

TECHNISCHE UNIVERSITÄT DRESDEN  
FAKULTÄT INFORMATIK  
INSTITUT FÜR TECHNISCHE INFORMATIK  
PROFESSUR FÜR VLSI-ENTWURFSSYSTEME, DIAGNOSTIK UND  
ARCHITEKTUR

## Bachelorarbeit

Evaluation einer modernen Zynq-Plattform am Beispiel der  
Implementierung einer Hough Transformation

Dominik Weinrich  
geboren am 11.08.1990 in Kassel  
(Mat.-Nr.: 3914410)

Betreuer Hochschullehrer:  
Prof. Dr.-Ing. habil. Rainer G. Spallek

Betreuer:  
Oliver Knodel

Dresden, Datum



# Aufgabenstellung



# **Selbstständigkeitserklärung**

Ich versichere, dass ich die vorliegende Studienarbeit zum Thema

**Evaluation einer modernen Zynq-Plattform am Beispiel der Implementierung einer  
Hough Transformation**

selbstständig verfasst und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt, nur die angegebenen Quellen benutzt und die in den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Die Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Name, Dresden, Datum

## **Wettbewerbsrechtlicher Hinweis**

Die bloße Nennung von Namen, Produkten, Herstellern und Firmennamen dient lediglich als Information und stellt keine Verwendung des Warenzeichens sowie keine Empfehlung des Produktes oder der Firma dar.



# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>VII</b>
<b>Tabellenverzeichnis</b>	<b>IX</b>
<b>Listings</b>	<b>XI</b>
<b>Abkürzungsverzeichnis</b>	<b>XIII</b>
<b>1 Einleitung und Motivation</b>	<b>1</b>
<b>2 Grundlagen</b>	<b>3</b>
2.1 Zielarchitektur - Zynq Systemarchitektur . . . . .	3
2.2 Hardware/Software Codesign . . . . .	3
2.3 High Level Synthese . . . . .	3
2.4 Hough Transformation . . . . .	3
2.4.1 Umwandlung eines RGB Bildes in Graustufen . . . . .	3
2.4.2 Gauß-Filter . . . . .	4
2.4.3 Canny Edge Detection . . . . .	5
2.4.4 Circle Hough Transformation . . . . .	5
2.4.5 Optimierungen . . . . .	5
<b>3 Hardware/Software Codesign am Beispiel einer Hough Transformation</b>	<b>7</b>
3.1 Softwareimplementierung . . . . .	7
3.2 Iterative Auslagerung einzelner Komponenten auf den FPGA . . . . .	7
<b>4 Evaluation</b>	<b>9</b>
<b>5 Fazit und Ausblick</b>	<b>11</b>
<b>Literaturverzeichnis</b>	<b>i</b>



# **Abbildungsverzeichnis**



# **Tabellenverzeichnis**



# Listings



# **Abkürzungsverzeichnis**

**ABS** Antiblockiersystem

**CPU** Central Processing Unit

**ESP** Elektronisches Stabilitätsprogramm

**FPGA** Field Programmable Gate Array

**GPU** Graphics Processing Unit

**HLS** High Level Synthese

**HW** Hardware

**SW** Software



# 1 Einleitung und Motivation

Vor einigen Jahren hatten die meisten Autos ein ABS<sup>1</sup> und ein ESP<sup>2</sup> zur Unterstützung bei Gefahrenbremsungen und zur Stabilisierung der Fahrt in scharfen Kurven. Mittlerweile ist die Liste der Fahrassistentenzsysteme deutlich größer geworden und umfasst unter anderen auch Adaptive Geschwindigkeitsregelanlagen, Spurhalte- und Spurwechselassistenten, Einparkhilfen und Autonome Notbremssysteme. Viele dieser Systeme basieren auf Bildbearbeitungs- und Bildanalysealgorithmen, welche gerade in Gefahrensituationen schnellstmöglich ausgewertet werden müssen. Hierfür ist die Nutzung einer CPU<sup>3</sup> meist keine ausreichende Lösung, da diese zwar eine vergleichsweise hohe Taktrate besitzt, aber aktuell mit bis zu 8 Kernen nicht genug Parallelität bietet, um große Bilder effektiv auswerten zu können. Solche Probleme werden daher vermehrt in Hardware ausgelagert, durch einen FPGA<sup>4</sup> oder eine GPU<sup>5</sup> gelöst.

Diese Bachelorarbeit behandelt eine solche Auslagerung von Software in Hardware. Dazu wird im zweiten Kapitel einleitend die Zielarchitektur vorgestellt und ein Einblick in die Grundlagen des HW<sup>6</sup>/SW<sup>7</sup> Codesigns, so wie der HLS<sup>8</sup> gegeben. Im dritten Kapitel wird am Beispiel einer Hough Transformation eine HLS durchgeführt. Dazu wird zunächst die Softwareimplementierung vorgestellt. Anschließend werden iterativ einzelne Komponenten der Hough Transformation von einer CPU auf einen FPGA ausgelagert.

Das vierte Kapitel behandelt die Auswertung des HW/SW Codesigns hinsichtlich des erbrachten Speedups und des Ressourcenverbrauchs und im fünften Kapitel wird abschließend ein Fazit gezogen und ein Ausblick in das Thema gegeben.

---

<sup>1</sup>Antiblockiersystem

<sup>2</sup>Elektronisches Stabilitätsprogramm

<sup>3</sup>Central Processing Unit

<sup>4</sup>Field Programmable Gate Array

<sup>5</sup>Graphics Processing Unit

<sup>6</sup>Hardware

<sup>7</sup>Software

<sup>8</sup>High Level Synthese



## 2 Grundlagen

### 2.1 Zielarchitektur - Zynq Systemarchitektur

### 2.2 Hardware/Software Codesign

### 2.3 High Level Synthese

### 2.4 Hough Transformation

Die Hough-Transformation ist ein Verfahren zur Erkennung von beliebigen parametrisierbaren Objekten in einem Bild. Dabei können durch ein Votingverfahren auch unvollständige, z.B. teilweise verdeckte Objekte, erkannt werden.

Da die Hough-Transformation nur auf binäre Gradientenbilder angewandt werden kann, muss ein Eingabebild zunächst in ein solches umgewandelt werden. Hierfür sind mehrere Schritte erforderlich, die in den folgenden Unterkapiteln behandelt werden.

#### 2.4.1 Umwandlung eines RGB Bildes in Graustufen

Damit eine Kantenextraktion mit dem Canny-Algorithmus durchgeführt und somit ein binäres Gradientenbild erzeugt werden kann, muss vorher eine Umwandlung eines Farbbildes in ein Graustufenbild erfolgen. Damit wird außerdem die Größe des Bildes und damit die Größe, der zu verarbeitenden Daten deutlich reduziert. Anstatt drei Farbkanäle (R, G, B) mit je 8 Bit pro Pixel verarbeiten zu müssen, muss in den nachfolgenden Schritten nur noch ein Kanal mit 8 Bit pro Pixel verarbeitet werden.

Für jedes Pixel werden die drei Farbkomponenten rot, grün und blau (R, G, B) in einen Intensitätswert  $I$  umgerechnet, welcher die Helligkeit des Pixels beschreibt. Es gibt eine Vielzahl verschiedener Ansätze, die sich unterschiedlich gut zur Kantenextraktion eignen. An dieser Stelle soll nicht genauer auf die Unterschiede eingegangen werden. Zwei detaillierte Vergleiche zu dem Thema finden sich in [KC] und [AMS]. Ein hinreichend guter und einfacher Algorithmus ist GLuminance, welcher die Intensität wie folgt berechnet:

$$I = 0.3 * R + 0.59 * G + 0.11 * B \quad (2.1)$$

### 2.4.2 Gauß-Filter

Einige Bilder enthalten ein Bildrauschen, welches bei einer späteren Kantenextraktion zu fehlerhaften Kanten führen kann. Um dieses Rauschen zu verringern wird zur Glättung des Bildes ein Gauß-Filter angewandt. Dieses addiert, abhängig von seiner Größe, benachbarte Pixel gewichtet auf. Das zu filternde Pixel hat hierbei die höchste Gewichtung. Umso weiter ein Pixel von diesem entfernt ist, desto kleiner wird dessen Gewichtung. Anschließend wird der Wert normiert, indem er durch die Summe aller Gewichtungen geteilt wird.

$$h(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \quad h(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2.2)$$

Die eindimensionale Impulsantwort  $h(x)$  entspricht der Funktion der Normalverteilung. Die zweidimensionale Impulsantwort  $h(x, y)$  des Gauß-Filters ergibt sich aus dem Produkt der Impulsantworten in x- und y-Richtung. Die Argumente  $x$  und  $y$  bezeichnen jeweils die Entfernung in horizontaler und vertikaler Richtung zum Ursprung und  $\sigma$  bezeichnet die Standardabweichung der Normalverteilung.

Ein Filterkernel für einen Gauß-Filter lässt sich nun über die zweidimensionale Impulsantwort berechnen. Ein Kernel der Größe  $N \times N$  mit  $N \in \{3, 5, \dots\}$  berechnet sich wie folgt:

$$A = \sum_{x=-B}^{B} \sum_{y=-B}^{B} h(x, y) \quad B = (N - 1)/2$$

$$\frac{1}{A} \begin{pmatrix} h(-B, -B) & \dots & \dots & h(0, -B) & \dots & \dots & h(B, -B) \\ h(-B, -1) & \dots & h(-1, -1) & h(0, -1) & h(1, -1) & \dots & h(B, -1) \\ h(-B, 0) & \dots & h(-1, 0) & h(0, 0) & h(1, 0) & \dots & h(B, 0) \\ h(-B, 1) & \dots & h(-1, 1) & h(0, 1) & h(1, 1) & \dots & h(B, 1) \\ h(-B, B) & \dots & \dots & h(0, B) & \dots & \dots & h(B, B) \end{pmatrix} \quad (2.3)$$

In der Bildverarbeitung wird häufig nur eine Approximation eines Gaußkernels verwendet, die auf dem Pascalschen Dreieck beruht. Diese bietet den Vorteil, dass der Kernel aus ganzen Zahlen aufgebaut ist und zur Normierung durch eine Zweierpotenz geteilt werden kann. Für einen Kernel der Größe  $N$  wird die  $N$ -te Zeile des Pascalschen Dreiecks in ein Feld mit  $N$  Elementen geladen. Anschließend wird der Kernel wie folgt aufgebaut: Die  $i$ -te Zeile des Kernels entsteht durch Multiplikation des Feldes mit dem  $i$ -ten Element des Feldes. Der Faktor zur Normierung ist dann  $A = 2^N$ .

Durch Ausnutzen der Separierbarkeit des Gauß-Filters kann die Rechenzeit weiter reduziert werden.....

**2.4.3 Canny Edge Detection**

**2.4.4 Circle Hough Transformation**

**2.4.5 Optimierungen**



# **3 Hardware/Software Codesign am Beispiel einer Hough Transformation**

## **3.1 Softwareimplementierung**

## **3.2 Iterative Auslagerung einzelner Komponenten auf den FPGA**



## 4 Evaluation



## **5 Fazit und Ausblick**



# Literaturverzeichnis

- [AMS] AHMAD, Ijaz ; MOON, Inkyu ; SHIN, Seok J.: Color-to-grayscale algorithms effect on edge detection - A comparative study. In: EDITOR (Hrsg.) ; IEEE (Veranst.): *Electronics, Information, and Communication (ICEIC), 2018 International Conference on IEEE*
- [KC] KANAN, Christopher ; COTTRELL, Garrison W.: Color-to-Grayscale: Does the Method Matter in Image Recognition.