TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

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

Zwischenpräsentation der Bachelorarbeit

Dominik Weinrich
dominik.weinrich@tu-dresden.de

Dresden, 31.05.2018

Gliederung

- Aufgabenstellung
- Motivation
- Fortschritt
- Implementierung
- Herausforderungen
- Bisherige Ergebnisse
- Geplantes HW/SW Codesign

Aufgabenstellung

- Softwareimplementierung einer Hough Transformation
  - Grayscaler
  - Gauß-Filter
  - Canny Edge Detection
  - Circle Hough Transformation
- Iterative Auslagerung einzelner Komponenten auf den FPGA
- Evaluation

Motivation

- Anwendungsbereiche für Hough Transformation vielseitig
- Erkennung von Passanten in selbst fahrenden Automobilen
- Zählen von Objekten (Geld, GO-Spielsteine, ...) auf einem Bild
- Beschleunigung von hochgradig parallelen, rechenintensiven Aufgaben mittels FPGA oftmals notwendig, um Aufgaben echtzeitfähig zu machen

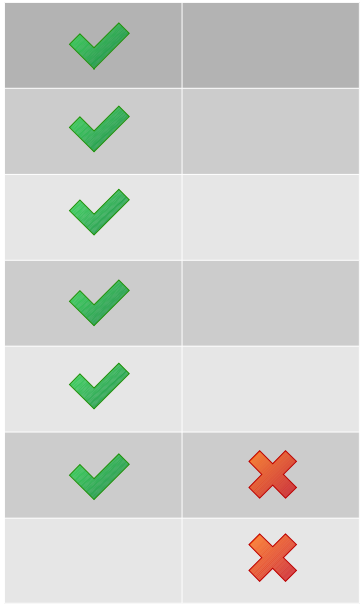
Abb. 1: Go-Spielbrett [1]

Abb. 2: Euromünzen [2]

Fortschritt

- Softwareimplementierung einer Hough Transformation
  - Grayscaler
  - Gauß-Filter
  - Canny Edge Detection
  - Circle Hough Transformation
- Iterative Auslagerung einzelner Komponenten auf den FPGA
- Evaluation

Zwischenstand



Implementierung

- Sprache: **C**
- Benutzte Bibliotheken/APIs:
  - **SDL2**
    - **Kann verschiedene Bildformate laden**
    - **Leichter Zugriff auf die Roh-Pixeldaten**
  - **OpenMP**
    - **Einfache Möglichkeit zur Parallelisierung des Codes**
    - **Über das Compilerflag -fopenmp kann leicht eine parallele und eine serielle Version erstellt werden**
- Modularer Aufbau für eine leichtere Anpassung für die HLS

Implementierung - Main

- Liest Argumente ein und wertet diese aus
- Initialisiert SDL2 und lädt Pixeldaten des Bildes in ein Array
- Ruft die folgenden Funktionen auf und misst deren Laufzeit
  - **uint8_t* grayscaler(uint32_t* input, ...);**
  - **uint8_t* gauss(uint8_t* input, ...);**
  - **uint8_t* canny(uint8_t* input, ...);**
  - **circle* hough(uint8_t* input, ...);**
- Gibt Anzahl der gefundenen Kreise aus und zeichnet diese mit dem Bresenham Algorithmus
- Enthält weitere Hilfsfunktionen

Implementierung - Grayscaler

- Wandelt ein 32 Bit Farbbild in ein 8 Bit Graustufenbild um
- Berechnet Farbanteile (r, g, b) des Pixels und addiert diese gewichtet zu einem Intensitätswert auf

Abb. 4: Eingabebild [2]

$$I = 0.3 \cdot r + 0.59 \cdot g + 0.11 \cdot b$$

Abb. 5: Ausgabebild

Implementierung - Gauss-Filter

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{\frac{-x^2+y^2}{2\sigma^2}}$$

- Ein Gauss-Filter vermischt benachbarte Pixel miteinander → eliminiert Bildrauschen
- Wendet dazu einen Filterkernel der Größe **k** auf einen Bildausschnitt der Größe **k** an
- Besondere Betrachtung für Randpixel
- Zur Beschleunigung dazu wird der Filterkernel in zwei eindimensionale Kernel zerlegt, welche nacheinander angewandt werden

$$\frac{1}{16}\begin{pmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{pmatrix}=\frac{1}{4}\begin{pmatrix}1\\ 2\\ 1\end{pmatrix}*\frac{1}{4}\begin{pmatrix}1 & 2 & 1\end{pmatrix}$$

Implementierung - Gauss-Filter

Abb. 6: Eingabebild

Abb. 7: Ausgabebild

Implementierung – Canny Edge Detection

- Findet Kanten mithilfe eines Gradientenfilters
- Sobelfilter ist gängigstes Filter

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Anwendung des Sobeloperators auf jedes Pixel in x- und y-Richtung → Gradientenbilder
- Die Kantenstärke berechnet sich dann über den euklidschen Betrag der partiellen Ableitungen

$$g(x,y) = \sqrt{g_x(x,y)^2 + g_y(x,y)^2}$$

- Die Richtung der Kante berechnet sich über den Arkustangens

$$\theta = \tan^{-1}\left(\frac{g_y(x,y)}{g_x(x,y)}\right)$$

Implementierung – Canny Edge Detection

- Durchführung einer Non-maximum suppression
  - Laufe über die Kante und Vergleiche Pixelwert mit benachbarten Werten
  - Setze Wert auf 0, wenn kleiner als einer der zwei Nachbarn
- Durchführung einer Hysterese
  - Schaue für jedes noch nicht gesetzte Pixel, ob Wert größer als **high_threshold** ist; Falls ja → Setze Pixel
  - Laufe in positiver und negativer Richtung entlang der Kante und schaue, ob die Werte größer **low_threshold** sind; Falls ja → Setze Pixel

Implementierung – Canny Edge Detection

Abb. 8: Eingabebild

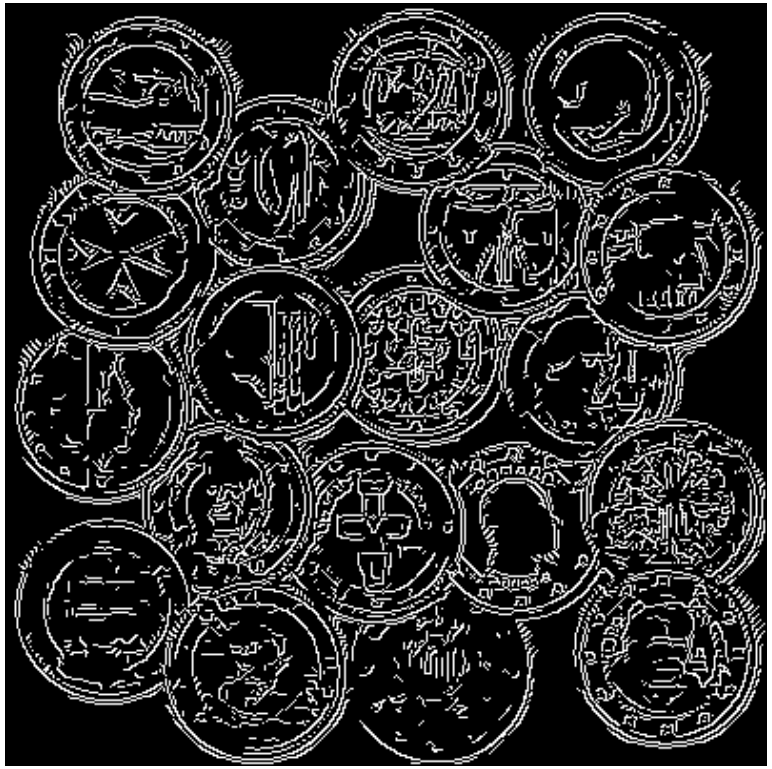


Abb. 9: Ausgabebild

Implementierung – Circle Hough Transformation

Implementierung – Circle Hough Transformation

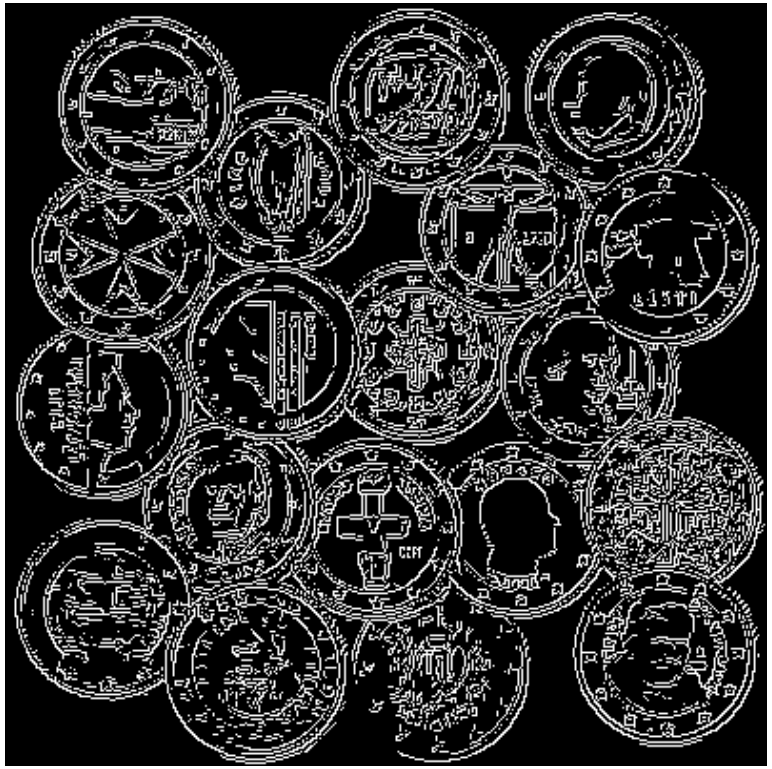


Abb. 10: Eingabebild

Abb. 11: Ausgabebild

Herausforderungen

- Erkennung von Objekten mit verschiedenen Radien
  - Lösungsansatz: Bias compensation
- Mehrfache Erkennung des selben Objekts
- Es werden viele Parameter benötigt, welche mühsam gesucht werden müssen
  - Möglicher Ansatz: Automatische Parameterbestimmung über neuronale Netze (nicht Teil der Arbeit)

Bisherige Ergebnisse

Geplantes HW/SW Codesign

- Durchführung der HLS für alle einzelnen Komponenten der Hough Transformation
- Auslagern einzelner Komponenten auf den FPGA
- Implementierung eines Controllers, welcher mit FPGA kommuniziert
- Testen, welche Komponenten einen Geschwindigkeitsvorteil erbringen

Quellen

- [1] Go-Spielbrett: https://www.japanwelt.de/media/image/go-spiel.jpg, 30.05.2018
- [2] Euro-Münzen: http://www.historia-hamburg.de/media/product/1ec/19-x-1-euro-satz-aus-19-euro-staaten-511.jpg, 30.05.2018
- [3] Ahmad, Ijaz; Moon, Inkyu ; Shin, Seok J.: Color-to-grayscale algorithms effect on edge detection – A comparative study. In: *Electronics, Information, and Communication (ICEIC), 2018 International Conference on* IEEE
- [4] Burger, Wilhelm; Burge, Mark J.: *Principles of Digital Image Processing: Fundamental Techniques.* London : Springer, 2009. - ISBN 978-1-84800-190-9
- [5] Burger, Wilhelm; Burge, Mark J.: *Digital Image Processing – An Algorithmic Introduction Using Java.* 2. London : Springer, 2016. - ISBN 978-1-4471-6683-2
- [6] Kanan, Christopger; Cottrell, Garrison W.: Color-to-Grayscale: Does the Method Matter in image Recognition