HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Zhicheng Wu
Herr
Yuqiang Wang

Detektion von Artefakten in Bildern und deren laufzeitoptimierte Umsetzung auf ARM

Mittweida, 2020

Fakultät: Ingenieurwissenschaften

BACHELORARBEIT

Detektion von Artefakten in Bildern und deren laufzeitoptimierte Umsetzung auf ARM

Autor:

Zhicheng Wu

Yuqiang Wang

Studiengang:

Elektro- und Informationstechnik

Seminargruppe:

EI17sA-BC

Erstprüfer:

Prof. Dr.-Ing. A. Lampe

Zweitprüfer:

M. Sc. H. Xu

Einreichung:

Mittweida, 21.11.2020

Verteidigung/Bewertung:

Mittweida, 2020

Faculty: Ingenieurwissenschaften

BACHELOR THESIS

The detection of artefacts in images and their run-time optimized implementation on ARM CPUs

author:
Zhicheng Wu

Yuqiang Wang

course of studies:
Elektro- und Informationstechnik

seminar group:
EI17sA-BC

first examiner:
Prof. Dr.-Ing. A. Lampe

second examiner:
M. Sc. H. Xu

submission:
Mittweida, 21.11.2020

defence/evaluation:
Mittweida, 2020

Bibliografische Beschreibung:

Zhicheng, Wu und Yuqiang Wang:

Cortex-A9 NEON Media Processing Engine Technical Reference Manual –2020

NEON Programmer's Guide –2013

ARM NEON support in the ARM compiler –2008

58 Seiten, Mittweida, Hochschule Mittweida, Fakultät Elektro- und Informationstechnik, Bachelorarbeit, 2020

Referat:

Die vorliegende Arbeit befasst sich mit Detektion von Artefakten in Bildern und deren laufzeitoptimierte Darstellung auf ARM Prozessoren. Das Ziel dieser Arbeit ist die Entwicklung und Optimierung von Grafiken basierend auf der Cortex-A9 NEON Media Processing Engine. Die verwendete Hardwaregerät ist Raspberry Pi 4 und das Umgebungssystem ist LINUX.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Programmverzeichnis

Abkürzungsverzeichnis

ARM ..... Advanced RISC Machines

RISC ..... Reduced Instruction Set Computer

SIMD ..... Single Instruction Multiple Data

SSH ..... Secure Shell

1 Einleitung

In der Computergrafik sind Artefakte in digitalen Bildern und unerwünschten Anzeigen sichtbar. Es gibt viele Gründe für diesen Fehler, wie z.B. Rauschen, Licht, menschliches Versagen oder Datenverarbeitungsfehler. Das Erkennen dieser Fehler ist in der künstlichen Intelligenz, in der Medizin, in Satellitenbildern und in anderen Bereichen von großer Bedeutung und steht auch im Schwerpunkt dieses Bachelorarbeit. Im Bereich der Gesichtserkennung haben beispielsweise unterschiedliche Lichter und Schatten einen großen Einfluss auf die Bilder.

Um den Zweck der Überprüfung der Artefakte auf dem Bild zu erreichen, wird der entsprechende C-Sprachcode geschrieben.

Aufgrund der großen Anzahl von Bildern und des großen Rechenaufwands kann man den Raspberry Pi verwenden, um die Berechnungsgeschwindigkeit zu erhöhen. Der Raspberry Pi ist leistungsstark und kostengünstig und verfügt über einen ARM-Prozessor mit NEON-Technologie, der mehrere Berechnungen gleichzeitig verarbeiten kann, wodurch sich die Rechenzeit erheblich verkürzt.

Diese Bachelorarbeit ist insgesamt in 7 Kapitel untergliedert.

Im ersten Kapitel werden die Einleitung und die Aufgabenstellung dargestellt.

Im zweiten Kapitel werden einige Grundkenntnisse über Raspberry Pi und den Workflow zum Hochladen und Ausführen von Code vorgestellt.

Das dritte Kapitel befasst sich mit der Bildrotation. Das Grundprinzip und der Code der Bildrotation werden vorgestellt.

Im vierten Kapitel wird hauptsächlich der Matlab-Code zu der Detektion von Artefakten im Bild vorgestellt.

In Kapitel fünf wird der MATLAB-Code von Kapitel 4 analysiert und in Form eines C-Sprachprogramms neu geschrieben.

In Kapitel sechs ersetzen wir einen Teil des C-Sprachcodes durch NEON-Code und vergleichen und bewerten die Laufgeschwindigkeit der beiden.

Das siebente Kapitel ist die Zusammenfassung der Bachelorarbeit.

Herr Wu, Zhicheng schreibt Kapitel 1, 2, 3, 6 und 7.

Herr Wang, Yuqiang beschäftigt mit Kapitel 4, 5, 6 und 7.

2 Raspberry Pi und Workflow

Die für diese Bachelorarbeit erforderliche Hardware-Ausrüstung sind PC und Raspberry Pi, und die erforderliche Software sind Matlab, Xftp 6 und Xshell 6. Xftp 6 ist eine Software zur Dateiübertragung und Xshell 6 ist eine Software zur Fernsteuerung von Raspberry Pi. Matlab wird hauptsächlich verwendet, um die Richtigkeit der experimentellen Ergebnisse zu überprüfen. Es wird in den folgenden Kapiteln vorgestellt.

2.1 Raspberry Pi

2.1.1 Hardware-Informationen

Raspberry Pi ist ein ARM-basiertes Mikrocomputer-Motherboard. Aus der Abbildung 6 kann man erkennen, dessen Größe nur der Größe einer Kreditkarte entspricht und dessen System auf Linux basiert. Es verfügt über alle Grundfunktionen eines PCs und kann an einen Monitor, eine Tastatur, eine Maus und andere Geräte angeschlossen werden. In diesem Bachelorarbeit wird die 4 GB RAM-Version von Raspberry Pi 4B verwendet.

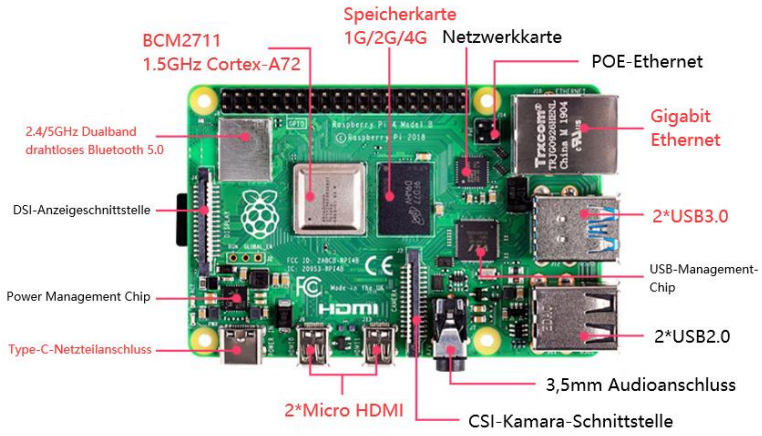

Abbildung 1: Raspberry Pi 4B

Raspberry Pi 4B hat die folgenden Parameter[1]:

- Ausgestattet mit einem 1,5GHz, 64-Bit, Quad-Core-Prozessor (Broadcom BCM2711, Quad-Core-Cortex-A72 (ARM v8) 64-Bit-SoC bei 1,5 GHz)
- 1 GB / 2 GB / 4 GB LPDDR4-Speicher
- Unterstützt Bluetooth 5.0, BLE
- Zwei USB 3.0- und zwei USB 2.0-Anschlüsse

[1] Raspberry Pi 4B Spezifikation: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

- Dualer Micro-HDMI-Ausgang, unterstützt 4K-Auflösung
- Antriebsstrom auf 3A erhöht

*In der Anlage kann man die detaillierte Installation und Inbetriebnahme von Raspberry Pi finden. (siehe Anlage 1: Installation und Inbetriebnahme)

Raspberry Pi hat die folgenden Vor- und Nachteile:

Vorteile:

- kleines Volumen
- kostengünstig
- bestimmte PC-Aufgaben und -Anwendungen ausführen kann

Nachteile:

- Es kann nicht mit der Leistung des PCs verglichen werden und kann keine komplexen Computeranwendungen ausführen.

2.1.2 ARM-Prozessor und NEON

Raspberry Pi 4 Modell B verwendet einen ARM-Prozessor wie unten gezeigt. Es handelt sich um eine RISC-Prozessorarchitektur (reduzierter Befehlssatz), die in vielen Designs eingebetteter Systeme weit verbreitet ist. „Reduced Instruction Set Computer (RISC, englisch für Rechner mit reduziertem Befehlssatz) ist eine Designphilosophie für Computerprozessoren“[4]. Der größte Vorteil des ARM-Prozessors besteht darin, dass es NEON-Anweisungen ausführen kann.

Abbildung 2: ARM-Prozessor

NEON ist eine SIMD-basierte Technologie. SIMD bedeutet, es kann mehrere Daten gleichzeitig mit einer Anweisung verarbeiten, und die Berechnungsgeschwindigkeit wird erheblich verbessert. Neon hat ein 64-bit D-Register (Doppelregister) oder ein 128-bit Q-Register (Quad-Register). Das bedeutet, dass zwei Single-Register zusammen als ein 64-bit-

D-Register benutzen können und vier Single-Register zusammen als ein 128-bit-Q-Register benutzen können. Ein 64-Bit-D-Register enthält[2]:

- 8x8-bit Ganzzahldaten
- 4x16-bit Ganzzahldaten
- 2x32-bit Ganzzahldaten
- 1x64-bit Ganzzahldaten

Ein 128-bit-Q-Register enthält:

- 16x8-bit Ganzzahldaten
- 8x16-bit Ganzzahldaten
- 4x32-bit Ganzzahldaten
- 2x64-bit Ganzzahldaten

NEON-Anweisungen sind nicht kompliziert, man kann „NEON Intrinsics“[1] verwenden, um NEON-Programme zu schreiben. Vorher sollte man den NEON-Datentyp definieren. NEON-Vektordatentypen werden nachfolgendem Format benannt:

<Basis><W>x<L>_t

In dem:

- <Basis> bezieht sich auf den grundlegenden Datentyp. Es enthält hauptsächlich int(Integer), uint(unsigned int), float(floating-point).
- <W> ist die Breite des Grundtyps.
- <L> ist die gespeicherte Datenmenge.

Zum Beispiel:

uint16x4_t ist ein Register, der 4 Gruppen von vorzeichenlosen 16-Bit-Ganzzahlen enthält, es gehört zum 64-Bit-D-Register.

Um das Beschleunigungsprinzip von NEON besser zu verstehen, jetzt sind 3 Gruppen von int32x4_t definiert[1]. Wie in der Abbildung 8 gezeigt, hat jedes Register 4 Elemente und die Größe jedes Elements beträgt 32-bit:

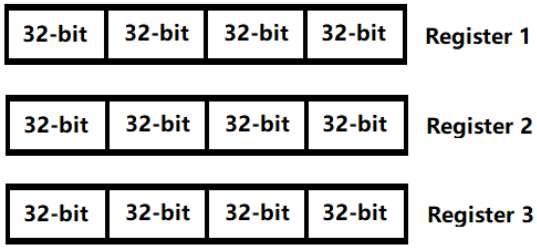

Abbildung 3: Das Beschleunigungsprinzip von NEON

Bei der Abbildung 9 handelt es sich um die Addition des NEON-Registers. Wenn man Ganzzahldaten von 1 bis 8 in Register 1 und Register 2 lädt und ihre Summe in Register 3 lädt, kann das NEON-Register gleichzeitig 1 + 5, 2 + 6, 3 + 7, 4 + 8 berechnen und das Ergebnis in Register 3 speichern, es ist nur eine Berechnung erforderlich. Aber die allgemeinen Register müssen nacheinander hinzugefügt werden, insgesamt 4 Berechnungen.

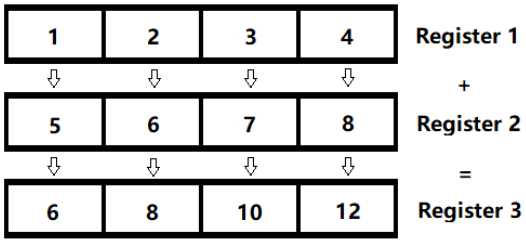

Abbildung 4: die Addition des NEON-Registers

Das bedeutet, dass NEON viele Daten gleichzeitig einlesen und bearbeiten kann, deshalb eignet es sich zur Optimierung der Rechengeschwindigkeit.

ARM-Prozessor und NEON haben die folgenden Vor- und Nachteile:

Vorteile:

- Eine große Anzahl von Registern wird verwendet, und die Befehlsausführungsgeschwindigkeit ist schneller.
- Der Adressierungsmodus ist flexibel und einfach, und die Ausführungseffizienz ist hoch.
- Schnelle Rechengeschwindigkeit

Nachteile:

- Aufgrund seines geringen Stromverbrauchs ist es schwierig, große und komplexe Berechnungen durchzuführen.
- Vor und nach der Operation dauert es einige Zeit, Daten aus dem allgemeinen Register in das NEON-Register zu lesen und zu schreiben.

2.1.3 Allgemeine Linux-Befehle und -Anweisungen

Im Folgenden gibt es einige häufig verwendete Linux-Befehle und Anweisungen[3], die nur als Referenz dienen:

- ***sudo reboot***
  die Raspberry Pi neustarten

- ***sudo poweroff***
  Ausschalten

- ***sudo ifconfig -a***

  die Netzwerkkonfigurationsinformationen des Raspberry Pi auflisten

- ***cd ~***

  in das Home-Verzeichnis des aktuellen Benutzers wechseln

- ***sudo shutdown -h 01:05***
  Timing Shutdown: Mit dem obigen Befehl wird die Shutdown-Zeit auf 1:05 Uhr morgens eingestellt.

- ***sudo halt***
  Ausschalten. Im Gegensatz zu ‘‘poweroff ‘‘ stoppt dieser Befehl alle CPU-Funktionen vor dem Herunterfahren. Bei der Ausführung werden alle Anwendungsprozesse beendet und der Synchronisierungssystemaufruf ausgeführt. Es wird empfohlen, diese Methode zum Herunterfahren zu verwenden.
- ***clear***
  der Text auf dem Terminal löschen
- ***cd / neontest / test***
  Gehe zum Katalog / neontest / test
- ***sudo raspi-config***
  die Konfigurationsoberfläche der Raspberry Pi-Funktion öffnen

Hinweis: Fügen Sie *sudo* vor einer Befehlszeile hinzu. Dies bedeutet, dass Sie diesen Befehl als Systemadministrator ausführen müssen.

2.2 Workflow

Es gibt normalerweise zwei Möglichkeiten, den Raspberry Pi zu steuern. Zum einen müssen Monitor, Tastatur und Maus direkt mit dem Raspberry Pi verbunden werden, zum anderen soll der Raspberry Pi mithilfe von Software wie Xshell oder der Remotedesktopverbindung des Microsoft-Systems (siehe Anlage 2: Remotedesktopverbindung) ferngesteuert werden. In dieser Bachelorarbeit wird Xshell 6 zur Fernsteuerung des Raspberry Pi verwendet.

Die detaillierten Verfahren zum Hochladen und Ausführen sind wie folgt:

1. die Raspberry Pi einschalten
2. Da für diese Arbeit keine grafische Benutzeroberfläche erforderlich ist, benötigt man nur ein Terminal. Xshell 6 wird verwendet, um sich remote am Terminal des Raspberry Pi anzumelden und damit Befehle und Programme auszuführen. Die Vorteile sind die einfache Benutzeroberfläche, die leistungsstarken Funktionen und die hohe Sicherheit. Hinweis: Die Voraussetzung für die Verwendung der Remote-Anmeldung ist die Aktivierung des SSH-Dienstes (SSH, English für Secure Shell). Der SSH-Dienst ist ein Protokoll, das Sicherheit für die Remote-Anmeldung und andere Netzwerkdienste bietet. (siehe Anlage 3: SSH-Verbindung)

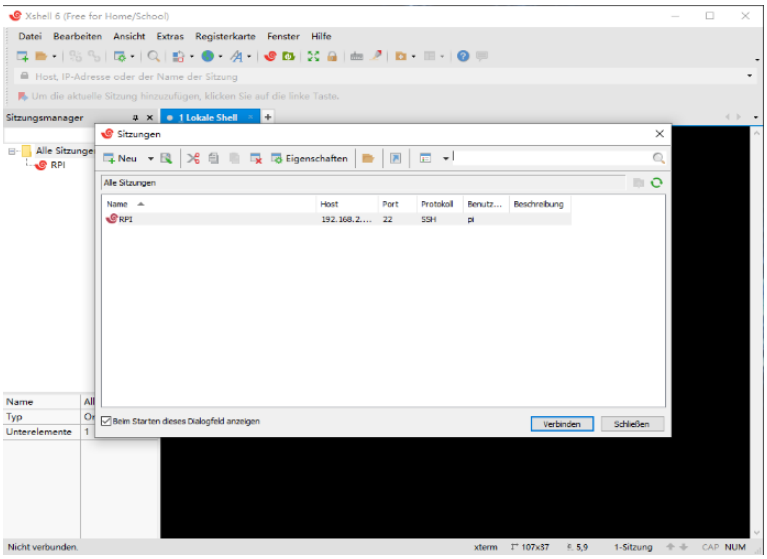

Abbildung 5: Xshell 6

3. Xftp 6 wird zum Hoch- und Herunterladen von Dateien verwendet. Die Datei kann jedoch nicht direkt ausgeführt werden, sondern muss kompiliert werden, um ein

ausführbares Programm zu generieren. Wie in der folgenden Abbildung gezeigt, kann man Dateien direkt vom PC auf das Raspberry Pi-System kopieren.

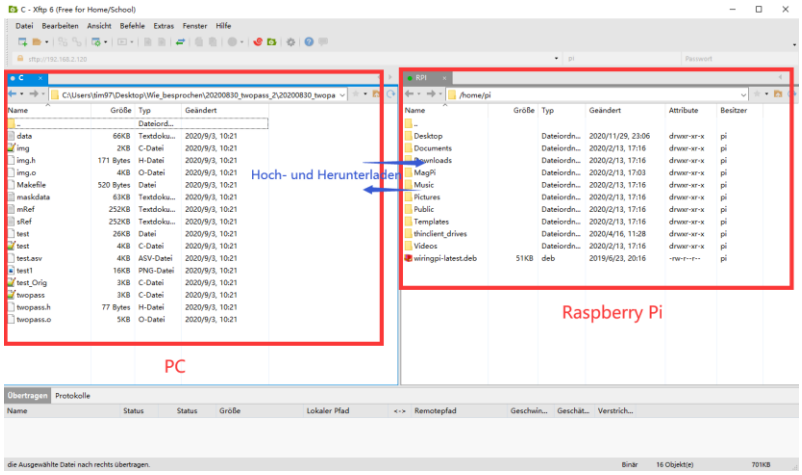

Abbildung 6: Xftp 6

4. Im Allgemeinen können Quellcodedateien in C-Sprache vom Raspberry Pi nicht erkannt werden. Einige Kompilierungsbefehle sollten zuerst ausgeführt werden, und dann kann der C-Sprachcode ausgeführt werden. Aufgrund der großen Anzahl von Programmierbefehlen kann man alle Programmierbefehle im Voraus in einer Datei namens *Makefile* speichern. Man kann die Funktion der automatischen Kompilierung über die Makefile-Datei realisieren, sodass der Kompilierungsbefehl nicht manuell eingegeben werden muss.

   Um die Bequemlichkeit von *Makefile* zu erklären, wurde ein kleines Programm geschrieben. Wie unten in der Abbildung gezeigt, sind die Befehle im roten Feld notwendige Kompilierungsbefehle für dieses Programm, und man muss diese Befehle eingeben, bevor man das Hauptprogramm ausführt. Wenn der Kompilierungsbefehl ausgeführt wird, wird im Programmordner eine ausführbare Datei mit dem Namen *main* generiert. Der Dateiname ist grün. Dies bedeutet, dass das Programm erfolgreich kompiliert wurde und auf dem Raspberry Pi ausgeführt werden kann.

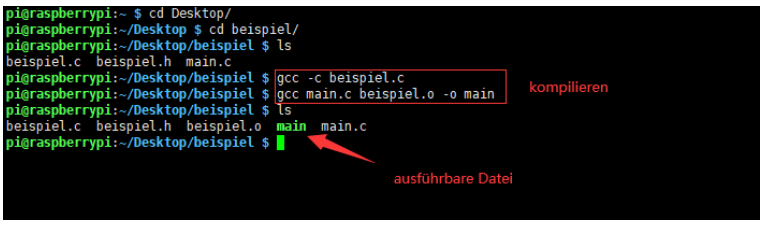

Abbildung 7: Programm kompilieren (Beispiel)

Dieses Programm ist eine Makefile-Datei, die im Voraus erstellt und mit dem Befehl *make* aufgerufen wurde.

```
1.  CC=gcc      //C-Compiler ist gleich gcc
2.
3.  main:main.c beispiel.o
4.   $(CC) main.c beispiel.o -o main     //das Hauptprogramm definieren
5.                                       //der Programmname lautet Beispiel
6.  beispiel.o:beispiel.c
7.   $(CC) -c beispiel.c
```

Programm 1: makefile(Beispiel)

Wie in der Abbildung gezeigt, es benötigt nur einen Befehl *make*, und das gesamte Projekt wird automatisch kompiliert, um eine ausführbare Datei zu generieren. Dies verbessert die Effizienz erheblich.

```
pi@raspberrypi:~/Desktop/beispiel $ ls
beispiel.c  beispiel.h  main.c  makefile
pi@raspberrypi:~/Desktop/beispiel $ make        nur ein Befehl
gcc -c beispiel.c
gcc main.c beispiel.o -o main
pi@raspberrypi:~/Desktop/beispiel $ ls
beispiel.c  beispiel.h  beispiel.o  main  main.c  makefile
pi@raspberrypi:~/Desktop/beispiel $
```

Abbildung 8: mit „make“

Dieses Bild zeigt die Ergebnisse dieses Programms. Wenn es im Terminal ausgeführt wird, wird der Satz „Bleib Gesund“ angezeigt. Das Programm wird korrekt ausgeführt:

```
pi@raspberrypi:~/Desktop/beispiel $ make
make: 'main' is up to date.
pi@raspberrypi:~/Desktop/beispiel $ ./main
Bleib gesund
```

Abbildung 9: Ergebnis

3 Rotation

Da einige Bilder schräg sind, sollte man sie zuerst drehen. In diesem Kapitel wird es erläutert, wie die Drehung des Bildes mit dem C-Programm realisiert wird. Zuerst verwendet man Matlab, um das Originalbild in ein Graustufenbild umzuwandeln und es um 45 Grad gegen den Uhrzeigersinn zu drehen (Rotation), und verwendet dann das C-Programm, um es um 45 Grad im Uhrzeigersinn zu drehen (Derotation). Wenn das endgültige Bild einen normalen und keinen Neigungswinkel zeigt, ist das Experiment erfolgreich. Der gesamte Prozess wird durch Matlab zeigt.

Das Matlab Programm wird bei Prof. Lampe geboten. In Anlage 4 findet man den vollständigen Code. (siehe Anlage 4: Matlab-Code für Rotation)

3.1 Das Prinzip der Rotation

Das Prinzip der Rotation ist relativ einfach. Im zweidimensionalen Raum kann die Drehung durch einen einzelnen Winkel θ definiert werden. Konventionell bedeutet ein positiver Winkel eine Drehung gegen den Uhrzeigersinn und ein negativer Winkel eine Drehung im Uhrzeigersinn. Zuerst setzt man das Koordinatensystem in die Mitte des Bildlaufbildes. Dann dreht man der Pixel (a, b) im Uhrzeigersinn relativ zum Ursprung um einen Winkel θ und erhalten schließlich ein neuer Pixel (A, B).

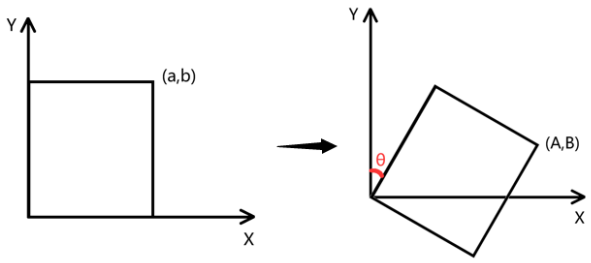

Abbildung 10: Allgemeine Methode

In der linearen Algebra ist eine Rotationsmatrix eine Transformationsmatrix, die verwendet wird, um eine Rotation in einem zweidimensionalen Raum durchzuführen. Verwendet man beispielsweise die folgende Konvention, die Matrix:

$$\begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix}$$

Man dreht einen Punkt in der xy-Ebene im Uhrzeigersinn um einen Winkel von θ relativ zur x-Achse. Um die Drehung auf einem ebenen Punkt mit Standardkoordinaten v = (x, y) durchzuführen, sollte dieser als Spaltenvektor geschrieben und mit der Matrix R multipliziert werden:

$$\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \tag{3-1}$$

$$A = cos\theta \cdot a + sin\theta \cdot b \tag{3-2}$$

$$B = -sin\theta \cdot a + cos\theta \cdot b$$

Für einen beliebigen Punkt (a + m, b + n) kann die folgende Formel erhalten werden

$$A' = cos\theta \cdot (a + m) + sin\theta \cdot (b + n) = A + m \cdot cos\theta + n \cdot sin\theta \tag{3-3}$$

$$B' = -sin\theta \cdot (a + m) + cos\theta \cdot (b + n) = B - m \cdot sin\theta + n \cdot cos\theta$$

Diese Formel kann als aktive Drehung des Vektors im zweidimensionalen Raum oder als passive Drehung des Koordinatensystems interpretiert werden.

Wie im roten Teil der Abbildung unten gezeigt. Da einige Pixel des gedrehten Bildes negative Koordinaten haben, was im Programm nicht zulässig ist, muss das Bild verkleinert werden.

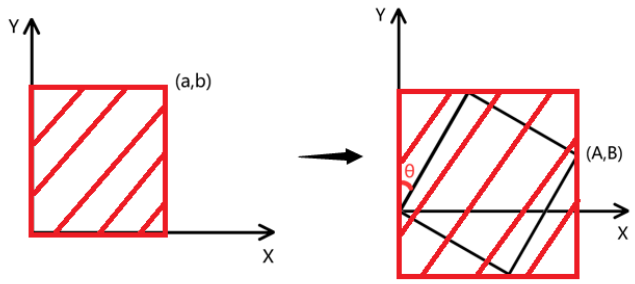

Abbildung 11: Gründe, das Bild zu verkleinern

Für die Bildskalierung wird häufig „Interpolation für den nächsten Nachbarn“ verwendet. Die Vorteile sind einfach und schnell. Wie in der folgenden Tabelle gezeigt, wird ein 4*4-Bild auf ein 2*2-Bild verkleinert.

Die Schritte bestehen darin, dass die Pixelwerte von 4 benachbarten Positionen in einem 4*4-Bild durch 4 geteilt und dann in ein 2 * 2-Bild geladen werden. „pos“ bedeutet Position. Man nimmt den Durchschnitt von jeweils 4 benachbarten Pixeln.

pos	**pos+1**		
pos+x	**pos+x+1**		

Tabelle 1: Interpolation für den nächsten Nachbarn

Zum Beispiel nimmt man den Durchschnitt der 4 Pixel im grünen Teil (pos, pos+1, pos+x, pos+x+1) und erhalten schließlich den Pixelwert in pos2. Das Ergebnis von Interpolation für den nächsten Nachbarn:

pos2	

Tabelle 2: Ergebnis von Interpolation für den nächsten Nachbarn

3.2 Programmablaufplan

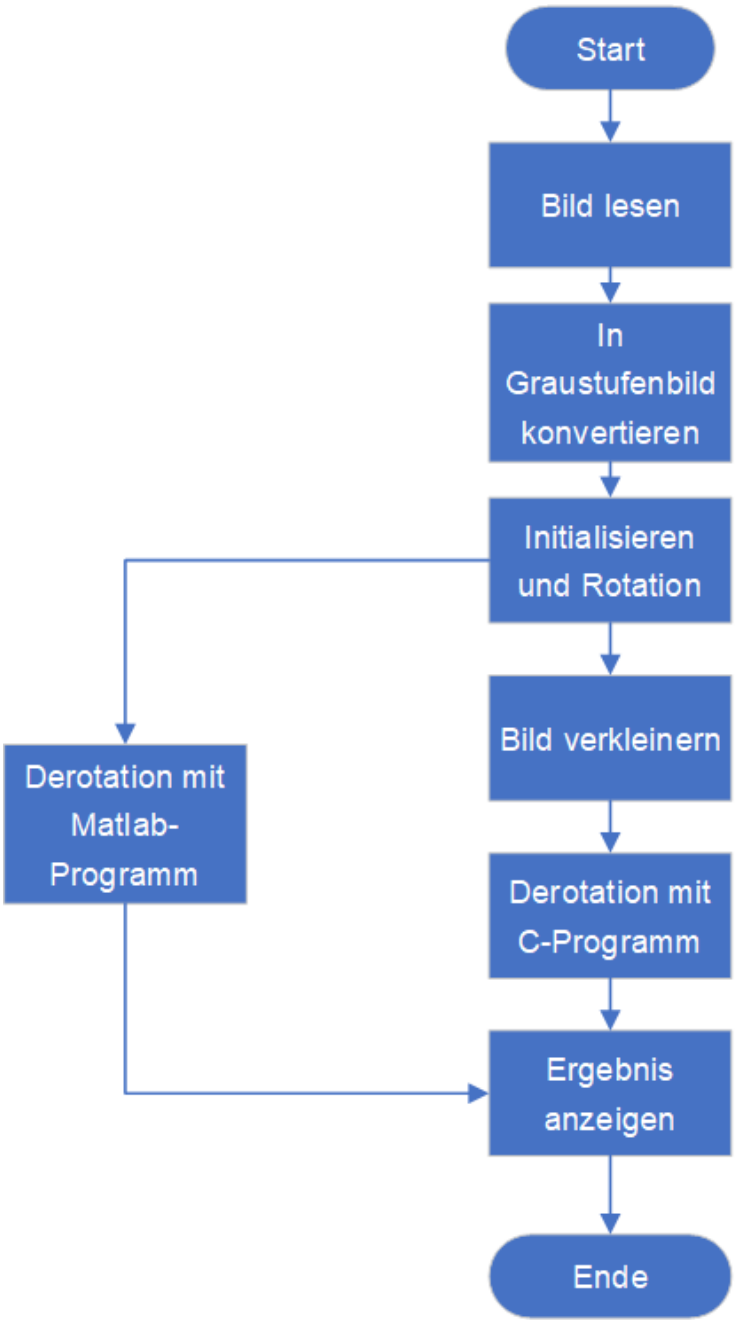

Abbildung 12: Programmablaufplan

3.3 Rotation mit Matlab

3.3.1 Bild lesen mit Matlab

Man benutzt Matlab, um das Bild zu lesen.

```
1.  close all     //alle Figurenfenster schließen
2.  clear      //alle Variablen im Arbeitsbereich löschen
3.
4.  ImRGB = imread('* filename');    //Name des Bild
5.  ImGray = rgb2gray(ImRGB);    //das RGB-Farbbild in ein Graustufenbild konvertieren
6.
7.  figure(1)              //das Ergebnis zeigen
8.  imshow(ImGray)
```

Programm 2: Bildeinlesen

Bevor man dieses Programm ausführt, sollten Bilder und Codes in einem Ordner gespeichert werden.

Die Eingangsvariable *filename* ist der Name des Bildes.

Abbildung 13: das Originalbild

Um die Anzahl unnötiger Berechnungen zu reduzieren, verwendet das gesamte Projekt Graustufenbild. Man konvertiert das RGB-Farbbild oben in das Graustufenbild unten:

Abbildung 14: das Graustufenbild

3.3.2 gedrehtes Bild erstellen

```
rotAngleGrad = 45;   // rotation angle in grad,  45 Grad ist gegen den Uhrzeigersinn

ImRot = imrotate(ImGray(:,:),rotAngleGrad,'nearest');    // gedrehtes Bild erstellen
//'nearest': Interpolation des nächsten Nachbarn
figure(2)     //das Ergebnis zeigen
imshow(ImRot)
```

Programm 3: gedrehtes Bild erstellen mit Matlab

Das Ergebnis der Erstellung eines gedrehten Bildes mit Matlab:

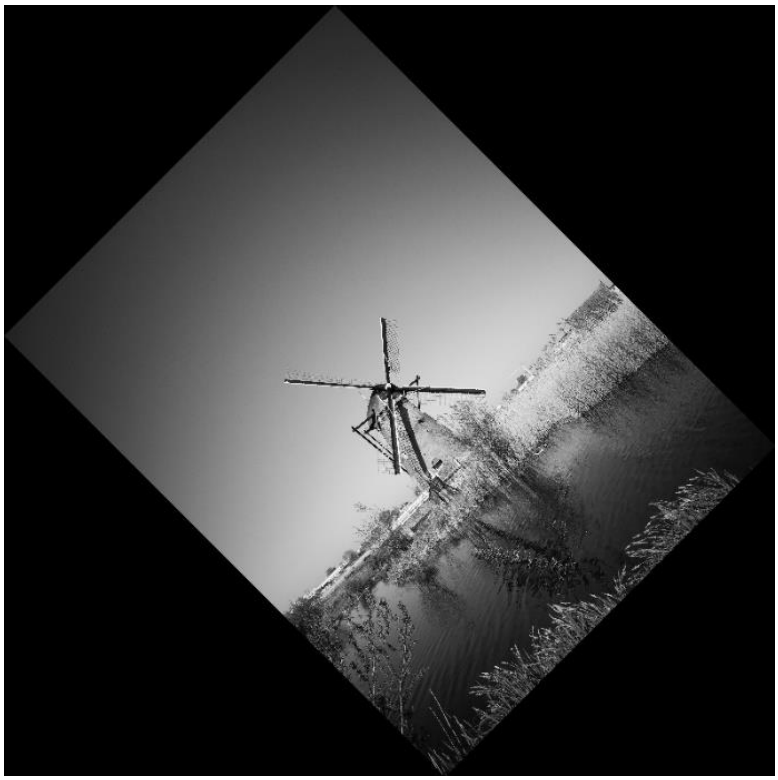

Abbildung 15: gedrehtes Bild erstellen mit Matlab

3.4 Derotation mit C-Programm

3.4.1 Variable setzen

```
1.  void rotation_rad(png_bytep m1,unsigned int a,unsigned int b,double rad,int output[a
    *b]){
2.   double y_acc_y, x_acc_y, y_acc_x, x_acc_x;     //Akkumulator initialisieren
3.   double c1x, c2x, c1y, c2y;
    //double ist ein Datentyp, es kann positiv oder negativ sein
4.   int pos,i,j;     //pos= Position
5.   unsigned int x=2*a;
6.   unsigned int y=2*b;  //m1 ist das Eingabebild, die Länge ist x und die Breite ist y
7.   int m2[a*b];
8.  //Da die Grenze des Bildes m1 nach dem Drehen des Originalbilds größer wird,
    muss man m1 um die Hälfte reduzieren, was m2 ist, seine Länge ist a und Breite ist b
9.
10.  c1y = cos(pi/2-rad);
11.  c2y = sin(pi/2-rad);
12.  c1x = cos(rad);
13.  c2x = sin(rad);
14.  y_acc_y = ceil(c2x*b)-1;   //die Zahl auf die nächste ganze Zahl verringern
15.  x_acc_y = 0;
```

Programm 4: Variable setzen

Zeile 1: „Der Radiant (Einheitenzeichen: rad) ist ein Winkelmaß, bei dem der Winkel durch die Länge des entsprechenden Kreisbogens im Einheitskreis angegeben wird. “

Umrechnungsmethode von Radianten und Winkel:

$$1rad = \frac{180°}{\pi} \approx 57.3° \tag{3-4}$$

Zeile 2 und Zeile 3: Der Akkumulator ist ein Register, in dem Zwischenergebnisse für Arithmetik und Logik gespeichert sind. Wenn man die Formeln von (3-3) vergleicht, kann man die Beziehung in der folgenden Abbildung erhalten:

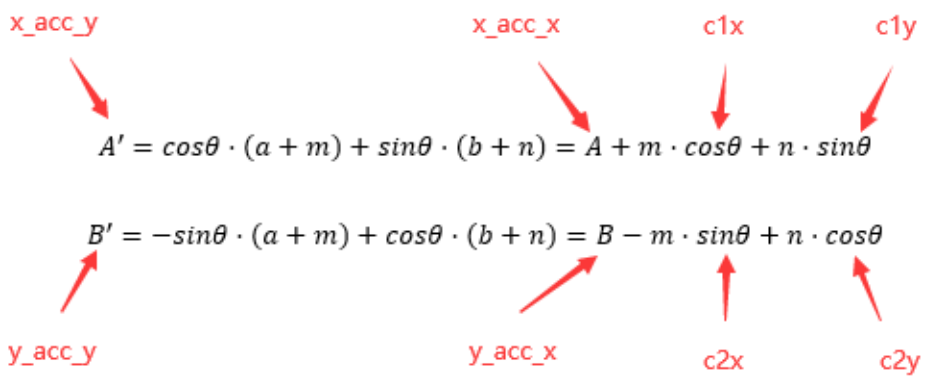

Abbildung 16: Die Bedeutung von 8 Variablen

Zeile 4: *pos* bedeutet die eindimensionale Koordinate des Punktes im Bild im Code. (i, j) bedeutet die zweidimensionalen Koordinaten des Punktes im Bild. i repräsentiert die Abszisse und j repräsentiert die Ordinate.

Zeile 14: Die *Ceil* wird in diesem Programm verwendet und dient dazu, die kleinste Ganzzahl zu verwenden, die größer als der Koordinatenwert ist. (z.B. 3.2→4, 6,8→7) Die Genauigkeit ist nicht so gut, aber genug für das Programm.

3.4.2 Bild verkleinern

```
1.  // Unterabtastung des Eingabebildes um den Faktor 4
2.  for(i=0; i<y; i+= 2){
3.      for(j=0; j<x; j+= 2){
4.          pos = i*x+j;          // zweidimensionale Koordinaten in eindimensionale Ko
    ordinaten konvertieren
5.          pos2=i/2*a+j/2;
6.          m2[pos2] = (m1[pos]+m1[pos+1]+
```

```
7.                  m1[pos+x]+m1[pos+x+1])>>2;     //die Summe der Werte von 4 Position
    en durch 4
8.       }
9.   }
```

Programm 5: Bild verkleinern

In der siebten Zeile des Programms: >> 2 bedeutet, dass die Binärziffer durch zwei nach rechts verschoben wird, d.h. diese Zahl durch 4 geteilt wird.

3.4.3 Derotation

```
1.      // Schleife in y-Richtung
2.      for(i=0;i< b-10;i++){
3.
4.          // Schleife in x-Richtung
5.          y_acc_x = y_acc_y;
6.          x_acc_x = x_acc_y;
7.
8.          // Schleife in x-Richtung
9.          for(j=0; j < a-10; j++){
10.
11.             output[i*a+j] = m2[(int)(y_acc_x+0.5)*a+(int)(x_acc_x+0.5)];
12.
13.             // Inkrementzähler in x-Richtung
14.             x_acc_x = x_acc_x + c1x;     // Nach Formel 3-5
15.             y_acc_x = y_acc_x - c2x;
16.         }
17.
18.         // Inkrementzähler in y-Richtung
19.         x_acc_y = x_acc_y + c1y;     // Nach Formel 3-6
20.         y_acc_y = y_acc_y + c2y;
21.     }
```

Programm 6: Derotation

Für jeden Punkt (a, b) im Bild sind die rechten und unteren Koordinaten wie in der Tabelle gezeigt:

(a, b)	(a+1, b)
(a, b+1)	…

Tabelle 3: Rotation

Nach Formel (3-3) gilt für (a + 1, b), wenn m = 1 und n = 0:

$$A' = cos\theta \cdot (a+1) + sin\theta \cdot (b+0) = A + cos\theta \tag{3-5}$$

$$B' = -sin\theta \cdot (a+1) + cos\theta \cdot (b+0) = B - sin\theta$$

Nach Formel (3-3) gilt für (a, b+ 1), wenn m = 0 und n = 1:

$$A' = cos\theta \cdot (a+0) + sin\theta \cdot (b+1) = A + sin\theta \tag{3-6}$$

$$B' = -sin\theta \cdot (a+0) + cos\theta \cdot (b+1) = B + cos\theta$$

Anzeige des Ergebnisses:

Abbildung 17: Rotationsergebnisses

Dieses Ergebnis zeigt, dass der Derotationsteil des C-Sprachprogramms gute Ergebnisse erzielt hat.

3.5 Zeitmessung

Aufgrund der unterschiedlichen Bildgröße kann auch die Laufzeit des Programms unterschiedlich sein. Also macht man die Zeitmessung und die Testergebnisse sind wie folgt:

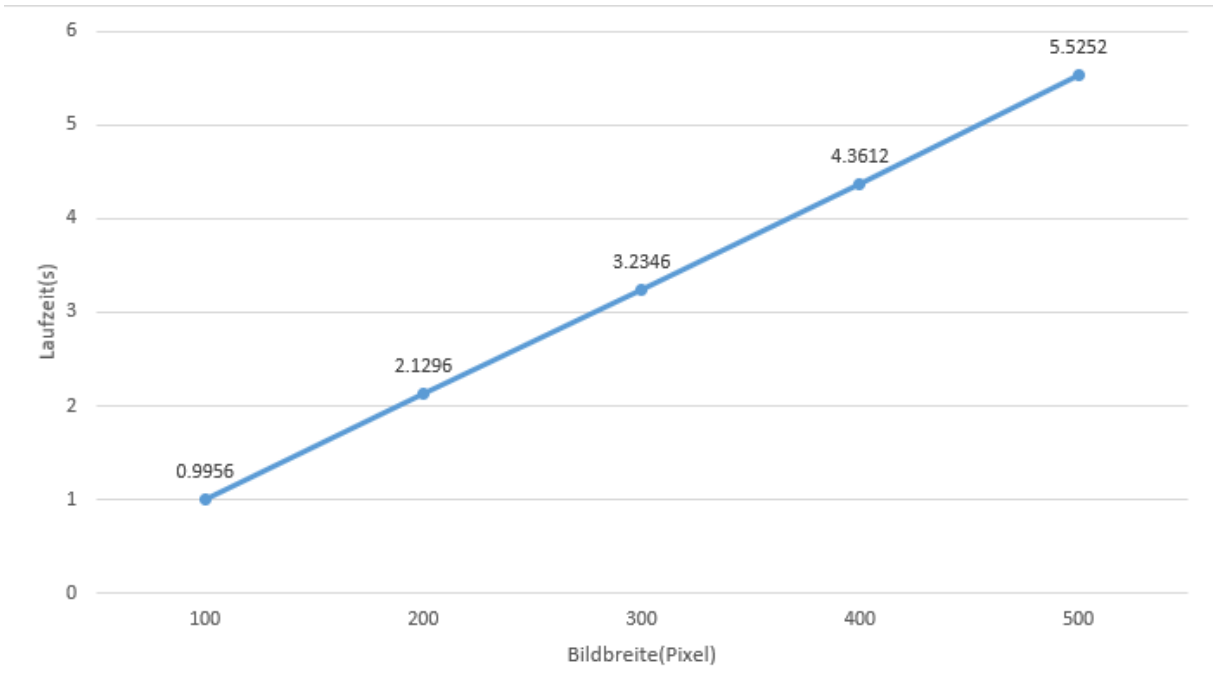

Abbildung 18: Zeitmessung

In der obigen Abbildung ist die Abszisse die Breite des Bildes (die Bildlänge ist auf 600 Pixel festgelegt) und die Ordinate ist die Laufzeit. Man kann aus dem Diagramm erkennen, dass die Laufzeit einen linearen Zusammenhang mit der Größe des Bildes hat. Je größer das Bild, desto länger die Laufzeit.

4 Artefakten und der Matlab-Code

4.1 Prinzip der Detektion

Als Probe werden viele Gesichtsfotos derselben Mannes ausgewählt. Diese Bilder wurden im Voraus zu Graustufenbildern verarbeitet. Im Allgemeinen besteht ein Bild aus drei Farben: Rot(R), Grün(G) und Blau(B), jeder Farbewertbereich liegt zwischen 0 und 255. Bei Graustufenbildern sind alle drei Werte gleich, und die Leute nennen diesen Wert den Grauwert.

$$Grauwert = R = G = B$$

Abbildung 19: Stichprobe

Es ist aus dem Bild ersichtlich, dass die Lichtintensität und der Lichtwinkel in den Beispielfotos unterschiedlich sind. Um die abnormalen Punkte aus jedem Foto der Probe zu finden, ist eine Referenz erforderlich.

Als Referenz berechnet man den Durchschnittswert der Pixel an derselben Position in jedem Bild. Nachdem jede Position berechnet wurde, ist das Referenzbild verfügbar. Um abnormale Punkte zu finden soll man nur das Bild mit dem Referenzbild vergleichen und deren Grauwerte subtrahieren. Je größer der Differenz ist, desto größer ist der Dispersionsgrad. Dann sind die Varianz und die Standardabweichung wertvoller, weil sie den Grad der Streuung zwischen den Probendaten und die Referenz besser zeigen können.

$$Referenz\ Mittelwert[x,y] = \frac{1}{N}\sum Bildgrauwert[x,y] \tag{4-1}$$

$$Referenz\ Varianz:\ \sigma^2[x,y] = \frac{1}{N}\sum(Bildgrauwert[x,y] - Referenz\ Mittelwert[x,y])^2 \tag{4-2}$$

$$Standardabweichung = \sigma$$

$$N\ steht\ für\ die\ Anzahl\ der\ Bilder$$

$$x\ und\ y\ sind\ die\ Spalten\ und\ Zeilen, indenen\ sich\ das\ Pixel\ befindet$$

Für ein einzelnes Bild sollte die folgenden Formeln verwendet werden, um den Mittelwert und die Standardabweichung zu berechnen.

$$Mittelwert = \frac{1}{Bildzeilen * Bildspalten}\sum Bildgrauwert(x,y) \tag{4-3}$$

$$Standardabweichung = \sqrt{\frac{1}{Bildzeilen * Bildspalten}\sum(Bildgrauwert(x,y))^2 - Mittelwert * Mittelwert}$$

4.2 Implementierung

Die Detektion wird schon durch ein Matlab-Programm erreicht. Der allgemeine Ablauf des Programms ist wie folgt:

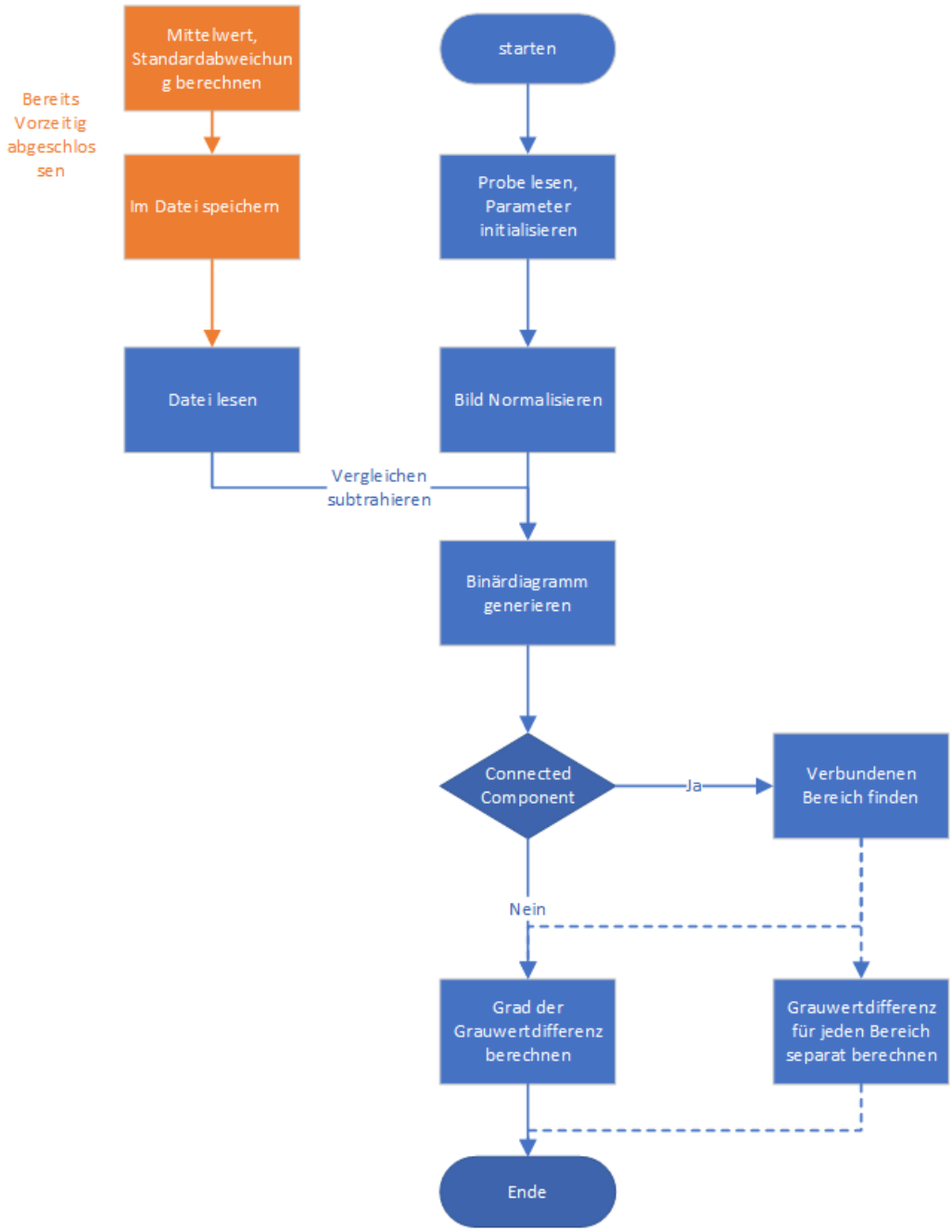

Abbildung 20: Programmablauf

Als nächstes folgt die ausführliche Einführung des Programms.

Das Programm in diesem Kapitel wird von Prof. Lampe geboten.

4.2.1 Vorbereitungsteil

In der Software Matlab werden alle Parameter im Workspace gespeichert. Vor Arbeitsbeginn sollen alle Elemente im Workspace gelöscht und andere von Matlab generierte Fenster geschlossen werden.

Danach werden die erforderlichen Parameter definiert und initialisiert. Dann die Bilder werden geladen.

Der Parameter ‘flagConnectedComponents‘ bestimmt die Richtung des nachfolgenden Programms. Man kann es auf 0 oder 1 setzen.

```
1. clear ;  %% clear all variables
2. close all;
3.
4. flagConnectedComponents = 1; %% set parameters - use or don't use split into smaller
   connected components
```

Programm 7: Initialisierung

Die Proben werden im Ordner ‘yaleB01‘ gespeichert, Dateien im pgm-Format in diesem Ordner werden hier ausgewählt. Der Grauwert aller Bilder wird im dreidimensionalen Vektor ImO gespeichert. Um den gleichen Typ wie andere nachfolgende Daten beizubehalten, wird der Datentyp in double geändert.

```
1. pathName='yaleB01\'; %% load image data
2. files=dir(pathName);
3. Im=[];
4. countIm = 0;
5.
6. for i=3:length(files)
7.     try
8.         if strcmp(files(i).name(end-2:end),'pgm')
9.             countIm = countIm+1;
10.            ImO(:,:,countIm) =double(imread([pathName '\' files(i).name]));
11.        end
12.    end
13. end
```

Programm 8: Bild lesen

Die Mittelwerte und die Varianz wurden im Voraus berechnet und in der Datei ‘StatisticReferences.mat‘ gespeichert. Hier kann man diese Datei direkt laden.

```
1. load('yaleB01\StatisticReferences.mat','meanImRef','stdImRef') %% load statistical
   references
```

```
2.
3.  th = 2; % set threshold in multiples of pixel-wise standard deviation
```

Programm 9: Datei lesen

‘th‘ ist der Schwellenwert, man kann den Erkennungsgrad ändern, indem man diesen Wert einstellt. Alle Differenz, die diesen Schwellenwert überschreiten, werden herausgefiltert, je größer der Wert ist, desto kleiner ist der gefilterte Bereich, weniger Pixels werden herausgesucht.

4.2.2 Mask Pixel Threshold

Das nächste Ziel ist es, dass einen Binärbild zu erstellen. Es kann auch als Matrix betrachtet werden, die nur aus 0 und 1 besteht. Das Binärbild kann die gefilterten Pixel am intuitivsten anzeigen und die nachfolgenden weiteren Berechnungen basieren ebenfalls auf dieses Bild. In diesem Programm heißt diese Matrix ‘Mask Pixel Threshold‘.

```
1.  for i = 1:countIm % loop over test images
2.      ImP = ImO(:,:,i); % load ith image
3.      numberRows = size(ImP,1); % set size of image
4.      numberCols = size(ImP,2);
5.      meanAcc = 0; % calculate mean and standard deviation of image
6.      powAcc = 0;
7.      for r = 1:numberRows  % normalize ith image
8.          for c =  1:numberCols
9.              meanAcc = meanAcc + ImP(r,c);
10.             powAcc = powAcc + ImP(r,c)*ImP(r,c);
11.         end
12.     end
13.     meanIm = meanAcc/numberRows/numberCols;
14.     powIm = powAcc/numberRows/numberCols;
15.     stdIm = sqrt(powIm - meanIm*meanIm);
16.     for r = 1:numberRows % subtract pixel-wise mean image and divide pixel-wise by
    standard deviation
17.         for c =  1:numberCols
18.             ImN(r,c) = (ImP(r,c)-meanIm)./stdIm*15+80;
19.         end
20.     end
21.
22.     for r = 1:numberRows
23.         for c =  1:numberCols
24.             ImwoMean(r,c) = (ImN(r,c) - meanImRef(r,c));
25.             DiffImNorm(r,c) = ImwoMean(r,c)/stdImRef(r,c);
26.         end
27.     end
28.
```

```
29.     maskPixelsThreshold = zeros(numberRows, numberCols); % find pixel indices where
    normalized difference image is larger than threshold and create binary mask
30.     numPixTotal = 0;
31.     for r = 1:numberRows
32.         for c =  1:numberCols
33.             if abs(DiffImNorm(r,c)) >= th
34.                 maskPixelsThreshold(r,c) = 1; % sum up all ones in the binary mask
    to get total number if differing pixels
35.                 numPixTotal = numPixTotal + 1;
36.             else
37.                 maskPixelsThreshold(r,c) = 0;
38.             end
39.         end
40.     end
41.
42.     ...
43.
44. end
```

Programm 10: Normalisierung und maskPixelsThreshold

In der fünften Zeile des Programms wird der Grauwert des Bildes in der Variablen ‘ImP‘ gespeichert. Der Grauwertbereich jedes Pixels liegt zwischen 0 und 255 und der Grauwert ist immer eine ganze Zahl. Dies kann dazu führen, dass die Daten bei der Berechnung der Varianz zu groß für einen Vergleich sind. Zur Vereinfachung der nachfolgenden Datenverarbeitung soll man das Bild normalisieren.

Die Normalisierung besteht darin, die zu verarbeitenden Daten durch einen bestimmten Algorithmus innerhalb eines bestimmten Bereichs zu begrenzen. Die spezifische Funktion der Normalisierung ist die statistische Verteilung einer einheitlichen Probe zusammenzufassen. Die Normalisierung macht nicht vergleichbare Daten vergleichbar sein, während die relative Beziehung zwischen den beiden vergleichenden Daten beibehalten wird.

$$ImN(r,c) = (ImP(r,c) - meanIm)/stdIm * 15 + 80 \tag{4-4}$$

Zur Normalisierung benutzt man diese Formel, denn später man subtrahieren ‘ImN‘ von ‘meanImRef‘ und die Zahlen in ‘meanImRef‘ ist einige statische Werte, ‘ImN‘ ist der normalisierte Wert. Offensichtlich kann man aus der Variablentabelle finden, dass die meisten von ihnen 80 überschreiten.

Variables - meanImRef

meanImRef

192x168 double

	1	2	3	4	5	6	7	8	9	10
1	80.8985	80.8355	81.5480	82.2857	83.1597	83.1258	82.6955	83.0487	83.2963	83.8689
2	80.5120	80.4469	81.3085	82.5066	82.8555	83.3850	83.2551	83.4385	83.8862	84.5260
3	79.5646	80.4628	81.5834	82.5633	83.2564	83.1652	82.6651	82.8938	84.0677	84.8428
4	80.0431	80.9733	81.9199	82.5067	82.8891	82.3602	82.2565	82.5841	83.4348	84.3441
5	80.8246	81.5232	81.7704	82.0431	82.1565	82.2392	82.0153	82.4983	83.3642	83.6424
6	80.6203	81.4177	82.0546	82.3812	82.5333	82.1248	82.1639	82.8747	83.3236	83.5816
7	81.7516	82.1137	82.0613	82.4260	82.8601	82.8723	82.8382	83.5484	83.5891	83.5442
8	82.2094	82.1272	82.1894	82.5180	82.6708	82.7835	83.2883	83.7175	84.1039	84.1635
9	82.0540	81.9008	82.2849	82.7126	82.7197	83.2047	83.6967	84.2304	84.5583	85.1264
10	82.0009	81.5915	81.9515	82.6877	83.1734	83.7077	84.3286	84.7562	84.9275	85.4838
11	81.7693	81.6558	82.2530	83.1166	83.7148	84.1342	84.5425	84.9997	85.2646	85.5948

Abbildung 21: Ein Teil von meanImRef

ImN

192x168 double

	20	21	22	23	24	25	26	27	28	29
1	3 78.0853	78.0853	68.2696	78.0853	78.0853	78.0853	78.0853	78.0853	87.9010	78.0853
2	5 78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853
3	5 68.2696	78.0853	68.2696	68.2696	78.0853	78.0853	78.0853	87.9010	78.0853	78.0853
4	5 78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853
5	3 87.9010	78.0853	78.0853	87.9010	78.0853	78.0853	78.0853	68.2696	87.9010	87.9010
6	3 87.9010	68.2696	78.0853	87.9010	87.9010	87.9010	87.9010	87.9010	87.9010	87.9010
7	3 78.0853	78.0853	78.0853	87.9010	87.9010	78.0853	78.0853	78.0853	78.0853	78.0853
8	3 78.0853	78.0853	78.0853	78.0853	78.0853	68.2696	78.0853	78.0853	68.2696	68.2696
9	3 78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853
10	3 78.0853	78.0853	87.9010	87.9010	78.0853	78.0853	78.0853	78.0853	78.0853	78.0853
11	3 87.9010	78.0853	78.0853	87.9010	87.9010	78.0853	78.0853	78.0853	78.0853	78.0853

Abbildung 22: Ein Teil von ImN

Schließlich hat man nach der Berechnung nur einige kleine Werte ‘DiffImNorm‘, um den Grad der Abnormalität jedes Pixels anzugeben. Dann kann man durch Ändern der Schwellenwert die Differenz filtern. Danach kann ‘Mask Pixel Threshold‘ erstellt werden. Der Teil, dessen absoluter Wert größer als der Schwellenwert ist, wird als 1 dargestellt, und die anderen werden als 0 markieren. Wenn sie in Form eines Bildes angezeigt werden, ist der weiße Bereich der herausgefilterte abnormale Bereich. Es ist nicht schwer aus Abbildung 29 zu erkennen: je größer der Schwellenwert ist, desto kleiner ist daher der gefilterte Bereich.

DiffImNorm

192x168 double

	1	2	3	4	5	6	7	8	9	10
1	-0.1209	-0.5371	-0.5939	-0.6457	-0.2263	-0.2232	-0.6670	-0.6919	-0.6997	-0.7123
2	-0.1029	-0.5276	-0.5561	-0.6031	-0.6525	-0.2297	-0.2160	-0.2339	-0.2650	-0.2973
3	-0.5014	-0.5068	-0.5383	-0.6109	-0.6554	-0.2262	-0.2078	-0.2102	-0.2528	-0.3067
4	-0.4931	-0.5371	-0.5854	-0.5910	-0.5907	-0.5912	-0.1781	-0.6330	-0.7014	-0.7640
5	-0.5073	-0.1388	-0.5586	-0.1701	-0.5883	-0.5760	-0.5914	-0.2099	-0.7285	-0.7647
6	-0.5117	-0.1377	-0.1643	-0.5798	-0.5823	-0.5849	-0.1833	-0.2275	-0.7322	-0.7598
7	-0.5191	-0.1575	-0.5339	-0.5464	-0.5786	-0.6077	-0.6443	-0.2501	-0.7239	-0.7435
8	-0.5130	-0.5238	-0.5265	-0.5443	-0.5787	-0.6098	-0.6498	-0.6857	-0.2741	-0.7314
9	-0.5109	-0.5113	-0.5273	-0.5677	-0.5948	-0.2103	-0.6484	-0.7011	-0.7313	-0.3167
10	-0.5027	-0.5076	-0.5410	-0.5897	-0.6118	-0.2314	-0.2658	-0.7199	-0.7538	-0.3372
11	-0.1337	-0.1372	-0.1668	-0.5934	-0.6211	-0.2531	-0.2811	-0.3046	-0.3191	-0.3361

Abbildung 23: DiffImNorm

Abbildung 24: Mask Pixel (th-Wert von links nach rechts:2 1,5 1)

4.2.3 Connected Components und Statistik des Grauwerts des abnormalen Bereichs

‘Connected Components‘ bezieht sich auf die miteinander verbundenen Pixel. Diese Pixel gehört zu einer gleichen Region. Es gibt zwei verschiedene Möglichkeiten, die benachbarten Pixel eines Pixels zu berechnen: Vierer-Nachbarschaft und Achter-Nachbarschaft. Das bedeutet, bei der Berechnung wird davon ausgegangen, dass mehrere benachbarte Pixel vorhanden sind.

Beispiel: Basierend auf dem roten Pixel in der Mitte

Vierer-Nachbarschaft: Nur B, D, E und G sind Nachbarn.

Achter-Nachbarschaft: Alle 8 Pixel (A bis H) sind Nachbarn.

A (x-1, y-1)	B (x, y-1)	C (x+1, y-1)
D (x-1, y)	Pixel (x, y)	E (x+1, y)
F (x-1, y+1)	G (x, y+1)	H (x+1, y+1)

Tabelle 4: Nachbar eines Pixels

Alle nebeneinander liegenden Pixel ungleich Null werden als zu demselben Bereich gehörend betrachtet.

Im folgenden Programm verwendet man Achter-Nachbarschaft Algorithmus.

```
1.  for i = 1:countIm
2.
3.     ...
4.      if numPixTotal > 0  % if there are such pixels, then calculate size of connected
    pixel areas
5.
6.          if flagConnectedComponents
7.  % clear variable dGV
8.              clear dGV
9.  % find connected components - this algorithm needs to be rewritten or reused
10.             listConnComp = bwconncomp(maskPixelsThreshold);
11.
12. % go through list of connected components
13.             for connComp = 1:listConnComp.NumObjects
14.
```

```
15.
16.                 numPix(connComp) = numel(listConnComp.PixelIdxList{connComp});
17.
18.
19.                 [indRows,indCols] = ind2sub([numberRows, numberCols],listConnComp.Pi
    xelIdxList{connComp});
20.                 dGV(connComp) =  0;
21.                 for pixel = 1:numPix(connComp)
22.                     dGV(connComp) =  dGV(connComp) + ImN(indRows(pixel), indCols(pix
    el))...
23.                         /meanImRef(indRows(pixel), indCols(pixel));
24.                 end
25.                 dGV(connComp) = dGV(connComp)/numPix(connComp);
26.             end
27.             dGVav = 0;  % calculate average gray value difference over all connected
    components
28.             for connComp = 1:length(listConnComp.PixelIdxList)
29.                 dGVav = dGVav + dGV(connComp)*numPix(connComp);
30.             end
31.             dGVav = dGVav/numPixTotal;
32.
33.         else
34.             dGVav = 0;
35.             for r = 1:numberRows
36.                 for c =  1:numberCols
37.                     if maskPixelsThreshold(r,c) == 1
38.                         dGVav = dGVav + ImN(r,c)/meanImRef(r,c);
39.                     end
40.                 end
41.             end
42.             dGVav = dGVav/numPixTotal;
43.         end
44.     else
45.
46.         dGVav = 0; % if there are no pixels detected
47.     end
48.
49. end
```

Programm 11: der durchschnittlichen Grauwert berechnen

Der Kernalgorithmus des zweiten Teils ist die Funktion ‘bwconncomp ()‘. Es kann die Connected Components finden und das Ergebnis wird in eine Struktur gespeichert.

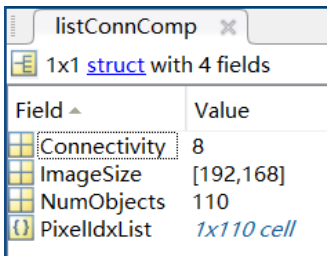

Abbildung 25: List für Connected Components

Man kann daraus sehen, dass es den Achter-Nachbarschaft Algorithmus verwendet (Connectivity 8), die Bildgröße (ImageSize) ist 192 mal 168 und es gibt insgesamt 110 abnormale Bereiche (NumObjects). Die PixelIdList zeichnet die in jedem Bereich enthaltenen Pixel auf. Es ist erwähnenswert, dass es Spalte für Spalte von oben nach unten berechnet wird.

Zum Beispiel wird eine beliebige Matrix als Mask Pixel Threshold. Es gibt 11 Pixel und 5 Bereiche.

1	0	1	1	0
0	0	0	0	0
1	1	1	0	1
0	0	0	0	1
1	0	0	1	1

Tabelle 5: Mask Pixel Threshold

Nach dem Algorithmus von ‘bwconncomp ()‘ sollte das Ergebnis so sein:

1	0	4	4	0
0	0	0	0	0
2	2	2	0	5
0	0	0	0	5
3	0	0	5	5

Tabelle 6: Ergebnis 1

Wenn man den zeilenweisen Algorithmus verwenden, sollte das Berechnungsergebnis so aussehen:

1	0	2	2	0
0	0	0	0	0
3	3	3	0	4

0	0	0	0	4
5	0	0	4	4

Tabelle 7: Ergebnis 2

Das muss man beachten, denn Matlab wird später verwendet, um zu überprüfen, ob das Ergebnis des C-Programms korrekt ist.

Als nächstes extrahiert und berechnet man die Grauwertdifferenz und den Durchschnittswert aller Regionen.

Auf diese Weise können alle abnormalen Bereiche separat berechnet werden. Man kann auch alle Pixel als Ganzes berechnen. Solange der ‘flagConnectedComponents‘ auf 0 gesetzt ist, springt das Programm zu Zeile 44, dann wird nur die durchschnittliche Grauwertdifferenz der gesamten Fläche berechnet. Das Endergebnis bleibt gleich.

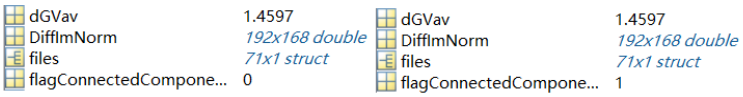

Abbildung 26: Das Endergebnis

5 Implementierung mit C-Programm

In diesem Kapitel werden alle Algorithmen des vorherigen Kapitels in C-Sprache geschrieben, die für Raspberry Pi verfügbar sind.

Um den Vergleich mit NEON zu erleichtern, muss auch die Laufzeit des Programms gemessen werden. Natürlich muss es sichergestellt werden, dass das Ergebnis mit dem von Matlab übereinstimmt, dazu muss man die Berechnungsergebnisse ausgeben und als '.txt' Datei speichern, dann mit Hilfe des Matlabs kann man die Richtigkeit zu überprüfen.

5.1 Header Datei

Eine Header-Datei ist in der Programmierung, insbesondere in den Programmiersprachen C++ und C, eine Textdatei, die Deklarationen und andere Bestandteile des Quelltextes enthält. Quelltext, der sich in einer Header-Datei befindet, ist im Allgemeinen zur Verwendung in mehreren Programmen oder mehreren Teilen eines Programmes vorgesehen. [5]

Die folgenden Header-Dateien werden in diesem Projekt verwendet.

```
1.  #include <math.h>
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <png.h>
5.  #include <assert.h>
6.  #include <time.h>
7.  #include <sys/time.h>
8.  #include <string.h>
9.  #include <dirent.h>
10. #include <unistd.h>
11. #include "img.h"
12. #include "twopass.h"
```

Programm 12: Header Dateien

In 'math.h' werden einige mathematische Funktion definiert. Der im Programm verwendete 'sqrt' ist hier enthalten, mit dieser Funktion kann man die Wurzel berechnen.

Die Abkürzung 'stdio' bedeutet "standard input and output". Beim Debuggen eines Programms muss man manchmal mit Hilfe der Funktion 'printf()' bestimmte Daten ausdrucken, damit man feststellen kann, ob das Programm ordnungsgemäß ausgeführt wird. Die Funktion 'printf()' ist in dieser Header-Datei definiert.

Die Funktionen ‘malloc()‘ und ‘free()‘ sind in Header-Datei ‘stdlib.h‘ vorhanden. Durch Aufrufen der Funktion ‘malloc()‘ kann man einer Variablen eine bestimmte Größe des Speicherplatzes zuweisen. Dann verwendet man ‘free()‘ am Ende des Programms, um den Speicher freizugeben.

Der neue Bilddatentyp ‘png_bytep‘ wird in Datei ‘png.h‘ definiert.

In der Header-Datei ‘assert.h‘ wird eine wichtige Funktion zum Debuggen ‘assert()‘ ist in dieser Datei definiert. Wenn der Ausdruck in Klammern ‘falsch‘ ist, wird das Programm beendet.

‘time.h‘ und ‘sys/time.h‘ definieren einige zeitbezogene Funktion und Strukturtypen, zum Beispiel: ‘gettimeofday()‘, ‘struct timeval‘.

‘string.h‘ ermöglicht Benutzern, einige Funktionen wie ‘strcmp()‘(Zeichenfolgenvergleich) und ‘strcpy()‘(String-Kopie) zum Verarbeiten von Zeichen oder Zeichenfolgen zu verwenden.

Um die Dateien übersichtlicher auszusehen, legt man alle Bilder in einem Ordner ab. Dann muss man die in der Header-Datei ‘dirent.h‘ definierten Funktionen ‘opendir‘ und ‘closedir‘ und in ‘unistd.h‘ definierte ‘chdir‘ verwenden, um die Ordner zu wechseln, die das Programm scannen kann.

Alle oben genannten Header-Dateien sind die Dateien, die mit dem System geliefert werden.

5.1.1 Header-Datei img.h

Nur zwei Funktionen werden in diese Datei definiert. Ihre Funktionen bestehen darin, Daten aus Bild zu lesen und Daten als Bild zu speichern. In dieser Arbeit wird nur die Funktion ‘readimg‘ benutzt.

Die erste Variable ist der Dateiname des Bildes, zum Beispiel ‘Beispiel.png‘. Es ist die einzige Eingangsvariable der Funktion.

‘*data‘ ist Daten aus dem Bild gelesen, mit anderen Worten ist es eine Matrix, die aus den Grauwertdaten des Bildes besteht.

‘*w‘ und ‘*h‘ sind Bildbreite und -länge.

```
1.  void readimg(const char *filename, png_bytep *data,size_t *w, size_t *h);
```

```
2. void writeImageData(const char* filename, png_bytep *data,size_t w, size_t h, size_t
   bitdepth );
```

Programm 13: img.h

Der Header-Datei ‘img.h‘ wird von Prof. Lampe geboten.

5.1.2 Header-Datei twopass.h

5.1.2.1. Prinzip des Twopass-Algorithmus

Der Twopass-Algorithmus ist zielt darauf ab, die ‘Connected Components‘ eines Binärgraphen zu berechnen und jeden Bereich zu nummerieren.

Der pseudocode ist:

```
1. algorithm TwoPass(data) is
2.     linked = []
3.     labels = structure with dimensions of data, initialized with the value of Backgr
   ound
4.
5.     First pass
6.
7.     for row in data do
8.         for column in row do
9.             if data[row][column] is not Background then
10.
11.                neighbors = connected elements with the current element's value
12.
13.                if neighbors is empty then
14.                    linked[NextLabel] = set containing NextLabel
15.                    labels[row][column] = NextLabel
16.                    NextLabel += 1
17.
18.                else
19.
20.                    Find the smallest label
21.
22.                    L = neighbors labels
23.                    labels[row][column] = min(L)
24.                    for label in L do
25.                        linked[label] = union(linked[label], L)
26.
27.     Second pass
28.
29.     for row in data do
30.         for column in row do
31.             if data[row][column] is not Background then
32.                 labels[row][column] = find(labels[row][column])
```

```
33.
34.     return labels
```

Programm 14: pseudocode[6]

Bei diesem Algorithmus muss man die Daten insgesamt zweimal scannen.

Im ersten Scan muss man alle ‘1‘ aus dem Binärbild finden, ihre benachbarten Elemente überprüfen und sie dann beschriften, gleichzeitig werden alle benachbarten Nummern Notieren. Hier wird 8-connected Nachbar Algorithmus verwendet. Die Nummerierungsregel lautet:

1. Von Zeile nach rechts scannen; die oberen linken, oberen, oberen rechten und linken Elemente werden als benachbarte Elemente betrachtet
2. Die benachbarten Elemente dieses Elements sind alle 0; dieses Element wird eine neue Nummer vergeben
3. Das Element hat benachbarte Elemente ungleich Null; es wird genauso nummeriert wie das benachbarte Element
4. Neben diesem Element befindet sich mehr als eine Zahl; es wird als Minimum daneben nummeriert und die verschiedene Werte werden notiert

Beim zweiten Scan soll man nur alle Nicht-Null-Elemente gemäß den aufgezeichneten miteinander verbundenen Werten neu nummerieren.

5.1.2.2. Twopass-Algorithmus in C

Hier wird das Programm streng in Übereinstimmung mit dem obigen Algorithmus geschrieben.

Die einzige Funktion heißt ‘int *bwconncomp1(int *bild, int x, int y)‘. Die drei Parameter dieser Funktion repräsentieren Bilddaten in Form eines eindimensionalen Arrays, Bildbereit und Bildlänge. Diese Funktion gibt schließlich ein eindimensionales Array aus, es ist das vom Twopass-Algorithmus berechnete Bild.

```
1.   int *rt=(int *)malloc(sizeof(int)*x*y);  //result
2.   int *z=(int *)malloc(sizeof(int)*x*y);
3.   int counter=1;
4.   int A,B,C,D,temp;
5.
6.                ...
7.
8.   return rt;
```

```
9.   free(rt);
10.  free(z);
```

Programm 15: Initialisierung

Zuerst werden einige Variablen deklariert, die später verwendet werden. Da hier die Funktion ‘malloc()‘ verwendet wird, sollte am Ende ‘free()‘ hinzugefügt werden.

‘*rt‘ ist der Ausgang der gesamten Funktion, das Sternzeichen vor dem Variablennamen ‘rt‘ zeigt an, dass es sich um einen Zeiger oder ein Array handelt. Mit anderen Worten, das Ergebnis der Funktionsausgabe ist eine Reihe von Zahlen, keine einzelne Zahl. Normalerweise ist die Verwendung der Funktion ‘malloc()‘ nicht erforderlich, wenn man eine Variable direkt definieren, weist das System ihr automatisch Speicher zu. Es gibt kaum einen Unterschied zwischen den beiden Methoden im Hauptprogramm.

```
1.   int *rt=(int *)malloc(sizeof(int)*x*y);  //definieren mit 'malloc'
2.   int rt[];        //direkt definieren
```

Programm 16: malloc

Achtung: Für die Funktion in der Header-Datei besteht der Unterschied darin, dass das System die Daten in der Variablen automatisch löscht, wenn die Variable direkt definiert ist, wenn das Hauptprogramm die Funktion aufruft und die Operation beendet. Wie oben erwähnt, ‘*rt‘ und ‘rt[]‘ stellen tatsächlich beide Zeiger dar, was in ‘rt‘ gespeichert ist, ist die Speicheradresse. Wenn diese Funktion beendet ist, werden die Daten im Speicher gelöscht und die Daten können nicht an die Funktion übergeben werden. Dies bedeutet, dass diese Funktion nicht das richtige Ergebnis ausgeben kann. Daher muss man hier die Funktionen ‘malloc()‘ und ‘free()‘ benutzen. Es ist sehr wichtig, die Daten in dem Speicher zu löschen, nachdem die Funktion das Ergebnis ausgegeben hat.

```
1.   for (int i=0;i<x*y;i++)
2.    z[i]=i;
```

Programm 17: Initialisierung der Array

Hier erstellt man eine Liste, alle Daten am Anfang entsprechen eigenen Bezeichnung. Diese Liste wird verwendet, um den Mindestwert der Pixel um den Pixel herum aufzuzeichnen. Der Grund ist, dass man alle Pixel im selben Bereich entsprechend dem Mindestwert beschriften muss. Nach der Einführung des Firstpass-Algorithmus wird dieser anhand von Beispielen ausführlich vorgestellt.

Firstpass:

In diesem Prozess muss man nur jeder Pixel (ungleich Null) gemäß den Regeln im vorherigen Abschnitt (5.1.2.1) nummerieren und die minimale benachbarte Anzahl in der Liste aufzeichnen. Im Programm verwendet man ein eindimensionaler Array zum Speichern von Bilddaten, daher sollte die Position des Pixels bei (i, j) im Array (i* x + j) sein, ‘x‘ steht für die Anzahl der Pixel in jeder Zeile. Der Grund dafür ist, dass in einem eindimensionalen Array die Pixel des Bildes in einer Reihe angeordnet sind.

Die Position eines Pixels und seiner benachbarten Pixel sollte wie folgt sein:

A (i-1)*x+j-1	B (i-1)*x+j	C (i-1)*x+j+1
D i*x+j-1	(Pixel) i*x+j	…
…	…	…

Tabelle 8: Acht-Nachbarschaft

Die roten Pixel im Bild sind die benachbarten Pixel, die überprüft werden müssen, da sie vor den blauen Pixeln berechnet wurden.

Ein mögliches Problem ist: Die gescannten Pixel befinden sich möglicherweise am Bildrand und haben diese benachbarten Pixel nicht. Dies kann zu Berechnungsfehlern führen, daher muss man einen anderen Algorithmus verwenden, um die Pixel am Rand zu berechnen.

Der vollständige Code für ‘Firstpass‘ lautet wie folgt:

```
1.     if(bild[i*x+j]==0)
2.      rt[i*x+j]=0;     //background;0 Pixel
3.     else {
4.      if(i!=0&&j!=0&&j!=x-1){   //Pixel in the middle
5.       A=rt[(i-1)*x+j-1];
6.       B=rt[(i-1)*x+j];
7.       C=rt[(i-1)*x+j+1];
8.       D=rt[i*x+j-1];
9.      }
10.     if(j==0){      //Pixel in the first row
11.      A=0;
12.      B=rt[(i-1)*x+j];
13.      C=rt[(i-1)*x+j+1];
14.      D=0;
15.     }
```

```
16.     if(j==x-1){       //Pixel in the last column
17.      A=rt[(i-1)*x+j-1];
18.      B=rt[(i-1)*x+j];
19.      C=0;
20.      D=rt[i*x+j-1];
21.     }
22.     if(i==0){       //Pixel in the first column
23.      A=0;
24.      B=0;
25.      C=0;
26.      D=rt[i*x+j-1];
27.     }
28.     if(i==0&&j==0){      //the first Pixel
29.      A=0;
30.      B=0;
31.      C=0;
32.      D=0;
33.     }
34.     if(A==0&&B==0&&C==0&&D==0){  //New isolated Pixel
35.      rt[i*x+j]=counter;
36.      counter++;
37.     }
38.     else {        //other situation
39.      temp=x*y;
40.      if(A!=0&&A<=temp)
41.       temp=A;
42.      if(B!=0&&B<=temp)
43.       temp=B;
44.      if(C!=0&&C<=temp)
45.       temp=C;
46.      if(D!=0&&D<=temp)
47.       temp=D;
48.      rt[i*x+j]=temp;     //Find neighbor's minimum value other than 0
49.      if(z[temp]<z[A])
50.       z[A]=z[temp];
51.      if(z[temp]<z[B])
52.          z[B]=z[temp];
53.      if(z[temp]<z[C])
54.       z[C]=z[temp];
55.      if(z[temp]<z[D])
56.       z[D]=z[temp];
57.     }
58.    }
```

Programm 18: Firstpass

Zeile 1~2: Zuerst bleiben 0 Pixel unverändert, wie in der ersten und zweiten Zeile des Programms geschrieben.

Zeile 4: Pixel befindet sich nicht am Bildrand. Benachbarte Pixel A B C D werden nach der normalen Methode berechnet.

Zeile 10: Der Pixel befindet sich ganz links im Bild. Man setzen A und D manuell auf ‘0‘, da ‘0‘ die Berechnungsergebnisse nicht beeinflusst und das System denkt, dass der Pixel immer noch die 4 benachbarten Pixel A B C D hat, es wird sichergestellt, dass die nachfolgenden Programme normal ausgeführt werden können.

Für Pixel in der letzten Spalte und der ersten Zeile des Bildes sowie der erste Pixel verwendet man auch die obige Methode (Programm 18 Zeile 10 ~ 28).

Bisher hat das Programm die benachbarten Pixel von Nicht-Null-Pixeln gefunden. Dann nummeriert man einfach nach der Nummerierungsregel und aktualisiert man die Liste (Programm 18 Zeile 34 bis Ende).

Zu diesem Zeitpunkt ist ‘Firstpass‘ abgeschlossen, aber die Liste ist noch nicht korrekt. Im folgenden Beispiel wird das Problem der Liste und ihre Verwendung ausführlich erläutert.

Beispiel: Das ist ein Bild nach der Firstpass-Algorithmus:

0	0	1	1	0	0
0	2	1	1	1	0
3	0	0	1	0	0
4	0	0	0	0	5
6	0	8	7	7	0

Tabelle 9: Bild nach der Firstpass-Algorithmus

In diesem Beispiel ist gemäß dem Acht-Nachbarschaft Algorithmus leicht zu erkennen, dass es zwei Bereiche gibt: roten Bereich und grünen Bereich. Dann ist die Liste, die diesem Bild entspricht, wie folgt:

i	1	2	3	4	5	6	7
Z[i]	1	1	1	1	5	1	5

Tabelle 10: Liste

‘i‘ ist der ursprüngliche Wert und ‘z[i]‘ ist der Sollwert. Nach dem Einstellen gemäß der Liste sieht das Bild wie folgt aus:

0	0	1	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	0	0	0	5
1	0	8	5	5	0

Tabelle 11: Bild nach der Einstellung

Aber dieses Bild ist immer noch nicht das, was man braucht. Es gibt zwei Bereiche auf dem Bild, daher sollte die maximale Anzahl 2 statt 5 sein. Man muss eine Anpassung an der Liste vornehmen. Dies sind die richtige Liste und Bild:

i	1	2	3	4	5	6	7
Z[i]	1	1	1	1	2	1	2

Tabelle 12: Liste 2

0	0	1	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	0	0	0	2
1	0	8	2	2	0

Tabelle 13: Ergebnis

Dieses Bild ist das Endergebnis, das man benötigt. Es soll das Berechnungsergebnis des Secondpass-Algorithmus. Man muss die Liste vor dem ‘Secondpass‘ anpassen.

Tatsächlich entspricht die Liste, die das Programm nach dem ‘Firstpass‘ erhält, keiner der obigen Tabellen, sondern sieht so aus:

i	1	2	3	4	5	6	7
Z[i]	1	1	2	3	5	4	5

Tabelle 14: Liste 3

Diese Liste zeigt nur: 3 und 2, 4 und 3, 6 und 4 sind benachbart. Tatsächlich gehören 2, 3, 4 und 6 zum selben Gebiet, daher sollten sie am Ende alle dieselbe Zahl haben. Zusammenfassend muss man die Liste zweimal verarbeiten.

```
1.  for(int i=0;i<x*y;i++)
2.   while(z[i]!=z[z[i]])
3.    z[i]=z[z[i]];      //Refresh the 'list'
4.  counter=0;
```

```
5.   for(int i=1;i<x*y;i++){      //Renumber
6.    if(i==z[i]){
7.     counter++;
8.     z[i]=counter;
9.    }
10.   else {
11.    z[i]=z[z[i]];
12.   }
13.  }
```

Programm 19: Einstellung

- Die Zahlen in den gleichen Bereich mit der gleichen Nummer setzen (Programm 19 Zeile 1~3)
- Die Nummern der Bereiche neuordnen (Programm 19 Zeile 5 bis Ende)

Dann hat man die richtige Liste. Im ‘Secondpass‘ soll man nur einfach die Pixel im Bild entsprechend dieser Liste neunummerieren.

Secondpass:

```
1.   for (int i=0;i<y;i++){     //According to the list to produce the final result
2.    for(int j=0;j<x;j++){
3.     if(rt[i*x+j]!=0)
4.      rt[i*x+j]=z[rt[i*x+j]];
5.    }
6.   }
```

Programm 20: Secondpass

Die gesamte Funktion ist bisher abgeschlossen.

5.1.2.3. Überprüfung der Richtigkeit mit Hilfe von Matlab

Um die Richtigkeit der obigen Funktion zu überprüfen, kann man das Ergebnis mit dem Berechnungsergebnis von Matlab vergleichen.

Zuerst werden die Ergebnisse der Twopass-Algorithmus als txt-Datei gespeichert. Als nächstes verwendet wieder Matlab.

```
1.  clear ;
2.  close all;
3.
4.  load('StatisticReferences.mat','meanImRef','stdImRef')
5.  data=importdata('C:\Users\wyq\Desktop\Arb\test\data.txt');
6.
```

```
7.  th=2;
8.
9.  numPixTotal=0;
10.
11. test1 = double(imread('C:\Users\wyq\Desktop\Arb\test\test1.png'));
12.
13. numberRows = size(test1,1);
14. numberCols = size(test1,2);
15.
16. meanAcc = 0;
17. powAcc = 0;
18. for r = 1:numberRows
19.     for c =  1:numberCols
20.         meanAcc = meanAcc + test1(r,c);
21.         powAcc = powAcc + test1(r,c)*test1(r,c);
22.     end
23. end
24. meanIm = meanAcc/numberRows/numberCols;
25. powIm = powAcc/numberRows/numberCols;
26. stdIm = sqrt(powIm - meanIm*meanIm);
27.
28.     for r = 1:numberRows
29.         for c =  1:numberCols
30.             ImN(r,c) = (test1(r,c)-meanIm)./stdIm*15+80;
31.         end
32.     end
33.
34.  for r = 1:numberRows
35.         for c =  1:numberCols
36.             ImwoMean(r,c) = (ImN(r,c) - meanImRef(r,c));
37.             DiffImNorm(r,c) = ImwoMean(r,c)/stdImRef(r,c);
38.         end
39.     end
40.
41.     for r = 1:numberRows
42.         for c =  1:numberCols
43.             if abs(DiffImNorm(r,c)) >= th
44.                 maskPixelsThreshold(r,c) = 1;
45.                 numPixTotal = numPixTotal + 1;
46.             else
47.                 maskPixelsThreshold(r,c) = 0;
48.             end
49.         end
50.     end
51.
52.  listConnComp = bwconncomp(maskPixelsThreshold);
```

Programm 21: Programm

Der obige Code ist dem vorherigen Kapitel sehr ähnlich. Man kann sehen, dass das Berechnungsergebnis von ‘twopss‘ ‘data‘ sind (Programm 21 Zeile 5) und das Ergebnis von

Matlab in der Struktur ‘listConnComp‘ enthalten ist. Tatsächlich sind ‘data‘ ein zweidimensionales Array, daher muss man Daten aus ‘listConnComp‘ extrahieren und ein zweidimensionales Array erstellen, da man nur auf diese Weise zwei Variablen subtrahieren kann.

```
1.      for b=1:numberRows
2.        for  c=1:numberCols
3.           matrix(b,c)=0;
4.        end
5.     end
6.     num=listConnComp.NumObjects;
7.
8.     for i=1:num
9.         numPix(i)=numel(listConnComp.PixelIdxList{i});
10.        [Rows,Cols]=ind2sub([numberRows,numberCols],listConnComp.PixelIdxList{i});
11.        for ii=1:numPix(i)
12.            matrix(Rows(ii),Cols(ii))=i;
13.        end
14.    end
```

Programm 22: Programm 2

Durch das obige Programm werden die Daten in ‘listConnComp‘ in einem zweidimensionalen Array ‘matrix‘.

Variables - matrix

matrix

192x168 double

	89	90	91	92	93	94	95	96	97	98	99	100	101
57	0	2	2	2	2	2	2	2	0	0	0	0	0
58	0	0	2	2	2	2	2	2	2	0	0	0	0
59	0	0	2	2	2	2	2	2	2	0	0	0	0
60	0	0	2	2	2	2	2	2	2	2	2	0	0
61	0	0	0	2	2	2	2	2	2	2	2	2	0
62	0	0	0	0	0	0	0	0	0	0	2	2	0
63	0	0	0	0	0	0	0	0	0	0	2	2	2
64	0	0	0	0	0	0	0	0	0	2	2	2	2
65	0	0	0	0	0	0	0	0	0	2	2	2	2
66	0	0	0	0	0	0	0	0	2	2	2	2	2
67	0	0	0	0	0	0	0	0	2	2	2	2	2

Abbildung 27: Ein Teil von ‘matrix‘

Dann soll man nur jedes entsprechende Element in ‘matrix‘ und ‘data‘ subtrahieren. Wenn das Berechnungsergebnis Null ist, sind die Daten dieser beiden Variablen gleich.

```
1.      falschzaehler=0;
2.     f=1;
3.
4.     for i=1:numberRows
5.        for ii=1:numberCols
6.           if(matrix(i,ii)-data(i,ii))
7.               falschzaehler=falschzaehler+1;
8.               falsch(1,f)=i;
```

```
9.                  falsch(2,f)=ii;
10.                 falsch(3,f)=matrix(i,ii);
11.                 falsch(4,f)=data(i,ii);
12.                 f=f+1;
13.             end
14.         end
15.     end
```

Programm 23: Programm zur Fehlerprüfung

Eine Variable (‘falschzaehler‘) wird gesetzt, um die Anzahl der Fehler zu zählen. Für alle Fälle zählt man auch die Position des falschen Pixels, seinen ‘data‘-Wert und ‘matrix‘-Wert. Dies wird in der Variablen ‘falsch‘ gespeichert.

Um seine Zuverlässigkeit widerzuspiegeln, ändert man manuell zwei beliebige Daten in ‘data‘ (data.txt).

Variables - data

data

192x168 double

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0

Variables - data

data

192x168 double

	1	2	3	4	5	6	7	8	9
1	0	0	0	3	0	0	0	0	4
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0

Abbildung 28: ‘data‘ original (oben) und geändert (unten)

Auf dem Bild kann man sehen, die beiden Zahlen in der ersten Zeile werden manuell geändert. Dies ist das Ergebnis:

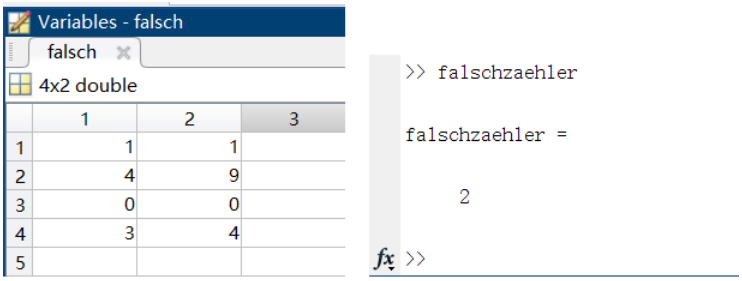

Abbildung 29: Ergebnis

Es stellt sich heraus, dass es zwei Fehler gibt. Die Fehlerdetails werden in ‘falsch‘ gespeichert: Eine Spalte steht für einen Fehler. Die erste Zeile ist die Zeile mit den falschen Pixeln. Die zweite Zeile ist die Spalte mit den falschen Pixeln. Die dritte Zeile ist der korrekte Wert des Pixels. Die vierte Zeile ist der aktuelle Wert des Pixels.

Es ist ersichtlich, dass dieses Programm zuverlässig ist. Dann ändert man die geänderten Daten wieder auf den ursprünglichen Wert und das Programm wird erneut ausgeführt.

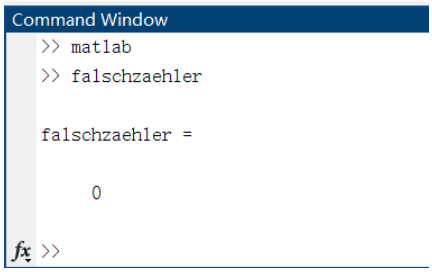

Abbildung 30: die Lösung

Jetzt ist ‘falschzaehler‘ gleich 0, mit anderen Worten, das Berechnungsergebnis von ‘twopass‘ stimmt mit dem von Matlab überein.

5.2 Anweisung zur Parametereinstellung

Im Allgemeinen kann das C-Programm mit dem Befehl ‘./‘ + *Programmname* in dem Ordner, der das C-Programm enthält, ausgeführt werden, und das Programm führt automatisch die Hauptfunktion ‘main‘ aus. In diesem Projekt lautet der Name des C-Programms ‘test‘, man kann den Befehl ‘./test‘ verwenden, um dieses Programm auszuführen.

In dieser Arbeit ändert man häufig Parameter wie die Anzahl der Ausführungen der Hauptprogrammschleife und den ‘th‘-Wert. Es ist sehr umständlich, jedes Mal den Code zu ändern, deshalb stellt man das Programm so ein, dass die Parameter im Programm durch externe Anweisungen geändert werden können. Das überarbeitete Hauptprogramm ist:

```
1.  int main(int argc,char** argv){
```

```
2.      unsigned int times;
3.      double thresholdStddevFactor;
4.      if(argc >= 2 && argv[1]!=NULL){
5.          times = strtol(argv[1],NULL,10);
6.          thresholdStddevFactor = (double)strtol(argv[2],NULL,10)/10;
7.      }
8.      else{
9.          times = 10000;
10.         thresholdStddevFactor = 2;
11.     }
12.     printf("\nexecute times:%d",times);
13.     printf("\nthreshold stdddev:%f\n\n",thresholdStddevFactor);
14.
15.     ...
16.
17. }
```

Programm 24: Einstellung der Eingabeparameter

Nach dieser Änderung wird die Anweisung als Zeichenfolgen im Array ‘argv‘ gespeichert. Wie im Programm 24 geschrieben (Zeile 5 und 6), wird das zweite Element in ‘argv‘ (argv[1],) der Variablen ‘times‘ zugewiesen, und ein Zehntel des dritten Elements wird der Variablen ‘thresholdStddevFactor‘ zugewiesen. Auf diese Weise kann man nach dem Befehl ‘./test‘ nur zwei Zahlen hinzufügen, um das Programm auszuführen und die Parameter des Programms festzulegen. Wenn keine Parameter eingegeben werden, setzt das Programm ‘times‘ auf 10,000 und ‘thresholdStddevFactor‘ auf 2.

Beispiel:

```
pi@raspberrypi:~/Desktop/ct $ ls
a.out            img.c  img.s            list.txt  maskdata.txt  rtDetected.txt  table.txt  test.c    test.s     twopass.h  yaleB01_P00A+010E-20.png
IMG_4631_25.png  img.h  listSameSeg.txt  Makefile  mRef.txt      sRef.txt        test       test.png  twopass.c  twopass.s
pi@raspberrypi:~/Desktop/ct $ ./test

execute times:10000
threshold stdddev:2.000000

x=168,y=192

Max=8064
meanacc=3418499
powacc=395712199
numPixTotal=233
```

Abbildung 31: Normale Ausführung

```
pi@raspberrypi:~/Desktop/ct $ ls
a.out            img.c  img.s            list.txt  maskdata.txt  rtDetected.txt  table.txt  test.c    test.s     twopass.h  yaleB01_P00A+010E-20.png
IMG_4631_25.png  img.h  listSameSeg.txt  Makefile  mRef.txt      sRef.txt        test       test.png  twopass.c  twopass.s
pi@raspberrypi:~/Desktop/ct $ ./test

execute times:10000
threshold stdddev:2.000000

x=168,y=192

Max=8064
meanacc=3418499
powacc=395712199
numPixTotal=233
```

Abbildung 32: Parameteränderung mit Befehl

5.3 Zeitmessung

Später muss man die Laufzeit des C-Programms mit der Laufzeit von NEON vergleichen, deshalb muss man die Zeit im Programm auf dem Bildschirm drucken.

```
1.      struct timeval tpstart,tpend;
2.      float timeuse;
3.
4.      gettimeofday(&tpstart,NULL);
5.
6.        (Zyklisches Programm)
7.
8.      gettimeofday(&tpend,NULL);
9.      timeuse=1000000*(tpend.tv_sec-tpstart.tv_sec)+tpend.tv_usec-tpstart.tv_usec;
10.     timeuse/=1000000;
11.
12.     printf("used time mean/std/numPixTotal calculation:%f seconds\n",timeuse);
```

Programm 25: Zeitmessung

Hier verwendet man die Funktion ‘gettimeofday()‘, um die Echtzeit vor und nach der Schleife zu messen. Die Laufzeit des Programms wird durch Subtrahieren der beiden gemessenen Zeitwerte erhalten.

In der Struktur ‘timeval‘ steht das Suffix ‘_sec‘ für Sekunden und das Suffix ‘_usec‘ für Mikrosekunden. Die Berechnung der neunten Zeile des Programms berechnet die Anzahl der Mikrosekunden, die das laufende Programm erfährt. Dann teilt man diese Zahl durch 1,000,000, um die Anzahl der Sekunden zu erhalten.

5.4 Hauptprogramm

Das Hauptprogramm wird in eine Schleife stellt wird, damit es zur Laufzeit mehrmals ausgeführt werden kann. Der Grund dafür ist, dass die für eine einzelne Berechnung erforderliche Zeit kurz (0,2 Millisekunde) und für den Vergleich nicht geeignet ist.

Achtung: Um klare Ergebnisse zu erhalten, berechnet das Programm jeweils nur ein Bild. Um die Genauigkeit der Berechnungsergebnisse sicherzustellen, nimmt man für mehrere Tests manuell Bilder aus der Probe auf. Bei der Berechnung wurde die ursprüngliche Berechnungsformel später an die Anforderungen von NEON angepasst. Daher existieren einige Variablen nur für mathematische Berechnungen und haben möglicherweise keine praktische Bedeutung.

5.4.1 Initialisierung

Vor der Schleife muss man das Bild initialisieren und Variablen deklarieren. Diese Größen ändern sich mit der Iteration des Programms nicht.

```
1.      size_t x,y;
2.     png_bytep bild;
3.
4.     readimg("yaleB01_P00A+010E-20.png",&bild,&x,&y);
5.     printf("x=%lu,y=%lu\n",x,y);
```

Programm 26: Bild lesen

Eine aufgenommene Bild wird direkt in den Ordner gelegen, der das Programm enthält. Die Funktion ‘readimg()‘ speichert die Grauwertinformationen des Bildes in der Variablen ‘bild‘, die Größe des Bildes wird in ‘x‘ und ‘y‘ gespeichert. Dann kann man die ‘printf()‘-Funktion verwenden, um die Werte von ‘x‘ und ‘y‘ anzuzeigen.

```
pi@raspberrypi:~/Desktop/ct $ ./test

execute times:10000
threshold stdddev:2.000000

x=168,y=192
```

Abbildung 33: Bildgröße

Nach dem Lesen des Bildes deklariert man einige Variablen für die spätere Verwendung:

```
1.     double meanImRef[y][x],stdImRef[y][x],Ref[y*x], LowThresh[y][x], HighThresh[y][x
   ];
2.     double meanIm, stdImInv, temp1, ratioGW[100];
3.     char maskDefectPixels[x*y];
4.     unsigned int rt[x*y], listSameSeg[100][100], numSameSeg[100],list[100],gvlist[10
   0];
5.     unsigned int accGreyValues[100], accMeanImRef[100], numNormaccGreyValues[100], n
   umNormaccMeanImRef[100];
6.
7.     unsigned int meanAcc, powAcc, wert, numPixTotal, numRegion;
8.     unsigned int  loopCount, i, j, pos, flag;
9.     int posk, k;
```

Programm 27: Variable deklarieren

Die meisten Variablennamen und -bedeutungen hier sind ungefähr die gleichen wie im Matlab-Programm, und die Array-Variable mit einer Größe von 100 liegt daran, dass die Anzahl der ‘Connected Components‘-Bereiche im Bild standardmäßig weniger als 100 beträgt.

Dann ähnlich wie beim Matlab-Programm hat man die Referenzdaten im Voraus als '.txt'-Dateien gespeichert, hier kann man diese Dateien direkt laden:

```
1.      FILE *fpm=fopen("mRef.txt","rb");
2.      fread(Ref,sizeof(double),y*x,fpm);
3.      fclose(fpm);
4.
5.      for(i=0;i<y;i++){
6.          for(j=0;j<x;j++){
7.              meanImRef[i][j]=Ref[y*j+i];
8.          }
9.      }
10.
11.     FILE *fps=fopen("sRef.txt","rb");
12.     fread(Ref,sizeof(double),y*x,fps);
13.     fclose(fps);
14.
15.     for(i=0;i<y;i++){
16.         for(j=0;j<x;j++){
17.             stdImRef[i][j]=thresholdStddevFactor*Ref[y*j+i];
18.         }
19.     }
```

Programm 28: Datei lesen

Auf diese Weise erhaltet man den Referenzmittelwert 'meanImRef' und die Referenzstandardabweichung 'stdImRef'. Wie man sehen kann, ist der Wert 'stdImRef' hier tatsächlich nicht der ursprüngliche Wert. Es wird später in der Formeltransformation ausführlich erläutet.

Abschließend wird die 'list' initialisiert und den Schwellenwert berechnet:

```
1.   for (i=1;i<100;i++)
2.    list[i]=i;
3.
4.      for(i=0;i<y;i++){
5.          for(j=0;j<x;j++){
6.              LowThresh[i][j]=meanImRef[i][j]-stdImRef[i][j];
7.              HighThresh[i][j]=meanImRef[i][j]+stdImRef[i][j];
8.          }
9.      }
```

Programm 29: Berechnung der Schwellwerte

5.4.2 Formeltransformation

Nach dem Testen macht das Programm bei Verwendung von NEON zur Berechnung der Division Fehler und im Vergleich zur Multiplikation nimmt die Berechnung der Division mehr

Zeit in Anspruch. Man muss die ursprüngliche Berechnungsformel transformieren und hauptsächlich die Division in Multiplikation ändern.

Die folgende Formel befindet sich im Matlab-Code:

$$meanIm = meanAcc/x/y \tag{5-1}$$

$$powIm = powAcc/x/y \tag{5-2}$$

$$stdIm = \sqrt{powIm - meanIm * meanIm} \tag{5-3}$$

$$ImN = (Bild - meanIm)/stdIm * 15 + 80 \tag{5-4}$$

$$ImwoMean = ImN - meanImRef \tag{5-5}$$

$$DiffImNorm = ImwoMean/stdImRef \tag{5-6}$$

$$(bool)maskPixel = (positiv)(|DiffImNorm| - th) \tag{5-7}$$

Die roten Variablen in der Formel sind die Variablen, die sich außerhalb der C-Programmschleife berechnet werden sollen. Sie können während des Transformationsprozesses nicht gelöscht werden. ‘maskPixel‘ in der Formel (5-7) ist eine Variable ähnlich wie Bool-Variable, sein Wert kann nur ‘1‘ oder ‘0‘ sein, und nur wenn die Formel hinter dem Gleichheitszeichen nicht negativ ist, ist ‘maskPixel‘ ‘1‘.

Man fügt ‘Inv‘ nach der Variablen hinzu, um den Kehrwert der ursprünglichen Variablen darzustellen. Dann kann man bekommen:

$$stdIm = \sqrt{powAcc * x * y - meanAcc * meanAcc}/\ （x * y） \tag{5-8}$$

$$ImN = \ （Bild - meanAcc/x/y） * stdImInv * 15 + 80 \tag{5-9}$$

$$DiffImNorm = \ （ImN - meanImRef）/stdImRef \tag{5-10}$$

$$(bool)maskPixel = (positiv)(|DiffImNorm| - th) \tag{5-11}$$

Weiter zusammenführen:

$$stdImInv = x * y/\sqrt{powAcc * x * y - meanAcc * meanAcc} * 15 \tag{5-12}$$

$$temp1 = ImN = （Bild - meanAcc/x/y） * stdImInv + 80 \tag{5-13}$$

$$(bool)maskPixel = (positiv)(ImN - (meanImRef + th * stdImRef)) \tag{5-14}$$

oder:

$$(bool)maskPixel = (negative)(ImN - (meanImRef - th * stdImRef)) \tag{5-15}$$

‘stdImInv‘ und ‘temp1‘ sind die Variablennamen im Programm. Die Ausdrücke in den Klammern in den beiden Berechnungen von ‘maskPixel‘ wurden berechnet:

$$HighThresh = meanImRef + stdImRef \tag{5-16}$$

$$LowThresh = meanImRef - stdImRef \tag{5-17}$$

Man kann diese beiden Formeln auch im vorherigen Abschnitt finden.

5.4.3 Programmschleife

Als nächstes werden die Formeln im letzten Abschnitt in das Programm geschrieben.

Zuerst muss man einige Variablen initialisieren und ‘meanAcc‘, ‘powAcc‘ und ‘stdImInv‘ berechnen:

```
1.      for(loopCount = 0; loopCount < times; loopCount++){
2.          // Initialization
3.          numPixTotal = 0;
4.          meanAcc = 0;
5.          powAcc = 0;
6.          numRegion = 0;
7.
8.          for (k=0; k < 100; k++){
9.              numSameSeg[k] = 0;
10.             accMeanImRef[k] = 0;
11.             numNormaccMeanImRef[k] = 0;
12.             accGreyValues[k] = 0;
13.             numNormaccGreyValues[k] = 0;
14.             gvlist[k]=0;
15.         }
16.
17.         // Calculation for meanAcc and powAcc
18.         for(i=0;i<y;i++){
19.             for(j=0;j<x;j++){
20.                 wert = bild[i*x+j];
21.                 meanAcc += wert;
22.                 powAcc += wert*wert;
```

```
23.             }
24.         }
25.
26.         meanIm = ((double)meanAcc)/x/y;
27.         stdImInv = x*y/sqrt((double)powAcc*x*y-
    ((double)meanAcc)*((double)meanAcc))*15;
28.
29.     ...
30.
31.     }
```

Programm 30: Beginn der Schleife

Die Variablen müssen aufgrund der Iteration in der Schleife initialisiert werden (Zeile 3~15). Die im obigen Programm berechneten Variablen sind für jedes Pixel festgelegt.

Als nächstes muss man das gesamte Bild scannen und mit der Berechnung gemäß der Formel fortfahren. Um die oben erwähnte Situation zu vermeiden, in der sich die Pixel am Rand befinden, ändert man die erste Pixelspalte manuell auf ‘0‘. Die Änderung einiger Pixel hier führt nicht zu einer großen Abweichung vom Ergebnis.

```
1.          for(i=0;i<y;i++){
2.              for(j=0;j<1;j++){
3.                  pos = i*x+j;
4.                  rt[pos] = 0;
5.                  maskDefectPixels[pos] = 0;
6.              }
7.          }
```

Programm 31: MaskPixel-Initialisierung

Obwohl der Twopass-Algorithmus im vorherigen Artikel eingeführt wurde, verwendet man eine andere Methode, da der Algorithmus das Bild zweimal scannt, es dauert sehr lange. Jetzt scannt das Programm das Bild nur einmal.

```
1.           for(i=0;i<y;i++){
2.              for(j=1;j<x;j++){
3.
4.                  pos = i*x+j;
5.                  temp1=(bild[pos]-meanIm)*stdImInv+80;
6.                  rt[pos] = 0;
7.
8.                  if(LowThresh[i][j] < temp1){
9.                      maskDefectPixels[pos] = 0;
10.                 }
11.                 else {
12.
```

```
13.                     maskDefectPixels[pos] = 1;
14.
15.           // accumulate area defect pixels
16.                     numPixTotal = numPixTotal+1;
17.                 }
18.             }
19.         }
```

Programm 32: Berechnung MaskPixels

Hier hat man den normalisierten Wert jedes Pixels berechnet und das ‘maskPixel‘ gemäß dem ausgewählten Schwellenwert erzeugt. Dann gibt man das Ergebnis als ‘.txt‘-Datei aus und überprüfen dann mithilfe von Matlab, ob die Berechnung korrekt ist:

```
1.      FILE *maskdata=fopen("maskdata.txt","w");
2.
3.     for(i=0;i<y;i++){
4.         for(j=0;j<x;j++){
5.             fprintf(maskdata,"%d,", maskDefectPixels[i*x+j]);
6.         }
7.         fprintf(maskdata,"\n");
8.     }
9.     fclose(maskdata);
```

Programm 33: Ergebnis exportieren

Man ändert den ursprünglichen Matlab-Code und fügt zur Überprüfung am Ende den folgenden Code hinzu:

```
1. data=importdata('C:\Users\wyq\Desktop\maskdata.txt');
2.
3.     falschzaehler=0;
4.     f=1;
5.
6.
7.       for i=1:numberRows
8.        for ii=1:numberCols
9.           if(maskPixelsThreshold(i,ii)-data(i,ii)) // Recording error
10.              falschzaehler=falschzaehler+1;
11.              falsch(1,f)=i;
12.              falsch(2,f)=ii;
13.              falsch(3,f)=maskPixelsThreshold(i,ii);
14.              falsch(4,f)=data(i,ii);
15.              f=f+1;
16.          end
17.       end
18.    end
```

Programm 34: Fehlerprüfung

Nach dem Ausführen des Programms werden die folgenden Ergebnisse erhalten.

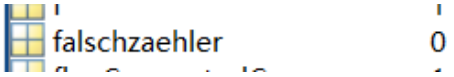

Abbildung 34: Überprüfungsergebnis

Wie in der Abbildung gezeigt, beträgt die Anzahl der Fehler ‘0‘ und das Berechnungsergebnis des C-Programms ist richtig.

Nachdem man das richtige ‘maskPixel‘ erhalten hat, werden den Grauwert und ‘Connected Components‘ berechnet. Sie müssen in die Mittel des Programm 34 eingefügt werden.

1. Connected Components:

```
1.      if(LowThresh[i][j] < temp1){
2.                      maskDefectPixels[pos] = 0;
3.                  }
4.                  else {
5.
6.                      maskDefectPixels[pos] = 1;
7.
8.                      flag = 0;
9.  // check if previously processed neighbouring pixels in 8 neighbourhood were
    detected
10.                     if (maskDefectPixels[pos-1] > 0 ){
11.                         rt[pos] = rt[pos-1];
12.                         flag = rt[pos];
13.                     }
14.
15.      if(i!=0){
16.       for (k = -1; k <= 1; k++){
17.        posk = pos-x-k;
18.        if (maskDefectPixels[posk] > 0){
19.         if (flag == 0){
20.          rt[pos] = rt[posk];
21.          flag = rt[pos];
22.         }
23.         else if (rt[posk] != flag){
24.          if(list[rt[posk]]<list[flag]){
25.            list[list[flag]]=list[rt[posk]]; //  build the list
26.            list[flag]=list[rt[posk]];
27.           }
28.           else {
29.            list[list[rt[posk]]]=list[flag];
30.            list[rt[posk]]=list[flag];
```

```
31.           }
32.
33.         }
34.        }
35.       }
36.         if (flag == 0) {     // increment region counter
37.           numRegion += 1;
38.           rt[pos] = numRegion;
39.         }
40.     }
41. }
```

Programm 35: Aktualisierung der Liste

Hier ist das Array ‘rt‘ das Berechnungsergebnis der ‘Connected Components‘. ‘flag‘ ist eine Variable, die zum vorübergehenden Speichern der benachbarten Pixelinformationen des Pixels verwendet wird. Der Anfangswert der Variable ‘flag‘ ist ‘0‘. Wenn Pixel ungleich Null vorhanden sind, ändert sich ‘flag‘ auf den Wert dieses Pixels (Programm Zeile 9 ~ 23). Dann wird die ‘list‘ aktualisiert.

2. Grauwert

```
1.         accGreyValues[rt[pos]] += temp1; // accumlate numerator and denominator for
   ratioGW
2.         accMeanImRef[rt[pos]] += meanImRef[i][j];
3.
4.         if (accMeanImRef[rt[pos]] > 1000){  // nomalize accumulated values to prevent
   overflow
5.             accMeanImRef[rt[pos]] = accMeanImRef[rt[pos]]-1000;
6.             numNormaccMeanImRef[rt[pos]] += 1;
7.        }
8.         if (accGreyValues[rt[pos]] > 1000){
9.             accGreyValues[rt[pos]] = accGreyValues[rt[pos]]-1000;
10.            numNormaccGreyValues[rt[pos]] += 1;
11.      }
```

Programm 36: Grauwertberechnung

Die Grauwerte werden gemäß der ‘list‘ summiert.

Am Ende des Schleifenprogramms teilt man den erhaltenen Grauwert durch den Referenzwert, um die Abweichungsrate zu erhalten:

```
1.  for( k= 1; k <= numRegion; k++){    // calculate ratioGW
2.     ratioGW[k] = (double)(accGreyValues[k]+numNormaccGreyValues[k]*1000)/
3.             (double)(accMeanImRef[k]+numNormaccMeanImRef[k]*1000);
4.     gvlist[list[k]]+=accGreyValues[k]+numNormaccGreyValues[k]*1000;
```

```
5.  }
```

Programm 37: Berechnung der Abweichungsrate

5.5 Ausgabeergebnis

Schließlich wird die Ergebnisse auf den Bildschirm gedruckt und als Datei ausgegeben:

```
1.      printf("meanacc=%d\n",meanAcc);
2.     printf("powacc=%d\n",powAcc);
3.     printf("numPixTotal=%d\n",numPixTotal);
4.     for( k= 1; k <= numRegion; k++){
5.         printf("%d Zaehler %d \t Normierung %d \t Nenner %d  \t Normierung %d \t rat
   ioGW=%f\n",k,accGreyValues[k],numNormaccGreyValues[k],
6.                 accMeanImRef[k],numNormaccMeanImRef[k], ratioGW[k]);
7.     }
8.     printf("used time mean/std/numPixTotal calculation:%f seconds\n",timeuse);
9.
10.    FILE *maskdata=fopen("maskdata.txt","w");  //save the data in 'txt' file
11.    FILE *rtDetected=fopen("rtDetected.txt","w");
12.    for(i=0;i<y;i++){
13.        for(j=0;j<x;j++){
14.            fprintf(maskdata,"%d,", maskDefectPixels[i*x+j]);
15.            fprintf(rtDetected,"%d,", rt[i*x+j]);
16.        }
17.        fprintf(maskdata,"\n");
18.        fprintf(rtDetected,"\n");
19.    }
20.    fclose(maskdata);
21.    fclose(rtDetected);
22.
23.    FILE *ListSameSeg=fopen("list.txt","w");
24. fprintf(ListSameSeg,"Nummer :Region   GV_Summe\n");
25.    for(i=1;i<100;i++)
26.        fprintf(ListSameSeg,"%d :%d   %d\n", i,list[i],gvlist[i]);
27.    fclose(ListSameSeg);
```

Programm 38: Ergebnisse ausgeben

Nach dem Ausführen des vollständigen Programms werden die folgenden Ergebnisse auf dem Bildschirm angezeigt.

```
pi@raspberrypi:~/Desktop/ct $ ./test

execute times:10000
threshold stdddev:2.000000

x=168,y=192
meanacc=3427495
powacc=405575613
numPixTotal=275
1 Zaehler 402    Normierung 0    Nenner 575      Normierung 0    ratioGW=0.699130
2 Zaehler 171    Normierung 0    Nenner 207      Normierung 0    ratioGW=0.826087
3 Zaehler 125    Normierung 2    Nenner 166      Normierung 3    ratioGW=0.671194
4 Zaehler 106    Normierung 0    Nenner 148      Normierung 0    ratioGW=0.716216
5 Zaehler 395    Normierung 1    Nenner 30       Normierung 2    ratioGW=0.687192
6 Zaehler 48     Normierung 0    Nenner 69       Normierung 0    ratioGW=0.695652
7 Zaehler 267    Normierung 0    Nenner 427      Normierung 0    ratioGW=0.625293
8 Zaehler 98     Normierung 0    Nenner 143      Normierung 0    ratioGW=0.685315
9 Zaehler 253    Normierung 0    Nenner 375      Normierung 0    ratioGW=0.674667
10 Zaehler 38    Normierung 0    Nenner 61       Normierung 0    ratioGW=0.622951
11 Zaehler 580   Normierung 0    Nenner 787      Normierung 0    ratioGW=0.736976
12 Zaehler 399   Normierung 0    Nenner 478      Normierung 0    ratioGW=0.834728
13 Zaehler 531   Normierung 1    Nenner 117      Normierung 2    ratioGW=0.723193
14 Zaehler 502   Normierung 0    Nenner 643      Normierung 0    ratioGW=0.780715
15 Zaehler 461   Normierung 2    Nenner 303      Normierung 4    ratioGW=0.571927
16 Zaehler 319   Normierung 0    Nenner 393      Normierung 0    ratioGW=0.811705
17 Zaehler 215   Normierung 0    Nenner 261      Normierung 0    ratioGW=0.823755
18 Zaehler 201   Normierung 1    Nenner 515      Normierung 1    ratioGW=0.792739
19 Zaehler 53    Normierung 0    Nenner 66       Normierung 0    ratioGW=0.803030
20 Zaehler 963   Normierung 0    Nenner 265      Normierung 1    ratioGW=0.761265
used time mean/std/numPixTotal calculation:2.181229 seconds
```

Abbildung 35: Programmergebnis

Man kann die Datei ‘list.txt‘ öffnen, um die zugehörigen Informationen ‘Connected Components‘ anzuzeigen.

```
Nummer :Region   GV_Summe
1 :1   402
2 :2   2648
3 :3   2498
4 :3   0
5 :2   0
6 :6   48
7 :3   0
8 :8   98
9 :9   1822
10 :9   0
11 :2   0
12 :12   399
13 :9   0
14 :2   0
15 :15   2461
16 :16   319
17 :17   215
18 :18   1201
19 :19   53
20 :20   963
21 :21   0
22 :22   0
23 :23   0
24 :24   0
25 :25   0
```

Abbildung 36: list.txt

Die Struktur der Daten in dieser Datei ist wie folgt:

$$Nummer: \ Die\ Bereichsnummer, zu\ der\ die\ Nummer\ gehört \qquad Gesamtgrauwert$$

Zum Beispiel gehören die Nummern 9, 10 und 13 zur Bereichsnummer 9, und ihr Gesamtgrauwert beträgt 1822.

Bisher wurde das C-Programm abgeschlossen. Der nächste Schritt besteht darin, einen Teil davon mit NEON umzuschreiben, um zu versuchen, die Berechnungsgeschwindigkeit zu erhöhen.

6 Beschleunigung mit NEON

In diesem Kapitel versucht man, mit NEON den Teil des C-Sprachcodes zu beschleunigen und zu testen.

Hinweis: In diesem Teil muss man "arm_neon.h" als Header-Datei laden.

6.1 Teil 1: Mittelwert und Standardabweichung

Dieser Abschnitt versucht, den folgenden Code zu beschleunigen:

```
1.  for(int i=0;i<y;i++){
2.   for(int j=0;j<x;j++){
3.    unsigned int pos=i*x+j;
4.    int wert=bild[pos];
5.    meanAcc=meanAcc+wert;
6.    powAcc=powAcc+wert*wert;
7.   }
8.  }
```

Programm 39: meanAcc und powAcc

Der Zweck dieses Programms besteht darin, ein Bild mit einer Größe von 192 mal 168 zu laden und die Summe aller Pixelwerte und die Summe der Quadrate aller Pixelwerte zu berechnen.

Um einen Datenüberlauf zu vermeiden, sollt man vor Arbeitsbeginn das entsprechende NEON-Register auswählen. Deshalb schätzt man die Größe von meanAcc: meanAcc repräsentiert die Summe der Grauwerte aller Pixel im Bild. Da die Bildgröße 192 * 168 beträgt und der maximale Grauwert jedes Pixels 255 beträgt, wird die Größe von meanAcc 192 * 168 * 255 nicht überschreiten. Es ist kleiner als:

192*168*256 = 21*8*3*64*256 = 63*2^17 < 2^23

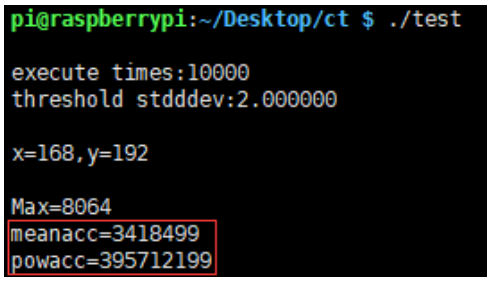

Abbildung 37: meanAcc

Aus der obigen Abbildung geht hervor, dass der tatsächliche meanAcc 3,418,499 beträgt. Es ist größer als 65,536(2^16). Zusammenfassend sollte man 32-Bit-Register wählen.

Als nächstes verwenden die Leute verschiedene Methoden zum Testen:

```
1.  int main(int argc,char** argv){
2.   size_t x,y;
3.   png_bytep bild;
4.   long times;
5.   if(argc >= 2 && argv[1]!=NULL)
6.    times = strtol(argv[1],NULL,10);
7.   else
8.    times = 1;
9.   printf("\nexecute times:%d\n\n",times);
10.  struct timeval tpstart,tpend;
11.  float timeuse0,timeuse,Multi;
12.  readimg("test.png",&bild,&x,&y);  //load the picture
13.  uint32_t *list=malloc(sizeof(uint32_t)*192*168);
14.  uint16_t *list_16=malloc(sizeof(uint16_t)*192*168);
15.  uint32_t *out1=malloc(sizeof(uint32_t)*4);
16.  uint32_t *out2=malloc(sizeof(uint32_t)*4);
17.  for(int i=0;i<192*168;i++){
18.   list[i]=bild[i];
19.   list_16[i]=bild[i];
20.  }
21.         ...
22.
23.  free(list);  //Release memory
24.  free(list_16);
25.  free(out1);
26.  free(out2);
27. }
```

Programm 40: Bild laden

Dieser ganze Teil ist die Programminitialisierung und das Laden von Bildern. Es wird auch aufgezeichnet, wie oft das gesamte Programm ausgeführt wird. Diese Funktion wird nur einmal aufgerufen.

Zeile 23: Menschen verwenden '*malloc()*', um Speicher manuell zuzuweisen.

Zeile 23: Der letzte Teil des Codes besteht darin, Speicher freizugeben.

```
1.  int meanAcc,powAcc;
2.   gettimeofday(&tpstart,NULL);  // start the timer
3.   for(int ttt=0;ttt<times;ttt++){
4.    meanAcc=0;powAcc=0;
5.    for(int i=0;i<192;i++){
```

```
6.     for(int j=0;j<168;j++){
7.      meanAcc = meanAcc + list[i*168+j];
8.      powAcc = powAcc + list[i*168+j]*list[i*168+j];
9.     }
10.   }
11.  }
12.  gettimeofday(&tpend,NULL);  //end the timer
13.  timeuse0=1000000*(tpend.tv_sec-tpstart.tv_sec)+tpend.tv_usec-tpstart.tv_usec;
14.  timeuse0/=1000000;
15.  printf("meanacc=%d\n",meanAcc);
16.  printf("powacc=%d\n",powAcc);
17.  printf("used time(C)=%f seconds\n\n",timeuse0);
```

Programm 41: Programm ausführen mit C-Programm

Der zweite Teil besteht darin, die C-Programm zu verwenden, um das gesamte Programm auszuführen. Dieser Teil verwendet die Linux-Timing-Funktion. Die Header-Datei lautet *#include <sys / time.h>*.

```
1.  uint32_t x0,x1;
2.   uint32_t awert_32[]={0,0,0,0};  // Set initial value
3.   uint32x4_t a,mult,meanacc,powacc;
4.   gettimeofday(&tpstart,NULL);
5.   for(int ttt=0;ttt<times;ttt++){
6.    meanacc = vld1q_u32(&awert_32[0]);
7.    powacc = vld1q_u32(&awert_32[0]);
8.    for(int i=0;i<192*168;i+=4){
9.     a = vld1q_u32(&list[i]);
10.    meanacc = vaddq_u32(meanacc,a);
11.    mult = vmulq_u32(a,a);
12.    powacc = vaddq_u32(powacc,mult);
13.   }
14.  }
15.  vst1q_u32(&out1[0],meanacc);
16.  vst1q_u32(&out2[0],powacc);
17.  x0=out1[0]+out1[1]+out1[2]+out1[3];  // Output result
18.  x1=out2[0]+out2[1]+out2[2]+out2[3];
19.  gettimeofday(&tpend,NULL);
20.  timeuse=1000000*(tpend.tv_sec-tpstart.tv_sec)+tpend.tv_usec-tpstart.tv_usec;
21.  timeuse/=1000000;
22.  printf("meanacc=%d \n",x0);
23.  printf("powacc=%d \n",x1);
24.  printf("used time(neon origin ver.):%f seconds\n",timeuse);
25.  printf("%f times faster\n\n",timeuse0/timeuse);
```

Programm 42: Versuch 1: 32x4 Q-Register

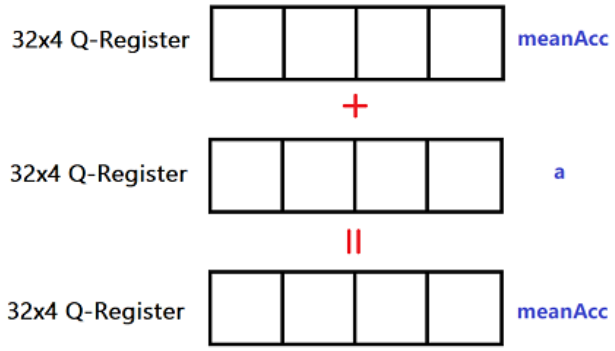

Abbildung 38: Schematische Darstellung-32x4 Q-Register

Diese Messung verwendet ein 32x4 Q-Register, 4 Elemente werden als Gruppe berechnet, der Anfangswert ist [0, 0, 0, 0]. Das Prinzip besteht darin, die Werte im ersten und im zweiten Register zu addieren und das Ergebnis in das erste Register zu schreiben.

In NEON-Funktionen repräsentieren Funktionen mit demselben Präfix denselben Funktionstyp:

vld1-: Diese Funktion dient zum Laden der Daten im Speicher in das Register.

vadd-: Seine Funktion ist das Hinzufügen von Daten.

vmul-: Die Elemente in den beiden Registern werden multipliziert.

vst1-: Das Ergebnis im Register wird in den Speicher ausgegeben.

Zeile 1: x0 ist das Ergebnis von meanAcc und x1 ist das Ergebnis von powAcc.

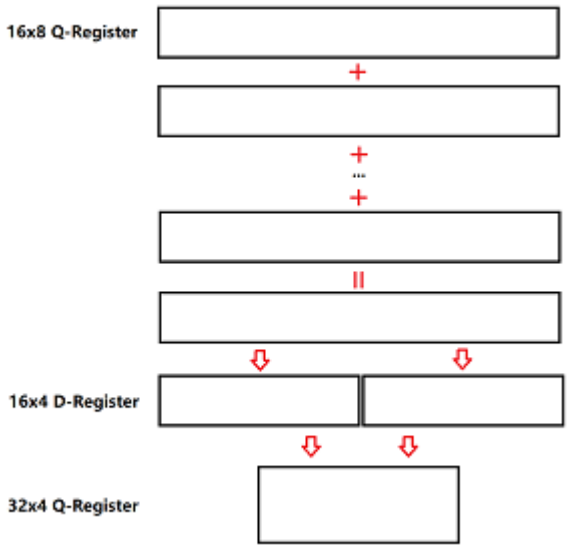

Abbildung 39: Schematische Darstellung

Die folgenden Experimente sind ähnlich wie die vorherigen Experimente. Die Auswirkung der Verwendung einer unterschiedlichen Anzahl von 16x4-D-Registern auf die experimentellen Ergebnisse wurde getestet.

```
1.  uint16x8_t rega1,rega2,rega5;
2.   uint16x4_t rega3,rega4;
3.   uint16_t awert_16[]={0,0,0,0,0,0,0,0};  // Set initial value
4.   gettimeofday(&tpstart,NULL);
5.   for(int ttt=0;ttt<times;ttt++){
6.    meanacc = vld1q_u32(&awert_32[0]);  //Load data to Neon-register
7.    powacc = vld1q_u32(&awert_32[0]);
8.    for(int i=0;i<192*168;i+=16){
9.     rega1 = vld1q_u16(&list_16[i]);
10.    rega2 = vld1q_u16(&list_16[i+8]);
11.    rega5 = vaddq_u16(rega1,rega2);
12.    rega3 = vget_high_u16(rega5);
13.    rega4 = vget_low_u16(rega5);
14.    meanacc = vaddw_u16(meanacc,rega3);
15.    meanacc = vaddw_u16(meanacc,rega4);
16.
17.    powacc = mulpow(rega1,powacc);
18.    powacc = mulpow(rega2,powacc);
19.   }
20.  }
21.  vst1q_u32(&out1[0],meanacc);
22.  vst1q_u32(&out2[0],powacc);
23.  x0=out1[0]+out1[1]+out1[2]+out1[3]; //Output result
24.  x1=out2[0]+out2[1]+out2[2]+out2[3];
25.  gettimeofday(&tpend,NULL);
26.  timeuse=1000000*(tpend.tv_sec-tpstart.tv_sec)+tpend.tv_usec-tpstart.tv_usec;
27.  timeuse/=1000000;
```

```
28. printf("meanacc=%d \n",x0);
29. printf("powacc=%d \n",x1);
30. printf("used time(neon 2nd ver.):%f seconds\n",timeuse);
31. printf("%f times faster\n\n",timeuse0/timeuse);
```

*Programm 43: Versuch 2: 16x8 Q-Register,dann 2*16x4->32x4 Register*

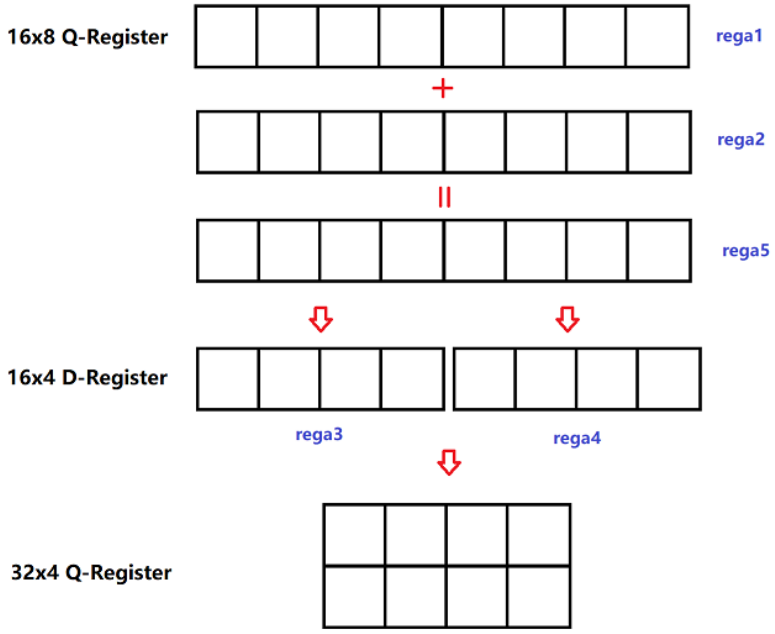

Abbildung 40: Ver. 2 - Schematische Darstellung

Das 16x8 Q-Register hat viele Kanäle und eine schnelle Berechnungsgeschwindigkeit. Das Ergebnis wird in einem 32-Bit-Register über zwei 16x4-D-Register gespeichert. Die Methode besteht darin, zuerst mit zwei 16x8-Q-Registern zu berechnen, das Ergebnis zum dritten 16x8-Q-Register hinzuzufügen, dann das Ergebnis in zwei 16x4-D-Register aufzuteilen und das Ergebnis schließlich zu 32x4-Q-Registern hinzuzufügen.

vget_high_u16: Daten werden in die oberen Bits des Registers geladen.

vget_low_u16: Daten werden in die unteren Bits des Registers geladen.

```
1. uint16x8_t rega6,rega7;
2.  gettimeofday(&tpstart,NULL);
3.  for(int ttt=0;ttt<times;ttt++){
4.   meanacc = vld1q_u32(&awert_32[0]);
5.   powacc = vld1q_u32(&awert_32[0]);
6.   for(int i=0;i<192*168;i+=24){
```

```
7.      rega1 = vld1q_u16(&list_16[i]);  // Load three sets of data at once
8.      rega2 = vld1q_u16(&list_16[i+8]);
9.      rega6 = vld1q_u16(&list_16[i+16]);
10.     rega5 = vaddq_u16(rega1,rega2);  // Add three sets of data
11.     rega7 = vaddq_u16(rega6,rega5);
12.     rega3 = vget_high_u16(rega7);
13.     rega4 = vget_low_u16(rega7);
14.     meanacc = vaddw_u16(meanacc,rega3);
15.     meanacc = vaddw_u16(meanacc,rega4);  // send the sum to register meanAcc
16.
17.     rega1 = vmulq_u16(rega1,rega1);  // calculate powAcc with similar operations
18.     rega3 = vget_high_u16(rega1);
19.     rega4 = vget_low_u16(rega1);
20.     powacc = vaddw_u16(powacc,rega3);
21.     powacc = vaddw_u16(powacc,rega4);
22.     rega2 = vmulq_u16(rega2,rega2);
23.     rega3 = vget_high_u16(rega2);
24.     rega4 = vget_low_u16(rega2);
25.     powacc = vaddw_u16(powacc,rega3);
26.     powacc = vaddw_u16(powacc,rega4);
27.     rega6 = vmulq_u16(rega6,rega6);
28.     rega3 = vget_high_u16(rega6);
29.     rega4 = vget_low_u16(rega6);
30.     powacc = vaddw_u16(powacc,rega3);
31.     powacc = vaddw_u16(powacc,rega4);
32.   }
33. }
34. vst1q_u32(&out1[0],meanacc);
35. vst1q_u32(&out2[0],powacc);
36. x0=out1[0]+out1[1]+out1[2]+out1[3];  //Output result
37. x1=out2[0]+out2[1]+out2[2]+out2[3];
38. gettimeofday(&tpend,NULL);
39. timeuse=1000000*(tpend.tv_sec-tpstart.tv_sec)+tpend.tv_usec-tpstart.tv_usec;
40. timeuse/=1000000;
41. printf("meanacc=%d \n",x0);
42. printf("powacc=%d \n",x1);
43. printf("used time(neon 3rd ver.):%f seconds\n",timeuse);
44. printf("%f times faster\n\n",timeuse0/timeuse);
```

Programm 44: Versuch 3: 16x8 Q-Register,dann 3*16x4->32x4 Register

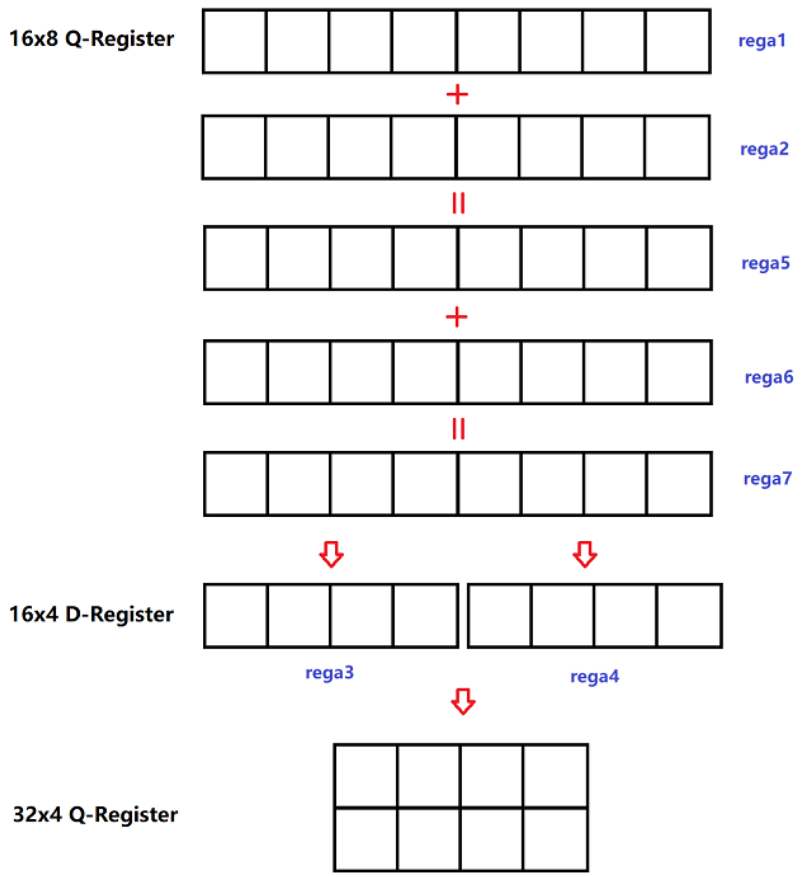

Abbildung 41: Ver. 3 - Schematische Darstellung

Um das Programm übersichtlicher und übersichtlicher zu gestalten, wird diese benutzerdefinierte Funktion verwendet:

```
1.  uint32x4_t mulpow(uint16x8_t xx,uint32x4_t yy){  //Calculate the square of xx and
    add it to yy
2.   uint16x4_t y1,y2;
3.   xx = vmulq_u16(xx,xx);
4.   y1 = vget_high_u16(xx);
5.   y2 = vget_low_u16(xx);
6.   yy = vaddw_u16(yy,y1);
7.   yy = vaddw_u16(yy,y2);
8.
9.   return yy;
10. }
```

Programm 45: Benutzerdefinierte Funktion

Leider ist die Berechnungsgeschwindigkeit langsam, wenn die benutzerdefinierte Funktion verwendet wird.

*Die nächsten drei Versuche sind ähnlich wie Versuch 3. (siehe Anlage 6: Versuch 4, 5, 6)

Die Anzahl der Programmläufe beträgt 10.000 Mal und die Ergebnisse:

```
pi@raspberrypi:~/Desktop/neontest $ ./test 10000

execute times:10000

meanacc=3757261
powacc=506884501
used time(C)=5.881601 seconds

meanacc=3757261
powacc=506884501
used time(neon origin ver.):2.559020 seconds
2.298380 times faster

meanacc=3757261
powacc=506884501
used time(neon 2nd ver.):3.137295 seconds
1.874736 times faster

meanacc=3757261
powacc=506884501
used time(neon 3rd ver.):2.414327 seconds
2.436125 times faster

meanacc=3757261
powacc=506884501
used time(neon 4th ver.):2.161771 seconds
2.720732 times faster

meanacc=3761263
powacc=506901719
used time(neon 5th ver.):2.541537 seconds
2.314190 times faster

meanacc=3757261
powacc=506884501
used time(neon 6th ver.):2.399462 seconds
2.451216 times faster
```

Abbildung 42: Testergebnis

Aus den Testergebnissen ist ersichtlich: Alle Versuche haben korrekte Ergebnisse erzielt und Versuch 4 hat den besten Beschleunigungseffekt. Es ist 2,7-mal schneller als das ursprüngliche C-Programm.

Als nächstes setzt man Versuch 4 in das ursprüngliche Programm ein, um die Laufzeit zu testen. Die Ergebnisse sind wie folgt:

```
pi@raspberrypi:~/Desktop/ct $ ./test

execute times:10000
threshold stdddev:2.000000

x=168,y=192
meanacc=3427495
powacc=405575613
numPixTotal=275
1 Zaehler 402    Normierung 0    Nenner 575      Normierung 0    ratioGW=0.699130
2 Zaehler 171    Normierung 0    Nenner 207      Normierung 0    ratioGW=0.826087
3 Zaehler 125    Normierung 2    Nenner 166      Normierung 3    ratioGW=0.671194
4 Zaehler 106    Normierung 0    Nenner 148      Normierung 0    ratioGW=0.716216
5 Zaehler 395    Normierung 1    Nenner 30       Normierung 2    ratioGW=0.687192
6 Zaehler 48     Normierung 0    Nenner 69       Normierung 0    ratioGW=0.695652
7 Zaehler 267    Normierung 0    Nenner 427      Normierung 0    ratioGW=0.625293
8 Zaehler 98     Normierung 0    Nenner 143      Normierung 0    ratioGW=0.685315
9 Zaehler 253    Normierung 0    Nenner 375      Normierung 0    ratioGW=0.674667
10 Zaehler 38    Normierung 0    Nenner 61       Normierung 0    ratioGW=0.622951
11 Zaehler 580   Normierung 0    Nenner 787      Normierung 0    ratioGW=0.736976
12 Zaehler 399   Normierung 0    Nenner 478      Normierung 0    ratioGW=0.834728
13 Zaehler 531   Normierung 1    Nenner 117      Normierung 2    ratioGW=0.723193
14 Zaehler 502   Normierung 0    Nenner 643      Normierung 0    ratioGW=0.780715
15 Zaehler 461   Normierung 2    Nenner 303      Normierung 4    ratioGW=0.571927
16 Zaehler 319   Normierung 0    Nenner 393      Normierung 0    ratioGW=0.811705
17 Zaehler 215   Normierung 0    Nenner 261      Normierung 0    ratioGW=0.823755
18 Zaehler 201   Normierung 1    Nenner 515      Normierung 1    ratioGW=0.792739
19 Zaehler 53    Normierung 0    Nenner 66       Normierung 0    ratioGW=0.803030
20 Zaehler 963   Normierung 0    Nenner 265      Normierung 1    ratioGW=0.761265
used time mean/std/numPixTotal calculation:2.266306 seconds
```

Abbildung 43: Laufzeit vor Änderung

```
pi@raspberrypi:~/Desktop/ct $ ./test

execute times:10000
threshold stdddev:2.000000

x=168,y=192
x=84,y=96
meanIm=106.259146
stdImInv=0.418831
bild=110
LowTh=42.457703
meanacc=3427495
powacc=405575613
numPixTotal=275
1 Zaehler 402    Normierung 0    Nenner 575      Normierung 0    ratioGW=0.699130
2 Zaehler 171    Normierung 0    Nenner 207      Normierung 0    ratioGW=0.826087
3 Zaehler 125    Normierung 2    Nenner 166      Normierung 3    ratioGW=0.671194
4 Zaehler 106    Normierung 0    Nenner 148      Normierung 0    ratioGW=0.716216
5 Zaehler 395    Normierung 1    Nenner 30       Normierung 2    ratioGW=0.687192
6 Zaehler 48     Normierung 0    Nenner 69       Normierung 0    ratioGW=0.695652
7 Zaehler 267    Normierung 0    Nenner 427      Normierung 0    ratioGW=0.625293
8 Zaehler 98     Normierung 0    Nenner 143      Normierung 0    ratioGW=0.685315
9 Zaehler 253    Normierung 0    Nenner 375      Normierung 0    ratioGW=0.674667
10 Zaehler 38    Normierung 0    Nenner 61       Normierung 0    ratioGW=0.622951
11 Zaehler 580   Normierung 0    Nenner 787      Normierung 0    ratioGW=0.736976
12 Zaehler 399   Normierung 0    Nenner 478      Normierung 0    ratioGW=0.834728
13 Zaehler 531   Normierung 1    Nenner 117      Normierung 2    ratioGW=0.723193
14 Zaehler 502   Normierung 0    Nenner 643      Normierung 0    ratioGW=0.780715
15 Zaehler 461   Normierung 2    Nenner 303      Normierung 4    ratioGW=0.571927
16 Zaehler 319   Normierung 0    Nenner 393      Normierung 0    ratioGW=0.811705
17 Zaehler 215   Normierung 0    Nenner 261      Normierung 0    ratioGW=0.823755
18 Zaehler 201   Normierung 1    Nenner 515      Normierung 1    ratioGW=0.792739
19 Zaehler 53    Normierung 0    Nenner 66       Normierung 0    ratioGW=0.803030
20 Zaehler 963   Normierung 0    Nenner 265      Normierung 1    ratioGW=0.761265
used time mean/std/numPixTotal calculation:2.033724 seconds
```

Abbildung 44: Laufzeit nach Änderung

Nach den beiden obigen Bildern ist die Laufzeit ca. 10% schneller. Diese Methode hat den gewünschten Effekt erzielt.

6.2 Teil 2: MaskPixelsThreshold

Als nächstes versucht man den folgenden Code zu beschleunigen:

```
1.  for(i=0;i<y;i++){
2.      for(j=1;j<x;j++){
3.              pos = i*x+j;
4.              temp1=(bild[pos]-meanIm)*stdImInv+80;  // normalized pixel
5.              rt[pos] = 0;
6.
7.              if(LowThresh[i][j] < temp1){
8.                   maskDefectPixels[pos] = 0;
9.               }
10.             else {
11.                 maskDefectPixels[pos] = 1;
12.             }
13.      }
14. }
```

Programm 46: original Programm

Der Zweck dieses Programms ist die Berechnung der MaskPixel.

Dieses Programm beginnt mit dem Scannen ab dem zweiten Pixel des Bildes und die Bildgröße beträgt 192 * 168. Nach dem Testen sollte das NEON-Register von Float32 * 4 verwendet werden, es kann nur 4 Pixel gleichzeitig berechnen. Am Ende gibt es also 3 zusätzliche Pixel und das Programm läuft schief ((192 * 168-1) / 4 ist keine ganze Zahl). Daher soll man stattdessen ab dem ersten Pixel berechnen (Zeile 2: j=1 → j=0).

Da dieses Programm MaskPixel durch Vergleichen der Größe von ‚LowThresh' und 'temp1' generiert, kann man die Vergleichsfunktion in NEON verwenden, um dieses Programm zu beschleunigen:

```
1.   uint32x4_t regb_out;
2.   float32x4_t regb1,regb2,regmean,regstd;
3.   float32_t konst80[4]={80.0,80.0,80.0,80.0};  // Constant array
4.   float32x4_t reg80=vld1q_f32(&konst80[0]);
5.   uint32_t res;
6.   uint32_t *mask_f32=malloc(sizeof(uint32_t)*4);
7.   float32_t *bild_32f=malloc(sizeof(float32_t)*192*168);
8.
9.   for(int i=0;i<192*168;i++){
10.   bild_32f[i]=bild[i];
11.  }
```

Programm 47: Parametereinstellungen

Dieser Teil ist die Definition von Variablen. Unten ist der Hauptteil:

```
1.  for(i=0;i<y;i++){
2.      for(j=0;j<x;j+=4){
3.
4.       regb1=vld1q_f32(&bild_32f[i*x+j]);
5.       regb1=vsubq_f32(regb1,regmean);
6.       regb1=vmulq_f32(regb1,regstd);
7.       regb1=vaddq_f32(regb1,reg80);
8.       vst1q_f32(&temp[0],regb1);
9.       regb2=vld1q_f32(&LowThresh[i*x+j]);
10.      regb_out=vcltq_f32(regb1,regb2);
11.      vst1q_u32(&mask_f32[0],regb_out);
12.
13.     for(j2=0;j2<4;j2++){
14.
15.      pos = i*x+j+j2;
16.      rt[pos] = 0;
17.      res=mask_f32[j2]>>31;
18.      maskDefectPixels[pos]= res & 1;
19.      }
20.   }
21. }
```

Programm 48: Mit NEON geschriebenem Programm

Zeile 5: ‘vsub-‘ repräsentiert die Subtraktion von zwei Datensätzen im Register.

Die Funktion ‘vcltq_f32(regb1,regb2)‘ vergleicht vier Datensätze im Register und gibt dann vier entsprechende 32-Bit-Ganzzahlen aus. Wenn rega1 kleiner als rega2 ist, wird ‘0xFFFFFFFF‘ ausgegeben. Andernfalls wird ‘0‘ ausgegeben [1]. Man möchte jedoch, dass das Ergebnis 1 oder 0 ist, sodass man das Ergebnis der Funktion um 31 Bit nach rechts verschiebt. Schließlich werden das Ergebnis und 1 UND-verknüpft.

Nach der Inspektion ist das Ergebnis der Programmoperation korrekt. Die Inspektionsmethode stimmt mit der Methode in Kapitel 5 überein.

Das Ergebnis der Laufzeit:

```
pi@raspberrypi:~/Desktop/ct $ ./test

execute times:10000
threshold stdddev:2.000000

x=168,y=192
x=84,y=96
meanIm=106.259146
stdImInv=0.418831
bild=110
LowTh=42.457703
meanacc=3427495
powacc=405575613
numPixTotal=275
1 Zaehler 402    Normierung 0    Nenner 575      Normierung 0    ratioGW=0.699130
2 Zaehler 171    Normierung 0    Nenner 207      Normierung 0    ratioGW=0.826087
3 Zaehler 125    Normierung 2    Nenner 166      Normierung 3    ratioGW=0.671194
4 Zaehler 106    Normierung 0    Nenner 148      Normierung 0    ratioGW=0.716216
5 Zaehler 395    Normierung 1    Nenner 30       Normierung 2    ratioGW=0.687192
6 Zaehler 48     Normierung 0    Nenner 69       Normierung 0    ratioGW=0.695652
7 Zaehler 267    Normierung 0    Nenner 427      Normierung 0    ratioGW=0.625293
8 Zaehler 98     Normierung 0    Nenner 143      Normierung 0    ratioGW=0.685315
9 Zaehler 253    Normierung 0    Nenner 375      Normierung 0    ratioGW=0.674667
10 Zaehler 38    Normierung 0    Nenner 61       Normierung 0    ratioGW=0.622951
11 Zaehler 580   Normierung 0    Nenner 787      Normierung 0    ratioGW=0.736976
12 Zaehler 399   Normierung 0    Nenner 478      Normierung 0    ratioGW=0.834728
13 Zaehler 531   Normierung 1    Nenner 117      Normierung 2    ratioGW=0.723193
14 Zaehler 502   Normierung 0    Nenner 643      Normierung 0    ratioGW=0.780715
15 Zaehler 461   Normierung 2    Nenner 303      Normierung 4    ratioGW=0.571927
16 Zaehler 319   Normierung 0    Nenner 393      Normierung 0    ratioGW=0.811705
17 Zaehler 215   Normierung 0    Nenner 261      Normierung 0    ratioGW=0.823755
18 Zaehler 201   Normierung 1    Nenner 515      Normierung 1    ratioGW=0.792739
19 Zaehler 53    Normierung 0    Nenner 66       Normierung 0    ratioGW=0.803030
20 Zaehler 963   Normierung 0    Nenner 265      Normierung 1    ratioGW=0.761265
used time mean/std/numPixTotal calculation:2.030261 seconds
```

Abbildung 45: Laufzeit vor Änderung

```
pi@raspberrypi:~/Desktop/ct $ ./test

execute times:10000
threshold stdddev:2.000000

x=168,y=192
x=84,y=96
meanacc=3427495
powacc=405575613
numPixTotal=275
1 Zaehler 402    Normierung 0    Nenner 570      Normierung 0    ratioGW=0.705263
2 Zaehler 171    Normierung 0    Nenner 206      Normierung 0    ratioGW=0.830097
3 Zaehler 125    Normierung 2    Nenner 151      Normierung 3    ratioGW=0.674389
4 Zaehler 106    Normierung 0    Nenner 148      Normierung 0    ratioGW=0.716216
5 Zaehler 395    Normierung 1    Nenner 16       Normierung 2    ratioGW=0.691964
6 Zaehler 48     Normierung 0    Nenner 71       Normierung 0    ratioGW=0.676056
7 Zaehler 267    Normierung 0    Nenner 420      Normierung 0    ratioGW=0.635714
8 Zaehler 98     Normierung 0    Nenner 140      Normierung 0    ratioGW=0.700000
9 Zaehler 253    Normierung 0    Nenner 379      Normierung 0    ratioGW=0.667546
10 Zaehler 38    Normierung 0    Nenner 61       Normierung 0    ratioGW=0.622951
11 Zaehler 580   Normierung 0    Nenner 775      Normierung 0    ratioGW=0.748387
12 Zaehler 399   Normierung 0    Nenner 470      Normierung 0    ratioGW=0.848936
13 Zaehler 531   Normierung 1    Nenner 111      Normierung 2    ratioGW=0.725249
14 Zaehler 502   Normierung 0    Nenner 640      Normierung 0    ratioGW=0.784375
15 Zaehler 461   Normierung 2    Nenner 201      Normierung 4    ratioGW=0.585813
16 Zaehler 319   Normierung 0    Nenner 394      Normierung 0    ratioGW=0.809645
17 Zaehler 215   Normierung 0    Nenner 261      Normierung 0    ratioGW=0.823755
18 Zaehler 201   Normierung 1    Nenner 527      Normierung 1    ratioGW=0.786509
19 Zaehler 53    Normierung 0    Nenner 67       Normierung 0    ratioGW=0.791045
20 Zaehler 963   Normierung 0    Nenner 277      Normierung 1    ratioGW=0.754111
used time mean/std/numPixTotal calculation:2.285535 seconds
```

Abbildung 46: Laufzeit nach Änderung

Aus diesen beiden Bildern ist ersichtlich, dass bei Verwendung von NEON die Laufzeit länger ist. Denn obwohl NEON mehrere Daten gleichzeitig berechnen kann, dauert es einige Zeit, Daten in und aus dem NEON-Register zu schreiben und zu lesen.

6.3 Einfluss des Schwellenwerts auf die Laufzeit

Durch die Kombination der vorherigen Inhalte verwendet man den schnellsten Algorithmus, um die Auswirkungen verschiedener Schwellenwerte auf die Laufzeit zu berechnen und die folgenden Ergebnisse zu erhalten:

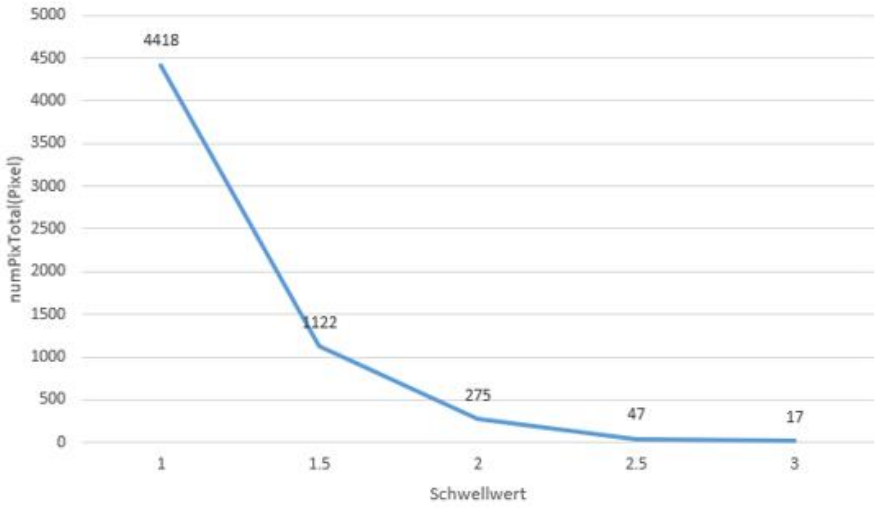

Abbildung 47: numPixelTotal-Schwellwert und Normalverteilung

Die obige Abbildung zeigt die Beziehung zwischen dem Schwellenwert und der Anzahl abnormaler Pixel, und ihre Beziehung folgt einer Normalverteilung. Je kleiner der

Schwellenwert ist, desto größer ist die Anzahl abnormaler Pixel. Und mehr abnormale Pixel bedeuten mehr Berechnungen, was zu einer längeren Laufzeit führt.

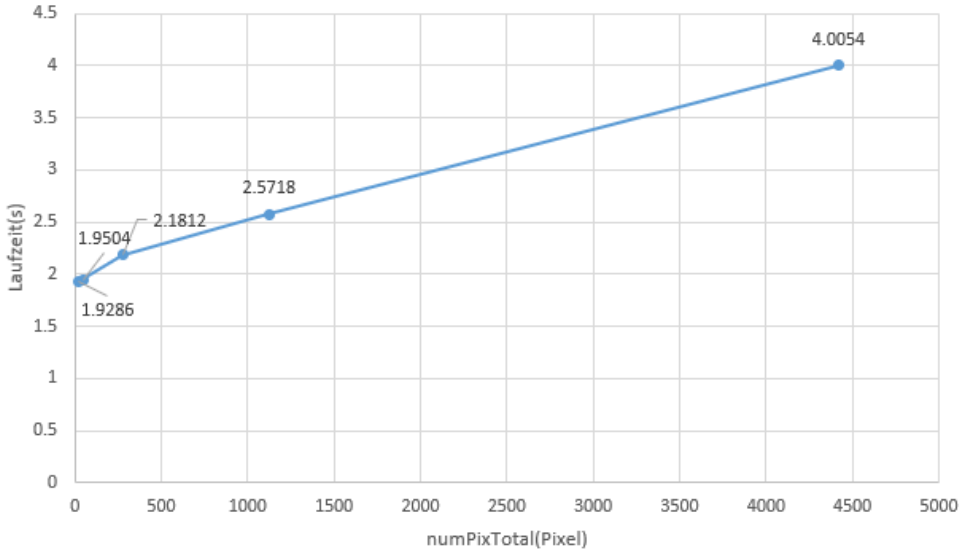

Abbildung 48: Laufzeit-numPixelTotal

Aus dem Diagramm der Anzahl abnormaler Pixel und der Laufzeit ist ersichtlich, dass sie nahezu linear sind.

6.4 Einfluss der Bildgröße auf die Laufzeit

Offensichtlich wirken sich unterschiedliche Bildgrößen definitiv auf die Laufzeit aus, sodass man kleinerer Bildteile aus den Bildern berechnet.

$$temp1 = ImN = \ （Bild - meanAcc/x/y） * stdImInv + 80 \qquad (6\text{-}1)$$

Da meanAcc in der Normalisierungsformel enthalten ist, ist es das Ergebnis der Ausführung des gesamten Bildes. Als nächstes wird nur die Normalisierung mit NEON und die nachfolgenden Algorithmen getestet. Die Testergebnisse sind in der folgenden Abbildung dargestellt:

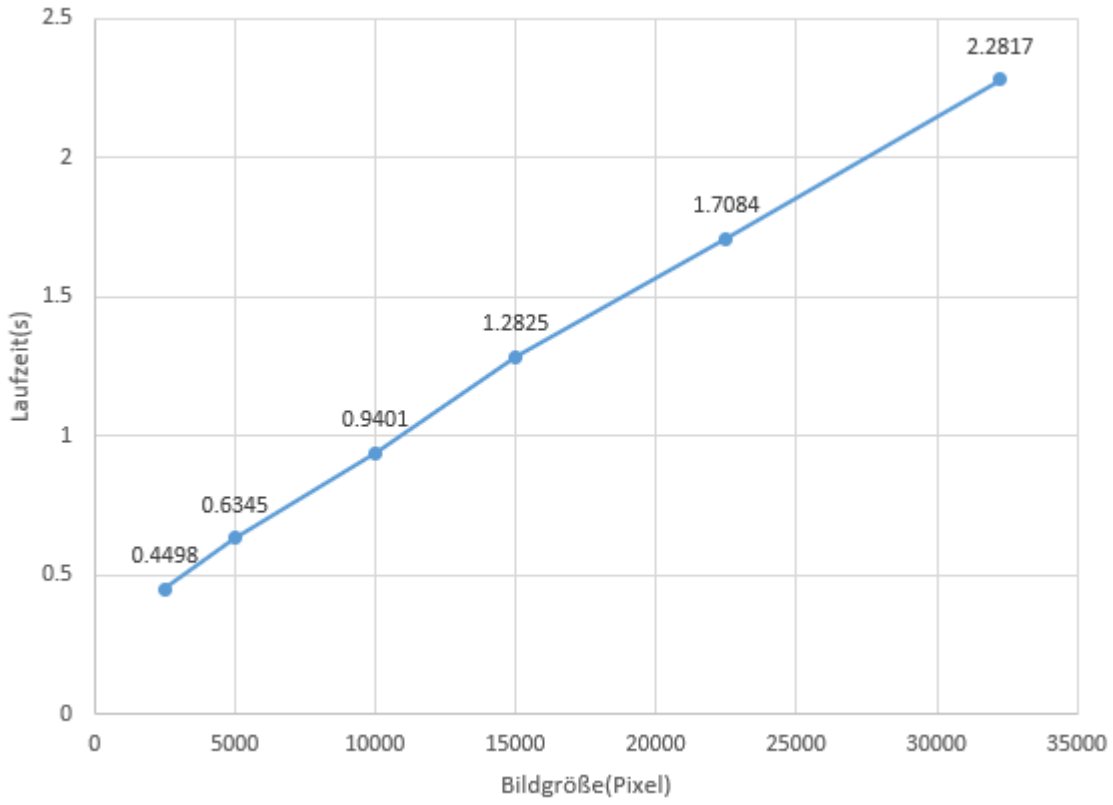

Abbildung 49: Laufzeit-Bildgröße

Aus der obigen Abbildung ist ersichtlich, dass die Größe des Bildes und die Laufzeit linear ist. Je größer das Bild, desto länger die Laufzeit.

7 Zusammenfassung

Diese Arbeit umfasst hauptsächlich der Bildrotationsalgorithmus, Konvertierung des MATLAB-Programms in das C-Programm und die Beschleunigung des Artefaktalgorithmus unter Verwendung von NEON. Am Ende wird es gefunden:

1. Die Drehung beliebiger Bilder wird realisiert (Siehe Kapitel 3).
2. Die Verwendung des optimalen Algorithmus mit NEON reduziert die Gesamtlaufzeit des Programms um 10% (Siehe Kapitel 6.1).

Beim Testen der Laufzeit des Programms stellten die Benutzer fest, dass NEON nur in einigen Fällen beschleunigen und in anderen Fällen sogar langsamer werden kann (Siehe Kapitel 6.2).

Literaturverzeichnis

[1] ARM Developer. (2020). Von https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/intrinsics?page=1 abgerufen

[2] ARM Limited,Company. (2013). *NEON Programmer's Guide.*

[3] LiBe.net. (2018). Von https://www.libe.net/themen/Linux_Befehle_Uebersicht.php abgerufen

[4] Wikipedia. (2021). Von https://de.wikipedia.org/wiki/Reduced_Instruction_Set_Computer abgerufen

[5] Wikipedia. (2021). Von https://de.wikipedia.org/wiki/Header-Datei abgerufen

[6] Wikipedia. (2021). Von https://en.wikipedia.org/wiki/Connected-component_labeling abgerufen

Anlagen

Anlage 1: Installation und Inbetriebnahme

- Vorbereitung
  1. SD-Karte mit installiertem Raspberry Pi-System (mindestens 8G)
  2. Netzteil: 5V/3A, ausgestattet mit USB-Typ-C Ausgangsanschluss.

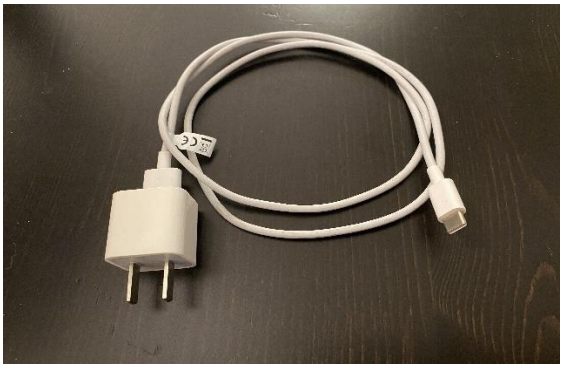

Abbildung 50: Netzteil

  3. Tastatur, Maus, Monitor und ein HDMI-Kabel zum Anschließen an den Monitor

- Schalten Sie den Raspberry Pi zum ersten Mal ein.

Abbildung 51: RPi einschalten

  1. Legen Sie zuerst die SD-Karte mit dem installierten System in den Raspberry Pi ein.
  2. Schließen Sie dann die USB-Tastatur und -Maus an den Raspberry Pi an.
  3. Verbinden Sie den Raspberry Pi und den Monitor mit einem HDMI-Kabel. Wenn Ihr Monitor über einen VGA-Schnittstellenausgang verfügt, benötigen Sie auch ein HDMI-zu-VGA-Kabel.
  4. Verbinden Sie den Raspberry Pi mit einem Netzwerkkabel mit dem Router. (Optional)
  5. Schließen Sie das Netzkabel an und schalten Sie das Gerät ein.

Wenn die rote Betriebsanzeige auf dem Raspberry Pi-Motherboard aufleuchtet und die grüne Anzeige gelegentlich blinkt, bedeutet dies, dass das System gestartet wurde. Wenn es gut geht, sehen Sie das Raspberry Pi Logo.

Abbildung 52: RPi Logo

Hinweis: Wenn der Monitor nicht angezeigt wird, liegt möglicherweise ein Problem mit dem HDMI-Kabel oder dem Adapter vor. Überprüfen Sie dies.

- Ersteinrichtung

  Führen Sie den Raspberry Pi zum ersten Mal aus, und Sie sehen den Assistenten für die Ersteinrichtung.

Abbildung 53: Ersteinrichtung 1

1. Stellen Sie Land, Sprache und Zeitzone ein:

Abbildung 54: Ersteinrichtung 2

2. Ein neues Passwort festlegen:

Abbildung 55: Ersteinrichtung 3

3. WIFI verbinden:

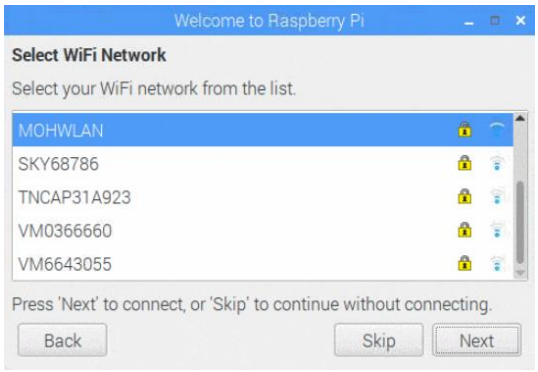

Abbildung 56: Ersteinrichtung 4

4. Erfolgreich eingestellt:

Abbildung 57: Ersteinrichtung 5

Anlage 2: Remotedesktopverbindung

Offensichtlich ist es sehr unpraktisch, den Raspberry Pi bei jedem Programmieren an das Display anzuschließen. Daher ist die Remotedesktopverbindung sehr hilfreich.

1. Installieren Sie den xrdp-Dienst auf Raspberry Pi mit dem Befehl:
   sudo apt-get install xrdp

2. Installieren Sie den Dienst tightvncserver auf Raspberry Pi mit dem Befehl:
   sudo apt-get install tightvncserver

3. Öffnen Sie das mit Windows gelieferte Remotedesktop-Verbindungstool
   [Startmenü] → [Windows-Zubehör] → [Remotedesktopverbindung]

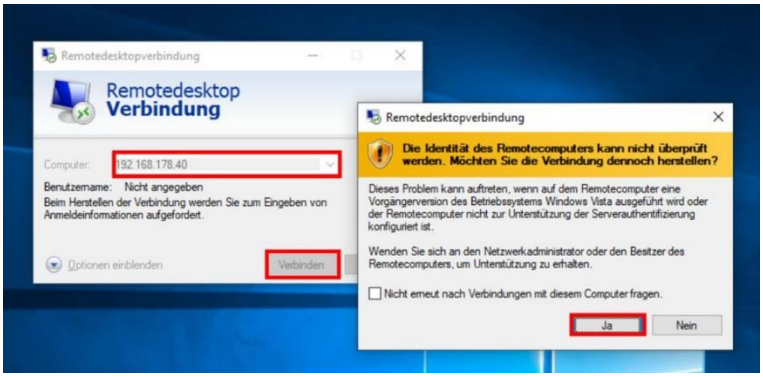

Abbildung 58: Remotedesktopverbindung 1

4. Geben Sie die IP-Adresse des Raspberry Pi ein. Um die Verbindung zu überprüfen, wählen Sie [Ja].

5. Geben Sie den Benutzernamen und das Kennwort der Verbindung ein. Der Standardbenutzername für Raspberry Pi lautet [pi] und das entsprechende Standardkennwort lautet [raspberry].

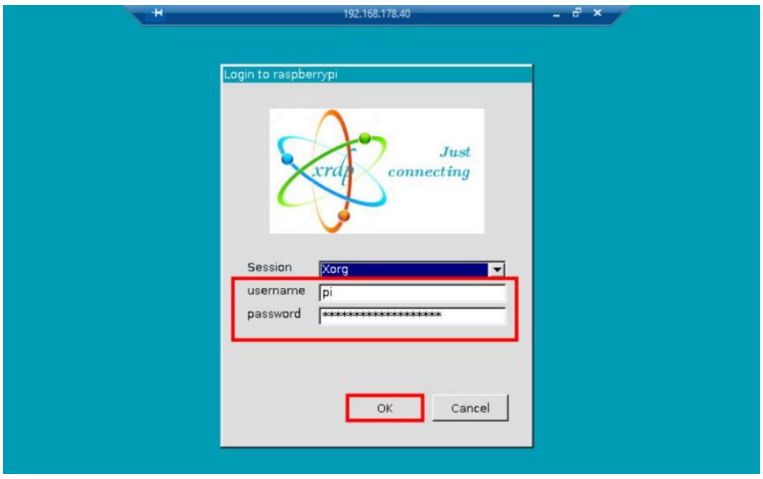

Abbildung 59: Remotedesktopverbindung 2

6. Verbindung erfolgreich.

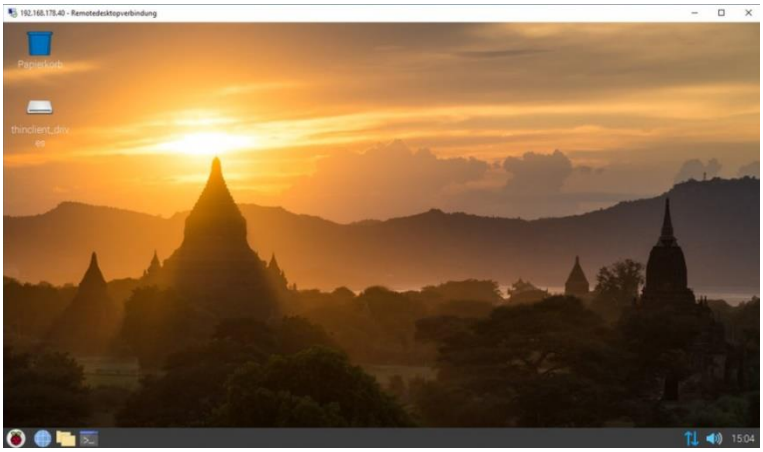

Abbildung 60: Remotedesktopverbindung 3

Anlage 3: SSH-Verbindung

Um den Raspberry Pi fernsteuern zu können, muss SSH aktiviert sein. (SSH, English für Secure Shell. Raspberry Pi deaktiviert SSH standardmäßig)

Für die SSH-Verbindung muss zunächst Folgendes bestätigt werden:

1. Ob der Raspberry Pi mit dem Netzwerk verbunden ist.

2. Ob bei Verwendung der Software die ausgefüllte IP-Adresse, der Benutzername und das Kennwort korrekt sind.

- Methode 1:

  Geben Sie in die Befehlszeile Folgendes ein:
  sudo /etc/init.d/ssh start
  drücken Sie die Eingabetaste, wie unten gezeigt:

Abbildung 61: SSH 1

Hinweis: Methode 1 ist vorübergehend. (muss nach dem Neustart neu gestartet werden)

- Methode 2:

  1. Öffnen Sie wie in der folgenden Abbildung gezeigt:
  Menü→ Preferences→ Raspberry Pi Configuration

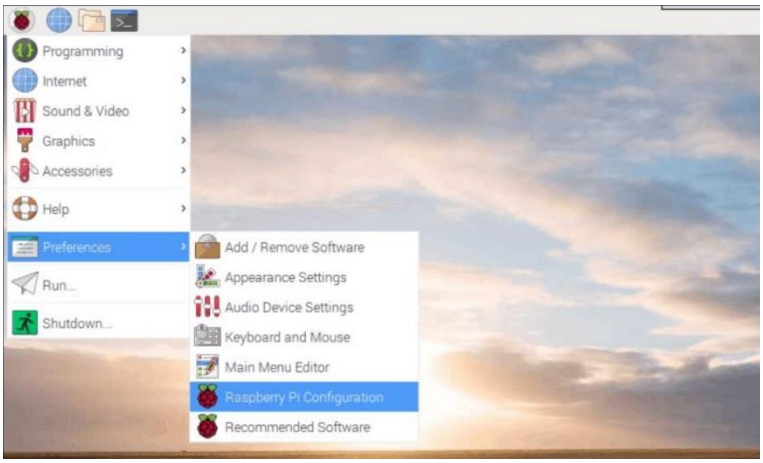

Abbildung 62: SSH 2

2. Klicken Sie auf "Interfaces" und wählen Sie "SSH-Dienst Enable". Wie nachfolgend dargestellt:

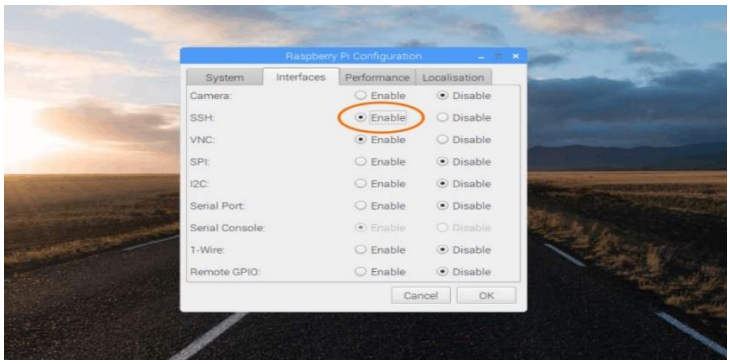

Abbildung 63: SSH 3

- Methode 3:

1. Geben Sie in die Befehlszeile Folgendes ein:
   sudo raspi-config
2. Drücken Sie die Eingabetaste, wie unten gezeigt:

```
pi@raspberrypi:~ $ sudo raspi-config
```

Abbildung 64: SSH 4

3. Wählen Sie "Interfacing Options" und drücken Sie die Eingabetaste, wie unten gezeigt:

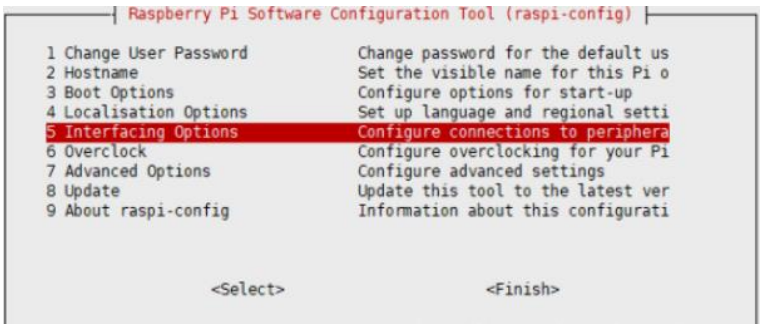

Abbildung 65: SSH 5

4. Wählen Sie "SSH" und drücken Sie die Eingabetaste, wie unten gezeigt:

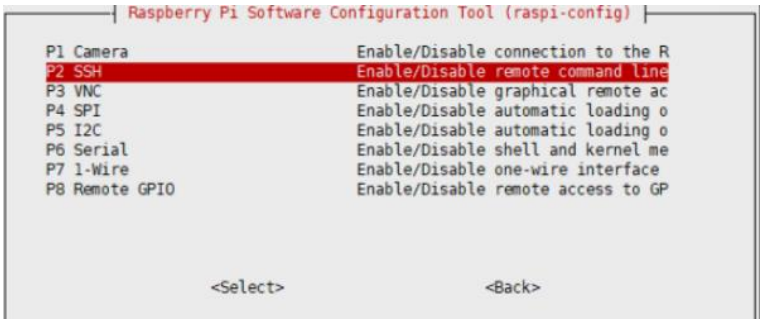

Abbildung 66: SSH 6

5. Wählen Sie abschließend "Finish" und warten Sie auf den Neustart

Anlage 4: Matlab-Code für Rotation

```
close all
clear

ImRGB = imread('IMG_0420.jpg');   % Name des Bild
ImGray = rgb2gray(ImRGB);

figure(1)
imshow(ImGray)

rotAngleGrad = 45;   % rotation angle in grad

% create rotated Image
ImRot = imrotate(ImGray(:,:),rotAngleGrad,'nearest');
figure(2)
imshow(ImRot)

%%

clear ImRes
rotAngleRad = rotAngleGrad/180*pi;


c1y = cos(pi/2-rotAngleRad);
c2y = sin(pi/2-rotAngleRad);
c1x = cos(rotAngleRad);
c2x = sin(rotAngleRad);

% initialize accumulators
y_acc_y = ceil(c2x*size(ImGray,2)); %find(ImRot(:,1)>0,1);
x_acc_y = 1;

% derotate image using nearest neighbour

% Schleife in y-Richtung
for y = 1:size(ImGray,1)-1

    y_acc_x = y_acc_y;
    x_acc_x = x_acc_y;

    % Schleife in x-Richtung
    for x = 1:size(ImGray,2)-1

        ind_y = round(y_acc_x);
        ind_x = round(x_acc_x);

        ImRes(y,x) = ImRot(ind_y,ind_x);

        % increment counter in x-direction
        x_acc_x = x_acc_x + c1x;
        y_acc_x = y_acc_x - c2x;
```

```
51.
52.     end
53.
54.     % increment counter in y-direction
55.     x_acc_y = x_acc_y + c1y;
56.     y_acc_y = y_acc_y + c2y;
57.
58. end
59.
60.
61. % Anzeige Ergebnis der Derotation
62. figure(3)
63. imshow(ImRes)
64.
65. % Vergleich Matlab-Derotation
66. figure(4)
67. imshow(imrotate(ImRot(:,:),-rotAngleGrad,'nearest'))
```

Anlage 5: Matlab-Code (Hauptprogramm)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Calculation of GV difference vs number of pixels for Yale Faces B testset
% 1st person
%
% http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%% clear all variables
clear ;
close all;

%% set parameters - use or don't use split into smaller connected components
% start with 0

flagConnectedComponents = 1;

%% load image data

pathName='yaleB01\';
files=dir(pathName);
Im=[];
countIm = 0;

for i=3:length(files)
    try
        if strcmp(files(i).name(end-2:end),'pgm')
            countIm = countIm+1;
            ImO(:,:,countIm) =double(imread([pathName '\' files(i).name]));
        end
    end
end

% show loaded images
figure(1)
montage(uint8(ImO))

%% load statistical references

load('yaleB01\StatisticReferences.mat','meanImRef','stdImRef')

%% find relevant differences between actual image and statistic images
% - this part is to be implemented and tested

% set threshold in multiples of pixel-wise standard deviation
th = 2;

% loop over test images
```

```
for i = 1:countIm

    % load ith image
    ImP = ImO(:,:,i);

    % set size of image
    numberRows = size(ImP,1);
    numberCols = size(ImP,2);

    % calculate mean and standard deviation of image
    meanAcc = 0;
    powAcc = 0;
    for r = 1:numberRows
        for c =  1:numberCols
            meanAcc = meanAcc + ImP(r,c);
            powAcc = powAcc + ImP(r,c)*ImP(r,c);
        end
    end
    meanIm = meanAcc/numberRows/numberCols;
    powIm = powAcc/numberRows/numberCols;
    stdIm = sqrt(powIm - meanIm*meanIm);

    % normalize ith image
    for r = 1:numberRows
        for c =  1:numberCols
            ImN(r,c) = (ImP(r,c)-meanIm)./stdIm*15+80;
        end
    end

    % subtract pixel-wise mean image and
    % divide pixel-wise by standard deviation
    for r = 1:numberRows
        for c =  1:numberCols
            ImwoMean(r,c) = (ImN(r,c) - meanImRef(r,c));
            DiffImNorm(r,c) = ImwoMean(r,c)/stdImRef(r,c);
        end
    end

    % find pixel indices where normalized difference image is larger than
    % threshold and create binary mask
    maskPixelsThreshold = zeros(numberRows, numberCols);
    numPixTotal = 0;
    for r = 1:numberRows
        for c =  1:numberCols
            if abs(DiffImNorm(r,c)) >= th
                maskPixelsThreshold(r,c) = 1;
                % sum up all ones in the binary mask to get total number if
                % differing pixels
                numPixTotal = numPixTotal + 1;
                   else
                       maskPixelsThreshold(r,c) = 0;
                   end
```

```
            end
        end

        % if there are such pixels, then calculate size of connected pixel areas
        if numPixTotal > 0

            if flagConnectedComponents
                % clear variable dGV
                clear dGV

                % find connected components - this algorithm needs to be
                % rewritten or reused
                listConnComp = bwconncomp(maskPixelsThreshold);

                % go through list of connected components
                for connComp = 1:listConnComp.NumObjects

                    % number of pixels in connected component number connComp
                    numPix(connComp) = numel(listConnComp.PixelIdxList{connComp});

                    % convert current list indices into row-column indices
                    [indRows,indCols] = ind2sub([numberRows, numberCols],listConnComp.PixelIdxList{connComp});

                    % extract pixels from actual and from mean image belonging to
                    % connected component connComp and calculate difference in
                    % gray value
                    dGV(connComp) =  0;
                    for pixel = 1:numPix(connComp)
                        dGV(connComp) =  dGV(connComp) + ImN(indRows(pixel), indCols(pixel))...
                            /meanImRef(indRows(pixel), indCols(pixel));
                    end
                    dGV(connComp) = dGV(connComp)/numPix(connComp);
                end

                % calculate average gray value difference over all connected components
                dGVav = 0;
                for connComp = 1:length(listConnComp.PixelIdxList)
                    dGVav = dGVav + dGV(connComp)*numPix(connComp);
                end
                dGVav = dGVav/numPixTotal;

            else
                % extract pixels from actual and from mean image belonging to
                % mask and calculate gray value difference
                dGVav = 0;
                for r = 1:numberRows
                    for c =  1:numberCols
                        if maskPixelsThreshold(r,c) == 1
```

```
                                dGVav = dGVav + ImN(r,c)/meanImRef(r,c);
                            end
                        end
                    end
                    dGVav = dGVav/numPixTotal;
                end
            else
                % if there are no pixels detected
                dGVav = 0;
            end

            %% assign result
            resNumPixTotal(i) = numPixTotal;
            resdGVav(i) = dGVav;

            % show actual image
            figure(7)
            imshow(uint8(ImP))

            % show normalized difference image
            figure(8)
            imshow((DiffImNorm),[0 5])

            % show results
            figure(9)
            plot(resNumPixTotal(i),resdGVav(i),'bd','Markersize',2,'Markerfacecolor',
    'b')
            hold on
            grid on

        end

        figure(9)
        grid on
        xlabel(['$Pixels \rightarrow$'], 'Interpreter','latex')
        ylabel(['d$GV \rightarrow$'], 'Interpreter','latex'
```

Anlage 6: C-Code (ohne Rotation oder NEON)

```
1.  #include <math.h>
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <png.h>
5.  #include <assert.h>
6.  #include <time.h>
7.  #include <sys/time.h>
8.  #include <string.h>
9.  #include <dirent.h>
10. #include <unistd.h>
11. #include "img.h"
12. #include "twopass.h"
13.
14. int main(int argc,char** argv){
15.  /***********input number to set the parameter***************/
16.     unsigned int times;
17.     double thresholdStddevFactor;
18.     if(argc >= 2 && argv[1]!=NULL){
19.         times = strtol(argv[1],NULL,10);
20.         thresholdStddevFactor = (double)strtol(argv[2],NULL,10)/10;
21.     }
22.     else{
23.         times = 10000;
24.         thresholdStddevFactor = 2;
25.     }
26.     printf("\nexecute times:%d",times);
27.     printf("\nthreshold stdddev:%f\n\n",thresholdStddevFactor);
28.
29.  /****************load the picture*************************/
30.     size_t x,y;
31.     png_bytep bild;
32.
33.     readimg("yaleB01_P00A+010E-20.png",&bild,&x,&y);
34.     printf("x=%lu,y=%lu\n",x,y);
35.
36.  /*******************initialization************************/
37.     double meanImRef[y][x],stdImRef[y][x],Ref[y*x], LowThresh[y][x], HighThresh[y][x
    ];
38.     double meanIm, stdImInv, temp1, ratioGW[100];
39.     char maskDefectPixels[x*y];
40.     unsigned int rt[x*y], listSameSeg[100][100], numSameSeg[100],list[100],gvlist[10
    0];
41.     unsigned int accGreyValues[100], accMeanImRef[100], numNormaccGreyValues[100], n
    umNormaccMeanImRef[100];
42.
43.     unsigned int meanAcc, powAcc, wert, numPixTotal, numRegion;
44.     unsigned int  loopCount, i, j, pos, flag;
45.     int posk, k;
46.
47.     struct timeval tpstart,tpend;
```

```
48.     float timeuse;
49.
50.     FILE *fpm=fopen("mRef.txt","rb");
51.     fread(Ref,sizeof(double),y*x,fpm);
52.     fclose(fpm);
53.     for(i=0;i<y;i++){
54.         for(j=0;j<x;j++){
55.             meanImRef[i][j]=Ref[y*j+i];
56.         }
57.     }
58.
59.  for (i=1;i<100;i++)  // List initialization
60.   list[i]=i;
61.
62.     FILE *fps=fopen("sRef.txt","rb");
63.     fread(Ref,sizeof(double),y*x,fps);
64.     fclose(fps);
65.     for(i=0;i<y;i++){
66.         for(j=0;j<x;j++){
67.             stdImRef[i][j]=thresholdStddevFactor*Ref[y*j+i];
68.         }
69.     }
70.
71.     for(i=0;i<y;i++){
72.         for(j=0;j<x;j++){
73.             LowThresh[i][j]=meanImRef[i][j]-stdImRef[i][j];
74.             HighThresh[i][j]=meanImRef[i][j]+stdImRef[i][j];
75.         }
76.     }
77.
78.  /***********Loop and measure time***********/
79.     gettimeofday(&tpstart,NULL);
80.     for(loopCount = 0; loopCount < times; loopCount++){
81.
82.         numPixTotal = 0;
83.         meanAcc = 0;
84.         powAcc = 0;
85.         numRegion = 0;
86.
87.         for (k=0; k < 100; k++){
88.             numSameSeg[k] = 0;
89.             accMeanImRef[k] = 0;
90.             numNormaccMeanImRef[k] = 0;
91.             accGreyValues[k] = 0;
92.             numNormaccGreyValues[k] = 0;
93.    gvlist[k]=0;
94.         }
95.
96.
97.         for(i=0;i<y;i++){
98.             for(j=0;j<x;j++){
99.                 wert = bild[i*x+j];
```

```
100.                        meanAcc += wert;
101.                        powAcc += wert*wert;
102.                    }
103.                }
104.
105.                meanIm = ((double)meanAcc)/x/y;
106.                stdImInv = x*y/sqrt((double)powAcc*x*y-
    ((double)meanAcc)*((double)meanAcc))*15;
107.
108.                for(i=0;i<y;i++){
109.                    for(j=0;j<2;j++){
110.                        pos = i*x+j;
111.                        rt[pos] = 0;
112.                        maskDefectPixels[pos] = 0;
113.                    }
114.                }
115.
116.                for(i=0;i<y;i++){
117.                    for(j=1;j<x;j++){
118.
119.                        pos = i*x+j;
120.                        temp1=(bild[pos]-meanIm)*stdImInv+80;  // normalized pixel
121.                        rt[pos] = 0;
122.
123.                        if(LowThresh[i][j] < temp1){
124.                            maskDefectPixels[pos] = 0;
125.                        }
126.                        else {
127.
128.                            maskDefectPixels[pos] = 1;
129.
130.                            flag = 0;
131.                            // check if previously processed neighbouring pixels in 8
     neighbourhood were detected
132.                            if (maskDefectPixels[pos-1] > 0 ){
133.                                rt[pos] = rt[pos-1];
134.                                flag = rt[pos];
135.                            }
136.
137.            if(i!=0){
138.             for (k = -1; k <= 1; k++){
139.              posk = pos-x-k;
140.              if (maskDefectPixels[posk] > 0){
141.               if (flag == 0){
142.                rt[pos] = rt[posk];
143.                flag = rt[pos];
144.               }
145.               else if (rt[posk] != flag){
146.                if(list[rt[posk]]<list[flag]){
147.                  list[list[flag]]=list[rt[posk]];
148.                  list[flag]=list[rt[posk]];
149.                 }
```

```
150.                 else {
151.                  list[list[rt[posk]]]=list[flag];
152.                  list[rt[posk]]=list[flag];
153.                 }
154.                //  build list
155.               }
156.              }
157.             }
158.                          }
159.
160.                          // increment region counter
161.                          if (flag == 0) {
162.                              numRegion += 1;
163.                              rt[pos] = numRegion;
164.                          }
165.
166.                          // accumlate numerator and denominator for ratioGW
167.                          accGreyValues[rt[pos]] += temp1;
168.                          accMeanImRef[rt[pos]] += meanImRef[i][j];
169.                          // nomalize accumulated values to prevent overflow
170.                          if (accMeanImRef[rt[pos]] > 1000){
171.                              accMeanImRef[rt[pos]] = accMeanImRef[rt[pos]]-1000;
172.                              numNormaccMeanImRef[rt[pos]] += 1;
173.                          }
174.                          if (accGreyValues[rt[pos]] > 1000){
175.                                  accGreyValues[rt[pos]] = accGreyValues[rt[pos]]-
    1000;
176.                                  numNormaccGreyValues[rt[pos]] += 1;
177.                              }
178.
179.                          // accumulate area defect pixels
180.                          numPixTotal = numPixTotal+1;
181.                      }
182.                  }
183.              }
184.
185.              // calculate ratioGW
186.              for( k= 1; k <= numRegion; k++){
187.                  ratioGW[k] = (double)(accGreyValues[k]+numNormaccGreyValues[k]*10
    00)/
188.                          (double)(accMeanImRef[k]+numNormaccMeanImRef[k]*1000);
189.          gvlist[list[k]]+=accGreyValues[k]+numNormaccGreyValues[k]*1000;
190.              }
191.
192.
193.          }
194.
195.          gettimeofday(&tpend,NULL);
196.          timeuse=1000000*(tpend.tv_sec-tpstart.tv_sec)+tpend.tv_usec-
    tpstart.tv_usec;
197.          timeuse/=1000000;
198.
```

```
199.        /*************output the result***************/
200.           printf("meanacc=%d\n",meanAcc);
201.           printf("powacc=%d\n",powAcc);
202.           printf("numPixTotal=%d\n",numPixTotal);
203.           for( k= 1; k <= numRegion; k++){
204.               printf("%d Zaehler %d \t Normierung %d \t Nenner %d  \t Normierung %d
     \t ratioGW=%f\n",k,accGreyValues[k],numNormaccGreyValues[k],
205.                       accMeanImRef[k],numNormaccMeanImRef[k], ratioGW[k]);
206.           }
207.           printf("used time mean/std/numPixTotal calculation:%f seconds\n",timeuse)
    ;
208.
209.           FILE *maskdata=fopen("maskdata.txt","w");
210.           FILE *rtDetected=fopen("rtDetected.txt","w");
211.           for(i=0;i<y;i++){
212.               for(j=0;j<x;j++){
213.                   fprintf(maskdata,"%d,", maskDefectPixels[i*x+j]);
214.                   fprintf(rtDetected,"%d,", rt[i*x+j]);
215.               }
216.               fprintf(maskdata,"\n");
217.               fprintf(rtDetected,"\n");
218.           }
219.           fclose(maskdata);
220.           fclose(rtDetected);
221.
222.           FILE *ListSameSeg=fopen("list.txt","w");
223.        fprintf(ListSameSeg,"Nummer :Region   GV_Summe\n");
224.           for(i=1;i<100;i++)
225.               fprintf(ListSameSeg,"%d :%d   %d\n", i,list[i],gvlist[i]);
226.           fclose(ListSameSeg);
227.
228.       }
```

Anlage 7: img.h

```
1.  #include "png.h"
2.  #include "assert.h"
3.  #include "stdio.h"
4.  #include "stdlib.h"
5.  #include "img.h"
6.
7.  void readimg(const char *filename, png_bytep *data,size_t *w, size_t *h){
8.   FILE *inputpng=fopen(filename,"rb");
9.   assert(inputpng!=NULL);
10.
11.  png_structp png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING,
12.                                                NULL, NULL, NULL);
13.   png_infop info_ptr = png_create_info_struct(png_ptr);
14.   png_init_io(png_ptr, inputpng);
15.   png_read_info(png_ptr, info_ptr);
16.   *w = png_get_image_width(png_ptr, info_ptr);
17.   *h = png_get_image_height(png_ptr, info_ptr);
18.
19.   *data = malloc(*h * png_get_rowbytes(png_ptr, info_ptr));
20.   for(unsigned int ii=0; ii<*h; ii++) {
21.     png_read_row(png_ptr,
22.                  *data + ii * png_get_rowbytes(png_ptr, info_ptr),
23.                  NULL);
24.       }
25.  png_read_end(png_ptr, info_ptr);
26.   png_destroy_read_struct(&png_ptr, &info_ptr, (png_infopp)NULL);
27.   fclose(inputpng);
28.  }
29.
30.  void writeImageData(const char* filename, png_bytep *data,
31.                     size_t w, size_t h, size_t bitdepth ) {
32.   FILE *outputpng = fopen(filename, "wb");
33.   assert(outputpng != NULL);
34.
35.   png_structp png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING,
36.                                                 NULL, NULL, NULL);
37.   png_infop info_ptr = png_create_info_struct(png_ptr);
38.   png_init_io(png_ptr, outputpng);
39.   png_set_IHDR(png_ptr, info_ptr, w, h, bitdepth,
40.                PNG_COLOR_TYPE_GRAY,
41.                //PNG_COLOR_TYPE_RGBA,
42.                PNG_INTERLACE_NONE,
43.                PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);
44.   png_write_info(png_ptr, info_ptr);
45.   /* png_set_swap(png_ptr); //correct byte order*/
46.   for(unsigned int ii=0; ii<h; ii+=1) {
47.     png_write_row(png_ptr, *data + ii*png_get_rowbytes(png_ptr, info_ptr));
48.   }
49.
```

```
50.   png_write_end(png_ptr, NULL);
51.   png_destroy_write_struct(&png_ptr, &info_ptr);
52.   fclose(outputpng);
53. }
```

Anlage 8: twopass.h

```
1.  #include <stdlib.h>
2.  #include <stdio.h>
3.  #include "twopass.h"
4.
5.  int *bwconncomp1(int *bild,int x,int y){
6.   int *rt=(int *)malloc(sizeof(int)*x*y);  //result
7.   int rt[];
8.   int *z=(int *)malloc(sizeof(int)*x*y);
9.   int counter=1;
10.  int A,B,C,D,temp;
11.
12.
13.  for (int i=0;i<x*y;i++)
14.   z[i]=i;          //list initialization
15.
16.  for (int i=0;i<y;i++){
17.   for(int j=0;j<x;j++){
18.    if(bild[i*x+j]==0)
19.     rt[i*x+j]=0;      //background;0 Pixel
20.    else {
21.     if(i!=0&&j!=0&&j!=x-1){   //Pixel in the middle
22.      A=rt[(i-1)*x+j-1];
23.      B=rt[(i-1)*x+j];
24.      C=rt[(i-1)*x+j+1];
25.      D=rt[i*x+j-1];
26.     }
27.     if(j==0){       //Pixel in the first row
28.      A=0;
29.      B=rt[(i-1)*x+j];
30.      C=rt[(i-1)*x+j+1];
31.      D=0;
32.     }
33.     if(j==x-1){       //Pixel in the last column
34.      A=rt[(i-1)*x+j-1];
35.      B=rt[(i-1)*x+j];
36.      C=0;
37.      D=rt[i*x+j-1];
38.     }
39.     if(i==0){       //Pixel in the first column
40.      A=0;
41.      B=0;
42.      C=0;
43.      D=rt[i*x+j-1];
44.     }
45.     if(i==0&&j==0){      //the first Pixel
46.      A=0;
47.      B=0;
48.      C=0;
49.      D=0;
```

```
    }
    if(A==0&&B==0&&C==0&&D==0){  //New isolated Pixel
     rt[i*x+j]=counter;
     counter++;
    }
    else {       //other situation
     temp=x*y;
     if(A!=0&&A<=temp)
      temp=A;
     if(B!=0&&B<=temp)
      temp=B;
     if(C!=0&&C<=temp)
      temp=C;
     if(D!=0&&D<=temp)
      temp=D;
     rt[i*x+j]=temp;    //Find neighbor's minimum value other than 0
     if(z[temp]<z[A])
      z[A]=z[temp];
     if(z[temp]<z[B])
         z[B]=z[temp];
     if(z[temp]<z[C])
      z[C]=z[temp];
     if(z[temp]<z[D])
      z[D]=z[temp];
    }
   }
  }
 }
 for(int i=0;i<x*y;i++)
  while(z[i]!=z[z[i]])
   z[i]=z[z[i]];      //Refresh the 'list'
 counter=0;
 for(int i=1;i<x*y;i++){      //Renumber
  if(i==z[i]){
   counter++;
   z[i]=counter;
  }
  else {
   z[i]=z[z[i]];
  }
 }

 for (int i=0;i<y;i++){     //The final result
  for(int j=0;j<x;j++){
   if(rt[i*x+j]!=0)
    rt[i*x+j]=z[rt[i*x+j]];
  }
 }

       return rt;
        free(rt);
        free(z);
```

```
102.        }
```

Anlage 9: Versuch 4, 5, 6

```
uint16x8_t rega8,rega9;
 gettimeofday(&tpstart,NULL);
 for(int ttt=0;ttt<times;ttt++){
  meanacc = vld1q_u32(&awert_32[0]);
  powacc = vld1q_u32(&awert_32[0]);
  for(int i=0;i<192*168;i+=32){
   rega1 = vld1q_u16(&list_16[i]); // Load four sets of data at once
   rega2 = vld1q_u16(&list_16[i+8]);
   rega6 = vld1q_u16(&list_16[i+16]);
   rega8 = vld1q_u16(&list_16[i+24]);
   rega5 = vaddq_u16(rega1,rega2); // Add four sets of data
   rega7 = vaddq_u16(rega6,rega5);
   rega9 = vaddq_u16(rega7,rega8);
   rega3 = vget_high_u16(rega9);
   rega4 = vget_low_u16(rega9);
   meanacc = vaddw_u16(meanacc,rega3);  // send the sum to register meanAcc
   meanacc = vaddw_u16(meanacc,rega4);

   rega1 = vmulq_u16(rega1,rega1); // calculate powAcc with similar operations
   rega3 = vget_high_u16(rega1);
   rega4 = vget_low_u16(rega1);
   powacc = vaddw_u16(powacc,rega3);
   powacc = vaddw_u16(powacc,rega4);
   rega2 = vmulq_u16(rega2,rega2);
   rega3 = vget_high_u16(rega2);
   rega4 = vget_low_u16(rega2);
   powacc = vaddw_u16(powacc,rega3);
   powacc = vaddw_u16(powacc,rega4);
   rega6 = vmulq_u16(rega6,rega6);
   rega3 = vget_high_u16(rega6);
   rega4 = vget_low_u16(rega6);
   powacc = vaddw_u16(powacc,rega3);
   powacc = vaddw_u16(powacc,rega4);
   rega8 = vmulq_u16(rega8,rega8);
   rega3 = vget_high_u16(rega8);
   rega4 = vget_low_u16(rega8);
   powacc = vaddw_u16(powacc,rega3);
   powacc = vaddw_u16(powacc,rega4);
  }
 }
 vst1q_u32(&out1[0],meanacc);
 vst1q_u32(&out2[0],powacc);
 x0=out1[0]+out1[1]+out1[2]+out1[3]; //Output result
 x1=out2[0]+out2[1]+out2[2]+out2[3];
 gettimeofday(&tpend,NULL);
 timeuse=1000000*(tpend.tv_sec-tpstart.tv_sec)+tpend.tv_usec-tpstart.tv_usec;
 timeuse/=1000000;
 printf("meanacc=%d \n",x0);
 printf("powacc=%d \n",x1);
 printf("used time(neon 4th ver.):%f seconds\n",timeuse);
```

```
51.  printf("%f times faster\n\n",timeuse0/timeuse);
```

*Programm 49: Versuch 4: 16x8 Q-Register,dann 4*16x4->32x4 Register*

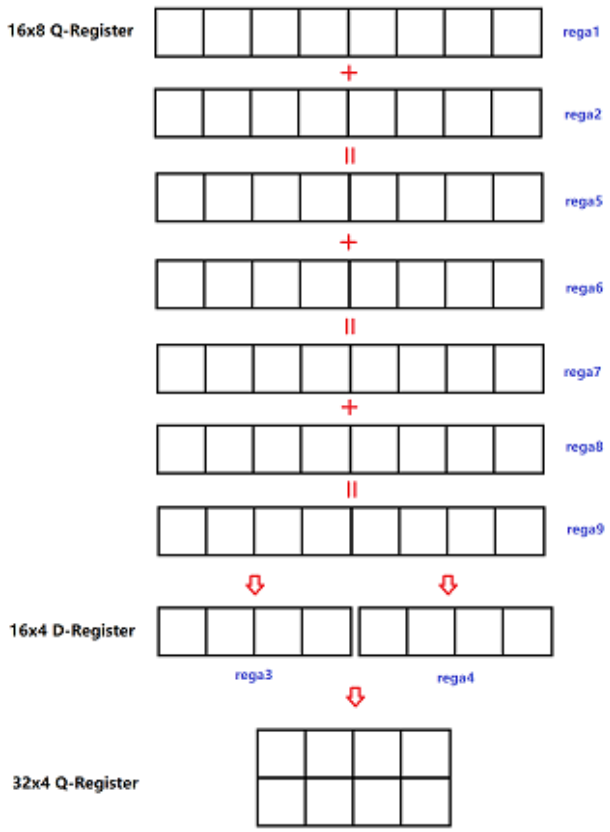

Abbildung 67: Ver. 4 - Schematische Darstellung

```
1.  uint16x8_t rega10,rega11;
2.   gettimeofday(&tpstart,NULL);
3.   for(int ttt=0;ttt<times;ttt++){
4.    meanacc = vld1q_u32(&awert_32[0]);
5.    powacc = vld1q_u32(&awert_32[0]);
6.    for(int i=0;i<192*168;i+=40){
7.     rega1 = vld1q_u16(&list_16[i]); // Load five sets of data at once
8.     rega2 = vld1q_u16(&list_16[i+8]);
9.     rega6 = vld1q_u16(&list_16[i+16]);
10.    rega8 = vld1q_u16(&list_16[i+24]);
11.    rega10 = vld1q_u16(&list_16[i+32]);
12.    rega5 = vaddq_u16(rega1,rega2);  // Add five sets of data
13.    rega7 = vaddq_u16(rega6,rega5);
14.    rega9 = vaddq_u16(rega7,rega8);
15.    rega11 = vaddq_u16(rega9,rega10);
16.    rega3 = vget_high_u16(rega11);
17.    rega4 = vget_low_u16(rega11);
18.    meanacc = vaddw_u16(meanacc,rega3); // send the sum to register meanAcc
```

```
19.    meanacc = vaddw_u16(meanacc,rega4);
20.
21.    rega1 = vmulq_u16(rega1,rega1);  // calculate powAcc with similar operations
22.    rega3 = vget_high_u16(rega1);
23.    rega4 = vget_low_u16(rega1);
24.    powacc = vaddw_u16(powacc,rega3);
25.    powacc = vaddw_u16(powacc,rega4);
26.    rega2 = vmulq_u16(rega2,rega2);
27.    rega3 = vget_high_u16(rega2);
28.    rega4 = vget_low_u16(rega2);
29.    powacc = vaddw_u16(powacc,rega3);
30.    powacc = vaddw_u16(powacc,rega4);
31.    rega6 = vmulq_u16(rega6,rega6);
32.    rega3 = vget_high_u16(rega6);
33.    rega4 = vget_low_u16(rega6);
34.    powacc = vaddw_u16(powacc,rega3);
35.    powacc = vaddw_u16(powacc,rega4);
36.    rega8 = vmulq_u16(rega8,rega8);
37.    rega3 = vget_high_u16(rega8);
38.    rega4 = vget_low_u16(rega8);
39.    powacc = vaddw_u16(powacc,rega3);
40.    powacc = vaddw_u16(powacc,rega4);
41.    rega10 = vmulq_u16(rega10,rega10);
42.    rega3 = vget_high_u16(rega10);
43.    rega4 = vget_low_u16(rega10);
44.    powacc = vaddw_u16(powacc,rega3);
45.    powacc = vaddw_u16(powacc,rega4);
46.   }
47. }
48. vst1q_u32(&out1[0],meanacc);
49. vst1q_u32(&out2[0],powacc);
50. x0=out1[0]+out1[1]+out1[2]+out1[3];  //Output result
51. x1=out2[0]+out2[1]+out2[2]+out2[3];
52. gettimeofday(&tpend,NULL);
53. timeuse=1000000*(tpend.tv_sec-tpstart.tv_sec)+tpend.tv_usec-tpstart.tv_usec;
54. timeuse/=1000000;
55. printf("meanacc=%d \n",x0);
56. printf("powacc=%d \n",x1);
57. printf("used time(neon 5th ver.):%f seconds\n",timeuse);
58. printf("%f times faster\n\n",timeuse0/timeuse);
```

Programm 50: Versuch5: 16x8 Q-Register,dann 5*16x4->32x4 Register

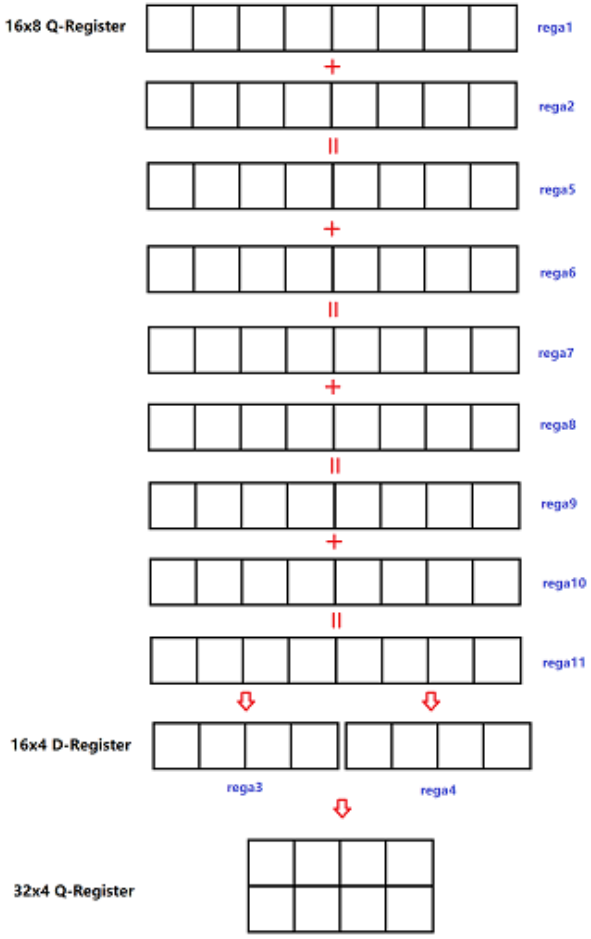

Abbildung 68: Ver. 5 - Schematische Darstellung

Achtung: Falsch, weil 192*168/40 nicht Integral ist.

```
uint16x8_t rega12,rega13;
 gettimeofday(&tpstart,NULL);
 for(int ttt=0;ttt<times;ttt++){
  meanacc = vld1q_u32(&awert_32[0]);
  powacc = vld1q_u32(&awert_32[0]);
  for(int i=0;i<192*168;i+=48){
   rega1 = vld1q_u16(&list_16[i]);  // Load six sets of data at once
   rega2 = vld1q_u16(&list_16[i+8]);
   rega6 = vld1q_u16(&list_16[i+16]);
   rega8 = vld1q_u16(&list_16[i+24]);
   rega10 = vld1q_u16(&list_16[i+32]);
   rega12 = vld1q_u16(&list_16[i+40]);
   rega5 = vaddq_u16(rega1,rega2); // Add six sets of data
   rega7 = vaddq_u16(rega6,rega5);
   rega9 = vaddq_u16(rega7,rega8);
   rega11 = vaddq_u16(rega9,rega10);
   rega13 = vaddq_u16(rega11,rega12);
   rega3 = vget_high_u16(rega13);
```

```
19.    rega4 = vget_low_u16(rega13);
20.    meanacc = vaddw_u16(meanacc,rega3); // send the sum to register meanAcc
21.    meanacc = vaddw_u16(meanacc,rega4);
22.
23.    rega1 = vmulq_u16(rega1,rega1); // calculate powAcc with similar operations
24.    rega3 = vget_high_u16(rega1);
25.    rega4 = vget_low_u16(rega1);
26.    powacc = vaddw_u16(powacc,rega3);
27.    powacc = vaddw_u16(powacc,rega4);
28.    rega2 = vmulq_u16(rega2,rega2);
29.    rega3 = vget_high_u16(rega2);
30.    rega4 = vget_low_u16(rega2);
31.    powacc = vaddw_u16(powacc,rega3);
32.    powacc = vaddw_u16(powacc,rega4);
33.    rega6 = vmulq_u16(rega6,rega6);
34.    rega3 = vget_high_u16(rega6);
35.    rega4 = vget_low_u16(rega6);
36.    powacc = vaddw_u16(powacc,rega3);
37.    powacc = vaddw_u16(powacc,rega4);
38.    rega8 = vmulq_u16(rega8,rega8);
39.    rega3 = vget_high_u16(rega8);
40.    rega4 = vget_low_u16(rega8);
41.    powacc = vaddw_u16(powacc,rega3);
42.    powacc = vaddw_u16(powacc,rega4);
43.    rega10 = vmulq_u16(rega10,rega10);
44.    rega3 = vget_high_u16(rega10);
45.    rega4 = vget_low_u16(rega10);
46.    powacc = vaddw_u16(powacc,rega3);
47.    powacc = vaddw_u16(powacc,rega4);
48.    rega12 = vmulq_u16(rega12,rega12);
49.    rega3 = vget_high_u16(rega12);
50.    rega4 = vget_low_u16(rega12);
51.    powacc = vaddw_u16(powacc,rega3);
52.    powacc = vaddw_u16(powacc,rega4);
53.   }
54. }
55. vst1q_u32(&out1[0],meanacc);
56. vst1q_u32(&out2[0],powacc);
57. x0=out1[0]+out1[1]+out1[2]+out1[3];  //Output result
58. x1=out2[0]+out2[1]+out2[2]+out2[3];
59. gettimeofday(&tpend,NULL);
60. timeuse=1000000*(tpend.tv_sec-tpstart.tv_sec)+tpend.tv_usec-tpstart.tv_usec;
61. timeuse/=1000000;
62. printf("meanacc=%d \n",x0);
63. printf("powacc=%d \n",x1);
64. printf("used time(neon 6th ver.):%f seconds\n",timeuse);
65. printf("%f times faster\n\n",timeuse0/timeuse);
```

Programm 51: Versuch 6: 16x8 Q-Register,dann 6*16x4->32x4 Register

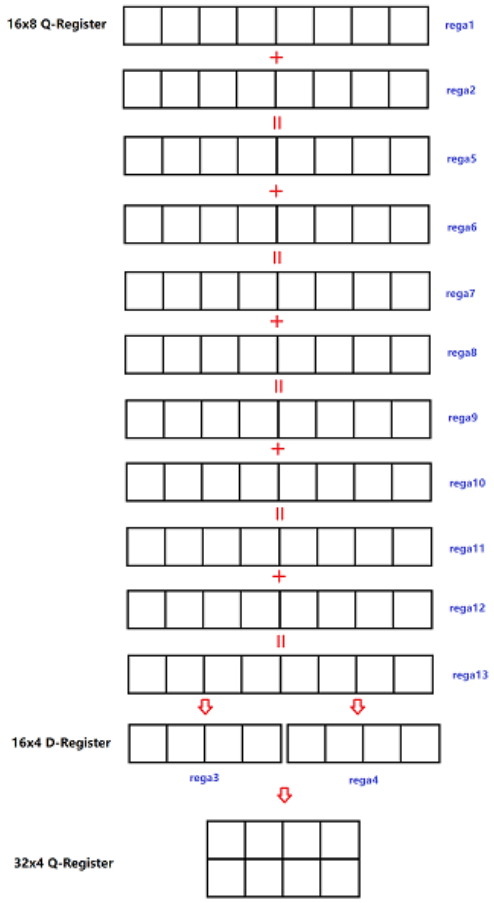

Abbildung 69: Ver. 6 - Schematische Darstellung

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, den 25.02. 2021

Unterschrift: