.

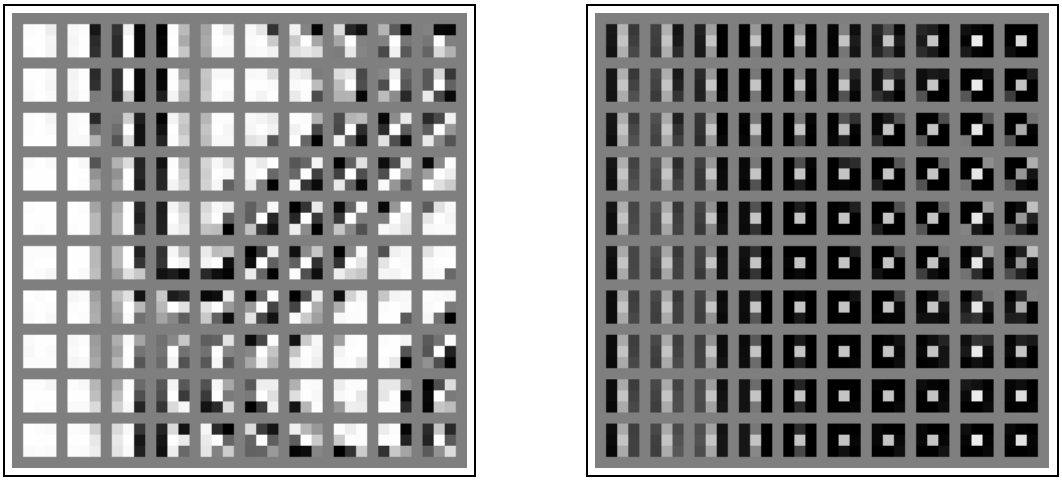
4.1.2 Bewegte Bildregionen

Nun verwenden wir anders als im vorigen Beispiel Bilder mit strukturiertem Hintergrund, der in beiden Bildern konstant bleibt. Im Vordergrund bewegen wir einen Bildausschnitt um eine vorgegebenen Abstand. So erhalten wir zwei Bilder mit definierter Verschiebung für alle Pixel außer denjenigen, die durch den bewegten Vordergrund in einem der Bilder verdeckt werden und im anderen sichtbar sind. Diese beiden Bilder sind in Abbildung 4.5

Abbildung 4.5: Zwei konstruierte Bilder mit verschobenem Ausschnitt

zu sehen. Dort befindet sich jeweils auf der rechten Seite eine violette Blume, die vom linken zum rechten Bild um ein Pixel nach rechts und ein Pixel nach oben verschoben positioniert ist.
Bei der Bestimmung der Korrespondenzen gibt es in diesem Beispiel drei Problemzonen, welche wir einzeln diskutieren wollen. Es gibt das allgemeine Problem, homogene Bereiche oder Bereiche mit nur wenig Kontrast zu entscheiden, da hier keine Grundinformation vorhanden ist, die durch Iteration der Ordnungsbedingung weitergegeben werden kann. Weiter kommt es an Kanten bewegter Objekte zu zwei unterschiedlichen Problemen. die zum einen aus der Verdeckung von Bildpunkten durch das Verschieben des Objektes resultieren und zum anderen durch das Auseinanderreißen in Nachbarschaften bestehender Ordnungen am Objektrand hervorgerufen werden.
Allgemein sind sowohl im Hintergrund als auch im verschobenen Ausschnitt durch eindeutigen Farbwechsel begrenzte homogene Regionen vorhanden. Betrachtet man zunächst nur die Farbähnlichkeiten, so kann in den homogenen Regionen keine Aussage zur Korrespondenz gemacht werden, da hier eine Vielzahl von Bildpunkten in Frage kommen würden. An ausgeprägten Farbkanten mit hohem Kontrast hingegen können viele Positionen für Korrespondenzen ausgeschlossen werden. Diese allgemein als Apertur-Problem[1] bezeichnete Eigenschaft schließt Korrespondenzen senkrecht zur Kante aus, aber kann parallel zur Kante keine Aussage machen. Das liegt daran, daß jeweils senkrecht zur Kante die Farbunterschiede sehr groß sind, aber parallel zur Kante viele Positionen sehr ähnliche Farbwerte aufweisen. Abbildung 4.6 zeigt links einen Ausschnitt aus der initialen Wahrscheinlichkeitsverteilung der Korrespondenzen, wo die Auswirkungen des Apertur-Problems zu sehen sind. Der Ausschnitt stammt genau aus der Mitte der Bilder und umfaßt 10×10 Bildpunkte. Wieder bedeuten dunkle Bereiche, daß viele Korrespondenzen ausgeschlossen werden können, während helle Positionen für Korrespondenzen in Frage kommen. In der linken Bildhälfte ist eine senkrechte und in der rechten Bildhälfte ist eine diagonale Präferenz zu erkennen.
Im rechten Teil der Abbildung 4.6 ist derselbe Ausschnitt der Wahrscheinlichkeitsverteilung nach drei Iterationsschritten dargestellt. Es ist zu sehen, daß das Apertur-Problem zunehmend gelöst werden kann, sofern entlang einer Kante auch weitere Positionen senkrecht zur Kante ausgeschlossen werden können. Dies ist besonders gut an dem Feld in der vierten Spalte der sechsten Zeile zu sehen. Die einzelnen Felder sind wieder durch ein grau gefärbtes Gitter getrennt. Dort ist bereits bei $t = 0$ eine Ecke festzustellen. Diese Information wird durch die Iteration weitergegeben und führt bei $t = 03$ dazu, daß entlang der beiden sich dort treffenden Kanten weitere Bildpunkte ihre Korrespondenzpartner einschränken. Es kristallisiert sich jeweils die zentrale Position als Maximum heraus. Besonders im linken Bildbereich sind allerdings noch viele Unsicherheiten in Ausrichtung der senkrechten Kante vorhanden, da hier noch keine weitere Information angekommen ist. Iterieren wir aber weiter, so wird auch dieser Bildbereich, der bei $t = 0$ noch homogen war, eindeutig entschieden. Da der Bildausschnitt aus dem in beiden Bildern konstanten Hintergrund stammt, nehmen alle Bildpunkte, wie zu erwarten ist, die zentrale Position des Suchraumes, also ihre eigene, als wahrscheinlichsten Korrespondenzpartner an.

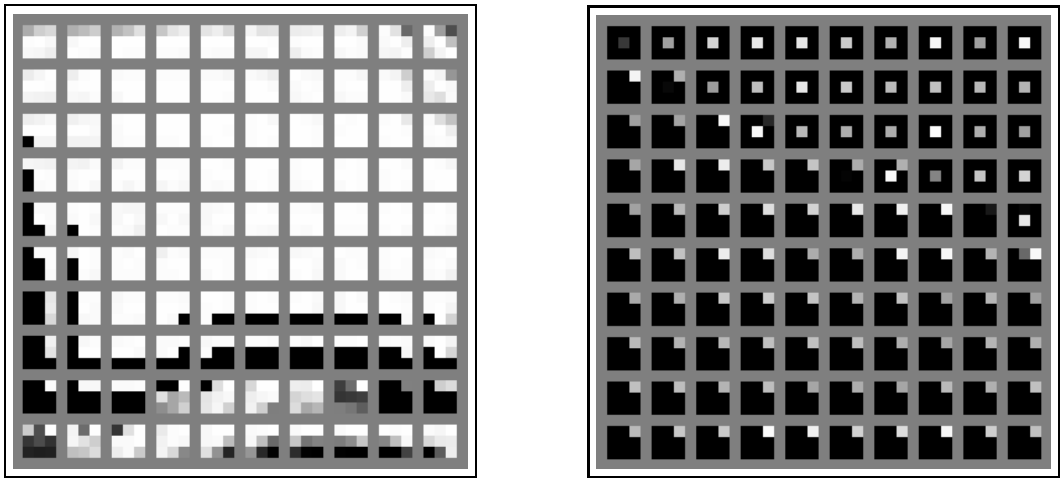
[1] Apertur-Problem: Korrespondenzaussage parallel zur Kante bei fehlender Begrenzung unbestimmt



$t = 0$ $t = 03$

Abbildung 4.6: vergrößerter Ausschnitt der Wahrscheinlichkeitsverteilungen mit Apertur-Problem und homogenen Gebieten

Betrachten wir nun den eigentlich interessanten Teil des Bildes, den mit der verschobenen violetten Blume. Hier wird wieder starke Information kontrastreicher Regionen wie Kanten und Ecken durch Iteration an benachbarte Bildpunkte weitergegeben. Zusätzlich kommt es aber nun an dem Rande des verschobenen Objektes zu gegensätzlichen Informationen. Da der Hintergrund konstant bleibt, das Objekt aber verschoben ist, sind nicht mehr alle Ordnungen innerhalb der Nachbarschaften zu erfüllen. Es bekommen alle Randpixel des Objektes und alle am Objektrand befindlichen Bildpunkte des Hintergrunds teilweise andere Nachbarn. Das Objekt ist um jeweils einen Pixel nach rechts und oben verschoben, sodaß links unten im zweiten Bild der Abbildung 4.5 Teile vom Hintergrund sichtbar sind, die im ersten Bild noch verdeckt waren, während rechts oben nun Bildpunkte verdeckt sind, die im ersten Bild sichtbar waren. Jeweils den Nachbarn der teilweise verdeckten Bildpunkten fehlt nun ein die Ordnung unterstützender Nachbar. Dadurch ist die Ordnungsbedingung nicht mehr exakt zu erfüllen. Zu fehlerhaften Korrespondenzen kommt es allerding nur, wenn die Information der neuen Nachbarn stärker ist als die der gebliebenen. Das ist besonders in den Bereichen der Verdeckung der Fall, kann aber auch auftreten, wenn eine kontrastreiche verschobene Objektkante neben einem homogenen Hintergrund vorkommt. Dann ist auch für Bildpunkte des Hintergrundes die Information der auf der Kante befindlichen Nachbarn stärker als die der eventuell sogar größeren Anzahl der Nachbarn auf dem konstanten Hintergrund. Folglich wird in diesem Bereich ein Korrespondenzpartner bevorzugt, der die Ordnung des Objektes befolgt, was damit ebenfalls eine Verschiebung bedeutet.
Abbildung 4.7 zeigt jeweils denselben Ausschnitt aus der Wahrscheinlichkeitsverteilung nach zwei verschiedenen Iterationsstufen. Der Ausschnitt beinhaltet den oberen rechten Rand des verschobenen Objektes und einen Teil des Hintergrunds. Links ist die initiale Verteilung bei $t = 0$ dargestellt. Es sind links und unten die durch den Objektrand zwischen zwei Blütenblättern erzeugten starken Informationen durch die vielen dunklen Positionen gekennzeichnet. Durch den hohen Kontrast zwischen Objektrand und



$t = 0$ $t = 15$

Abbildung 4.7: vergrößerter Ausschnitt der Wahrscheinlichkeitsverteilungen am Objektrand

Hintergrund können hier wieder viele Korrespondenzen jeweils senkrecht zum Rand ausgeschlossen werden. Wieder sind auch die Eigenschaften des Apertur-Problems durch mehrere mögliche, hell gekennzeichnete Positionen parallel zu den Kanten zu erkennen. Der mittlere Bereich dieses Ausschnitts repräsentiert den homogenen Bereich der im Hintergrund befindlichen orangen Blume. Hier sind keine Korrespondenzinformationen festzustellen, da hier aufgrund ihrer hohen Farbähnlichkeit zunächst alle Positionen möglich sind. In der oberen rechten Ecke ist wiederum etwas Information, hervorgerufen durch Farbänderungen vorhanden.
Iterieren wir diese Wahrscheinlichkeitsverteilung, so ergibt sich bei $t = 15$ das im rechten Teil der Abbildung 4.7 dargestellte Ergebnis. Dort ist festzustellen, daß links unten gemäß der Verschiebung des Objektes Korrespondenzen an den Positionen jeweils rechts oben der Felder sehr wahrscheinlich werden, während alle anderen Positionen im Suchraum ausgeschlossen werden. In der rechten oberen Ecke hingegen werden Korrespondenzen bevorzugt, die sich an derselben Position wie die Ausgangspixel befinden, was durch die helle Kennzeichnung der zentralen Positionen der Felder dargestellt ist. Dies ist auch zu erwarten, da der dort repräsentierte Hintergrund in beiden Bildern aus Abbildung 4.5 konstant geblieben ist. Soweit sind die Ergebnisse den Erwartungen durch Kenntnis der zugrundeliegenden Veränderung der beiden Bilder entsprechend. Interessant ist besonders der Bereich im Zentrum der Abbildung 4.7 rechts. Dort haben auch die Bildpunkte des Hintergrunds die Korrespondenzen an der Position rechts über ihnen angenommen. Dies kommt daher, daß bei $t = 0$ der links unten befindliche Objektrand sehr starke Kontrastinformation beinhaltete, während der Hintergrund nur sehr wenig Kontrast aufwies. Durch Iteration der Ordnungsbedingung wurde also nicht nur das Apertur-Problem gelöst, sondern weiter auch die Ordnung auf die benachbarten Bildpunkte übertragen. So kommt es an dieser Stelle zu falschen Korrespondenzaussagen, die darauf begründet sind, daß zum einen einige Bildpunkte durch Verdeckung durch das verschobene Objekt eigentlich keinen Korrespondenzpartner

haben und sich daher nach ihren Nachbarn richten. Zum anderen hatten ihre Nachbarn im Hintergrund keine eindeutige Information zur Verfügung, so daß die Information des verschobenen Objektes hier überwiegt. Die gegensätzlichen Informationen von dem Objekt links unten und vom Hintergrund oben rechts wandern bei jedem Iterationsschritt aufeinander zu. Entlang einer diagonalen Linie im oberen rechten Teil der Abbildung treffen sie sich und es entsteht die Grenze an der Stelle, wo beide Informationen gleich stark sind. Diese Grenze ist durch die unterschiedlich stark ausgeprägten Kanten in den Bildern nicht unbedingt in der Mitte zu finden, sondern in diesem Fall durch die scharfen Objektkanten bekräftigt, weit in den Hintergrund hinein gewandert. Zu beachten ist, daß es an dieser Grenze auch Bildpunkte gibt, für die sich zwei mögliche Korrespondenzen ergeben, oder gar kein passender Korrespondenzpartner zu bestimmen ist.
Vergleichen wir nun die wahren mit den ermittelten Korrespondenzen, so können wir diese Fehler bestimmen und graphisch darstellen. Abbildung 4.8 zeigt die Fehler der ermittelten Korrespondenzen, wobei die schwarzen Positionen zu erwartende Fehler durch Verdeckung darstellen, wo gar kein Korrespondenzpartner hätte gefunden werden dürfen. Die grauen Bereiche zeigen fehlerhafte Korrespondenzen, die durch die oben beschriebenen Probleme an den Objekträndern entstanden sind. Im übrigen sind alle anderen auch aus homogenen Bildteilen und aus dem bewegten Objekt stammenden Korrespondenzen korrekt ermittelt. Auch ist zu beachten, daß an Objekträndern, wo genügend Strukturinformation in Form von Farbkontrasten vorhanden ist, die oben beschriebenen Probleme nicht auftauchen.

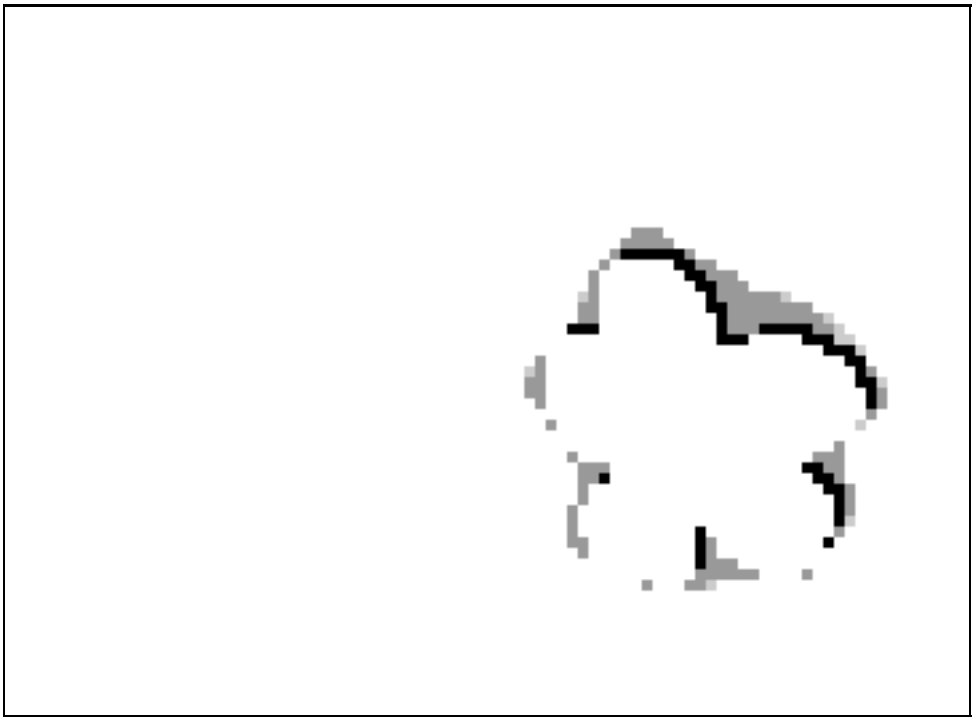
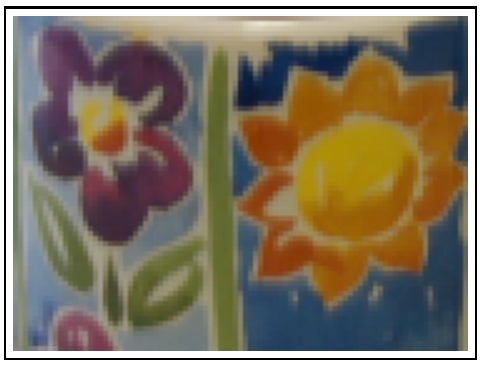


Abbildung 4.8: Darstellung fehlerhaft ermittelter Korrespondenzen

4.1.3 Subpixelpositionen

Betrachten wir nun den Fall, in dem Korrespondenzen nicht genau auf den diskreten Pixelpositionen liegen, sondern durch eine Bewegung wie Rotation die Bilder so verändert werden, daß einige Bildpunkte auf Subpixelpositionen zu liegen kommen. Die Bildpunkte so erzeugter Bilder befinden sich dann wieder an diskreten Pixelpositionen, wobei dazu aber deren Farbwerte interpoliert werden. So kann es vorkommen, daß es nicht mehr eine exakte Pixelposition gibt, die für eine Korrespondenz in Frage kommt, sondern, daß auch wie die Farbwerte interpoliert wurden, die Korrespondenz sich aus der Interpolation diskreter Positionen bildet.
Verändern wir nun ein Bild so, daß korrespondierende Merkmale nicht mehr genau auf den diskreten Pixelpositionen liegen. Dazu rotieren wir das Ausgangsbild um dessen Zentrum um einen Winkel α. So werden alle durch Bildpunkte repräsentierten Merkmale durch die Rotation in Abhängigkeit von α und ihrem Abstand $|r|$ vom Zentrum transformiert. Für kleine Winkel α ergeben sich Translationen der einzelnen Bildpunkte um den Abstand $d = 2\pi|r| \times \alpha/_{360}$ und der Richtung senkrecht zum Vektor $\vec{r}$. Da das daraus erzeugte Bild wieder aus diskreten Pixelpositionen besteht, werden dazu die Farbwerte durch Interpolation der Farbwerte der transformierten Bildpunkte gewonnen. Für die Korrespondenzfindung sind nun also nicht mehr exakt passende Bildpunkte vorhanden. Dennoch wollen wir unser Rechenwerk auf diese Problemstellung anwenden und erwarten, daß der jeweils am besten passende Bildpunkt als Korrespondenzpartner zu jedem Ausgangspixel gefunden wird. Abbildung 4.9 zeigt das verwendete Ausgangsbild der Dimension 91×67, welches zum Erhalt des zweiten Bildes schrittweise um $1, 2$ und 3 Grad im Uhrzeigersinn gedreht wird. Betrachten wir die Pixel am Bildrand, so ist hier bei ei-



$\alpha = 0$

$\alpha = 3°$

Abbildung 4.9: Ausgangsbild und gedrehtes Bild.

ner Rotation um 3 Grad bereits eine Verschiebung um bis zu $2,5$ Pixel zu erwarten. Da unsere Implementation mit einem Suchraum der Größe 3×3 arbeitet, also nur Verschiebungen um ein Pixel in einer $8er$ Nachbarschaft berücksichtigt werden können, ist im äußeren Bildbereich mit keinen oder fehlerhaft gefundenen Korrespondenzen zu rechnen. Nach $t = 15$ Iterationsschritten, die zum Entscheiden der homogenen Bildregionen nötig sind, erhalten wir die ermittelten Wahrscheinlichkeiten der Korrespondenzen, die jeweils auf den diskreten Positionen entsprechend des 3×3 Suchraumes liegen. Es kristallisiert

sich für über 90% der Bildpunkte ein eindeutiges Maximum der Wahrscheinlichkeiten der möglichen Korrespondenzpartner heraus. Die restlichen Bildpunkte haben Subpixelpositionen entsprechend mehrere mögliche Korrespondenzpartner, aus denen die endgültige Position, wie in Abschnitt 3.6 über die Nachverarbeitung beschrieben, interpoliert wird. Oder aber es sind besonders im Fall der Rotationen um 3 Grad durch den zu großen Abstand gar keine möglichen Korrespondenzpartner zu finden. Letztere bilden dann allerdings nur einen Anteil von unter 2%. Zu erwarten wäre, daß durch die mit zunehmendem Radius größer werdenen Verschiebungen ab einem Radius von 29 Pixeln bei unserem begrenzten Suchraum keine Korrespondenzen mehr gefunden werden können. Das bedeutet, daß in diesem Fall mit unserem Rechenwerk auch dann ein am besten passender Korrespondenzpartner innerhalb des Suchraumes ermittelt wird, wenn der wahre Partner außerhalb des Suchraumes liegt. Das liegt daran, daß der Suchraum für diese Anwendung zu klein gewählt ist.

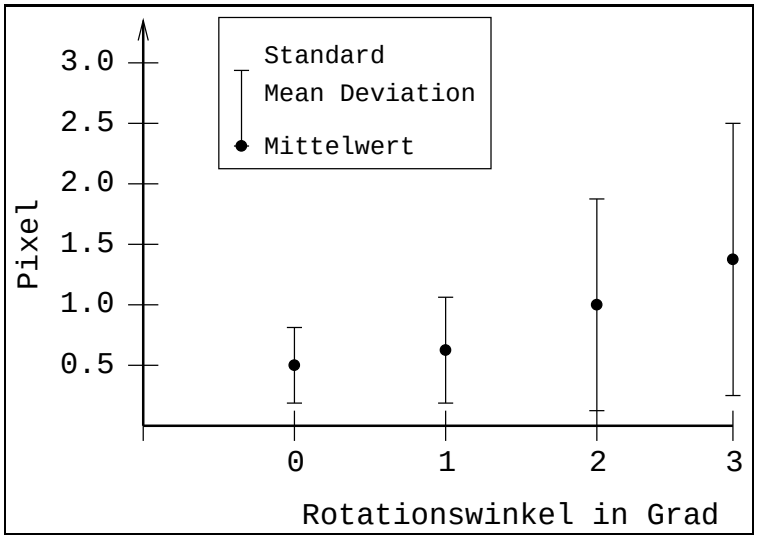


Abbildung 4.10: Fehler in Abhängigkeit des Rotationswinkels

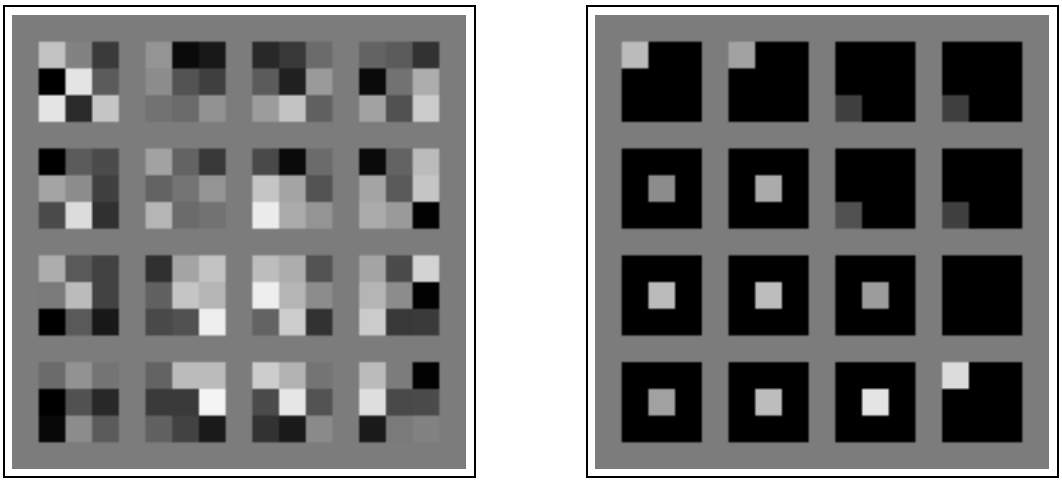
Betrachten wir die Fehler in den ermittelten Positionen der Korrespondenzen im Vergleich zu den aus der Konstruktion der Bilder bekannten wahren Positionen, so ergibt sich mit zunehmendem Rotationswinkel eine sehr große Fehlerspanne. Auch der durchschnittliche Fehler wächst bei zu großem Winkel aufgrund der oben beschriebenen Eigenschaften in Zusammenhang mit dem zu kleinen Suchraum. Bei einer Rotation um 1 Grad hingegen befindet sich der wahre Korrespondenzpartner für alle Bildpunkte innerhalb des Suchraumes. Daraus ergibt sich ein durchschnittlicher Fehler von etwa einem halben Pixel. Dieser Fehler ist bedingt durch die untersuchten diskreten Positionen zu begründen. Abbildung 4.10 zeigt den durchschnittlichen Fehler und den mittleren quadratischen Fehler[2], wobei der euklidische Abstand von wahrer und ermittelter Position der Korrespondenzen zugrunde liegt.
Abschließend kann also festgestellt werden, daß Korrespondenzen auf Subpixelpositionen

[2] `Standard Mean Deviation` ermittelt nach [Bro95]

mit einem Fehler von einem halben Pixel bestimmt werden können, aufgrund des begrenzten Suchraumes aber Einschränkungen bezüglich der maximalen wahren Verschiebungen gemacht werden müssen.

4.1.4 Rauschen

Bei den bisher betrachteten Beispielen waren die Farbwerte der Bildpunkte bis auf die Interpolation der Subpixel in beiden Bildern exakt gleich. Stammen die zu verarbeitenden Bilder allerdings aus unterschiedlichen Kameraaufnahmen, so ist mit Unterschieden in Form von Rauschen zu rechnen. Diese Eigenschaft realer Kameras wollen wir durch additives Gauß'sches Rauschen simulieren. Es wird dazu jeder Farbwert der drei Kanäle *(Rot, Grün, Blau)* eines jeden Bildpunktes unabhängig voneinander durch Addition eines zufällig generierten Fehlers mit vorgegebener Varianz verändert. Wir verwenden das Ausgangsbild aus Abbildung 4.9 und generieren daraus durch Verrauschen das korrespondierende Bild. Die Varianz des Rauschens beträgt bis zu 10% des Wertebereichs der Farbkanäle. Daraus ergeben sich unter Berücksichtigung der Ordnung des gesamten Bildes Korrespondenzen an exakt derselben Position, während zunächst für einzelne Bildpunkte aufgrund der veränderten Farben andere Partner besser geeignet erscheinen. Wenden wir die Ordnungsbedingung an und iterieren diese im Rechenwerk, so zeigt sich, daß zusammenhängende Regionen ihre Korrespondenzpartner in gleicher Nachbarschaft finden. Das bedeutet, daß einzelne Fehler in den initialen Wahrscheinlichkeiten ausgeglichen werden, falls die Informationen der Nachbarschaft überwiegen. Probleme gibt es nun allerdings in homogenen Regionen, da dort geringe Unterschiede in den Farbähnlichkeiten einen größeren Anteil der überhaupt vorhandenen Informationen ausmacht, als es in Regionen mit hohem Farbkontrast der Fall ist. Wir erkennen in Abbildung 4.11, die einen vergrößerten Ausschnitt aus der initialen und der iterierten Wahrscheinlichkeitsverteilung zeigt, daß sich die durch das Rauschen hinzugefügten Fehler unterschiedlich auswirken.
Beide Ausschnitte repräsentieren dieselben Bildpunkte aus einem homogenen Bildbereich. Links sind die initialen Wahrscheinlichkeiten dargestellt, die aus der Betrachtung der Farbähnlichkeiten gewonnen sind. Durch das Rauschen sind die zugrundeliegenden Farbwerte derart verfälscht, daß nicht mehr alle Positionen für eine Korrespondenz gleich wahrscheinlich sind, wie es in einer homogenen Region zu erwarten wäre. Daraus ergeben sich lokale Präferenzen für unterschiedliche Positionen. Durch Iteration der Ordnungsbedingung wird hier allerdings trotzdem für die Hälfte der Bildpunkte die richtige zentrale Position ermittelt. Für die andere Hälfte der Bildpunkte war die initiale Fehlinformation so groß, daß auch durch das Betrachten der Ordnung keine eindeutige einheitliche Korrespondenzposition bestimmt werden konnte. Dieses ist nun ein extremes Beispiel mit großem initialem Fehler. Im Einzelfall könnte man in solchen Fällen bereits in der Vorverarbeitung den verwendeten Parameter $\sigma = 0.16$ erhöhen, was dann nicht mehr so scharfe Kontraste ergibt, aber dafür nicht schon initial zu viele, eventuell richtige, Korrespondenzen ausschließt. Bei geringerem Rauschen sind die Ergebnisse auch bei konstantem σ in einem Rahmen, der den Erwartungen entspricht.
Wieder wurden die wahren mit den ermittelten Korrespondenzen verglichen und die Fehler in Abbildung 4.12 für verschiedene Rauschanteile dargestellt. Bis zu einem Rauschan-



$t = 0$ $t = 15$

Abbildung 4.11: Wahrscheinlichkeitsverteilung in konstanter homogener Bildregion bei Rauschanteil von 10%

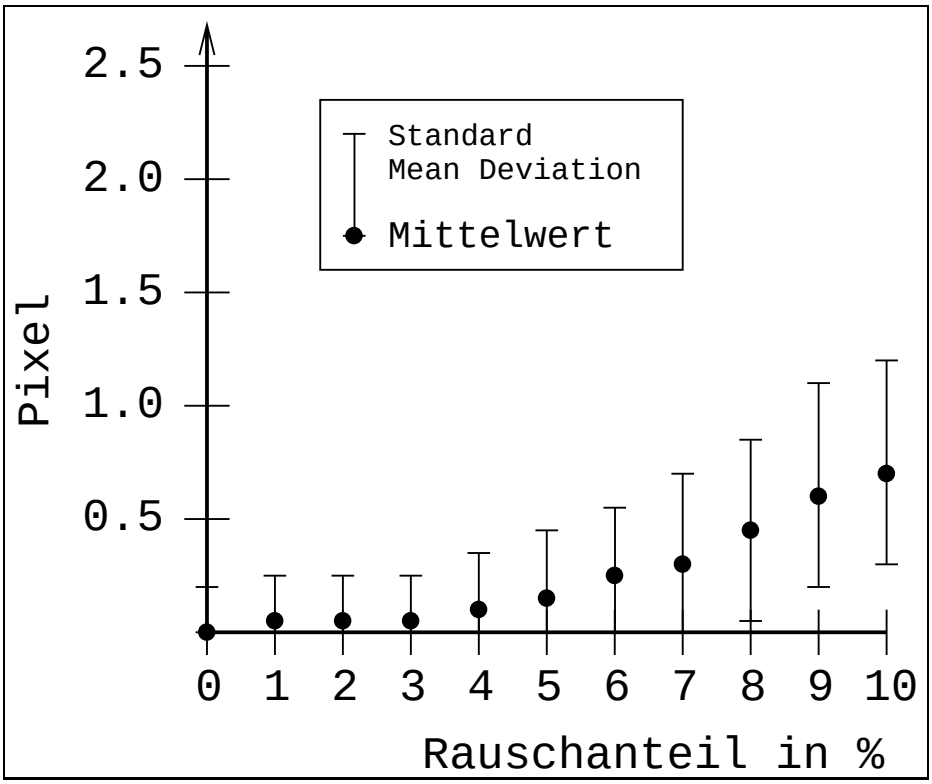


Abbildung 4.12: Positionsfehler in Abhängigkeit des Rauschanteils

teil von 4% liegt der Mittelwert der Positionsfehler, aber auch der mittlere quadratische Fehler[2] der Einzelpositionen unterhalb von einem halben Pixel. Diese Grenze ist durch den diskreten Suchraum auch in realen Anwendungen zu erwarten.

4.2 Anwendungen

Um den optischen Fluß zu ermitteln, wird normalerweise eine Folge von Bildern betrachtet, wobei Informationen aus vorangegangenen Bildern in darauf folgenden Schritten weiterverarbeitet werden. Wir können nur einen Schritt betrachten, da unser Verfahren nur mit jeweils zwei Bildern arbeitet. Daher ist ein Vergleich zu anderen Verfahren nicht direkt möglich. Trotzdem werden hier die Ergebnisse dieser Implementation, angewendet auf reale Situationen, vorgestellt. Vorschläge zur Erweiterung des Verfahrens sind in Kapitel 5 im Ausblick zu finden.

4.2.1 Yosemite Sequenz

Abbildung 4.13 zeigt zwei Bilder aus der Yosemite Sequenz, kreiert von L. Quann [Qua84], mit einer Auflösung von 320×240 Bildpunkten und je $8bit$ Grauwerten. Diese Sequenz stellt einen Flug durch eine Gebirgsschlucht dar. Zu erwarten ist, daß sich besonders der Vordergrund auf die Kamera zu bewegt, während der Hintergrund am Horizont konstant bleibt. Über dem Horizont befinden sich Wolken, die eine Eigenbewegung ausführen. Zusätzlich ändert sich deren Helligkeit und Größe. Für die Anwendung wurde die Auflösung um den Faktor 4 verringert, damit die zu erwartenden Korrespondenzen innerhalb des verwendeten Suchraumes liegen.

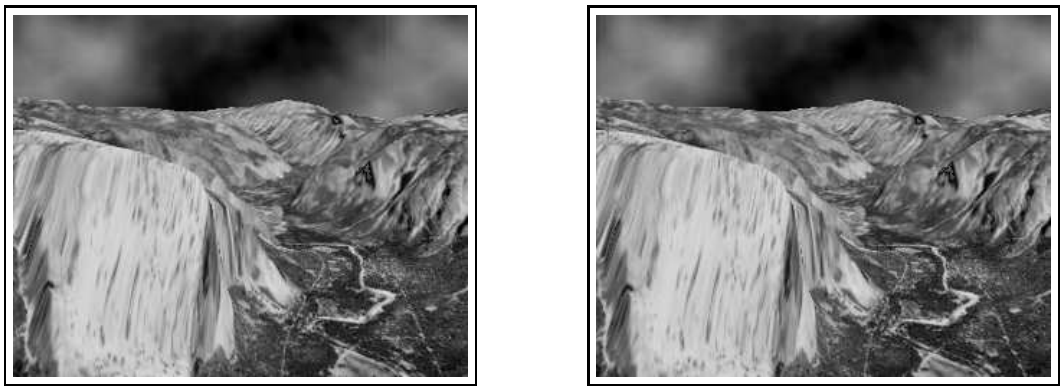


Abbildung 4.13: Yosemite-Sequenz: Bild 2 und 3

In Abbildung 4.14 sind die Ergebnisse der Anwendung auf diese Problemstellung in Form eines Flußdiagramms dargestellt. Von jedem Ausgangspixel ist, wie in Abschnitt 3.6 über die Nachverarbeitung beschrieben, ein Verschiebungsvektor ermittelt worden, der auf den

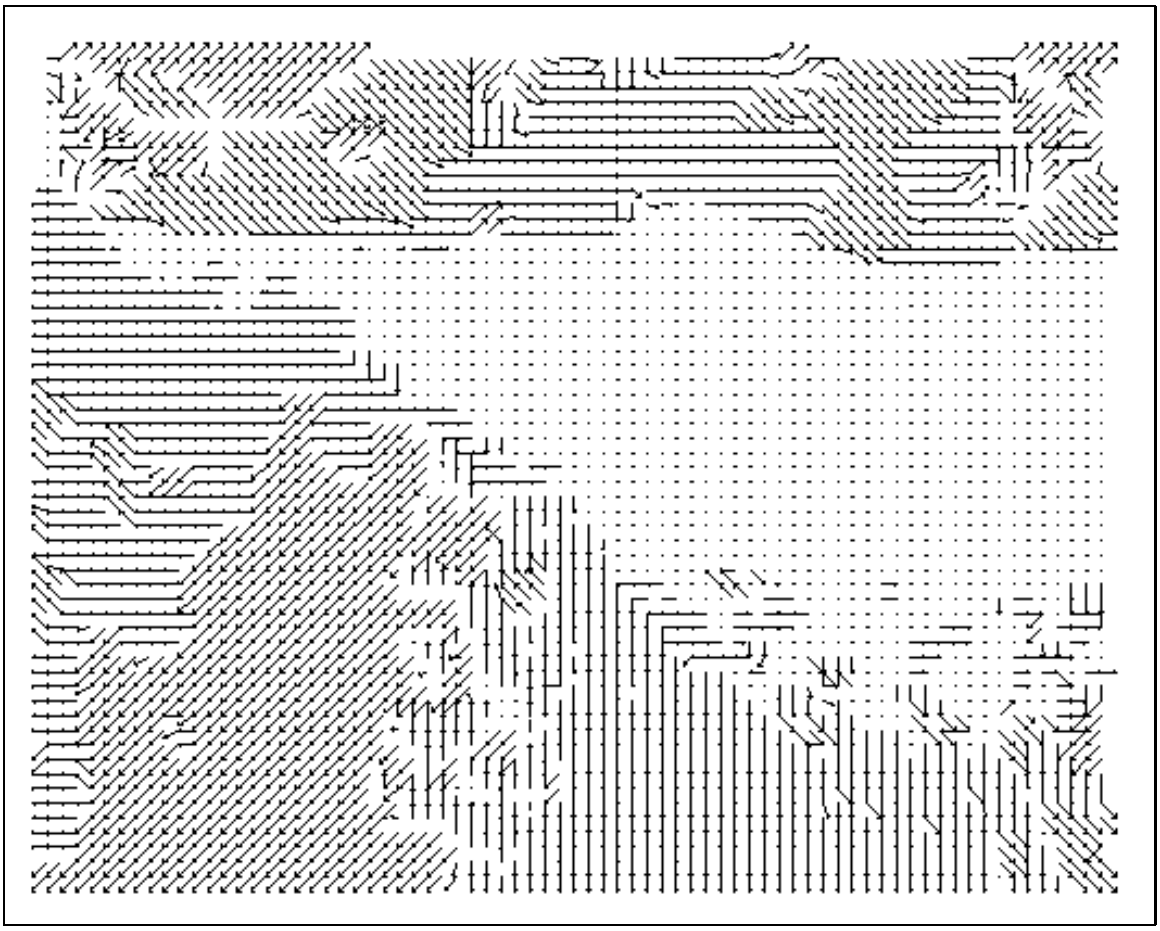


Abbildung 4.14: Flußdiagramm zur Yosemite-Sequenz

Ort des nach $t = 15$ Iterationsschritten am besten passenden Korrespondenzpartner zeigt. Zu erkennen ist, daß bis auf lokale Unregelmäßigkeiten ein Fluß ermittelt wurde, der den Erwartungen entspricht. Im unteren und linken Teil des Bildes wandern die Bildpunkte in Richtung des Bildrandes, was einer Annäherung der abgebildeten Positionen der zugrunde liegenden $3D$-Szene an die Kamera entspricht. Im rechten mittleren Teil des Bildes hingegen, wo der Hintergrund konstant bleibt, sind keine Verschiebungen festgestellt und daher keine Pfeile eingezeichnet worden. Der obere Bildrand, der den Himmel repräsentiert, weist einen ungeordneten Fluß auf, was in Zusammenhang mit den uneinheitlichen Änderungen der Wolken steht. Dieses Beispiel zeigt eine gute Anwendbarkeit der entwickelten Implementation auf diese Problemstellung.

4.2.2 Kamerabildfolge

Abbildung 4.15 zeigt zwei Bilder einer selbst aufgenommenen Sequenz. Die abgebildete Person bewegt sich vor einem konstanten Hintergrund nach rechts. Hintergrund und Person beinhalten sowohl strukturierte als auch homogene Bereiche. Die Bilder sind mit einer Auflösung von 320×240 Pixeln im RGB-Format von einer einfachen Kamera mit einer Bildfrequenz von 20 Bildern pro Sekunde aufgenommen. Zur Anwendung werden die Bilder in der Auflösung halbiert, damit die zu erwartenden Disparitäten der Suchraumgröße entsprechen.

Bild A

Bild B

Abbildung 4.15: Reale Kamerabilder einer gehenden Person

Abbildung 4.16 zeigt den daraus ermittelten optischen Fluß in Form von Verschiebungsvektoren. In der Darstellung ist nur für jeden zweiten Bildpunkt ein Vektor eingezeichnet, dessen Länge verdoppelt wurde, so daß die Vektoren bei der vorliegenden Druckqualität erkennbar sind. Es ist zu erkennen, daß die Bewegung der Person bis auf Störungen am Rand größtenteils richtig erkannt wurde. Wieder tritt allerdings das im Abschnitt 4.1 über bewegte Bildregionen in Verbindung mit Abbildung 4.7 erwähnte Problem an Objekträndern auf. Hier wird am rechten Objektrand die Grenze zum Hintergrund nicht richtig erkannt und für einen Teil des Hintergrundes ebenfalls eine Verschiebung nach rechts ermittelt.

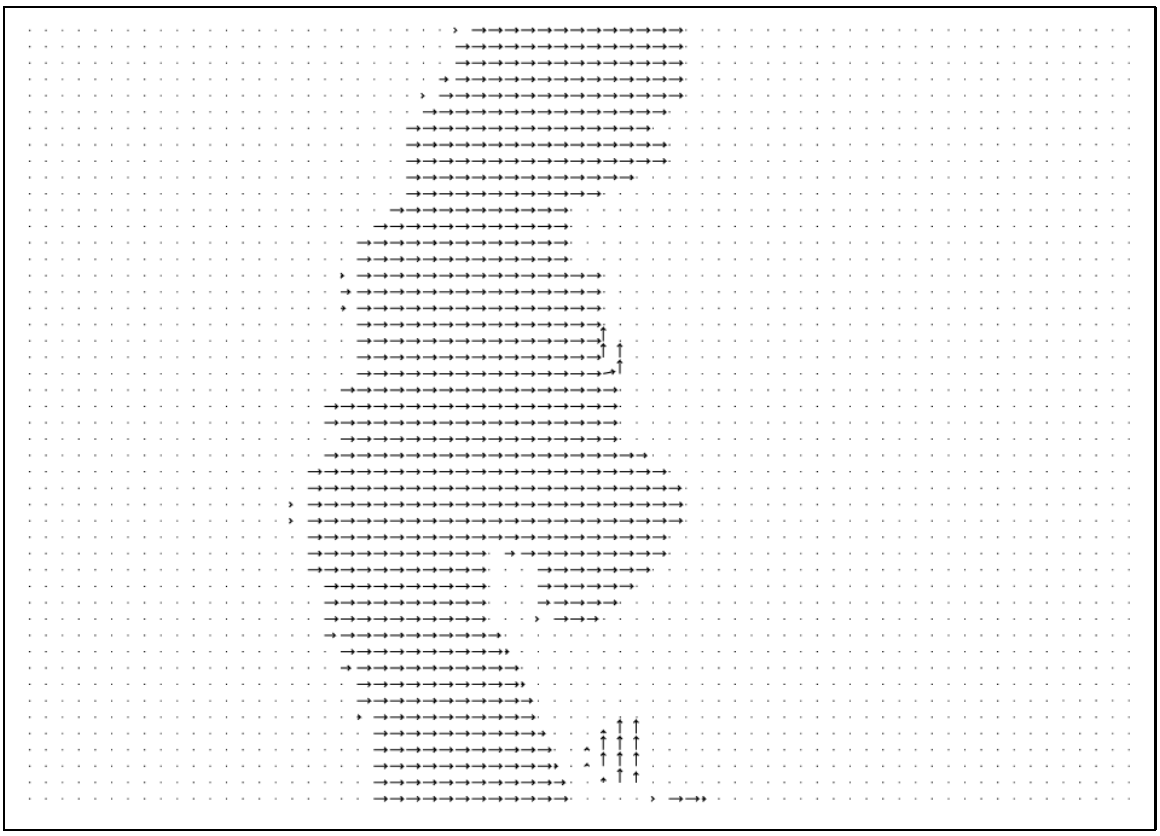


Abbildung 4.16: Flußdiagramm zu gehender Person

4.3 Vergleich mit anderen Systemen

Ziel dieser Arbeit war es, ein System zu entwickeln, welches Geschwindigkeitsvorteile gegenüber einer vorhandenen Implementation bietet. Da häufig verschiedene Bildgrößen verwendet werden, aber auch unterschiedlich große Disparitäten zugelassen werden, ist ein davon unabhängiges Maß nötig. Dazu definieren wir ein Vergleichsmaß.

$$\mathsf{PDS} := (\mathsf{Pixel} \times \mathsf{Disparität})/\mathsf{Sekunde} \tag{4.1}$$

Es gibt an, wieviele Pixel mit welcher maximalen Disparität in einer Sekunde verarbeitet werden können. Daraus ergibt sich dann bei festgelegter Bild- und Suchraumgröße die maximal erreichbare Bildfrequenz für die Anwendung. Der entwickelte Koprozessor allein erreicht bei einem Betrieb mit $20\mathsf{MHz}$ und 15 ausgeführten Iterationsschritten ca. 2 Millionen PDS. Dies würde bei Bildern der Dimension 100×100 eine theoretische Bildfrequenz von 20Hz ermöglichen.
Da aber der Datentransfer zwischen FPGA und Rechner nocheinmal in etwa ebensoviel Zeit benötigt wie die Iteration, beträgt die effektive Leistung des gesamten Systems 1 Million PDS. In der Anwendung ist zusätzlich noch die in Abschnitt 3.1 beschriebene Vorverarbeitung nötig. Diese ist ebenfalls abhängig von der verwendeten Bild- und Suchraumgröße. Sie ist auf dem verwendeten Hauptrechner, einem AMD Athlon bei $800\mathsf{MHz}$ mit einer Leistung von 0.5 Millionen PDS, realisiert. Die nötige Verarbeitungszeit ist aber nicht unbedingt additiv zur der des FPGAs. Durch die selbständige Steuerung des Koprozessors arbeitet dieser unabhängig vom Hauptrechner, so daß hier ebenfalls wie innerhalb des FPGA das Pipeline-Prinzip angewendet werden kann. Dennoch ist in der zur Verfügung stehenden Testumgebung die Vorverarbeitung im Vergleich zur wesentlich rechenaufwendigeren Iteration des Koprozessors in der Pipeline das bremsende Element. Allein auf den Koprozessor und den nötigen Datentransfer bezogen beträgt die Verarbeitungszeit bei einem Bilderpaar der Dimension 100×100 und einer Suchraumgröße von 3×3 $100\mathsf{ms}$, was eine Bildrate von bis zu $10\mathsf{Hz}$ erlaubt. Dies ist im Vergleich zu der Implementation *Acre* [Per] auf dem von Perwass [PS02] verwendeten Rechner mit $1.5\mathsf{GHz}$ ein Geschwindigkeitsvorteil von $20:1$. Das gesetzte Ziel, die Verarbeitungszeit zu erhöhen, konnte also erreicht werden. Das entwickelte System liegt mit einer Rechenleistung von $1.0 \times 10^6\mathsf{PDS}$ in einem Rahmen, der auch von vergleichbarer Hardware erreicht wird. Tabelle 4.1 zeigt eine Übersicht über andere Korrespondenzfindungsverfahren, die verwendete Hardware und die erreichte Rechenleistung. Es zeigt sich, daß wesentlich größere Rechenleistung nur in Verbindung mit mehr, parallel genutzter Hardware erzielt wird.

Autor	Hardware	Verfahren	PDS ($\times 10^6$)
Kayaalp[KE88]	Datacube	Laplace-Gauß Korrelation	6.7
Moll[OBM+93]	PAM	Normalisierte Korrelation	7.4
Kanade[TAK+96]	$5\times$VME-Board	LOG+L_1 Korrelation	30.0
Konolige[Kon97]	DSP 33MHz	LOG+L_1	1.8
Woodfill[WH97]	$16\times$ FPGA 33MHz	Census Transformation	77.0
Perwass[PS02]	AMD 1500MHZ	pdf Iteration	0.05
vorliegende Arbeit	FPGA 20MHz	pdf Iteration	1.0

Tabelle 4.1: Geschwindigkeitsvergleich

Kapitel 5

Schlußfolgerung

Diese Arbeit beschäftigt sich mit der Realisierung eines speziellen Verfahrens zur Korrespondenzfindung. Die Problemstellung war, das Verfahren auf einem FPGA zu implementieren und dabei die besonderen Eigenschaften des Verfahrens bezüglich der Parallelisierbarkeit in Verbindung mit den Möglichkeiten der Hardware zu nutzen.
Es wurde ein System entwickelt, welches unabhängig von außen funktioniert, die Problemstellung erfüllt, in der Anwendung zufriedenstellende Ergebnisse liefert und Vorteile im Vergleich zu einer herkömmlichen Implementierung bietet. Neben der konstruktionsbedingten festen Annahme zweier Parameter ist das System auf die benötigten Bilddaten parametrisierbar. Festgelegt ist zum Einen die Größe des verwendeten Suchraums, um die Komplexität der Schaltung und die Datenmenge zu begrenzen und zum Anderen die Schärfe der verwendeten Ordnungsbedingung, um eine einfach zu realisierende Funktion zu definieren. Die Parallelisierbarkeit des Verfahrens ist in einem sinnvollem Grad verwirklicht worden, der in einem durch äußere Umstände begrenzten Rahmen liegt. Dabei stellten sich besonders der Datentransfer und die Speicherung als begrenzende Faktoren heraus. Die Möglichkeiten der Hardware nutzend wurde das Ziel der Parallelität weiter bis in die Realisierung der einzelnen Rechenschritte verfolgt. Dabei wurde stets ein Kompromis zwischen den gegensätzlichen Faktoren Ressourcenbedarf und Zeitbedarf gesucht, um im vorgegebenen Rahmen zu bleiben.
Damit zusammenhängend wurde eine geeignete Art der Datencodierung gewählt und eine Organisation der Daten entwickelt, die an deren Verarbeitung angepaßt ist. Neben der Konstruktion eines Rechenwerkes, welches mehrfach implementiert wird, wurde ein dazu passendes Steuerwerk entwickelt, so daß sich zusammen ein Prozessor nach dem SIMD-Prinzip realisieren läßt. Dadurch ergibt sich ein System, welches bis auf den nötigen Datenaustausch selbständig arbeitet.
Die Einbettung dieses Systems in einen Rechner zur Anwendung ist durch geeignete Vor- und Nachverarbeitung beschreiben. In verschiedenen Experimenten wurde das System auf Funtionalität getestet. Bis auf sich aus der konstruktionsbedingten Einschränkung des Suchraumes ergebenden entsprechenden Einschränkungen der Ergebnisse sind keine implementationsbedingten Fehler festzustellen. Die verwendete Rechengenauigkeit und der fest gewählte Parameter der Ordnungsbedingung haben sich als geeignet erwiesen. In der Anwendung ergeben sich unter den gesetzten Rahmenbedingungen zufriedenstellen-

de Ergebnisse. Der erzielte Geschwindigkeitsvorteil gegenüber einer existierenden, nicht optimierten Implementation[1] auf einem herkömmlichen, sequentiell arbeitenden Rechner des derzeitigen Standes der Technik[2] liegt bei einem Faktor von $20:1$. Damit liegt das System unter Berücksichtigung des Hardwareaufwandes in einem mit anderen Systemen ähnlicher Realisierung[3] vergleichbaren Rahmen.
Es bleiben die Probleme zu nennen, die während der Entwicklung auftauchten. Neben dem in der Hardware integrierten SRAM, welches nur begrenzte Zugriffsgeschwindigkeit zuläßt, und somit die Verarbeitungsgeschwindigkeit des Systems herunterbremst, ist der PCI-Bus im Datentransfer zwischen FPGA-Karte und Hauptrechner das bremsende Element. Es hat sich gezeigt, daß die Verarbeitungsgeschwindigkeit hätte verdoppelt werden können, sofern das Speicherelement Daten in dieser Geschwindigkeit Lesen und Schreiben könnte. Dieses Problem ist durch die Verwendung der speziellen Hardware *microEnable* bedingt. Unabhängig davon ist durch fehlende Treiberunterstützung seitens des Herstellers [Sil] ein schneller Datentransfer über den PCI-Bus mittels DMA in Verbindung mit dem verwendeten Betriebssystem *Linux* nicht möglich. Hier ist ein großer Zeitverlust in Kauf zu nehmen. Der Transfer der in unserem Fall benötigten Datenmenge nimmt daher ebensoviel Zeit in Anspruch, wie deren interne Verarbeitung. Dieses Problem konnte auch durch den Transfer in großen zusammenhängenden Blöcken und unter Verwendung von FiFo-Zwischenspeichern nicht zufriedenstellend gelöst werden. Weiter ist der in der Implementierung auf 3×3 Positionen eingeschränkte Suchraum für Korrespondenzen bei realen Anwendungen zu klein. Mit üblicher Kameraauflösung von 256×256 Pixeln sind im Fall von Stereo-Sehen im Allgemeinen größere Disparitäten zu erwarten und auch im optischen Fluß ist bei der erzielten Bildrate mit größeren Bewegungen zu rechnen.

5.1 Ausblick

Es hat sich gezeigt, daß das entwickelte System eine sowohl effektive als auch effiziente Umsetzung des Korrespondenzfindungsverfahren von Perwass und Sommer [PS02] ist. Abschliessend sollen nun Verbesserungsvorschläge und Anregungen zu weiteren Projekten in diesem Bereich genannt werden.
Wie in Abschnitt 3.1 angesprochen ist die Vorverarbeitung in dieser Implementation nicht in den FPGA integriert. Dies ließ sich mit der Realisierung der Iterationsschritte nicht vereinbaren, da die Hardware-Ressourcen begrenzt sind. Durch die schnelle Rekonfigurierbarkeit des FPGAs ist es allerdings möglich, die Prozessorarchitektur innerhalb von $20ms$ neu zu definieren. Dies kann zusammen mit der Verwendung des statischen Speicherelementes nach Scalera [SML98] vorteilhaft genutzt werden, indem auf verschiedene Aufgaben optimierte Architekturen nacheinander zum Einsatz kommen, während auch bei einem Wechsel der Architektur die bearbeitete Datenmenge im Speicher zur Verfügung stehenbleibt. So kann der zeitaufwendige Datentransfer über den PCI-Bus verringert wer-

[1] *Acre*: verwendet in [PS02] und verfügbar unter [Per]
[2] AMD Athlon XP1800+ (1.53GHz) unter Windows XP
[3] siehe unter [TAK+96, KE88, OBM+93, WH97]

den.
Im Zusammenhang mit einer Hardware-Realisierung der Vorverarbeitung ist es auch möglich, die Ausgangsdatenmenge nicht über den PCI-Bus in den FPGA zu transportieren, sondern eine Kamera direkt an die vorhandene externe Schnittstelle der FPGA-Karte anzuschließen. Dies hat den Vorteil, daß dann nur noch die Ergebnisse der Berechnung über den PCI-Bus transportiert werden müssen und der gesamte Datenfluß eine Pipeline bilden kann. Es ist in diesem Fall mit einem Geschwindigkeitsvorteil um den Faktor 2 zu rechnen.
Es ist auch möglich, die Vorverarbeitung selbst zu erweitern, um auch andere Bedingungen außer den Farbähnlichkeiten der Bildpunkte zum Erstellen der initialen Korrespondenzannahmen zu nutzen. Eine Erweiterung wäre die Betrachtung mehrerer Schritte einer Bildfolge, um Ergebnisse aus vorhergehenden Bildern in darauf folgende Berechnungen mit einfließen zu lassen. Im Fall des optischen Flusses ist hierdurch mit einem Stabilitätsvorteil zu rechnen. Die Iteration der Ordnungsbedingung kann unabhängig von der Vorverarbeitung von dem entwickelten System weiterverwendet werden.
Durch die Implementierung der Hardwareschaltungen in Form von C++ Klassen ist deren Weiterverwendung und -entwicklung unabhängig von ihrem speziellen, vorgestellten Einsatzzweck möglich. In Verbindung mit einem FPGA, der größere Hardware-Ressoucen zur Verfügung stellt, wäre eine weitere Vervielfachung der entwickelten Rechenwerke denkbar, so daß mehr Parallelität möglich ist. Die hinzugewonnene Rechenleistung könnte genutzt werden, um weitete Geschwindigkeitsvorteile zu erzielen, oder einen größeren Suchraum zu betrachten. Um ein universelles, echtzeitfähiges System zu erhalten, welches mit $25\mathsf{Hz}$ Bilder nach dem PAL-Standard mit einer Auflösung von 768×576 Pixeln verarbeiten kann, müßte die Rechenleistung allerdings um den Faktor 100 gesteigert werden. Das größere Problem wäre allerdings, die Datenmenge in ebenso erhöhter Geschwindigkeit zu transportieren.
Ein Ansatz zur Lösung des Kommunikationsflaschenhalses ist es, wie B. Draper [DNB$^+$] vorschlägt, einen FPGA nicht auf einer externen Zusatzkarte zu betreiben, sondern ihn in Anlehnung an die Pentium-MMX-Architektur in direkte Verbindung mit der Haupt-CPU zu setzen. So könnte man bestehende Datentransfer- und Speicherungsmechanismen nutzen und zusätzlich eine individuelle, auf ihre Anwendung optimierte Funktionalität ermöglichen.

Literaturverzeichnis

[Bel96] P.N. Belhumeur. A bayesian approach to binocular stereopsis. *International Journal of Computer Vision*, 19:237–262, 1996.

[Bro95] I.N. Bronstein. *Taschenbuch der Mathematik*, Seiten 626, 652. Verlag Harri Deutsch, Frankfurt am Main, zweite Auflage, 1995.

[DNB+] B. Draper, W. Najjar, W. Böhm, J. Hammes, B. Rinker, C. Ross, M. Chawathe und J. Bins. Compiling and optimizing image processing algorithms for fpga's. Dep. of Computer Science, Colorado State University, Fort Collins, USA.

[Gau] W.H. Gaulert. Tutorial: Hardware Description Language VHDL. http://www.vhdl-online.de.

[HS81] B. Horn und B. Schunck. Determining optical flow. In *Artificial Intelligence*, 17, Seiten 185–203. 1981.

[Jäh97] B. Jähne. *Digitale Bildverarbeitung*. Springer Verlag, Heidelberg, vierte Auflage, 1997.

[KE88] A. Kyaalp und J. Eckmann. A pipeline architecture for near real-time stereo range detection. In *Proceedings of SPIE Mobile Robotics III*, Seiten 279–286. 1988.

[Kon97] K. Konolige. Small vision systems: Hardware and implementation. In *International Symposium on Robotics Research*. October 1997.

[Kor02] K. Kornmesser. *The FPGA Development System CHDL*. Lehrstuhl für Informatik V, Universität Mannheim, erste Auflage, 2002.

[KP] I.V. Klotchkov und S. Pedersen. A codedesign case study: Implementing arithmetic functions in fpgas. Technical University of Denmark, Lyngby, 1996.

[Kös01] K. Köser. Fpga design development for real-time edge detection. Cognitive Systems Group, Institut of Computer Science and Applied Mathematics, Christian-Albrechts-Universität of Kiel, 2001.

[Lhu98] M. Lhuillier. Efficient dense matching for textured scenes using region growing, 1998. GRAVIR-IMAG and INRIA Rhone-Alpes, France.

[LQ99] M. Lhuillier und L. Quan. Robust dense matching using local and global geometric constraints, 1999. ZIRST-GRAVIR-IMAG, France.

[MH99] R. Männer und S. Hezel. Schnelle Berechnung von 2D-FIR-Filteroperationen mittels FPGA-Koprozessor microEnable. In *Proceedings of 21. DAGM Symposium, Bonn*. 1999.

[MMMH01] R. Männer, K. Mühlmann, D. Maier und J. Hesser. Calculating dense disparity maps from color stereo images, an efficient implementation. In *CVPR*. 2001. Lehrstuhl für Informatik V, Universität Mannheim.

[MP43] W. McCulloch und W. Pitts. A logical calculus of the ideas immanent in nervous activity. In *Bulletin of Mathematical Biophysics*, Band 5, Seiten 115–133. 1943.

[Nof99] K.-H. Noffz. *microEnable Benutzerhandbuch*. Silicon Software GmbH, erste Auflage, 1999.

[OBM+93] O.Faugeras, B.Hotz, M.Mathieu, T.Vieville, Z.Zhang, P.Fau, E.Theron, L.Moll, G.Berry, J.Vuillemin, P.Bertin und C.Proy. Real-time correlation-based stereo: algorithm, implementation and applications. Technischer Bericht, INRIA, 1993.

[Per] C.B.U. Perwass. Dichte Korrespondenzfindung: Implementierung A.C.R.E. http://www.perwass.de.

[PGR01] C.B.U. Perwass, C. Gebken und T. Rabsch. *The Kiel CHDL Component Library Manual*. Institut für Informatik, Christian Albrechts Universität zu Kiel, erste Auflage, 2001.

[PS01] C.B.U. Perwass und G. Sommer. A fuzzy logic algorithm for dense image point matching. In *Proceedings of Vision Interface*, Seiten 39–47. 2001.

[PS02] C.B.U. Perwass und G. Sommer. Dense Image Point Matching through Propagation of Local Constraints. Technischer Bericht 0205, Institut für Informatik und praktische Mathematik, Christian-Albrechts-Universität, Kiel, 2002.

[QM55] W. Quine und McCluskey. A way to simplify truth funktions. *American Mathematical Monthly*, 62, 1955.

[Qua84] L. Quann. Yosemite sequence. http://www.ai.sri.com/videos, 1984. SRI International, Artificial Intelligence Center.

[SH98] S.J. Sangwine und R.E.N. Horne. *The Color Image Processing Handbook*. Champman & Hall, 1998.

[Sil] Silicon Software GmbH, Mannheim. http://www.silicon-software.de.

[SML98] S.M. Scalera, J. Murray und S. Lease. A mathematical benefit analysis of context switching reconfigurable computing, 1998. Sanders, A Lookhed Martin Comp.

[SS98] D. Scharstein und R. Szeliski. Stereo matching with nonlinear diffusion. *International Journal of Computer Vision*, 28(2):155–174, 1998.

[TAK+96] T.Kanade, A.Yoshida, K.Oda, H.Kano und M.Tanaka. A stereo machine for video-rate dense depth mapping and its new applications. In *IEEE Conference on Computer Vision and Pattern Recognition*, Seiten 196–202. 1996.

[TS02] U. Tietze und C. Schenk. *Halbleiter-Schaltungstechnik*. Springer Verlag, 12. Auflage, 2002.

[Wan97] M. Wannemacher. *Das FPGA-Kochbuch*. International Thomson Publishing, vierte Auflage, 1997.

[WH97] J. Woodfill und B. Von Herzen. Real-time stereo vision on the parts reconfigurable computer. In *IEEE Symposium of Custom Computing Machines*. 1997. Rapid Prototypes, Inc.

[WHZ] J. Woodfill, B. Von Herzen und R. Zabih. Frame-rate robust stereo on a pci board. Computer Science Department, Cornell University, 1999.

[Xil] Xilinx. Inc. http://www.xilinx.com.

[ZEM] A. Zuloaga, J. Ezquerra und J.L. Martin. Hardware architecture for optical flow estimation in real time. University of Basque Country, Bilbao, Spanien, 1998.

[ZEMB] A. Zuloaga, J. Ezquerra, J.L. Martin und U. Bidarte. Optical flow estimator using vhdl for implementation in fpga. University of Basque Country, Bilbao, Spanien, 1998.

An dieser Stelle möchte ich gerne allen danken, die mich bei der Ausführung dieser Arbeit unterstützt haben. Dabei danke ich besonders Christian Perwass für die Betreuung, verbunden mit Diskussion, Anregung und Unterstützung bei der Realisierung der Ideen. Weiter geht mein Dank an die Systembetreuer Gerd Diesner und Henrik Schmidt für die Lösung der technischen Probleme im Zusammenhang mit den Hard- und Softwarekomponenten.

Hiermit bestätige ich, daß ich die vorliegende Arbeit selbständig angefertigt und ausschließlich die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel, den 25. April 2003