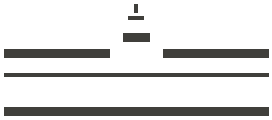


Westfälische
Wilhelms-Universität
Münster

Ausarbeitung zum Praktikum

zu der Vorlesung Computer Vision

veranstaltet von:

Prof. Dr. X. Jiang

Version vom 21. März 2017

Liste der noch zu erledigenden Punkte

Kapitel 1

Einführung

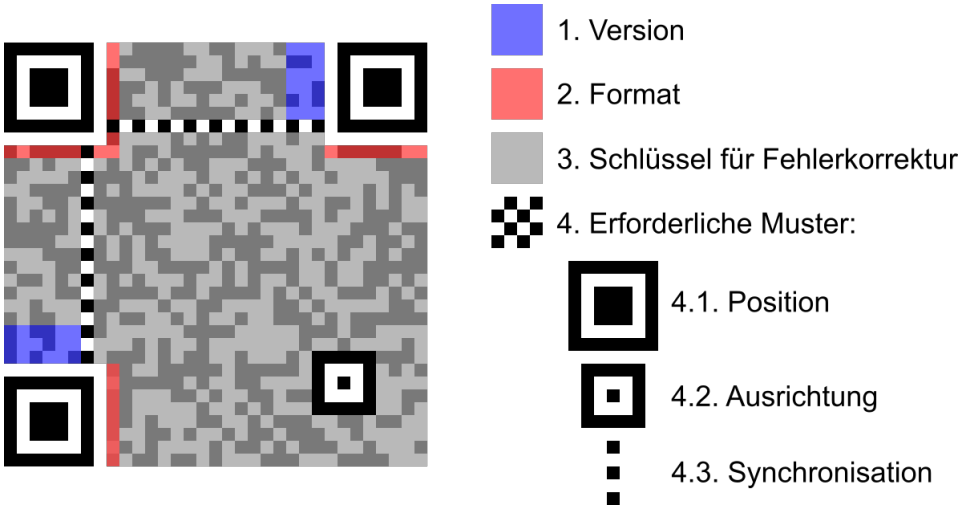
1.1 Aufgabenstellung

Ziel dieses Praktikums war es eine Software zu konstruieren um QR-Codes in Bildern zu lokalisieren und danach zu extrahieren. Dabei sollten sich die Teams auf QR-Codes, die dem ISO/IEC Standard 18004 entsprechen, beschränken. Unter der Annahme, dass sämtliche QR-Codes auf den Eingabebildern planar waren, sollten perspektivische Transformationen oder ähnliche Verzerrungen entfernt werden. Zusätzlich sollte sich jedes Team darum kümmern ein *Dataset* zur Analyse und späteren Evaluation zu erschaffen.

1.2 Aufbau des QR-Codes

Um das spätere Vorgehen der Lokalisierung des QR-Codes nachvollziehen zu können, wollen wir kurz auf den Aufbau des QR-Codes eingehen. Wie in der Abbildung **??** zu sehen ist, sind im Code die Informationen zur Version, Format und der Fehlerkorrektur enthalten.

Abbildung 1.1: QR-Code Strukturbeispiel



Quellen angeben!

Des weiteren ist klar zu erkennen, dass der Code in drei von vier Ecken das Muster 4.1 vorweist. Diese dienen zur Bestimmung der Orientierung. Zusätzlich wird das Muster 4.2 (*Finder Pattern*) verwendet um die Ausrichtung noch prezieser zu bestimmen. Bei größeren QR-Codes werden weitere Ausrichtungsmuster

(Muster 4.2) eingefügt. Zwischen den *Finder Patterns* befindet sich ein Streifen mit abwechselnd „schwarz - weißen“ Punkten (*Modulen*). Das sind die so genannten Synchronisations Muster oder auch *Timming Patterns*.

Die Größe der QR-Codes ist beschränkt durch die Anzahl der *Module*. Die Anzahl der *Module* liegt zwischen 21×21 und 177×177. Beispielsweise sei der Code in Abbildung **??** einer der Größe 21×21. Dann hätte ein *Finder Pattern* die länge von 7 *Modules*. Die Anzahl ergibt sich aus der einzigartigen $1:1:3:1:1$ Struktur eines *Finder Patterns*. Diese Fakten werden später ausschlagegebend sein bei der Rasterisierung des QR-Codes.

1.3 Die verwendette Bibliothek OpenCV

OpenCV[1] ist eine Bibliothek mit Algorithmen spezialisiert auf „Computer Vision“. Sie wurde für die Programmiersprachen C/C++ geschrieben und steht unter BSD Lizenz. Es gibt mehrere Versionen der Bibliothek, die aktuellste ist die 3.2. Unser Programm setzt allerdings die Version 2.4, funktioniert aber auch mit neueren Version z.B .

Phill fragen

[1] WWW-Seite von dem Projekt *OpenCV*: `http://opencv.org/`

Kapitel 2

Bildvorverarbeitung

Die Bildvorverarbeitung ist ein essenzieller Schritt um die QR-Codes richtig lokalisieren zu können. Das Eingabebild soll in dieser Phase erstmal in ein Graustufenbild umgewandelt werden und danach binarisiert werden. *OpenCV* bietet die Möglichkeit bilder direkt als Graustufenbild einzulesen oder sie in eins umzuwandeln. Das in Graustufen vorhandene Bild kann weiter verarbeitet werden. Die Klasse `ImageBinarization` ist für die binarisierung zuständig.

Listing 2.1.:

```
Mat ImageBinarization::run(Mat image, int thresholdMethod) {

    this->image = image;
    computeSmoothing();
    createHistogram();

    printThresholdMethod(thresholdMethod);
    switch (thresholdMethod) {
    case Global:
        setThresholdValue();
        computeGlobalThreshold();
        break;

    case LocalMean:
        computeLocalThreshold(ADAPTIVE_THRESH_MEAN_C, 11, 0);
        break;

    case LocalGaussian:
        computeLocalThreshold(ADAPTIVE_THRESH_GAUSSIAN_C, 11, 0);
        break;
    }
    return binarizedImage;
}
```

Listing 2.1 zeigt den Ablauf der Binarisierung. In Zeile 4 wird die Methode `computeSmoothing` aufgerufen, die eine Gaußglättung auf dem Bild durchführt, um eventuelles Rauschen zu mindern. Im nächsten Schritt wird ein Farbstufen-Histogramm mithilfe der `computeHistogram` Methode berechnet. Um die best Mögliche Binarisierung zu erreichen, wurden drei Methoden zur Schwellwert Berechnung implementiert:

- globales Schwellwertverfahren,
- Mittelwert basiertes Schwellwertverfahren,
- Gauß'sches Schwellwertverfahren.

Im ersten durchlauf wird das globale Schwellwertverfahren angewandt. Sollte dies keine gültige Ergebnisse liefern (keine drei *Finder Pattern* enthalten), so wird das nächste Verfahren gewählt.

Abhängig von der Wahl werden zwei verschiedene *OpenCV* Methoden verwendett. Bei der globalen Schwellwert Berechnung wird auf die `threshold` Methode wie sie im Listing 2.2 steht zurückgegriffen. Der Methode werden das Ein- und Ausgabe Bild, die Schwellwertgrenze, der maximale Farbwert und die Wahl des Algorithmus übergeben.

Listing 2.2.:

```
void ImageBinarization::computeGlobalThreshold() {
    threshold(blurredImage, binarizedImage,
              threshold_value, max_BINARY_value, CV_THRESH_OTSU);
}
```

Wir haben uns hier auf den Algorithmus von Otsu[1] geeinigt, da er sehr gute Ergebnisse mit einer geringen Laufzeit liefert.

Bei den lokalen Schwellwertverfahren hingegen wird die Methode `adaptiveThreshold` verwendett. Listing 2.3 veranschaulicht die Methode die für die zwei lokalen Verfahren verwendet wird. Für die jeweiligen Verfahren wird der Parameter `adaptiveMethod` passend übergeben.

Listing 2.3.:

```
void ImageBinarization::computeLocalThreshold(int adaptiveMethod,
                                              int blockSize, int C) {
    adaptiveThreshold(blurredImage, binarizedImage,
                      max_BINARY_value, adaptiveMethod,
                      THRESH_BINARY, blockSize, C);
}
```

Abbildung

Abbildung zeigt das Resultat.

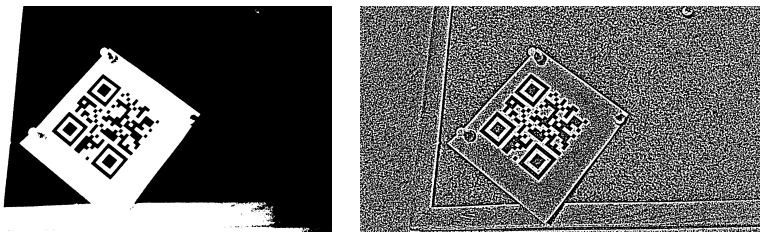


Abbildung 2.1: Das Resultat der Binarisierung mit den jeweiligen Verfahren. Links global und Rechts lokal.

[1] Mehr Information zu dem Algorithmus auf der zugehörigen *OpenCV* Seite:`http://docs.opencv.org/3.1.0/d7/d4d/tutorial_py_thresholding.html`

Kapitel 3

Patternidentifikation