|  |
| --- |
| C:\Users\holzer\Downloads\qr_code\matlab-qrcode.jpg |
| Matlab QR-Code-Reader  Bericht  **Modul: BZG1301 "Programmierung in Matlab/Octave" Dozent: Marx Stampfli Autor: Joel Holzer**  **Version: 1.0, 24.01.2016** |
| **Berner Fachhochschule**  Technik und Informatik  Informatik |

Zusammenfassung

Inhaltsverzeichnis

[1 Ausgangslage & Idee 4](#_Toc441165057)

[2 Vorgehen 5](#_Toc441165058)

[2.1 Zeitplan (Soll-Ist-Vergleich) 5](#_Toc441165059)

[2.2 Eingesetzte Tools & Technologien 5](#_Toc441165060)

[3 Angewendete Methoden & Konzepte 6](#_Toc441165061)

[3.1 Übersicht 6](#_Toc441165062)

[3.2 Inhalt des QR-Codes auslesen 6](#_Toc441165063)

[3.2.1 Übersicht 6](#_Toc441165064)

[3.2.2 Schritt 1 – QR-Code-Bild in Graustufen- und Binary-Bild konvertieren 7](#_Toc441165065)

[3.2.3 Schritt 2 – Begrenzungsmuster suchen und Bild zuschneiden 7](#_Toc441165066)

[3.2.4 Schritt 3 – QR-Code-Version berechnen 7](#_Toc441165067)

[3.2.5 Schritt 4 – Format-Infos auslesen 7](#_Toc441165068)

[3.2.6 Schritt 5 – Angewendete Maske berechnen 7](#_Toc441165069)

[3.2.7 Schritt 6 – Ausrichtungsmuster suchen und einfärben 7](#_Toc441165070)

[3.2.8 Schritt 7 – Daten (Binär) auslesen 7](#_Toc441165071)

[3.2.9 Schritt 8 – Zeichensatz und Textlänge ermitteln 7](#_Toc441165072)

[3.2.10 Schritt 9 – Daten nach ISO 8859-1 konvertieren und Text ausgeben 7](#_Toc441165073)

[4 Ergebnisse 7](#_Toc441165074)

[5 Zeitplan 7](#_Toc441165075)

[6 Persönliches Fazit des Autors 7](#_Toc441165076)

[7 Tabellenverzeichnis 8](#_Toc441165077)

[8 Abbildungsverzeichnis 8](#_Toc441165078)

[9 Literaturverzeichnis 8](#_Toc441165079)

[10 Versionskontrolle 8](#_Toc441165080)

# Ausgangslage & Idee

QR-Codes sind in der heutigen Zeit ein weit verbreitetes Medium um Menschen einen schnellen Zugriff auf Daten zu ermöglichen. Mit einem klassischen QR-Code-Scanner (Gerät) oder einem Mobiltelefon kann der QR-Code gescannt und so dessen Information ausgelesen und dem Benutzer in verständlicher Form angezeigt werden. Oft werden QR-Codes verwendet, um URLs von Webseiten zu Speichern. Beim Scan des QR-Codes mit dem Mobiltelefon wird dem Benutzer dann die entsprechende Webseite im Browser angezeigt. Weitere Anwendungsbereiche sind bei IT-Lösungen im Mobile Computing oder Internet of Things (kurz IOT) zu finden. Dort werden QR-Codes hauptsächlich verwendet um Daten zwischen verschiedenen mobilen Geräten oder Things (bei IOT) auszutauschen.

QR-Codes sind aber nicht nur auf die Speicherung von Text beschränkt, sondern ermöglichen die Speicherung beliebiger binärer Daten (bis max. 2956 Byte).

Für Softwareentwickler steht eine Vielzahl von Libraries zur Verfügung, welche die einfache Integration eines QR-Code-Readers in eigene Applikationen ermöglichen. Dazu sind meistens weder Grundwissen über den Aufbau von QR-Codes, noch Kenntnisse über die Bildverarbeitung erforderlich. Die ganze Analyse des QR-Code-Bilds und die Datenextraktion aus dem Bild werden meistens durch Third Party Libraries durchgeführt.

Auch der Autor dieser Arbeit hat bereits mehrfach Erfahrungen mit dem Einsatz von QR-Code-Reader-Libraries im Android-Umfeld gesammelt. Doch wie kann ein QR-Code ohne diese Libraries gelesen werden?

Im Rahmen dieser Arbeit wird genau dieser Frage auf den Grund gegangen. Dazu wird mit Hilfe von Matlab von Grund auf einen QR-Code-Reader entwickelt. Auf die Verwendung von Matlab QR-Code-Libraries wird verzichtet.

Die zu entwickelte Matlab-Applikation soll folgende Funktionalitäten beinhalten:

* **Angabe des Bildpfads:**   
  Pfad zur Bilddatei (PNG), welche einen 2D QR-Code beinhaltet, kann angegeben werden.
* **Erkennung des QR-Codes in einem Bild:**   
  Die Applikation erkennt durch die Anwendung klassischer Methoden aus der Bildanalyse den QR-Code in der angehängten Bilddatei. Der QR-Code in der Bilddatei kann nur ein Teil des Gesamtbilds sein und sich irgendwo im Bild befinden.  
  Einschränkung: In einer ersten Version wird die Funktionsweise so eingeschränkt, dass der QR-Code im Bild in den Farben Schwarz/Weiss sein muss. Je nach Verlauf des Projekts wird die Applikation so erweitert, dass der QR-Code irgendwelche kontrastreichen Farben beinhalten darf.
* **Extraktion des Inhalts:**Die Applikation kann den Inhalt von QR-Codes, welche Text (Buchstaben, Zahlen, Zeichen) nach ISO-8859-1 beinhalten, aus dem QR-Code extrahieren und den Text dem Benutzer auf dem Bildschirm anzeigen.
* **Unterstützung verschiedener Versionen von QR-Codes:**Es gibt verschiedene Versionen von QR-Codes. Diese unterscheiden sich in der Anzahl Module. Je mehr Module ein QR-Code beinhaltet, desto mehr Daten können gespeichert werden. Der entwickelte QR-Code-Reader soll alle QR-Code Versionen, von Version 1 mit 21 x 21 Modulen bis hin zu Version 40 mit 177 x 177 Modulen lesen können.

Daneben wurde dieser Bericht erstellt. Dieser soll dem Leser einen Einblick in die Umsetzung der genannten Applikation ermöglichen. Zuerst folgt eine Erläuterung des Vorgehens (Zeitplan, Tools und Technologien). Im Hauptteil geht der Autor detailliert auf die angewendeten Methoden und Konzepte ein und erläutert deren Zweck und die Art der Umsetzung in der realisierten Matlab-Anwendung. Danach folgt eine Betrachtung des Ergebnisses und zu guter Letzt wird der Bericht mit einem persönlichen Fazit des Autors abgeschlossen.

# Vorgehen

## Zeitplan (Soll-Ist-Vergleich)

Nachfolgender Zeitplan zeigt den Vergleich zwischen den geplanten Umsetzungsterminen (Soll) und den effektiven Umsetzungsterminen (Ist) der durchgeführten Tasks. Die Soll-Felder sind Blau markiert, während die Ist-Felder Grün markiert sind. Gegenüber dem Soll-Zeitplan sind die beiden Tasks „GUI für Dateiauswahl und Darstellung aller Lese-Schritte“, sowie der Task „Code-Dokumentation“ hinzugekommen. Die Erstellung eines GUIs war zu Beginn des Projekts nicht vorgesehen, wurde dann vom Entwickler jedoch als zusätzliches Feature umgesetzt.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tätigkeit** | **KW 48**  **25.11** | **KW 49** | **KW 50** | **KW 51** | **KW 52** | **KW 53** | **KW 01** | **KW 02** | **KW 03** |
| **2.12** | **9.12** | **16.12** | **23.12** | **30.12** | **6.1** | **13.1** | **20.1** |
| Einlesen in QR-Code-Theorie |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Realisierung: Einlesen eines Bilds und Erkennung der Begrenzungsmuster |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Realisierung: Erkennung der übrigen standardisierten Muster und Ausrichtung des QR-Codes |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Realisierung: Extraktion der Daten |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Realisierung: Demaskierung der Daten |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Realisierung: Dekodierung und Ausgabe der Daten |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Realisierung: GUI für Dateiauswahl und Darstellung aller Lese-Schritte |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Realisierung: Code-Dokumentation |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Finalisierung der Dokumentation |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Präsentation und Abgabe |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Tabelle 1 Zeitplan (Soll-Ist-Vergleich)

## Eingesetzte Tools & Technologien

Nachfolgende Tabelle zeigt die bei der Umsetzung der Applikation zum Einsatz kommenden Tools und Technologien:

|  |  |
| --- | --- |
| Anwendungsbereich | Tool oder Technologie |
| Umsetzung der Funktionalität zum Auslesen der QR-Codes | Matlab R2015a |
| GUI-Erstellung mit Matlab | In Matlab integrierter GUI-Editor „GUIDE“ |
| Generierung von QR-Codes | <http://www.qrcode-monkey.de/#text> |
| Versionsverwaltung Programmcode | GitHub |
| Betriebssystem Entwicklungscomputer | Windows 7 |

Tabelle 2 Eingesetzte Tools & Technologien

# Angewendete Methoden & Konzepte

## Übersicht

Die Umsetzung des QR-Code-Readers kann grob in zwei Teile unterteilt werden: Erstens die Erstellung der Routine zum Auslesen des QR-Codes, zweitens die Erstellung des GUIs, welches die Auswahl einer Bilddatei ermöglicht und den Leseprozess des QR-Codes Schrittweise dokumentiert.

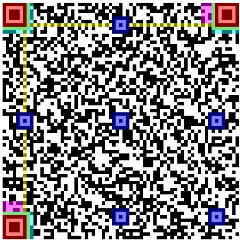
Dieses Kapitel beleuchtet die beiden Teile der realisierten Matlab-Anwendung und die den Teilen zugrunde liegende Theorie über QR-Codes.

## Inhalt des QR-Codes auslesen

### Übersicht

In einem QR-Code sind diverse Informationen versteckt, welche von der erstellten Anwendung in mehreren Schritten ausgelesen werden. Die Anwendung führt folgende Schritte durch:

1. Bild mit dem QR-Code wird in ein Graustufen-Bild und danach in ein Binary-Bild konvertiert.
2. Die drei Begrenzungsmuster (Finder Patterns) des QR-Codes werden gesucht. Das Bild wird mit diesen drei Begrenzungsmustern so zugeschnitten, dass dieses nur noch aus dem QR-Code besteht.
3. Nun errechnet die Anwendung aus der Grösse der Begrenzungsmuster die Grösse eines einzelnen Moduls aus. Damit kann die Version des QR-Codes bestimmt werden. Je nach Version des QR-Codes (1-40) unterscheidet sich das Vorgehen bei den nachfolgenden Schritten.
4. Liest die Format-Infos aus. Dies ist ein Block von 5 Modulen unterhalb des oberen-linken Begrenzungsmuster, welcher für die Berechnung der angewendeten Maske verwendet wird.
5. Errechnet aus den bei Schritt 4 erlangten Format-Infos die angewendete Maske. Es gibt 8 verschiedene Masken, wovon bei der QR-Code-Erstellung eine über den QR-Code gelegt wurde um die Farbe gewisser Module zu verändern. Diese Maske wird benötigt, um die Ursprungswerte der Module zu errechnen.
6. Sucht die Ausrichtungsmuster und färbt diese im Bild rot ein. Ausrichtungsmuster sind nur in QR-Codes ab Version 2 vorhanden. Die Kenntnis der Ausrichtungsmuster ist nötig, weil an dieser Stelle im QR-Code keine Daten vorhanden sind.
7. Liest die Daten des QR-Codes von unten rechts nach oben links ein. Jedes Modul wird mit der bei Schritt 5 errechneten Maske demaskiert und dann ein binärer Wert (0 oder 1) erhalten. Das Ergebnis ist ein binärer String, welcher jedoch nicht nur den im QR-Code hinterlegte Text, sondern auch der Zeichensatz des Texts, sowie die Text-Länge, beinhaltet.
8. Ermittelt, aus dem bei Schritt 7 gelesenen Binären-String, den Zeichensatz und die Textlänge.
9. Wandelt den Text-Teil des bei Schritt 7 gelesenen Binären-String mit dem bei Schritt 8 gelesenen Zeichensatz (Nur ISO 8859-1 unterstützt) um. Diese ist dann der Text, welcher vom Ersteller im QR-Code hinterlegt wurde.

Nachfolgende Abbildung zeigt die erwähnten Muster (Patterns) eines QR-Codes (Details folgen in den kommenden Kapiteln).

Rot = Begrenzungsmuster (Finder Patterns)  
Blau = Ausrichtungsmuster (Alignment Patterns). Ab Version 2

vorhanden.  
Grün = Format- und Fehlerbehebungs-Block  
Pink = Versionsnummer. Nur bei Version 7 und höher vorhanden.

Kommt in der erstellten Applikation nicht zur Anwendung.

Gelb = An dieser Stelle sind keine Daten vorhanden.

Übriges: Daten-Bits (Inhalt).

Abbildung 1: QR-Code mit farblicher Darstellung  
der verschiedenen Muster (Patterns). [1]

### Schritt 1 – QR-Code-Bild in Graustufen- und Binary-Bild konvertieren

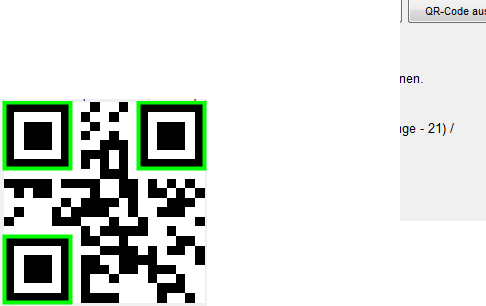
Ein Binärbild ist eine Grafik, welche nur die zwei Farben Schwarz und Weiss beinhaltet. Um ein Binärbild zu erstellen, muss das farbige RGB-Bild zuerst in ein Graustufen-Bild umgewandelt werden. Das Graustufenbild wird dann unter der Angabe eines bestimmten Schwellenwerts (Threshold) in das Binärbild konvertiert. Der Schwellenwert bestimmt, ab welchem Grauwert ein Pixel in Schwarz oder in Weiss umgewandelt wird.

Mit der Graustufen- und Binär-Konvertierung im QR-Code-Reader wird erreicht, dass dieser nicht nur Schwarz-Weisse, sondern auch andersfarbige QR-Codes verarbeiten kann. Wichtig dabei ist, dass die beiden Farben sich gegenüber einen grossen Kontrast aufweisen (z.B. Blau und Weiss, Oder Blau und Gelb). Nach der Binär-Konvertierung ist für die Weiterverarbeitung in jedem Fall ein Schwarz-Weiss-Bild vorhanden (Dunkle Farbe wird Schwarz, Helle Farbe Weiss).

Abbildung 2: Graustufen- und Binär-Bild von einem QR-Code

Die erstellte Anwendung führt die Graustufen- und Binär-Konvertierung in der erstellten Funktion „convertImageToBinary“ in der Matlab-Datei „convertImageToBinary.m“ durch. Zum Einsatz kommen die beiden Matlab-Funktionen „rgb2gray“ und „im2bw“. Als Schwellenwert wurde der Standardwert 0.5 der Matlab-Funktion „im2bw“ verwendet.

### Schritt 2 – Begrenzungsmuster suchen und Bild zuschneiden

Die vom User angegebene, und in Schritt 1 zu einem Binärbild konvertierte, Bilddatei muss nicht nur aus dem QR-Code bestehen. Der QR-Code kann nur ein Teil des Bilds sein und sich irgendwo im Bild befinden. Bevor die Matlab-Anwendung den QR-Code auslesen kann, muss sie diesen im Bild lokalisieren. Dazu sucht die Anwendung im Bild nach den Begrenzungsmuster (Finder Patterns) des QR-Codes.

Ein QR-Code beinhaltet 3 gleiche Begrenzungsmuster, jeweils im oberen linken Ecken, im unteren linken Ecken und im oberen rechten Ecken. Diese Begrenzungsmuster sehen bei jedem QR-Code genau gleich aus. Jedes Begrenzungsmuster ist ein Quadrat mit 7 x 7 Modulen. Die äussersten Module auf jeder Seite sind Schwarz, danach folgt eine Schicht weisser Module, bevor dann im Innern ein Quadrat mit 3 x 3 Schwarzen Modulen kommt. Das gibt von aussen nach innen gesehen ein Schwarz-Weiss Verhältnis von 1:1:3:1:1.

Abbildung 3: QR-Code mit Grün-markierten Begrenzungsmustern

Die realisierte Matlab-Anwendung sucht die 3 Begrenzungsmuster mit Hilfe von „Connected-component labeling“. Dies ist eine Methode aus der Computer Vision um zusammenhängende Pixel im Bild zu erkennen. Die 3 Begrenzungsmuster weisen eine grosse, bei allen Begrenzungsmustern gleiche Anzahl, zusammenhängende Pixel auf.

Mit Hilfe der nachfolgenden Zeilen macht die Anwendung ein Connected-component labeling und erstellt eine Struktur, in welcher an jeder Position ein zusammenhängender Bereich mit deren Grösse und deren Rand (Bounding Box) gespeichert wird.

|  |
| --- |
| % Connected component labeling  [labeled, numberOfObjects] = bwlabel(binaryImage, 8);  % Creates a structure for every object in labeled (3 finder pattern, 1 border % pattern and alignement patterns)  structLabeledObjects = regionprops(labeled, 'all'); |

Die erstellte Struktur wird dann mit Hilfe zweier verschachtelter For-Schleifen durchlaufen. Dabei wird die Fläche jedes zusammenhängenden Bereichs mit den Flächen der anderen zusammenhängenden Bereiche verglichen. Sobald 3 zusammenhängende Bereiche die gleiche Fläche haben, handelt es sich um einen Kandidat für ein Begrenzungsmuster. Es kann jedoch auch sein, dass es zufälligerweise noch andere zusammenhängende Bereiche im QR-Code gibt, von welchen genau 3 dieselbe Fläche aufweisen. Daher werden in einem ersten Schritt alle zusammenhängende Bereiche ermittelt und diese danach nochmals miteinander verglichen. Die Begrenzungsmuster weisen immer einen grösseren zusammenhängenden Bereich als die übrigen zusammenhängenden Bereiche auf.

Nachdem die Anwendung die Begrenzungsmuster gefunden hat, schneidet diese einerseits das Bild auf die Grösse des QR-Codes zu (mit Hilfe der Methode „imcrop“), andererseits berechnet diese die Länge/Breite eines einzelnen Moduls (in Pixel). Die Länge/Breite eines Begrenzungsmuster ist bekannt und auch die Tatsache, dass ein Begrenzungsmuster pro Seite 7 Module lang ist. Die Seitenlänge in Pixel durch 7 ergibt daher die Seitenlänge eines Moduls in Pixel. Die errechnete Seitenlänge eines Moduls wird in späteren Schritten benötigt.

Der Code für das Suchen der Begrenzungsmuster und Zuschneiden des Bilds befindet sich in der Funktion „findFinderPatternsAndCropImage“ in der Datei „findFinderPatternsAndCropImage.m“.

### Schritt 3 – QR-Code-Version berechnen

QR-Codes existieren von Version 1 bis Version 40. Je grösser die Version eines QR-Codes, aus desto mehr Modulen besteht dieser und desto mehr Daten kann dieser speichern. Ein QR-Code der Version 1 besteht beispielsweise aus 21 x 21 Modulen und kann 17 Zeichen nach ISO 8859-1 speichern. Bei jeder QR-Code-Version kommen pro Seitenlänge 4 Module dazu, d.h. 25 x 25 Module bei Version 2, 29 x 29 Module bei Version 3, usw. Version 40 mit 177 x 177 Modulen kann bis zu 2953 Zeichen nach ISO 8859-1 speichern.

Ursprünglich war geplant, dass der realisierte QR-Code-Reader QR-Codes aller Versionen lesen kann. Aus für den Autor unerklärlichen Gründen führte der QR-Code-Reader jedoch ab Version 6 zu einem Fehlverhalten beim Auslesen, d.h. es wurde ein falscher Text ausgelesen. Auch eine Mehrtägige Fehlersuche und Durchforstung von diversen Webseiten zum Aufbau von QR-Codes führten nicht zur Auffindung des Fehlers. Daher hat sich der Autor entschieden, die Anwendung auf QR-Codes bis Version 5 zu beschränken. Bereits mit Version 5 ist es möglich, 106 Zeichen nach ISO 8859-1 zu speichern, was für die meisten Anwendungsgebiete von QR-Codes ausreicht.

1. Nun errechnet die Anwendung aus der Grösse der Begrenzungsmuster die Grösse eines einzelnen Moduls aus. Damit kann die Version des QR-Codes bestimmt werden. Je nach Version des QR-Codes (1-40) unterscheidet sich das Vorgehen bei den nachfolgenden Schritten.

### Schritt 4 – Format-Infos auslesen

### Schritt 5 – Angewendete Maske berechnen

### Schritt 6 – Ausrichtungsmuster suchen und einfärben

### Schritt 7 – Daten (Binär) auslesen

### Schritt 8 – Zeichensatz und Textlänge ermitteln

### Schritt 9 – Daten nach ISO 8859-1 konvertieren und Text ausgeben

## Erstellung des GUIs

# Ergebnisse

# Zeitplan

# Persönliches Fazit des Autors

# Tabellenverzeichnis

[Tabelle 1 Zeitplan (Soll-Ist-Vergleich) 5](#_Toc441160264)

[Tabelle 2 Eingesetzte Tools & Technologien 5](#_Toc441160265)

# Abbildungsverzeichnis

Abbildung 1: QR-Code mit farblicher Darstellung der verschiedenen Muster (Patterns). 6

# Literaturverzeichnis

1. **QR Codes**  
   *Webseite swisseduc.ch - Informatik, Stand 25.11.2015*[*http://www.swisseduc.ch/informatik/theoretische\_informatik/qr\_codes/docs/unterlagen\_lernende.pdf*](http://www.swisseduc.ch/informatik/theoretische_informatik/qr_codes/docs/unterlagen_lernende.pdf) 6

<http://ch.mathworks.com/discovery/matlab-gui.html>

<https://de.wikipedia.org/wiki/Bin%C3%A4rbild>

<https://en.wikipedia.org/wiki/Connected-component_labeling>

# Versionskontrolle

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Datum** | **Beschreibung** | **Autor** |
| 1.0 | 24.01.2016 | Dokument finalisiert und freigegeben. | Joel Holzer |