Der QR-Code

Inhaltsverzeichnis

1. Vorwort

Seit längerer Zeit sind die kleinen Pixelquadrate allgegenwärtig, ob auf Tickets der Bahn oder bei Werbeanzeigen. Inzwischen ist sogar geplant solche QR-Codes auf der Rückseite von QSL-Karten aufzudrucken um die Vermittlung weiter zu automatisieren oder um es möglich zu machen die QSO-Daten des mit der Karte bestätigten QSOs direkt in das eigene Logbuchprogramm einlesen zu können.

Bild 1: Beispiel von QR-Codes auf der Rückseite einer QSL-Karte

2. Historisches

Der QR-Code wurde 1994 von der japanischen Firma „Denso-Wave“ entwickelt. Inzwischen sind die QR-Codes international und in ISO/IEC 18004:2006 genormt. Die Benutzung von QR-Codes ist lizenzfrei.

3. Die Randbedingungen

Der QR-Code sollte deutlich universeller sein als z.B. der Bar-Code. Es sollte möglich sein, QR-Bilder mit einer Kamera aufzunehmen und die Information mit Hilfe eines Analyseprogramms zu dekodieren. Die Dekodierung sollte auch dann noch funktionieren wenn die Randbedingungen wie Beleuchtung, Kontrast, Bildschärfe, Lage zum Bild nicht optimal sind. Je nach Anwendung sollte es möglich sein Nachrichten von ganz unterschiedlicher Länge und Art (von ganz kurzen Datenblöcken bis zu längeren Datensätzen, von reinem Text, reinen Ziffern oder beliebigen Bitmustern) in jeweils dafür optimierten QR-Codes zu verschlüsseln. Parallel dazu sollte auch die

Datensicherheit, also die Zahl erlaubter Lesefehler bei der die Nachricht trotzdem noch fehlerfrei dekodiert werden kann wählbar sein.

3.1. Das Datenvolumen

Abhängig von der Größe der QR-Struktur (also der Zeilenanzahl) und der gewünschten Datensicherheit lassen sich in einem QR-Quadrat bis zu 7089 Ziffern oder 4296 Buchstaben/Ziffern unterbringen.

3.2. Die Sicherheit der Daten

Für die Sicherung der Daten gibt es vier verschiedene Stufen die mit Buchstaben bezeichnet werden:

L: Die Originaldaten können bei einer Fehlerrate von 7% noch rekonstruiert werden.

M: Die Originaldaten können bei einer Fehlerrate von 15% noch rekonstruiert werden.

Q: Die Originaldaten können bei einer Fehlerrate von 25% noch rekonstruiert werden.

H: Die Originaldaten können bei einer Fehlerrate von 30% noch rekonstruiert werden.

Abhängig von der gewählten Datensicherheit wird ein bestimmter Anteil des Platzes im QR-Quadrat durch die Datensicherungsinformation belegt.

3.3. Die optischen Probleme

Je nach Kameraposition und Beleuchtung kann es zu folgenden durch die Optik bedingten Problemen kommen:

- Je nach Entfernung und Brennweite der Kamera können sich die Bildgrößen stark unterscheiden
- Das Bild kann gegenüber der optimalen Lage in allen Richtungen gedreht sein, ja sogar auf dem Kopf stehen.
- Es kann zu perspektivischen Verzerrungen kommen.
- Spiegelungen können dazu führen dass Teile des Bildes nicht erkennbar sind.
- Es kann passieren dass Teile des Bildes durch andere Objekte abgedeckt sind.
- Es kann passieren dass Teile des Bildes unscharf sind.

4. Die Geometrie

4.1. Die Größe

Wie an dem Bild 1 zu erkennen ist, bestehen QR-Codes aus mit quadradischen Punkten versehenen

Quadraten. Die Kantenlänge kann von minimal 21 Punkten bis maximal 177 Punkten gehen. Diese verschiedenen Varianten werden „Versionen“ genannt wobei es insgesamt 40 Versionen gibt. Die Kantenlänge (also die Zahl von Zeilen bzw. Spalten) ergibt ich zu:

$$Seitenlänge = (version - 1) * 4 + 21;$$

Nicht mitgerechnet ist dabei der weiße Rand um das Quadrat, der nötig ist damit die Decodiersoftware die Größe des Bildes sicher bestimmen kann.

4.2 Figuren zur Bestimmung der Lage

Ganz auffällig sind die drei Figuren bestehend aus ineinander geschachtelten Quadraten links oben, rechts oben und links unten. Sie sind auch deshalb so auffällig weil sie jeweils einen weißen Rad besitzen, also einen Bereich in dem es keine schwarzen Punkte gibt. Die Erkennungssoftware muss das Bild rechnerintern so drehen, dass diese Figuren oben und links unten stehen. Gelingt das nicht, ist das Bild spiegelverkehrt.

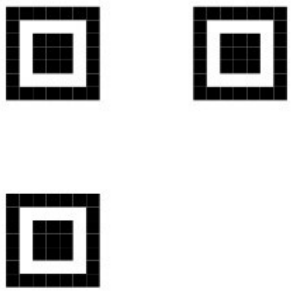

Bild 2: Quadrate zur Bestimmung der Lage

4.3 Weitere Informationen zur Bestimmung der Bildgeometrie

Ab der Version 2 gibt es noch weitere, weit kleinere Quadrate mit einem schwarzen Punkt in der Mitte die gleichmäßig über das Bild verteilt sind. Sie sind nicht ganz so auffällig weil sie im Gegensatz zu den drei oben beschriebenen Quadraten keinen weißen Rand haben. Mit Hilfe dieser kleinen Quadrate lassen sich perspektifische Verzerrungen des Gesamtbildes erkennen und entspechend korrigieren.

Nicht direkt zu erkennen sind zwei weitere fixe Bestandteile im Bild: Zwischen den beiden oberen Quadraten gibt es eine Zeile in der sich die Punkte in der Folge weiß-schwarz-weiß.....schwarz-weiß abwechseln. Zwischen dem Quadrat links oben und links unten gibt es eine ebensolche Spalte. Dieser schwarz/weiß-Wechsel hilft der Dekodiersoftware die Zeilen und Spaltenabstände zu bestimmen um jedem Punkt im Bild die richtige Position zuordnen zu können.

Schließlich gibt es noch einen einzelnen fixen schwarzen Punkt etwas rechts über dem linken unteren Quadrat dessen Sinn unklar ist.

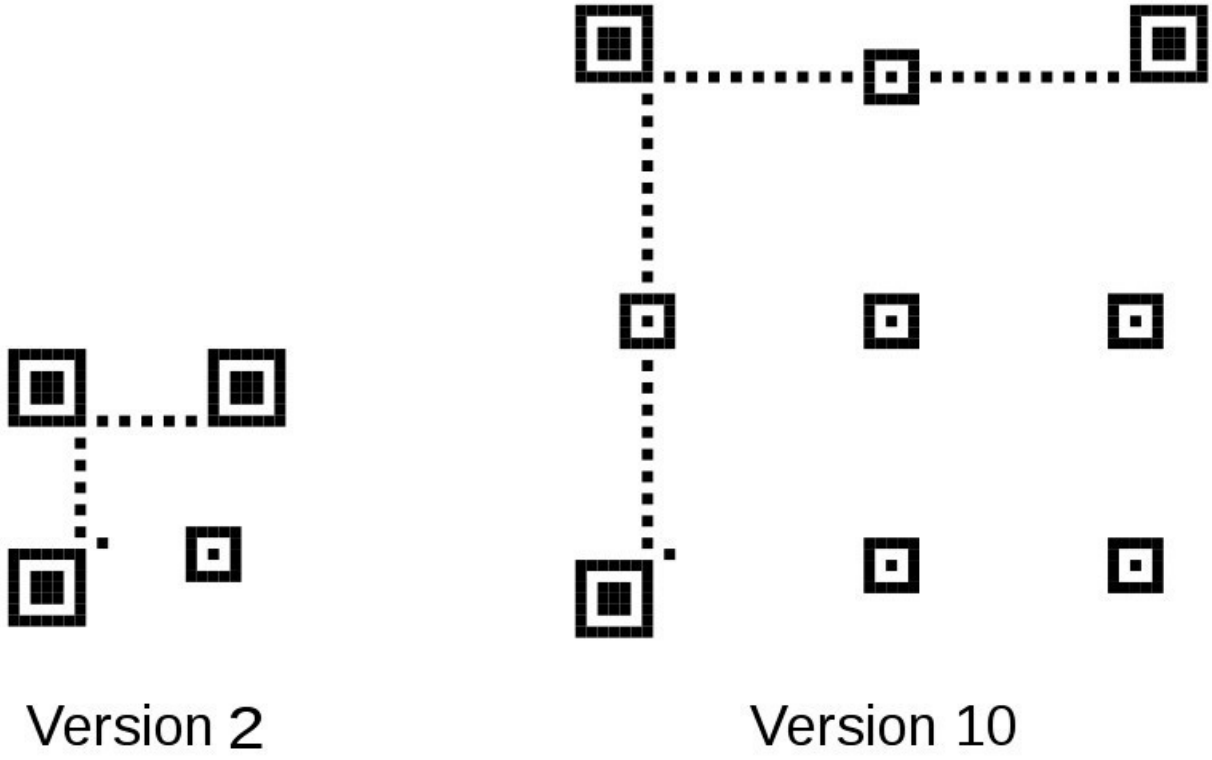

Bild 3: Feste Anteile in den QR-Quadraten

4.4 Bereich zur Beschreibung der Kodierungsart (Formatinfo)

Damit die Dekodierungsprogramme mitkriegen was es da zu dekodieren gibt, wurden in dem QR-Quadrat zwei Bereiche für eine 15-Bit lange Formatinformation reserviert. Nachdem ohne diese Information keine Dekodierung möglich ist, ist die Formatinformation mit einer Menge Sicherungsinformation versehen und im QR-Code doppelt hinterlegt. Ebenso sind ab Version 7 zwei weitere Bereiche für die Versionsinformation (grün) reserviert.

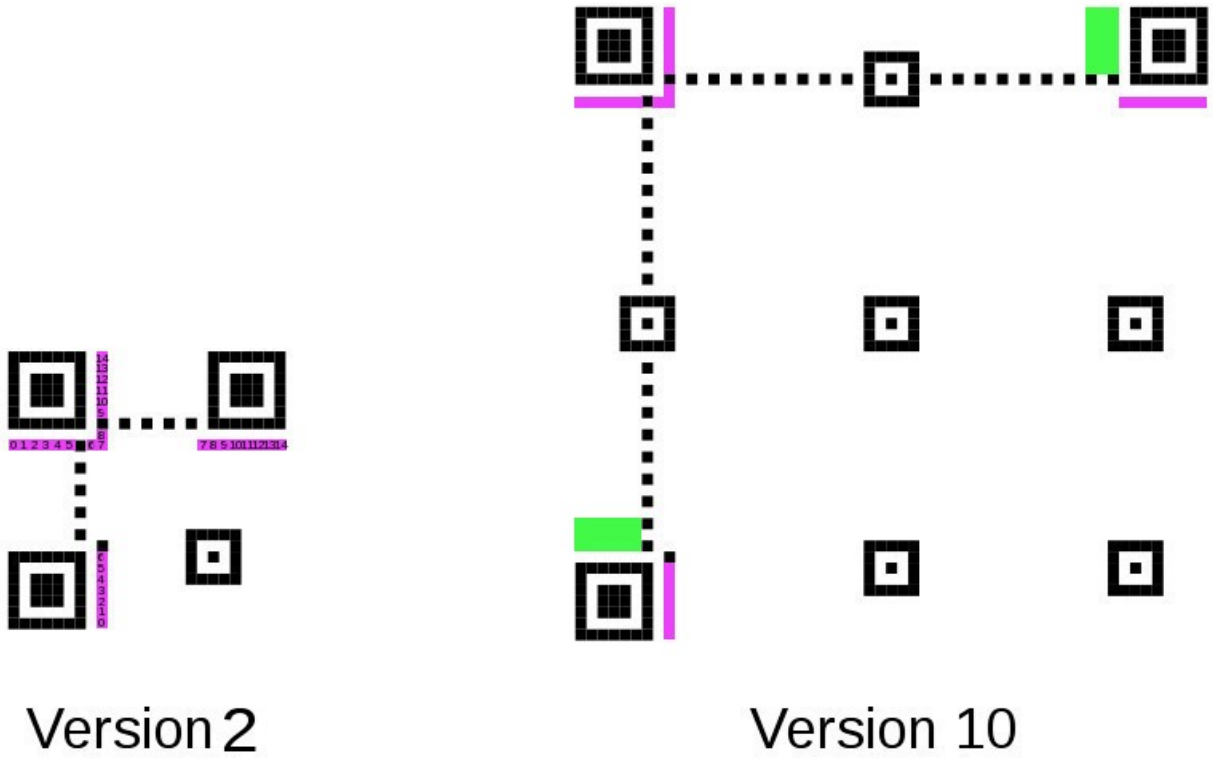

Bild 4: Die für die Formatinformation (rot) und Versionsinformation (grün) reservierten Bereiche

5. Aufbereitung der Daten

Im ersten Schritt werden die Daten – wenn möglich – möglichst effektiv komprimiert. Dazu wird erst einmal untersucht woraus diese Daten bestehen:

- Nur Ziffern,

- Nur die Großbuchstaben A...Z, die Ziffern 0...9 und die Zeichen $,%,*,.,+,-,/,: und Leerzeichen

- beliebige Bitmuster, wobei diese als ASCII-Zeichen interpretiert werden.

5.1. Komprimierung von Ziffern

Um einzelne Ziffern (0 … 9) binär zu kodieren braucht man 4 Bits. Würde man die Kodierungen von drei Ziffern einfach hintereinandersetzen, so würde man dafür 3x4=12 Bits verbrauchen. Weit effektiver ist es statt dessen die Ziffern in Dreiergruppen zusammenzufassen und dann als Binärzahl zu kodieren. In dem Fall kommt man dann mit 10 Bit anstatt mit 12 Bit aus. Bleiben dann am Ende noch zwei Ziffern über, werden die als 7Bit-Zahl angehängt, eine einzelne Ziffer am Ende lässt ich schließlich nicht mehr komprimieren und muss deshalb als 4Bit-Zahl angehängt werden.

5.2 Komprimieren der Daten von normalen Texten

Bestehen die auszugebenden Daten nur aus Großbuchstaben, den Ziffern und den oben vorgestellten Satzzeichen wird wie hier beschrieben komprimiert: Die Zeichen werden in der unten gezeigten Liste aufgereiht und mit einer Ordnungsnummer 0 bis 44 versehen. Die zu kodierenden Zeichen werden paarweise zusammengefasst. Die Ordnungsnummer für das erste Zeichen wird mit der Zahl 45 multipliziert und die der zweiten aufaddiert. Auf diese Weise lässt sich jedes Zeichenpaar in 11 Bit zuammenfassen. Die 11-Bit-Blöcke werden dann einfach hintereinandergesetzt

Zeichen	Wert		Zeichen	Wert
0	0		N	23
1	1		O	24
2	2		P	25
3	3		Q	26
4	4		R	27
5	5		S	28
6	6		T	29
7	7		U	30
8	8		V	31
9	9		W	32
A	10		X	33
B	11		Y	34
C	12		Z	35
D	13		Leerzeichen	36
E	14		$	37
F	15		%	48
G	16		*	39
H	17		+	40
I	18		-	41
J	19		.	42
K	20		/	43
L	21		:	44
M	22			

Tabelle 1: Zuordnung von Zeichen zur zahlenmäßigen Kodierung

Beispiel:

Soll die Buchstabenfolge HE kodiert werden, so ergibt sich für das Zeichen H die Zahl 17, für das Zeichen E die Zahl 14. Das Buchstabenpäärchen HE hat entspricht damit der Zahl

45 * 17 + 14 = 779

Hat man einen längeren Text (wie z.B. „HELLO WORLD“), sieht man die aus den Buchstabenpaaren errechneten Zahlen als 11-Bit lange Bitmuster, die man lückenlos aneinanderreihen kann. Da wird noch ein kleiner Vorspann davorgesetzt in dem steht aus wievielen Zeichen die Nachricht besteht und aus was die kodierten Daten bestehen (also z.B nur aus Ziffern oder wie in unserem Fall Ziffern/Buchstaben). Schließlich werden an die Nachricht mindestens vier Nullen angehängt und das Ergebnis bis zum nächsten vollen Byte mit Nullen aufgefüllt. Ist dieser String für die vorgesehene QR-Version zu kurz, wird der Rest mit einem speziellen Füllmuster aufgefüllt. Diese Reihe kann man dann wiederum in 8-Bit lange Blöcke zerlegen und erhält auf diese Weise eine Folge von Zahlen. Im QR-Code würde der Satz „HELLO WORLD“ letztendlich so aussehen:

32 91 11 120 209 144 120 77 67 64 236 17 236 17 236 17

wobei diese Zahlenreihe mit dem oben schon erwähnten Füllmuster (17 236) auf die für Version 1 und Sicherungstufe H nötigen 16 Byte aufgefüllt wurde.

5.3 Behandlung von nicht komprimierbaren Bytes

Gibt es keine Einschränkung auf bestimmte Zeichen, ist auch keine Komprimierung möglich. In diesem Fall wird mit einer Kennung begonnen die die Daten als „nicht komprimierte Bytes“ identifiziert, danach kommt die Anzahl der Bytes gefolgt von den Bytes selbst. Am Ende werden m (wie in den anderen Fällen auch) mindestens 4 Bits „0“ angehängt dann bis zum vollen Byte mit Nullen aufgefüllt, und schließlich das Füllmuster 17 236 bis zum Ende des möglichen Datenbereiches angehängt.

6. Generierung der Sicherungsinformation

Um die in dem QR-Bild eingetragenen Daten in Fall von Lesefehlern möglichst gut rekonstruieren zu können werden an die Nutzdaten nach dem Reed-Solomon Verfahren erstellte Sicherungsdaten angehängt.

6.1 Datensicherung nach Reed-Solomon

Die Reed-Solomon Datensicherung ist eine Block-Code Sicherung die nicht nur beim QR-Code sondern auch in vielen anderen Fällen (Datensicherung bei Festplatten und CDs oder bei Verfahren wie JT65) verwendet wird. Deshalb soll hier eine kurze Beschreibung der Ideen und deren

Realisierung folgen.

Für die Datensicherung wird etwas benutzt, was man auf den ersten Blick für ein Verfahren zur Sicherung von Daten nicht erwarten würde: Polynome. Ganz kompakt geschrieben sind sie so definiert:

$$y=\sum_{n=0}^{k} a_n x^n;$$

Man kann es auch etwas anders schreiben:

$$y=a_k x^k+a_{k-1}x^{k-1}+\ldots+a_1 x^1+a_0 x^0;$$

Zur genaueren Erläuterung hier ein kleines Beispiel mit konkreten Zahlen:

$$y=-2x^3+50x^2+6x+8; \quad \text{mit} \quad a_3=-2; a_2=50; a_1=6 \text{ und } a_0=8;$$

Setzt man in diese Formel für x beliebige Werte ein, erhält man für jeden x-Wert genau einen y-Wert. Diese Ergebnisse lassen sich auch in einem x-y-Diagramm darstellen.

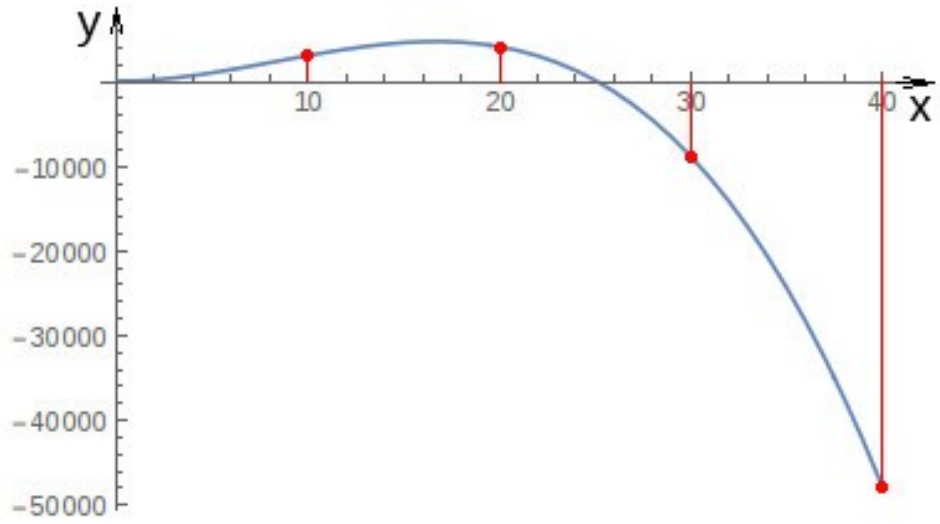

Bild 5: Diagramm zum Beispielpolynom

Nimmt man sich aus diesem Diagramm vier beliebige X/Y-Päärchen heraus (in dem Beispiel sind das 10/3068, 20/4128, 30/-8812, 40/-47752), kann man aus diesen Wertepaaren das ursprüngliche Polynom rekonstruieren. Das gilt nicht nur für dieses Beispiel sondern für alle Polynome:
Wenn zu einem beliebigen Polynom mit dem Grad „k“ „k+1“ sich voneinander unterscheidende Stützstellen bekannt sind, lässt sich das Polynom eindeutig bestimmen. „Eindeutig bestimmen“ soll heißen, dass man die Koeffizienten , in unserem Beispiel die Zahlen -2, 50, 6 und 8 aus den vier Stützstellen berechnen kann.
Zugegeben, die allgemeine Formel zur Berechnung der Koeffizienten sieht etwas verwirrend aus.

$$L(x)=\sum_{j=0}^{k} y_j l_j(x); \qquad l_j:=\prod_{0\le m\le k;\, m\ne j}^{n} \frac{x-x_m}{x_j-x_m}; \text{und den Stützstellen} \qquad (x_0,y_0)\ldots(x_j,y_j)\ldots(x_k,y_k)$$

Diese Formel ist bei den Mathematikern als Lagrange-Interpolation bekannt.

Wenn man sich mehr Stützstellen aussucht als unbedingt nötig, kann man die Datensicherheit erhöhen denn jetzt kann man sich aus der Summe aller Stützstellen eine beliebige, ausreichende Auswahl zusammenstellen und daraus das Originalpoynom erstellen. Probiert man bei den vorhandenen Stützstellen alle Kombinationen durch, wird man im Idealfall in allen Fällen zu den gleichen Koeffizienten kommen. Ist aber eine der Stützstellen nicht korrekt, so werden alle Kombinationen in denen die fehlerhafte Stützstelle auftaucht zu abweichenden Koeffizienten führen. Auf diese Weise lässt sich die fehlerhafte Stützstelle aussortieren.

Aus einem anderen Blickwinkel betrachtet:

Ersetzt man das „x“ in dem der Formel $y=\sum_{n=0}^{k} a_n x^n;$ durch $\sin(\omega t)$ oder die noch allgemeinere Form $e^{j\omega t}$ dann erhält man:

$y(t)=\sum_{n=0}^{k} a_n e^{njwt};$ Diese Formel beschreibt die Fourieranalyse einer Kurve die aus k + 1 harmonischen Vielfachen der Frequenz ω besteht wobei die Koeffizienten der Frequenzanteile die Nachricht bilden. Man kann das Reed-Solomon Verfahren damit auch als eine spezielle Anwendung der Fourrieranalyse sehen.

Zurück zu unserem eigentlichen Problem, nämlich der Frage wie sich eine Sicherung von Daten realisieren lässt. Wir haben oben die Zahlenfolge gesehen die bei der Kodierung der Wörter „HELLO WORLD“ entsteht. Setzt man genau diese Zahlen (aus Kapitel 5.2) in ein Polynom ein kommt man zu folgender Gleichung.

$$y=32x^{15}+91x^{14}+11x^{13}+120x^{12}+209x^{11}+114x^{10}+220x^{9}+77x^{8}+64x^{7}+64x^{6}+236x^{5}+17x^{4}+236x^{3}+17x^{2}+236x+17$$

Sucht man sich bei diesem Polynom 16 Stützstellen heraus, kann man aus diesen Stützstellen das Orignalpolynom und damit die Originalnachricht rekonstruieren. Alle 16 Stützstellen sind für eine Rekonstruktion natürlich nur dann nötig, wenn man das Originalplynom überhaupt nicht kennt. Sind Teile davon bekannt kommt man für die Rekonstruktion der fehlenden Information mit weit weniger Stützstellen aus. Damit wäre ein Verfahren gefunden bei dem man aus Zusatzinformationen (den Stützstellen) das Orginal, also unseren Text rekonstruieren kann. So schön dieses Verfahren der Datensicherung auch aussehen mag, es hat ein gravierendes Problem. Sehen wir uns die ersten 15 Stützstellen an, so kommt man zu folgenden Ergebnissen:

x	y
0	17
1	1764
2	3821213
3	1024575602
4	62599315665
5	1586492169872
6	22667027757869
7	216821853474558
8	1542364478370737
9	8739201459598460
10	41352268458010077
11	169048661233449674
12	612271646214258353
13	200276901730005672
14	6005789811856607405
15	16708020671155111382

Tabelle 2: Stützstellen des Polynoms für den Text HELLO WORLD.

Sieht man sich diese Tabelle an wird schnell klar, dass die y-Werte schon für einen kleinen Text so extrem groß werden dass die QR-Codes hoffnungslos aus den Nähten platzen würden.

Die Lösung hat man schließlich in der Arbeit eines französischen Mathematikers Evriste Galois (1811-1831) gefunden. Der hatte sich mit mathematischen Körpern beschäftigt. Ein Körpert ist in der Mathematik eine Menge von Zahlen folgende Eigenschaft haben: Führt man mit beliebigen Elementen aus dieser Menge die vier Grundrechenarten (Adition, Subtraktion, Multiplikation und Division) aus, so müssen die Ergebnisse dieser Grundrechenarten wieder zu Elementen (Zahlen) aus dieser Menge führen. So ist z.B. die Menge der ganzen Zahlen kein Körpr weil es Divisionsergebnisse gibt die keine ganzen Zahlen mehr sind. Galois ist noch einen Schritt weiter gegangen und hat endliche Körper untersucht, also Mengen mit einer endlichen Anzahl von Elementen.

Ein Galois-Körper ist ein endlicher Bereich von Zahlen mit der Eigenschaft dass man mit Zahlen aus diesem Zahlenbereich die vier Grundrechenarten (Addition, Subtraktion, Multiplikation und Division) ausführen kann wobei die Ergebnisse wieder in dem endlichen Zahlenbereich liegen. Es

ist natürlich klar, dass es dann für diese „Grundrechenarten“ andere Regeln braucht als die die man sonst so kennt. Auch die Rechenergebnisse sind deshalb nicht das was man bei „normalen“ Berechnungen erwarten würde. Galois konnte nun zeigen, dass diese speziellen Rechenregeln für alle endlichen Körper gelten bei denen die Elementanzahl die Potenz einer Primzahl ist.
Beim QR-Code wird als Primzahl die 2 benutzt und als Exponent die 8; es ist also ein Galoiskörper GF(2^8) mit 256 Einzelementen. Nachdem alle Berechnungen (auf die hier nicht näher eingegangen werden soll) auf dem Galois-Körper bleiben, also zwischen 0 und 255 liegen, bleiben die Ergebnisse in eben diesem Bereich. Die den Stützstellen entsprechenden Werte benötigen deshalb ebenfalls jeweils nur ein Byte. Für die Sichtungstiefe „H“ müssen für die 16 Datenwörter 10 Codewörter berechnet werden. In unserem Fall ist das Ergebnis dann die Zahlenfolge

196 35 39 119 235 215 231 226 93 23

Verglichen mit den in der Tabelle 2 gezeigten Stützstellen nur noch ein kleiner Bruchteil.

7. Verschachtelung (Interleaving) der Daten

Bei ganz kurzen Nachrichten – z.B. solchen die in ein Quadrat der Version 1 passen – werden Nutz- und Sicherungsinformation einfach aneinandergehängt:

32 91 11 120 209 144 120 77 67 64 236 17 236 17 236 17 196 35 39 119 235 215 231 226 93 23

Tabelle 3: Kodierter Text „HELLO WORLD“ (grün) und angehängte H-Sicherungsinformation (rot)

Weit komplizierter sieht es bei höheren Versionen aus. Dort wird die gesamte Nachricht zuerst in einzelne Blöcke zerlegt. Diese Blöcke können unterschiedlich lang sein. Blocklängen und Anzahlen können sich von Version zu Version und von Sicherungstiefe zu Sicherungstiefe erheblich unterscheiden. Hier ein Beispiel für eine 62 Byte lange Nachricht Version 5 Sicherungstiefe Q:

67 85 70 134 87 38 85 194 119 50 6 18 6 103 38
246 246 66 7 118 134 242 7 38 86 22 198 199 146 6
182 230 247 119 50 7 118 134 87 38 82 6 134 151 50 7
70 247 118 86 194 6 151 50 16 236 17 236 17 236 17 236

Zu dieser Nachricht wird für jeden Nachrichtenblock ein Block Sicherungsdaten erzeugt:

213 199 11 45 115 247 241 223 229 248 154 117 154 111 86 161 111 39
87 204 96 60 202 182 124 157 200 134 27 129 209 17 163 163 120 133
148 116 177 212 76 133 75 242 238 76 195 230 189 10 108 240 192 141
235 159 5 173 24 147 59 33 106 40 255 172 82 2 131 32 178 236

Jetzt werden die Nutzdaten von links nach rechts spaltenweise gelesen und nacheinander aufgereiht:

67 246 182 70 85 246 230 247 70 66 247 118 . . . 103 146 151 236 38 6 50 17 7 236

Danach wird mit den Sicherungsdaten genauso verfahren:

213 87 148 235 199 204 116 159 11 96 177 5 . . . 161 163 240 32 111 120 192 178 39 133 141 236

Diese beiden Zahlenreihen werden einfach aneinandergehängt:

67 246 182 70 . . . 50 17 7 236 213 87 148 235 … 39 133 141 246

8. Das Eintragen der Daten

Die im Punkt „Verschachtelung der Daten“ entstandene Zahlenreihe wird jetzt von links beginnend in einer mäanderförmigen Zickzacklinie von rechts unten nach linkes unten in das QR-Quadrat eingetragen. Graphisch sieht das dann so aus:

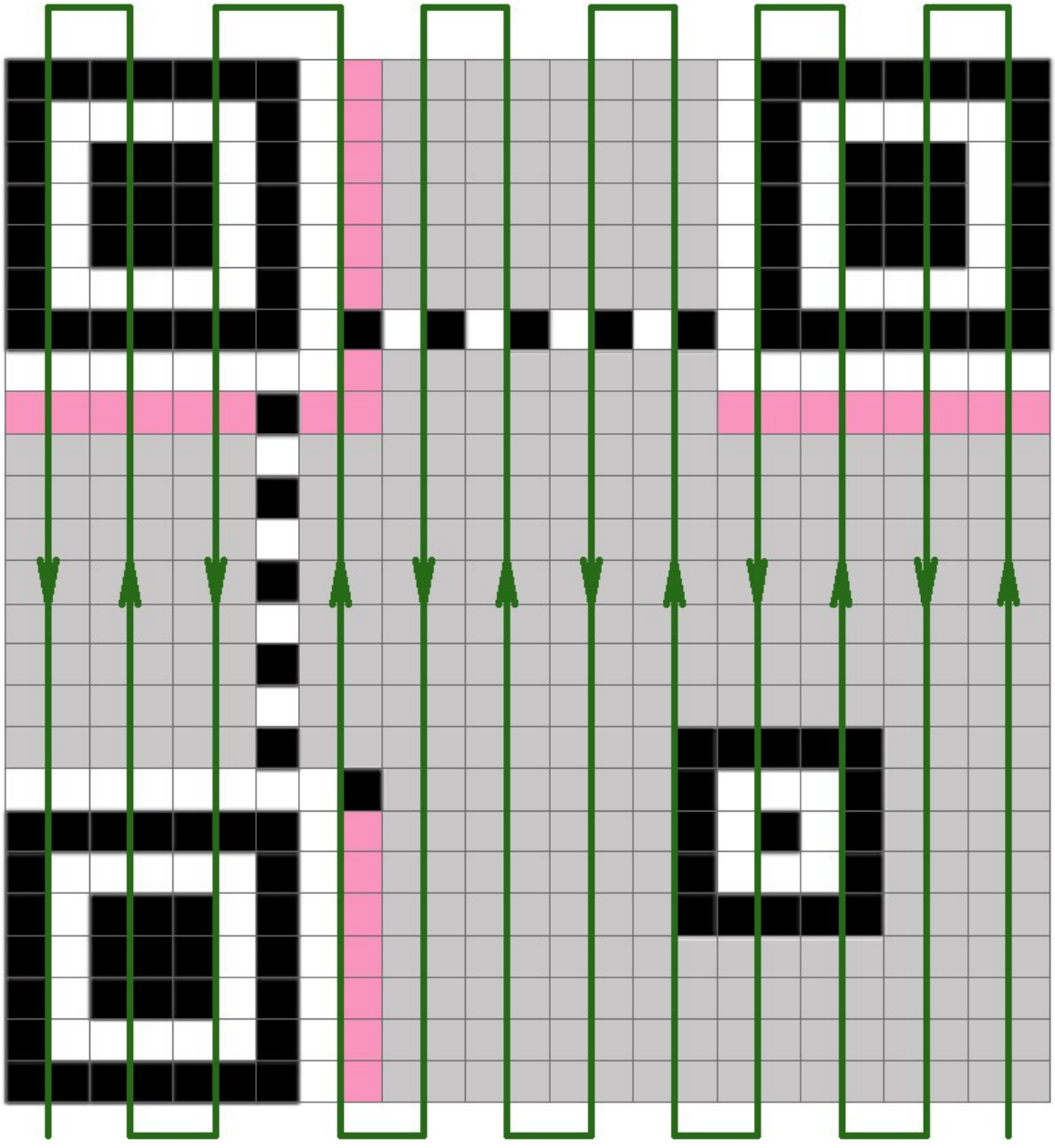

Bild 6: Füllrichtung der Nutzdaten auf der grauen Fläche

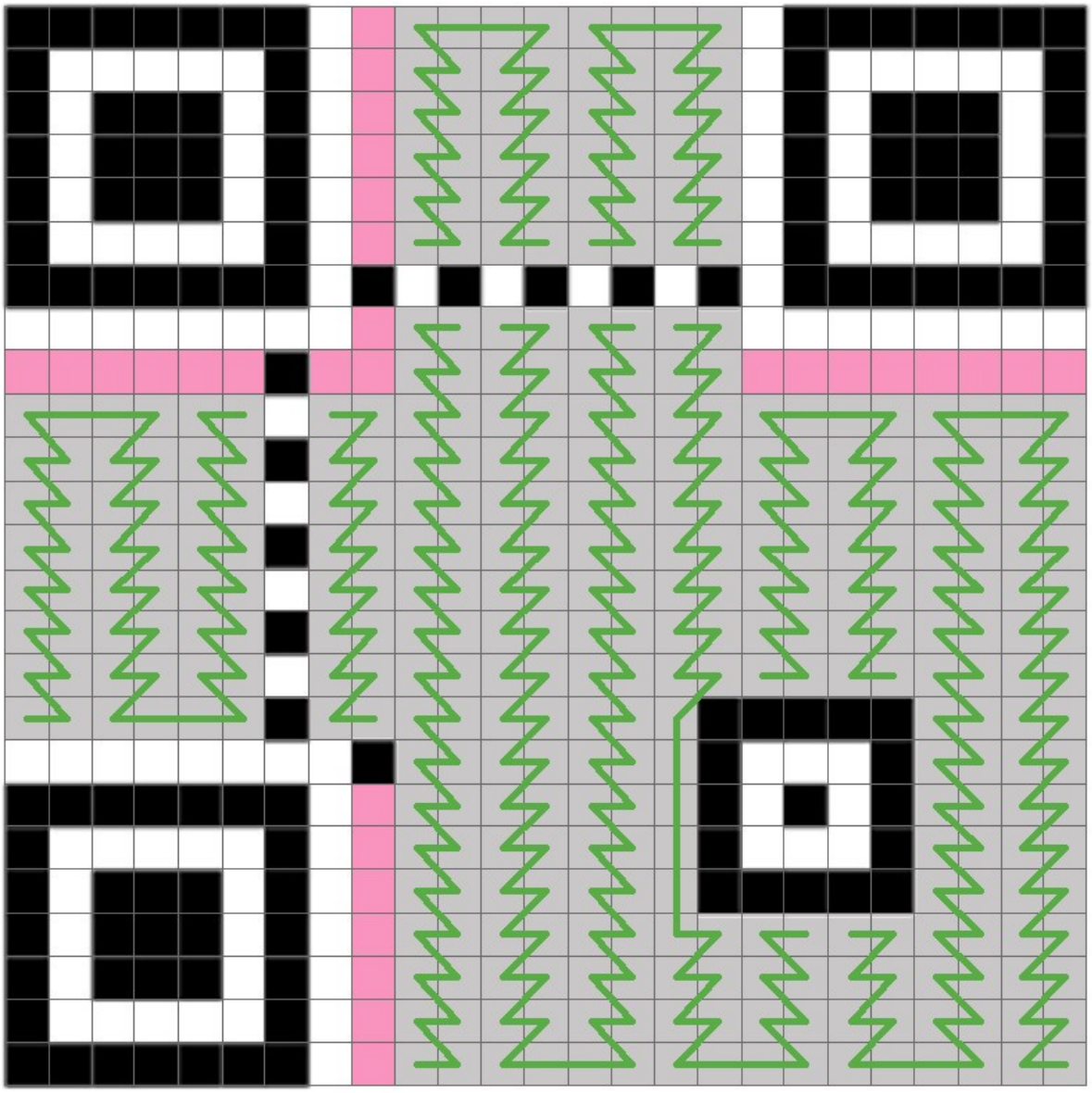

Bild 7: Zickzackmuster in dem die einzelnen Positionen entlang der Mäanderkurven befüllt werden.

9. Maskierung

Wird ein QR-Quadrat der Version 5 mit den im Absatz „Verschachtelung“ beschriebenen Daten gefüllt sieht das Ergebnis so aus:

Bild 8: Nicht „maskiertes“ Ergebnis direkt nach dem Eintrag von Nutzdaten in das Version 5 Quadrat.

Sieht man sich das Ergebnis genauer an, so kann man in dem Quadrat einige größere zusammenhängende schwarze oder weiße Flächen erkennen. In der Nähe solcher Flächen ist die genaue Positionsbestimmung einzelner Punkte unsicher – speziell wenn es bei der Aufnahme des QR-Bildes zu optisch bedingten Verzerrungen gekommen ist. Deshalb versucht man solche Flächen zu vermeiden. Nachdem man die Nutzdaten selbst nicht ändern kann, bleibt nur das Bild nachträglich zu verändern. Das Verfahren das dabei angewendet wird nennt man „Maskierung“. Dazu wird über das QR-Quadrat eine Maske gelegt, die aus einem vorgebenen Muster besteht. Danach werden alle Positionen die unter schwarzen Maskenpunkten liegen „invertiert“, alle anderen bleiben unverändert. Mit „invertiert“ ist gemeint dass aus einem schwarzen Punkt ein weißer wird und umgekehrt. Maskiert man ein beliebiges Bild mit einem vorgegebenen Muster kann man erreichen dass die ursprünglichen schwarzen oder weißen Flächen aufgelöst werden.

9.1. Maskierungsmuster

Maskiert man ein beliebiges Bild mit einem vorgegebenen Muster kann man zwar erreichen dass die ursprünglichen schwarzen oder weißen Flächen aufgelöst werden, es können aber ebensogut an anderer Stelle wieder solche Flächen entstehen. Diesem Problem begegnet man indem man acht

verschiedene solche Maskierungsmuster definiert.

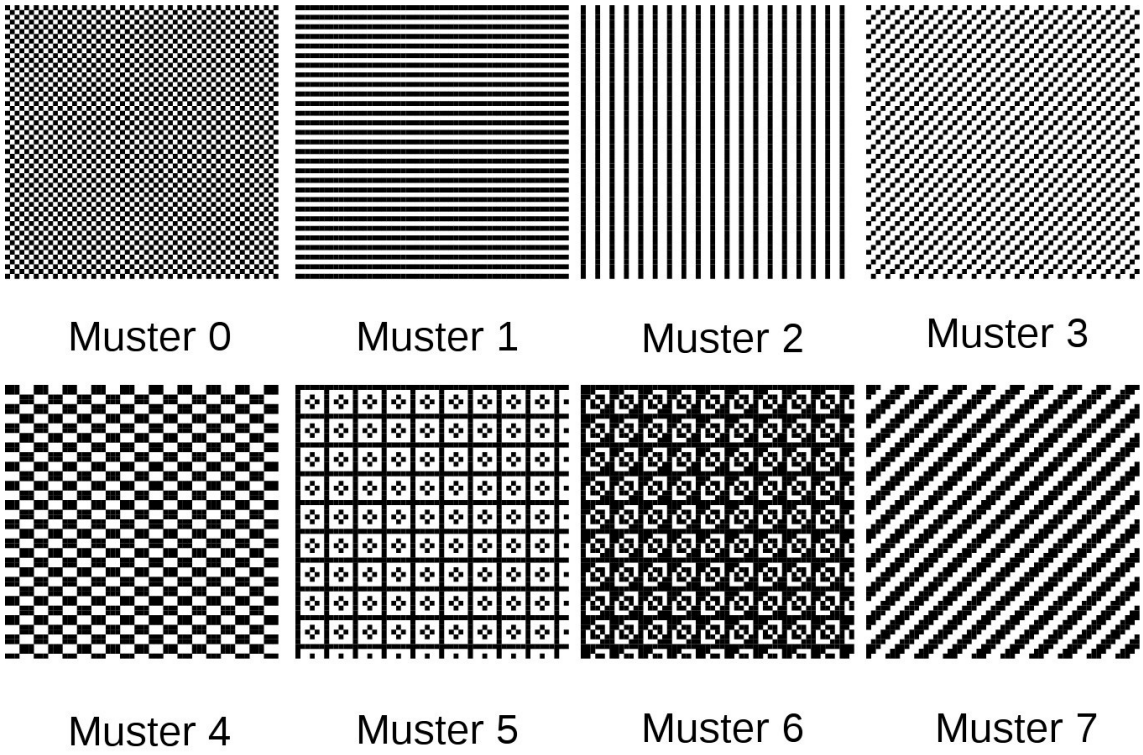

Bild 9: Die verschiedenen Muster der Masken

Würde man diese Masken auf die gesamte QR-Fläche anwenden, würden auch die bereits beschriebenen festen Teile in Pixelmuster aufgelöst, Lage und Geometrie können nach der Maskierung also nicht mehr erkannt werden. Die Maskierungsmuster müssen also auf die Bereiche beschränkt bleiben in denen die Nutz- und Sicherungsinformation steht.

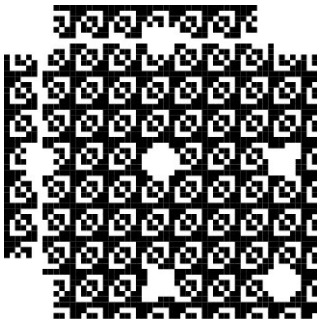

Bild 10 : Auf die Nutz- und Sicherungsdaten reduzierte Maske (im Beispiel Version 10, Muster 6)

Das Orignalbild wird dann mit jedem dieser Muster maskiert. Danach werden die acht Ergebnisse miteinander verglichen. Das Bild bei dem das Bewertungsverfahren die meisten Änderungen und die kleinste Zahl zusammenhängender Flächen erkennt wird schließlich ausgewählt.

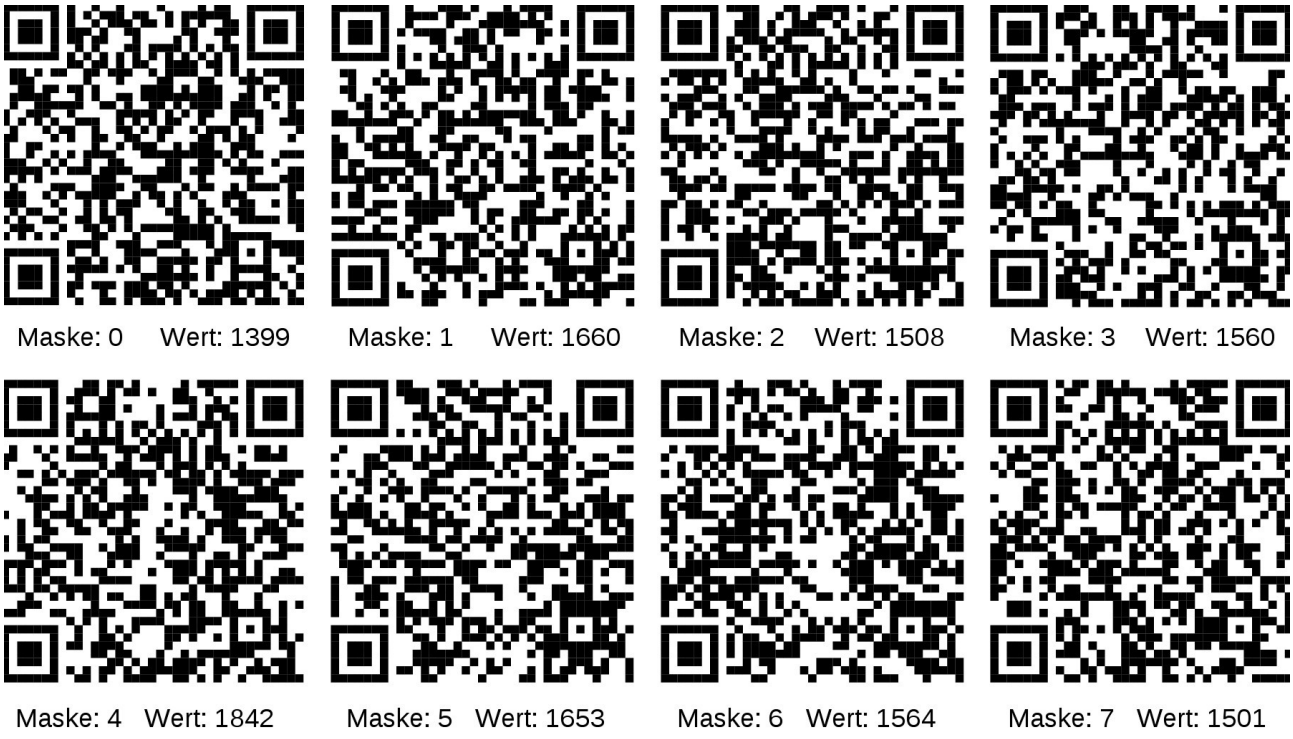

Bild 11: Die Nutzdaten des Bildes 8 mit den 8 Masken maskiert

Im Bild 10 sind die Ergebnisse nach der Anwendung der acht unterschiedlichen Masken zu sehen. Der dabei angegebene Wert ist eine Zahl die die Güte der Dekodierbarkeit (z.B. Anzahl und Größe der zusammenhängenden Flächen angibt. Dabei liefert das Bild mit dem niedrigsten Wert die besten Ergebnisse. In unserem Beispiel ist das das Bild mit der Maske 0, also der QR-Code der letztendlich ausgewählt wird.

10. Quellenverzeichnis

www.onky.com/qr-code-tutorial

fbnm.h-da.de/~ochs/mathe1/skript/galois.pdf