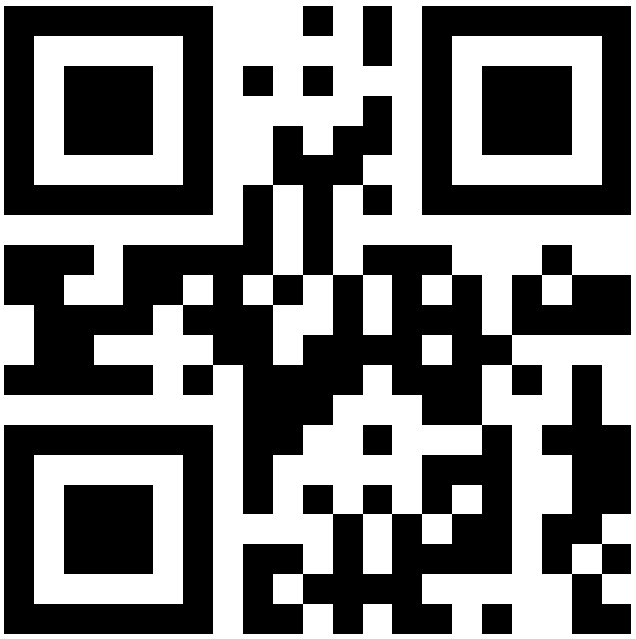

Matlab QR-Code-Reader

Bericht

Modul: BZG1301 "Programmierung in Matlab/Octave"
Dozent: Marx Stampfli
Autor: Joel Holzer
Version: 1.0, 23.01.2016

Berner Fachhochschule
Technik und Informatik
Informatik

Zusammenfassung

QR-Codes sind ein Medium, welches hauptsächlich genutzt wird, um kurze Textinformationen weiterzugeben. Häufig werden QR-Codes in der Werbung eingesetzt, um Personen, welche den QR-Code mit dem Smartphone scannen, den Aufruf der eigenen Webseite zu ermöglichen.
Auf den ersten Blick sind QR-Codes ein Bild aus schwarzen und weissen, zufällig angeordneten, Feldern. Doch hinter jedem QR-Code steckt ein System. Der im QR-Code versteckte Text wurde mit mehreren, fest vorgegebenen Schritten zu einem QR-Code umgewandelt. Beim Lesen des QR-Codes wird dasselbe Vorgehen rückwärts angewendet, um vom QR-Code den Ausgangstext zu erhalten.

Im Rahmen dieser Arbeit im Modul „Programmierung in Matlab/Octave“ an der Berner Fachhochschule erstellte der Autor einen QR-Code-Reader mithilfe von Matlab. Daraus entstand einerseits eine Matlab-Anwendung mit GUI, andererseits der vorliegende Bericht. Der Bericht verschafft dem Leser einen Überblick über die realisierte Anwendung und führt in die Thematik des Lesens von QR-Codes ein. Zudem wird das Vorgehen des erstellten QR-Code-Readers Schritt-für-Schritt wiedergegeben.

Als Resultat der Arbeit liegt eine Matlab-Anwendung vor, welche QR-Codes der Version 1 bis Version 5 lesen kann. Die zu lesenden QR-Codes müssen von einem QR-Code-Generator (z.B. Webseite) erstellt werden. Sie dürfen sich jedoch irgendwo im Bild befindet und auch aus irgendwelchen kontrastreichen Farben (nicht zwingend schwarz und weiss) bestehen. Der Benutzer kann über ein GUI die Bilddatei des QR-Codes angeben und dann den Leseprozess starten. Die Matlab Anwendung liest den QR-Code in einem Verfahren aus 9 Schritten ein und stellt die Ergebnisse der jeweiligen Schritte im GUI dar. Beim letzten Schritt wird dem Benutzer der im QR-Code gespeicherte und nach ISO-8859-1 dekodierte Text angezeigt.

Inhaltsverzeichnis

1	Einleitung	4
2	Vorgehen	5
	2.1 Zeitplan (Soll-Ist-Vergleich)	5
	2.2 Eingesetzte Tools & Technologien	5
3	Angewendete Methoden & Konzepte	6
	3.1 Übersicht	6
	3.2 Inhalt des QR-Codes auslesen	6
	3.2.1 Übersicht	6
	3.2.2 Schritt 1 – QR-Code-Bild in Graustufen- und Binary-Bild konvertieren	7
	3.2.3 Schritt 2 – Begrenzungsmuster suchen und Bild zuschneiden	7
	3.2.4 Schritt 3 – QR-Code-Version berechnen	8
	3.2.5 Schritt 4 – Format-Infos auslesen	9
	3.2.6 Schritt 5 – Angewendete Maske berechnen	10
	3.2.7 Schritt 6 – Ausrichtungsmuster suchen und rot einfärben	11
	3.2.8 Schritt 7 – Daten (Binär) auslesen	12
	3.2.9 Schritt 8 – Zeichensatz und Textlänge ermitteln	12
	3.2.10 Schritt 9 – Daten nach ISO 8859-1 konvertieren und Text ausgeben	13
	3.3 Erstellung des GUIs	13
4	Ergebnisse	14
	4.1 User Interface	14
	4.2 Anforderungsabdeckung der realisierten Anwendung	14
5	Tabellenverzeichnis	15
6	Abbildungsverzeichnis	15
7	Literaturverzeichnis	15
8	Versionskontrolle	16

1 Einleitung

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) [1][1].

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 dem Tabellen-, Abbildungs- und Literaturverzeichnis abgeschlossen.

[1] Der Quellenverweis [1] und alle weiteren sind im Literaturverzeichnis nachzuschlagen.

2 Vorgehen

2.1 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 2.12	KW 50 9.12	KW 51 16.12	KW 52 23.12	KW 53 30.12	KW 01 6.1	KW 02 13.1	KW 03 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)

2.2 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

3 Angewendete Methoden & Konzepte

3.1 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.

3.2 Inhalt des QR-Codes auslesen

3.2.1 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 berechnet die Anwendung aus der Grösse der Begrenzungsmuster die Grösse eines einzelnen Moduls. 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 unterhalb des Begrenzungsmusters oben links Dieser Block wird für die Berechnung der angewendeten Maske verwendet.
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. An den Stellen der Ausrichtungsmuster sind keine Daten vorhanden, daher muss der QR-Code-Reader über Kenntnis derer Positionen verfügen.
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 des hinterlegten Texts.
9. Wandelt den Text-Teil des bei Schritt 7 gelesenen binären Strings mit dem bei Schritt 8 gelesenen Zeichensatz (Nur ISO 8859-1 unterstützt) um. Dies ist 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) [2].

3.2.2 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 [3].

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 [4].

3.2.3 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 Begrenzungsmustern (Finder Patterns) des QR-Codes.

Abbildung 3: QR-Code mit Grün-markierten Begrenzungsmustern

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 [5].

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 [6].

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".

3.2.4 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 [1].

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.

Die Version des QR-Codes wird mit folgender Formel berechnet:

$$Version = 1 + \frac{Anzahl\ Module\ pro\ Seitenlänge - 21}{4}$$

Wobei „Anzahl Module pro Seitenlänge" 21, 25, 29, etc. ist.

Der Code für die Berechnung der Versionsnummer befindet sich in der Funktion „calculateQrCodeVersion" in der Datei „calculateQrCodeVersion.m"

3.2.5 Schritt 4 – Format-Infos auslesen

Im nächsten Schritt liest der QR-Code-Reader die Format-Infos aus dem QR-Code. Aus dieser binären Zahl kann in Schritt 5 die Maske errechnet werden, welche bei der Erstellung des QR-Codes über den QR-Code gelegt wurde. Mit dieser Maske werden dann die Module demaskiert, um den schlussendlichen Wert (1 für Schwarz, 0 für Weiss) eines Moduls zu erhalten (Mehr dazu, siehe Kapitel 0).
Der Format-String, d.h. eigentlich müsste dieser Format-Error-Correction-String heissen, handelt es sich um einen 2-Fach in jedem QR-Code vorkommenden Bereich. Die beiden Bereiche enthalten dieselbe Information und sind in nachfolgendem Bild (linker Ausschnitt) rot markiert.
Der Bereich 1 beginnt unterhalb des linken oberen Begrenzungsmuster (Abstand zu Begrenzungsmuster beträgt eine Zeile) und geht dann rechts vom oberen linken Begrenzungsmuster wieder hoch. Er ist 15 Module lang. Der zweite Bereich beginnt beim unteren linken Begrenzungsmuster von unten nach oben und wird dann beim oberen rechten Begrenzungsmuster von links nach rechts fortgesetzt [5].

Dieser Format-Error-Correction-String besteht, wie im rechten Bereich des Bildes sichtbar, aus den 3 Teilbereichen „Error correction level“, „Mask pattern“ und „Format error correction.“ Da aufgrund von zu grossem Aufwand in der realisierten Anwendung auf eine Fehlerkorrektur verzichtet wurde, ist nur der Mask-Pattern-Bereich relevant, d.h. die 3 grün markierten Module.

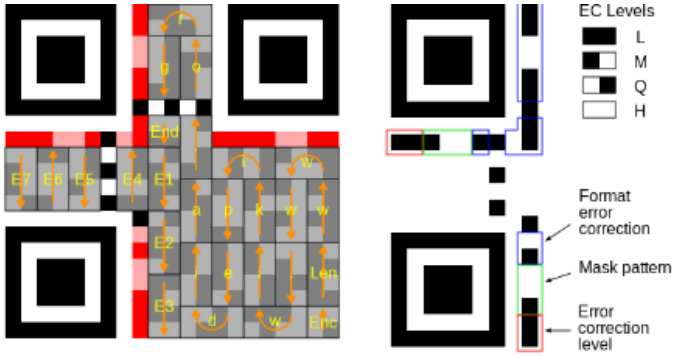

Abbildung 4: Format-Infos (Error correction, Mask pattern und Format error correction) [5]

Die realisierte Matlab-Anwendung ermittelt die Werte der ersten 5 Module (Error correction level und Mask pattern) im oberen linken Format-Bereich. Dazu errechnet sie durch Kenntnis der Pixelgrösse eines Moduls und der Tatsache, dass sich die Format-Infos in der 9.Zeile des QR-Codes beginnen, die X-Y-Koordinaten der 5 Module und fragt im Binärbild die entsprechenden Farbwerte ab.
Ist der Farbwert eines Moduls 0, so handelt es sich um ein schwarzes Modul, sonnt um ein weisses. Schwarze Module führen dann zu einer 1 im binären String, weisse zu einem 0.

Der QR-Code im rechten Bereich des obenstehenden Bilds würde daher der Format-String 11100 ergeben (Zuerst 3 schwarze Module, dann 2 weisse).

Der Code für das Auslesen der Format-Infos befindet sich in der Funktion „readFormatInfoAsBinary“ in der Datei „readFormatInfoAsBinary.m“

3.2.6 Schritt 5 – Angewendete Maske berechnen

Um nun aus dem in Schritt 5 erlangten Format-String den Maskenwert zu errechnen, wird in einem ersten Schritt folgende Berechnung durchgeführt:

$$Binärer\ Format\ String \quad XOR \quad 10101$$

Beim Format-String „11100“ XOR „10101“ würde sich „01000“ ergeben. Um die angewendete Maske zu ermitteln, sind nur die 3 letzten Zeichen, in diesem Fall „000“ relevant. Diese binäre Zahl wird dann in eine Dezimalzahl, hier 0, umgerechnet. Bei diesem QR-Code wurde also die Maske 0 angewendet.

Total existieren 8 verschiedene Masken (Nummer 0 – 7). Bei der Erstellung eines QR-Codes wird eine dieser 8 Masken über den Ursprungs-QR-Code gelegt. Welche Maske zur Anwendung kommt, hängt mit dem Aussehen des Ursprungs-QR-Code zusammen. Das Ziel dieser Maskierung ist, grosse einfarbige Flächen im QR-Code zu verhindern. Ein QR-Code sollte möglichst „unruhig“ sein und eine grosse Abwechslung von schwarzen und weissen Modulen aufweisen. Welche Maske dass bei der Erstellung eines QR-Codes angewendet ist, lässt sich aus 4 verschiedenen Penalty-Regeln errechnen (Mehr dazu, siehe Quelle [9]). Dieser Penalty-Wert wird für jede der 8 Masken berechnet und schlussendlich die Maske mit dem tiefsten Penalty-Wert angewendet.

Mit der Maske wird eine Art „Folie“ über den QR-Code gelegt. Jedes Modul, welches in der Maske Schwarz ist, wird umgedreht (d.h. Weiss wird zu Schwarz und umgekehrt). Bei den weissen Modulen in der Maske wird der Modulwert beibehalten.

Nachfolgende Abbildung zeigt die 8 Masken.

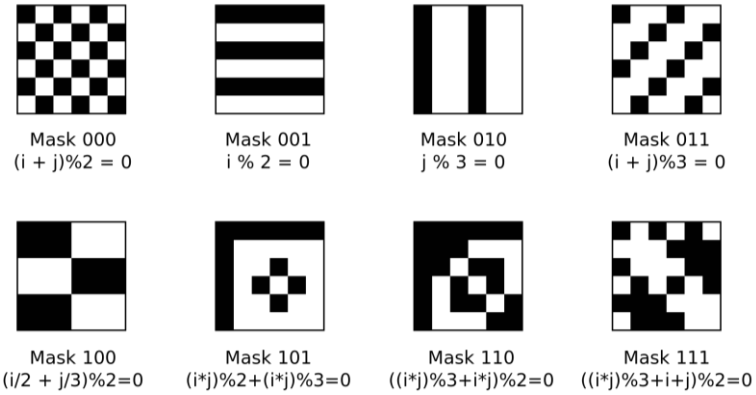

Abbildung 5: Mögliche Masken [7]

Um für ein spezifisches Modul zu errechnen, ob dieses bei einer spezifischen Maske umgedreht werden muss oder nicht, liegt jeder Maske eine Formel zu Grunde [8]. Diese ist in der obenstehenden Abbildung unterhalb der jeweiligen Maske zu sehen.

Dabei sind:

i: Spalte des Moduls, von links nach rechts, beginnend mit 0
j: Zeile des Moduls, von oben nach unten, beginnend bei 0

Ergibt die Berechnung für ein Modul 0, so wird das Modul umgekehrt.

Maske 1 (001) für das Modul (Feld) im rechten unteren Ecken bei einem QR-Code der Grösse 21 x 21 Module würde folgende Berechnung ergeben (i = 20, j = 20):

$$20\ \%\ 2 = 0 \quad \rightarrow Modul\ würde\ umgedreht\ werden.$$

Der Code für die Berechnung der angewendeten Maske ist in der Funktion „calculateMask“ in der Datei „calculateMask.m“ zu finden. Die Anwendung der Masken-Formel für die einzelnen Module folgt bei Schritt 7.

3.2.7 Schritt 6 – Ausrichtungsmuster suchen und rot einfärben

Bevor die Matlab-Anwendung bei Schritt 7 die Daten auslesen kann, muss noch der letzte Bereich des QR-Codes, welcher keine relevanten Daten beinhaltet, ermittelt werden. Dies sind die sogenannten Ausrichtungsmuster. Diese sind erst ab QR-Codes der Version 2 vorhanden und dienen dazu, die Ausrichtung des QR-Codes besser zu bestimmen.

Nachfolgende Tabelle zeigt die Anzahl Ausrichtungsmuster in den verschiedenen QR-Code-Versionen:

Version des QR-Codes	Anzahl Ausrichtungsmuster
1	0
2 – 6	1
7 – 13	6
14 – 20	13
21 – 27	22
28 – 34	33
35 – 40	46

Tabelle 3 Anzahl Ausrichtungsmuster in verschiedenen QR-Code-Versionen

Die Ausrichtungsmuster sind vom Aufbau her ähnlich wie die Begrenzungsmuster. Sie messen 5 x 5 Module und sind nach dem Schwarz-Weiss Verhältnis 1:1:1:1:1 aufgebaut.

Die umgesetzte Matlab-Anwendung sucht die Ausrichtungsmuster nach demselben Prinzip wie die Begrenzungsmuster (siehe Kapitel 3.2.3). Mit Hilfe von „Connected-component labeling“ werden zusammenhängende Bereiche gesucht. Die Grösse der Begrenzungsmuster ist aus Schritt 2 bekannt, d.h. als Kandidaten für ein Ausrichtungsmuster kommen nur zusammenhängende Bereiche in Frage, wessen Fläche kleiner ist als die der Begrenzungsmuster.

Alle als Ausrichtungsmuster in Frage kommenden Bereich werden zusätzlich der Prüfung unterzogen, ob sie dem Schwarz-Weiss Verhältnis von 1:1:1:1:1 entsprechen. Ist dies der Fall, so handelt es sich definitiv um ein Ausrichtungsmuster.

Die Matlab-Anwendung erstellt nun aus dem Binärbild ein RGB-Bild mit schwarzen und weissen Modulen. Alle Ausrichtungsmuster werden rot eingefärbt. Beim Lesen der Daten (Schritt 7) kann der QR-Code-Reader aufgrund der Modulfarbe ermitteln, ob das Modul Daten enthält (Schwarz oder Weiss) oder ob es sich um ein Ausrichtungsmuster ohne Daten handelt.

Der Code für das Finden und Einfärben der Ausrichtungsmuster befindet sich in der Funktion „findAndColorizeAlignmentPatterns“ in der Datei „findAndColorizeAlignmentPatterns.m“.

3.2.8 Schritt 7 – Daten (Binär) auslesen

Beim Schritt 7 geht es nun darum, unter Anwendung der aus Schritt 1-6 erlangten Informationen (Begrenzungsmuster, Maske, Ausrichtungsmuster, etc.) die Daten aus dem QR-Code zu lesen.

Die Anwendung liest die Daten von unten rechts beginnend, wie in den Pfeilen der nachfolgenden Abbildung dargestellt, aus. Gelesen werden immer 2 Spalten beim Hochgehen und wieder 2 Spalten beim Runtergehen. Innerhalb eines Hoch- oder Runter-Prozesses werden die Module in der in der rechten Abbildung dargestellten Reihenfolge gelesen.

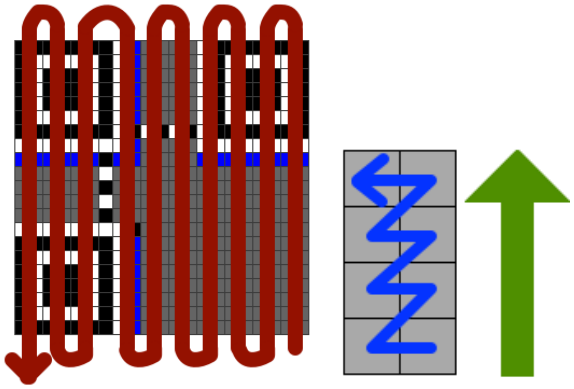

Abbildung 6: Vorgehen beim Auslesen der Daten [9]

Gelesen werden jedoch nur die Module, bei welchen es sich nicht um einen Teil eines Begrenzungs- oder Ausrichtungsmuster, um den Bereich mit den Format-Error-Informationen oder um das Fixed-Pattern handelt (Alle in der Abbildung 7 nicht rot markierten Bereiche). Das Fixed-Pattern ist die Verbindungslinie zwischen den 3 Begrenzungsmustern. Diese beinhaltet auch keine Daten.

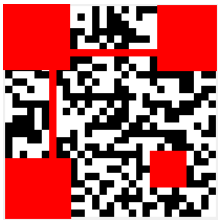

Abbildung 7 : Bereiche im QR-Code ohne Daten

In der Matlab-Anwendung wurden 2 Funktionen geschrieben, eine Funktion für das Aufwärtslesen, die andere Funktion für das Abwärtslesen. Ausgelesen wird für jedes Modul dessen Farbwert. Handelt es sich um ein rotes Modul (nur Ausrichtungsmuster sind im analysierten RGB-Bild rot markiert), so wird dieses übersprungen. Sonst wird der Farbwert ausgelesen und dieser Farbwert mit der in Kapitel 3.2.6 erläuterten Masken-Formel demaskiert. Der erhaltene Wert wird in 1 (Schwarz) oder 0 (Weiss) umgewandelt und so von unten rechts nach oben links der binäre Daten-String erstellt.

Der Code für das Auslesen und Demaskieren der Daten befindet sich in der Datei „readData.m“.

3.2.9 Schritt 8 – Zeichensatz und Textlänge ermitteln

Der in Schritt 7 ermittelte binäre Daten-String beinhaltet nicht nur die binären Daten vom gespeicherten Text, sondern auch noch Informationen über die Text-Länge und über den Zeichensatz.

Die 4 ersten Bits (die 4 zuerst gelesenen Module unten rechts) geben Informationen über den Zeichensatz des im QR-Code gespeicherten Texts. Es gibt 4 verschiedene Zeichensätze, die durch folgende binären Werte gekennzeichnet sind:

Binärer Wert	Zeichensatz
0001	Nur numerische Werte (10 Bits für 3 Zeichen)
0010	Nur alphanumerische Grossbuchstaben (11 Bits für 2 Zeichen)
0100	ISO 8859-1 (8 Bits pro Zeichen)
1000	Kanji Kodierung (Japanische Zeichen, 13 Bits pro Zeichen)

Tabelle 4: Zeichensätze für die Dekodierung von QR-Codes [5]

Der entwickelte QR-Code Reader kann nur QR-Codes nach ISO 8859-1 verarbeiten. Ermittelt die Anwendung in diesem Schritt einen anderen Zeichensatz, wird dies dem Benutzer auf dem GUI mitgeteilt und der QR-Code nicht weiterverarbeitet. Ein Grossteil der verbreiteten QR-Codes sind mit ISO 8859-1 kodiert, denn damit sind Buchstaben, Zahlen, Sonderzeichen und Umlaute möglich.

Nach den ersten 4 Bits folgen 8 Bits, wessen Dezimalwert Auskunft über die Länge des gespeicherten Texts gibt.

Der Zeichensatz wird in der Funktion „readFormatInfoAsBinary“ ermittelt, die Länge des gespeicherten Texts in der Funktion „convertToIso“ in der Datei „convertToIso.m“.

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

Zu guter Letzt wandelt der QR-Code-Reader nun die verbleibenden Bits des Datenstrings (ab Position 13) nach ISO 8859-1 um. Dazu wandelt er immer 8 Bits in eine Dezimalzahl um. Diese Dezimalzahlen werden in einer Matrix abgelegt und danach mit der Matlab-Funktion „strcat“ einerseits aneinander gekettet, andererseits in den Zahlen-, Buchstaben-, oder Sonderzeichen-Wert nach ISO 8859-1 umgewandelt.

Die Umwandlung nach ISO 8859-1 geschieht in der Funktion „convertToIso“ in der Datei „convertToIso.m“.

3.3 Erstellung des GUIs

Nach der Fertigstellung des Leseprozesses hat sich der Autor dieser Arbeit entschieden, die Matlab-Anwendung um ein GUI zu erweitern, welches dem Benutzer Schritt für Schritt den Leseprozess des QR-Codes aufzeigt. Diese Idee ist hauptsächlich entstanden, weil im Internet nur sehr wenige QR-Code-Reader verfügbar sind, welche den Benutzer Informationen zu den Lese-Schritten liefern. Andererseits wollte eine Anwendung geschaffen werden, die bei Demonstrationen gut und verständlich den Aufbau von QR-Codes aufzeigen kann.

Matlab R2015a bietet bereits in der Grundausstattung den GUI-Editor „GUIDE“ zur einfachen Erstellung von GUIs an. Dieser ermöglicht, aus ungefähr 10 GUI-Komponenten, ein GUI zusammen zu klicken.
Das erstellte GUI ist im Kapitel 0 zu betrachten.

Dem erstellten GUIDE-Formular ist eine Matlab-Datei hinterlegt (gui.m). In dieser werden die Aktionen der GUI-Elemente programmiert, beispielsweise was beim Laden der Ansicht oder bei einem Klick auf einen Button, geschehen soll. Umgesetzt sind diese als Callback-Funktionen.

Das erstellte GUI führt bei einem Klick auf den Button „QR-Code auslesen“ nacheinander die 9 Schritte zum Einlesen des QR-Codes aus. Nach jedem Schritt wird das Ergebnis des jeweiligen Schritts im GUI dargestellt. Die farbigen Kästchen zur Markierung von Bereichen im QR-Code, oder auch die Texte in den Bildern (z.B. Spalten/Zeilen oder Textkodierung) wurden mit den Matlab-Befehlen „text“ und „rectangle“ umgesetzt, d.h. diese befinden sich nicht im QR-Code-Bild, sondern wurden im GUI darüber gelegt.

4 Ergebnisse

4.1 User Interface

Nachfolgende Abbildung zeigt das User Interface der erstellten Anwendung. Im Eingabefeld kann der Benutzer den Pfad zur QR-Code-Bilddatei angeben. Zudem kann er über den Button „Datei auswählen“ eine Datei aus dem Dateisystem auswählen. Dazu steht ihm ein klassischer FileOpen-Dialog zur Verfügung. Sobald der Benutzer auf den Button „QR-Code auslesen“ klickt, beginnt die Matlab-Anwendung mit dem Auslesen des QR-Codes und stellt nach Abschluss jedes Schritts dessen Ergebnis im GUI dar.
Hat der Benutzer keine Bilddatei ausgewählt oder ist diese nicht im PNG-Format, so wird beim Klick auf „QR-Code auslesen“ eine entsprechende Fehlermeldung angezeigt und der Leseprozess nicht gestartet.

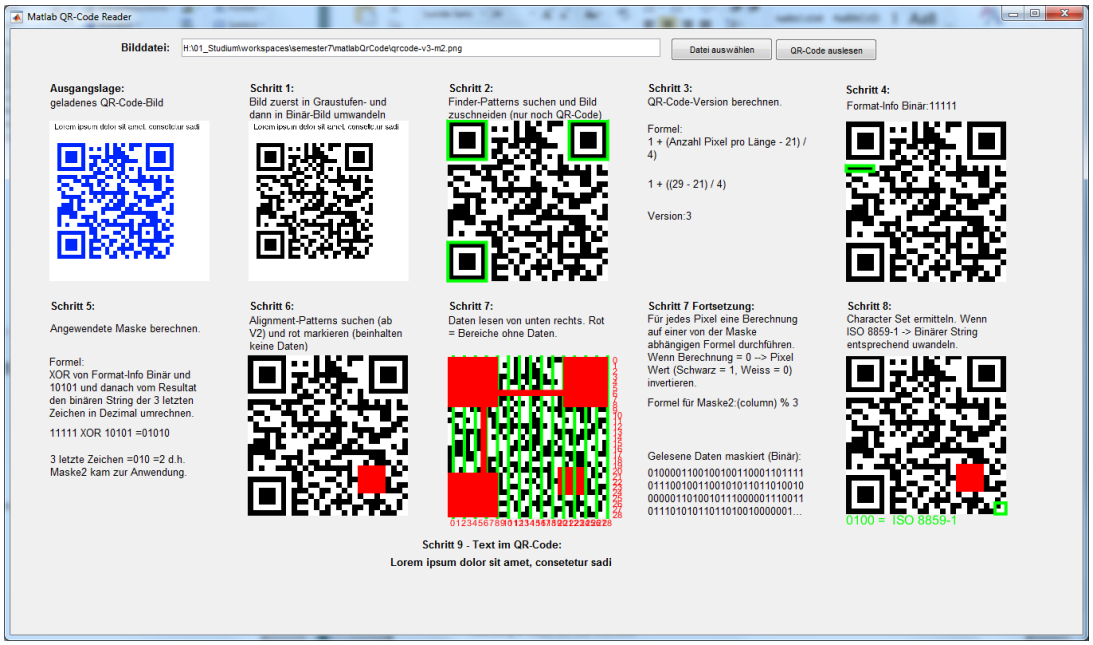

Abbildung 8: User Interface der erstellten Anwendung

4.2 Anforderungsabdeckung der realisierten Anwendung

Anforderung	Abdeckung
Angabe des Bildpfads	Wurde umgesetzt. Einerseits ist die Angabe des Bildpfads möglich, andererseits kann das Bild direkt mit einem FileOpen-Dialog aus dem Dateisystem ausgewählt werden.
Erkennung des QR-Codes im Bild	Wurde umgesetzt. Die Anwendung erkennt den QR-Code, sofern er sich irgendwo im Bild befindet und kann das Bild auf die QR-Code-Grösse zuschneiden.
Extraktion des Inhalts	Wurde umgesetzt. Der QR-Code-Reader kann QR-Codes mit der Kodierung ISO-8859-1 auslesen.
Unterstützung verschiedener Version von QR-Codes	Wurde teilweise umgesetzt. Der QR-Code-Reader kann nur Version 1-5 verarbeiten. Ab Version 6 tritt beim Lesen ein Fehler auf, wessen Ursache auch nach mehrtägiger Fehlersuche nicht ermittelt werden konnte.

Tabelle 5: Anforderungsabdeckung der realisierten Anwendung

Zusätzlich zu den definierten Anforderungen wurde das GUI zur Bedienung der Anwendung erstellt, welches dem Benutzer den Leseprozess Schritt für Schritt ersichtlich macht.

5 Tabellenverzeichnis

Tabelle 1 Zeitplan (Soll-Ist-Vergleich)....5
Tabelle 2 Eingesetzte Tools & Technologien....5
Tabelle 3 Anzahl Ausrichtungsmuster in verschiedenen QR-Code-Versionen....11
Tabelle 4: Zeichensätze für die Dekodierung von QR-Codes [5]....12
Tabelle 5: Anforderungsabdeckung der realisierten Anwendung....14

6 Abbildungsverzeichnis

Abbildung 1: QR-Code mit farblicher Darstellung der verschiedenen Muster (Patterns) [2]....6
Abbildung 2: Graustufen- und Binär-Bild von einem QR-Code....7
Abbildung 3: QR-Code mit Grün-markierten Begrenzungsmustern....7
Abbildung 4: Format-Infos (Error correction, Mask pattern und Format error correction) [5]....9
Abbildung 5: Mögliche Masken [7]....10
Abbildung 6: Vorgehen beim Auslesen der Daten [9]....12
Abbildung 7 : Bereiche im QR-Code ohne Daten....12
Abbildung 8: User Interface der erstellten Anwendung....14

7 Literaturverzeichnis

[1] Information capacity and versions of the QR Code
Webseite qrcode.com, Stand 25.11.2015
http://www.qrcode.com/en/about/version.html 4

[2] QR Codes
Webseite swisseduc.ch - Informatik, Stand 25.11.2015
http://www.swisseduc.ch/informatik/theoretische_informatik/qr_codes/docs/unterlagen_lernende.pdf 6

[3] Schwellenwertverfahren
Wikipedia, Stand 09.04.2015
https://de.wikipedia.org/wiki/Schwellenwertverfahren 7

[4] im2bw
MathWorks Dokumentation, Stand 09.12.2015
http://ch.mathworks.com/help/images/ref/im2bw.html 7

[5] QR code
Wikipedia, Stand 12.01.2016
https://en.wikipedia.org/wiki/QR_code 7

[6] Connected-component labelling
Wikipedia, Stand 25.08.2015
http://www.thonky.com/qr-code-tutorial/data-masking 7

[7] Bild zu den QR-Code-Masken
Wikimedia, Stand 12.01.2016
https://commons.wikimedia.org/wiki/File:QR_Code_Mask_Patterns.svg 10

[8] QR Mask Patterns Explained
Webseite Thonky, Stand 12.05.2015
http://www.thonky.com/qr-code-tutorial/mask-patterns 10

[9] Data Masking
Webseite Thonky, Stand 12.05.2015
http://www.thonky.com/qr-code-tutorial/data-masking

8 Versionskontrolle

Version	Datum	Beschreibung	Autor
1.0	23.01.2016	Dokument finalisiert und freigegeben.	Joel Holzer