EQ2440: Visual Based In-Flight File Transfer

Team Green 2013

Ahsan Mahmood
Brice Lavorel
Hanwei Wu
Jia Wang
Lukas Jornitz
Yasin El Guennouni

Supervisor: Per Zetterberg
Assistant: Iqbal Hussain

Abstract

This paper describes a way to transfer files between two smart phones without using radio transmission. Instead the information is sent using the screen on one phone and received using the camera of the other phone. The project started with using the standard QR code. However the standard was not followed completely, several tweaks were implemented in order to make the communication handle larger amounts of data, run faster and be more robust. To encode more data to the QR code color codes were used; this made it possible to encode 3 times more data.

The final system was run on Samsung Galaxy S3 phones using Android 4.2. In the final model a file that is up to 754 kilobytes can be transmitted with lowest error correction.

Keywords Visual channel, QR code, Android implementation

Table of Contents

Acknowledgement

We would like to take the chance to thank Per Zetterberg for making this course possible. He has also been great support during the course as well, always coming to the lab and seeing if everything is going as planned. We would also like to thank our project assistant Iqbal Hussain for giving us feedback and insights on different areas in the project. We would also want to thank the other groups for the pleasant time spent in the lab.

Chapter 1

Introduction

1.1 Background

The ban of devices using radio transmitters on board aircrafts is well known in some parts of the world. The common excuse is that the radio transmitters may interfere with the flight instruments such as the navigation system. But the main reason, according to the Federal Communications commission is that it interferes with networks on the ground [1]. Most base stations on the ground are slightly tilted forward in order to enable users close to the base station service. However, the antennas that are commonly used have a front and a back lobe, tilting the front lobe forward makes the back lobe rise up [2]. If the passengers onboard have their cell phones on they will connect to many base stations on the ground and disorder the operator's cell plans. The airline companies are creating new solutions to the problem by offering the passengers services as Wi-Fi on board [3]. However the services are limited and sometimes also expensive to use.

In this report a method that does not use the common radio transmission techniques will be described, and by doing so the ban of radio transmitters will be worked around.

In this system a visual based file transfer will be carried out, using the screen on one phone as a transmitter and the camera on another phone as a receiver. This method avoids any use of frequencies that might interfere with other systems, whether it is in the aircraft or on the ground.

In order to enable quick and reliable communication the Quick Response Code (QR Code) will be used. The main advantages with QR code are that it can store large amounts of data, both alpha numeric characters and in binary bits, [4] and it also runs very fast. Besides from that it has error correction; which makes the solution very robust. This makes it possible to read the code even if a part of it is missing or is unclear.

1.2 Problem Description

The layout of the problem solved in this project is illustrated in Figure 1. Initially the data is broken down into small pieces on the transmitter side and each piece is encoded into a QR image. Then these images will be displayed on an Android device at a certain rate.

On the other end of the system, namely the receiving end, the QR codes are read by the camera on another Android device and saved onto its SD card. Further the received images will be processed to decode the message and then combined in order to recover the initial file.

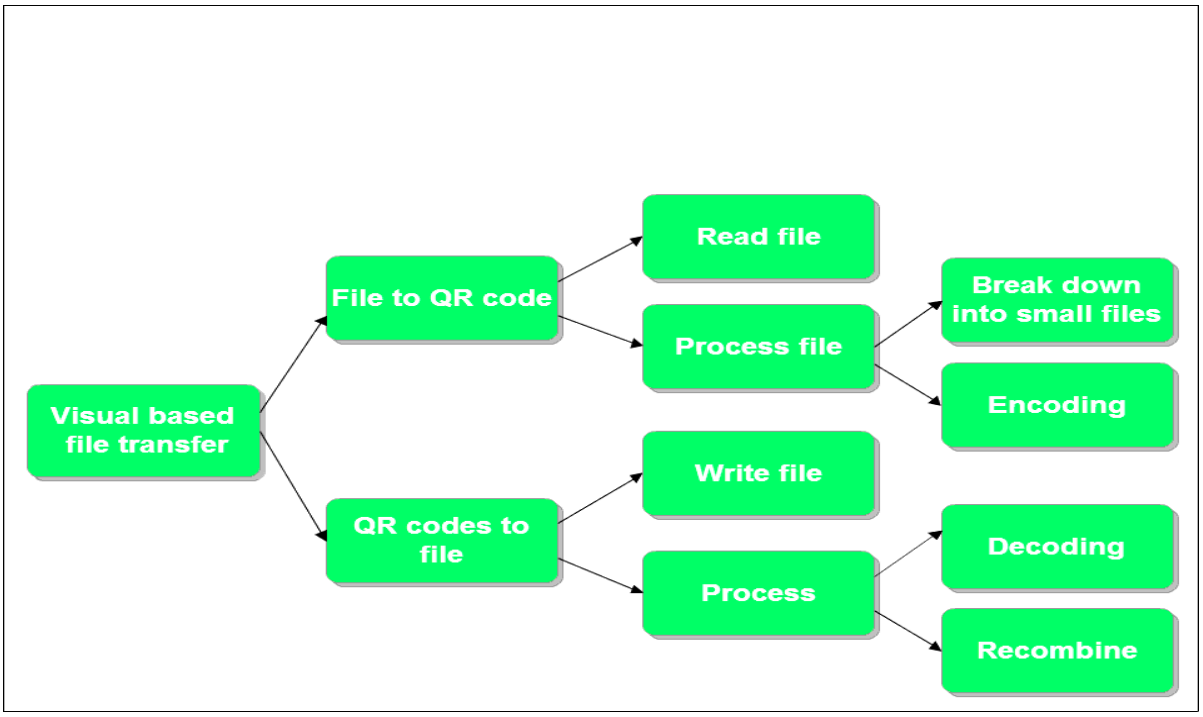


Figure 1: Flow chart of the system

Chapter 2

Theory

2.1 Communication System

In this project the system can be seen as a communication system with all the different parts included. The phone that is displaying the QR code is the transmitting part and the other phone using camera to read codes is the receiver. The system utilizes a visual channel which is the screen. There is also noise that disturbs the signal, this is caused by reflections and shadows on the screen

2.2 QR Codes

Each QR Code symbol should be constructed of encoding region and function patterns, namely finder, separator, timing patterns, and alignment patterns. Function patterns should not be used for encoding the data. The symbol should be surrounded by a quiet zone border. Figure 2 illustrates the structure of a Version 7 QR Code symbol.

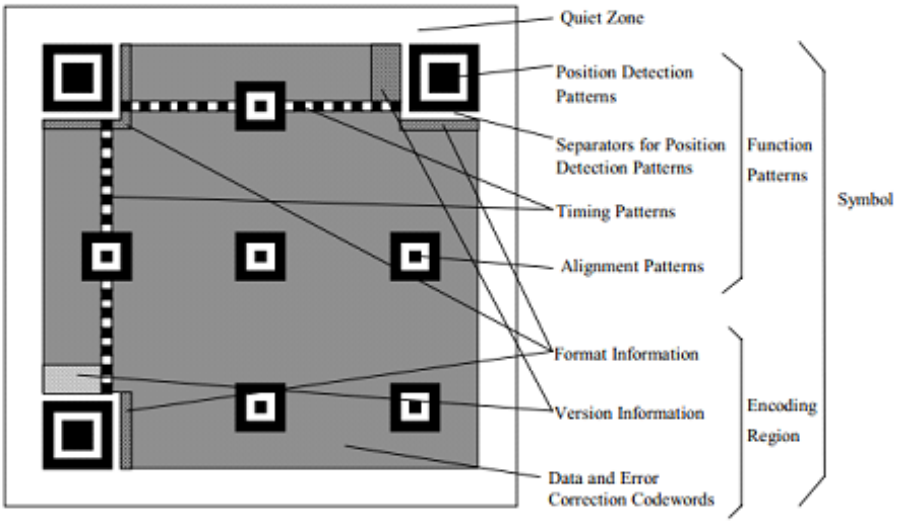


Figure 2: Structure of a Version 7 QR Code symbol [9]

2.2.1 Symbol Versions and Sizes

There are forty versions of QR Code symbols from 1 to 40. Version 1 measures 21 * 21 modules, Version 2 measures25 *25 modules and so on, increasing by the factor of 4 modules. For Version 40 which contains 177 *177 modules.

2.2.2 Finder Pattern

At the upper left, upper right and lower left corners of the symbol there are three Position Detection Patterns. The width ratio of each Position Detection Pattern is 1:1:3:1:1 as illustrated in Figure 3. The Finder pattern is used to rapidly find the QR Code symbol in the field of view.

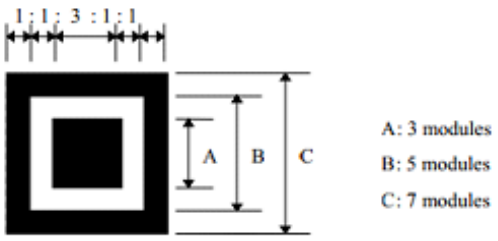


Figure 3: Ratio of the finder pattern [10]

2.2.3 Separators

A one-module wide Separator is placed between the Position Detection Pattern and Encoding Region, as illustrated in Figure 1; the Separators are constructed of all light modules.

2.2.4 Timing Pattern

Timing Patterns consist of a one module wide row or column of alternating dark and light modules, starting and ending with a dark module. The horizontal Timing Pattern is at row 7 of the symbol between the Detection Patterns; the vertical Timing Pattern similarly is at column 7 of the symbol between the Position Detection Patterns. They enable the symbol density and version to be determined and provide datum positions for determining module coordinates.

2.2.5 Alignment Patterns

The Alignment Patterns consist of three superimposed concentric squares which are constructed of dark 5* 5modules, light 3*3 modules and a single central dark module. The number of Alignment Patterns depends on the version of symbol. They easily enable to detect the symbol at the good position.

2.2.6 Encoding Region

This region should contain the data, error correction codeword, version Information and Format Information.

2.2.7 Error Correction Coding

The error correction codeword will be generated using a method called Reed-Solomon Error correction.

2.2.8 Quiet Zone

Quiet zone nominally should be light modules. And this region surrounds the symbol on all four sides. The quite zone enables the QR code to be distinguishable from the background.

Chapter 3

Method

3.1 Encoding

3.1.1 Data Analysis

Standard QR Codes include several modes to allow characters to be converted into symbol characters in efficient ways. In this project, only byte mode is used as it can encode any file type. The next step is to select the required Error Correction Level. If the user has not specified the symbol version to be used, select the smallest version that will accommodate the data.

3.1.2 Data Encoding

In this project, 8-bits mode is used to encode the data then split the resulting bit stream into 8-bit codeword. Pad Characters have to be added so as to fill the number of data codeword's required for the version if the data is not enough for that version.

8-Bit/Byte Mode

In this mode, one 8 bit codeword directly represents ASCII value of the input data as shown in Table 1.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Table 1: ASCII values

Terminator

The end of data in the symbol is signaled by the Terminator sequence 0000. This may be omitted if the data bit stream completely fills the capacity of the symbol, or abbreviated if the remaining capacity of the symbol is less than 4 bits. The message bit stream should then be extended to fill the data capacity of the symbol corresponding to the Version and Error Correction Level.

Bit Stream to Codeword Conversion

The resulting message bit stream should be divided into 8 bits codeword. If the final codeword is not exactly 8 bits in length, it shall be made 8 bits long by padding 0. Then message bit stream should be extended to fill the capacity of simple, by the addition of the Padding Codewords 11101100 and 00010001 alternately. In certain versions of symbol, it may be necessary to add 3, 4 or 7 Remainder Bits (all zeros) to the end of the message in order exactly to fill the symbol capacity.

EXAMPLE: (for Version 1-H symbol)

Input data: 012345

1. Convert data to its binary value:

0 00000000

1 00000001

2 00000010

3 00000011

4 00000100

5 00000101

2. Connect the binary data in sequence:

00000000 00000001000000100000001100000100000000101

3. Add Character Count and mode Indicator to binary (10 bits for version 1-H):

No. of input data characters: 6 00000110

Mode indicator: 0100

4. Add padding bit and divided it into each 8 bits.

00000110 01000000 00000000 00010000 00100000 00110000 01000000 01010000

To encode a sequence of input data in a given mode, the steps defined in sections 2.2.1 to 2.2.3 shall be followed.

Table 2 defines the Mode Indicators for each mode. Table 3 defines the length of the Character Count Indicator,

Table 2 — Mode indicators

Mode	Indicator
ECI	**0111**
Numeric	**0001**
Alphanumeric	**0010**
8-bit Byte	**0100**
Kanji	**1000**
Structured Append	**0011**
FNC1	**0101** (First position) **1001** (Second position)
Terminator (End of Message)	**0000**

Table 3 — Number of bits in Character Count Indicator

Version	Numeric Mode	Alphanumeric Mode	8-bit Byte Mode	Kanji Mode
1 to 9	**10**	**9**	**8**	**8**
10 to 26	**12**	**11**	**16**	**10**
27 to 40	**14**	**13**	**16**	**12**

3.1.3 Error Correction Coding

The error correction codeword will be generated using a method called Reed-Solomon Error correction.

Steps of Polynomial Long Division

The steps for polynomial long division are:

1. Find the appropriate term to multiply the divisor. The result of the multiplication should have the same first term as the dividend.

2. Subtract the result from the dividend. Repeat steps 1 and 2 until it is no longer possible to multiply by an integer, or in other words, it would be necessary to multiply by a fraction. The number at the bottom of is the remainder.

Galois Field Arithmetic

GF (256) contains the numbers from 0 to 255. Mathematical operations in GF (256) are cyclical, meaning that if a number is larger than 255, it will be necessary to use modulo operation to get a number still in the Galois Field.

In the Galois Field, negative numbers have the same value as positive numbers, so:

$-n = n.$

In other words, always use the absolute value of the numbers involved in Galois Field arithmetic. This means that addition and subtraction within the Galois Field is the same thing.

Generator Polynomial

The generator polynomial is a polynomial that is created by multiplying (x - α0) ... (x - αn-1) where n is the number of error correction codeword that must be generated.

Generating Error Correction Codeword

Divide the Message Polynomial by the Generator Polynomial, after dividing the two polynomials, there will be a remainder. The error correction codeword are the coefficients of the remainder.

3.1.4 Structure of Final Message

Now the data codeword and their corresponding error correction codeword should be available. As mentioned in the previous step, larger QR codes requires one to break the data codeword into smaller blocks, and generate error correction codeword separately for each block. When this is the case, the data blocks and error correction codeword must be interleaved according to table 4. Interleaving for data and error correction codewords are carried out separately. For each part, one codeword is taken from each block and interleaved with corresponding codewords from the other blocks. If the number of codewords in blocks is different, then the initial blocks should have lesser number of codewords. Number of codewords in error correction blocks is always the same.

Determine how many blocks are required for the data interleaving.

The **error correction table** in Appendix B shows how many data blocks are required for each version and error correction level.

	Data codewords					Error correction codewords			
Block 1	D1	D2	.....	D11		E1	E2	.....	E22
Block 2	D12	D13	.....	D22		E23	E24	.....	E44
Block 3	D23	D24	.....	D33	D34	E45	E46	.....	E66
Block 4	D35	D36	.....	D45	D46	E67	E68	.....	E88

Table4: Interleaving [8]

3.1.5 Placement in Matrix

If the data and error correction codeword are interleaved, and the final string of bits is obtained, then the next step is to put them in the QR code matrix along with the required function patterns.

Put the Finder Patterns into the Matrix

First, put the finder patterns into the matrix. The finder pattern (shown below) consists of an outer black square that is 7 modules by 7 modules, an inner white square that is 5 modules by 5 modules, and a solid black square in the center that is 3 modules by 3 modules.

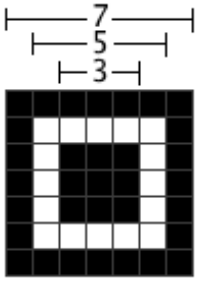


Figure 4: Finder pattern in modules [7]

QR code scanners can search for this ratio of light to dark modules to detect the finder patterns and correctly orient the QR code for decoding. The finder patterns are always placed in the top left, top right, and bottom left corners of the QR code, no matter which version is in use.

Add the Separators into the Matrix

The separators are lines of white modules, one module wide, that are placed beside the finder patterns to separate them from the rest of the QR code. The separators are only placed beside the edges of the finder patterns that touch the inside of the QR code. For example:

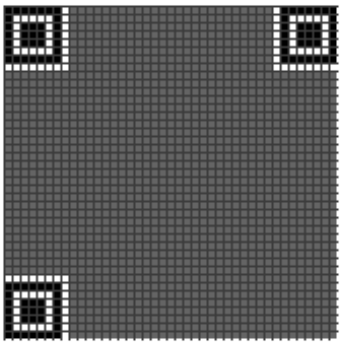


Figure 5: White separators surrounding the finder patterns [7]

Add the Alignment Patterns into the Matrix

QR codes that are version 2 and larger are required to have alignment patterns. An alignment pattern, shown below, consists of a 5 module by 5 module black square, an inner 3 module by 3 module white square, and a single black module in the center.

The locations at which the alignment patterns must be placed are defined in the alignment **pattern locations table** in Annex A**.**

Add the Timing Patterns into the Matrix

The timing patterns are two lines, one horizontal and one vertical, of alternating dark and light modules. The horizontal timing pattern is placed on the 7th row of the QR code between the separators. The vertical timing pattern is placed on the 7th column of the QR code between the separators. The timing patterns always start and end with a dark module. Alignment patterns may overlap timing patterns because their light and dark modules always coincide with the light and dark modules of the timing patterns.

Add the Dark Module and Reserved Areas

The dark module must be added, and there are areas of the matrix that must be reserved for the format and version bits, which will be added later.

Place the Data Bits

The data bits are placed starting from the bottom-right of the matrix and proceeding upward in a column that is 2 modules wide. When top of the column is reached, continue with the next 2-module column to the left and continues downwards. Whenever edge of a column is

reached, move on to the next 2-module column and change direction. While moving upwards/downwards, first place on right part of the 2-module column and then in the left part. If a function pattern or reserved area is encountered, the data bit is placed in the next unused module.

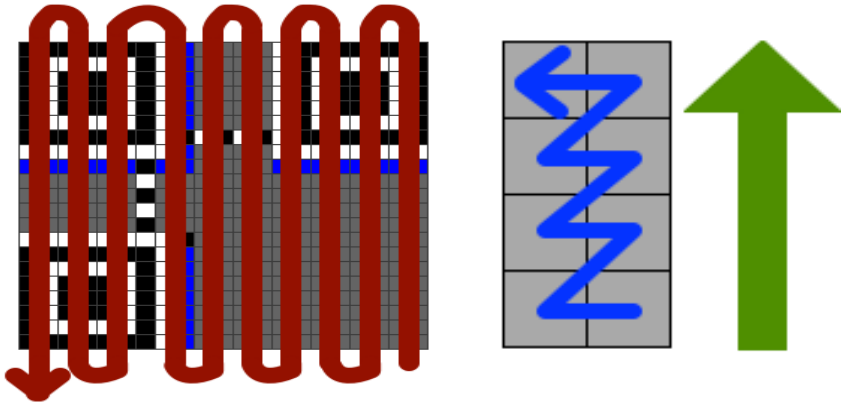


Figure 6: Placement of data bits [7]

3.1.6 Masking

A mask pattern decides which modules are dark and which are light according to a particular rule. The purpose of this step is to modify the QR code to make it easy to scan for a QR code reader.

The Mask Patterns

Each mask pattern uses a formula to determine whether or not to change the color of the current bit. The coordinates of the current bit are put into the formula, and if the result is 0, then the opposite bit at that coordinate is used.

Here is the list of the mask pattern formulas:

Mask Number	
0	(row + column) mod 2 == 0
1	(row) mod 2 == 0
2	(column) mod 3 == 0
3	(row + column) mod 3 == 0
4	(floor(row / 2) + floor(column / 3)) mod 2 == 0
5	((row * column) mod 2) + ((row * column) mod 3) == 0

6 (((row * column) mod 2) + ((row * column) mod 3)) mod 2 == 0

7 (((row + column) mod 2) + ((row * column) mod 3)) mod 2 == 0

Determining the Best Mask

Note that the entire matrix (including function patterns and reserved areas) is evaluated, even though the masking is only applied to data and error correction modules. For the purpose of evaluation, the reserved areas are considered to all be light modules, not dark modules.

The four penalty rules can be summarized as follows:

Feature	Evaluation condition	Points
Adjacent modules in row/column in same color	No. of modules = (5 + i)	$N_1 + i$
Block of modules in same color	Block size = m × n	$N_2 \times (m - 1) \times (n - 1)$
1:1:3:1:1 ratio (dark:light:dark:light:dark) pattern in row/column		N_3
Proportion of dark modules in entire symbol	$50 \pm (5 \times k)\%$ to $50 \pm (5 \times (k + 1))\%$	$N_4 \times k$

Table 5: Mask [7]

- The first rule gives the QR code a penalty for each group of five or **more** same-colored modules in a row (or column).
- The second rule gives the QR code a penalty for each 2x2 area of same-colored modules in the matrix.
- The third rule gives the QR code a large penalty if there are patterns that look similar to the finder patterns.
- The fourth rule gives the QR code a penalty if more than half of the modules are dark or light, with a larger penalty for a larger difference.

Mask pattern results in the lowest penalty score is the mask pattern that must be used for the final output.

3.1.7 Format and Version Information

Format Information consists of 15 bits. First two bits represent the error correction level and next three represent the mask used for encoding. The remaining 10 bits are used for error correction of the format information. The first two bits for different error correction levels are classified as:
01 → Level L
00 → Level M
10 → Level Q
11 → Level H

Version Information consists of 18 bits. First 6 bits represent the version of used QR symbol while remaining 12 bits are used for error correction. Version information is used only if version>6.

Generate Error Correction Bits for Format String and Version string

When five bits for the format string are available, then they are used to generate ten error correction bits. This step uses BCH Error Correction, but it is a little easier because in this case it is just required to generate ten error correction bits.

Get the Generator Polynomial

When generating the format string's error correction codeword's, then it is suggested by the QR code specification to use the following generator polynomial:

X10 + x8 + x5 + x4 + x2 + x + 1

This can be converted to a binary string by only taking the coefficients of each term. The coefficient of x10 is 1, the coefficient of x9 is 0 since x9 is not present in the polynomial. In other words, the binary string that represents the generator polynomial for this step is 10100110111

Calculate the Error Correction Bits

The next step is to divide the format string bits by the generator polynomial. To do this, first create a 15-bit string by putting ten 0s to the right of the format string. Now remove any 0s from the left side, and then the division is performed.

The steps for division are:

1. Pad the generator polynomial string on the right with 0s to make it the same length as the current format string.

2. XOR the padded generator polynomial string with the current format string.

3. Remove 0s from the left side of the result.

One has to be careful since it is needed to divide the polynomials until the resulting format string is 10 or fewer bits long. Therefore, before each division step, a check to make sure that the current format string is 11 bits or longer is done.

The Version Information areas are the 6* 3 module block above the Timing Pattern and immediately to the left of the top right Position Detection Pattern Separator, and the 3* 6

module block to the left of the Timing Pattern and immediately above the lower left Position Detection Pattern separator

Example:

Version number: 7

Data: 000111

BCH bits: 110010010100

Format Information module pattern: 000111110010010100.

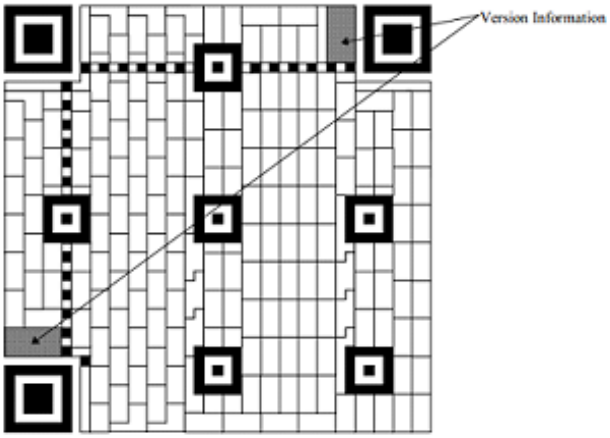


Figure 7: Version information [8]

3.1.8 Placement of Version Information in QR Code

There are two rectangular areas where the version information string must be placed: one on the bottom-left and other on the top right. We are using two blocks for redundancy as both contain the same information.

Bottom Left Version Information Block

The dimension of bottom left version information block is 3×6. The following table explains how to arrange the bits of the version information string in the bottom-left version information area. The 0 represents the RIGHT most (least significant) bit of the version information string, and the 17 represents the LEFT most (most significant) bit of the version information string.

00	03	06	09	12	15
01	04	07	10	13	16
02	05	08	11	14	17

Table 6 [7]

Top Right Version Information Block

The dimension of the top right version information block is 6×3. The following table explains how to arrange the bits of the version information string in the top-right version information area. The 0 represents the right-most (least significant) bit of the version information string, and the 17 represents the LEFT most (most significant) bit of the version information string.

00	01	02
03	04	05
06	07	08
09	10	11
12	13	14
15	16	17

Table: 7 [7]

3.1.9 Placement of Format Information in QR Code

Format information is placed around the first finder pattern module. Redundant information is split into half and placed around other finder patterns. The 0 represents the right-most (least significant) bit of the format information string, and the 14 represents the LEFT most (most significant) bit of the format information string.

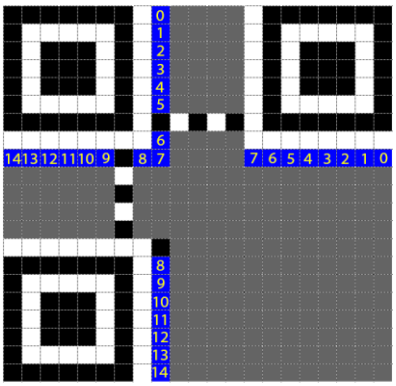


Figure 8: Format Information in QR Symbol [8]

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	0	0	0	1	0	1	1	1	1

Table 8: Format Information [8]

3.1.10 Summary for Generating Single QR Code

- Determine which encoding mode to use
- Encode the data
- Generate error correction codeword's
- Interleave blocks if necessary
- Place the data and error correction bits in the matrix
- Apply the mask patterns and determine which one results in the lowest penalty
- Add format and version information

3.1.11 Colored QR Codes

In order to achieve higher data rates, colored QR codes in RGB color space were used. In this way, it was possible to transmit three QR codes at the same time.

For generating colored codes, the only difference from the normal case is that instead of only using black and white pictures, three pictures (red, green and blue) is used. These pictures are added together resulting in one colored code having eight possible colors for transmission.

The easiest way to add capacity to your code is to add color to it which enables you transmit them together. As QR code does not have to be standard black and white in order to be scanned. It is possible to embed multiple colors and apply a color gradient without affecting scan ability. The only rule of thumb is that the dark modules are placed on a colored background.

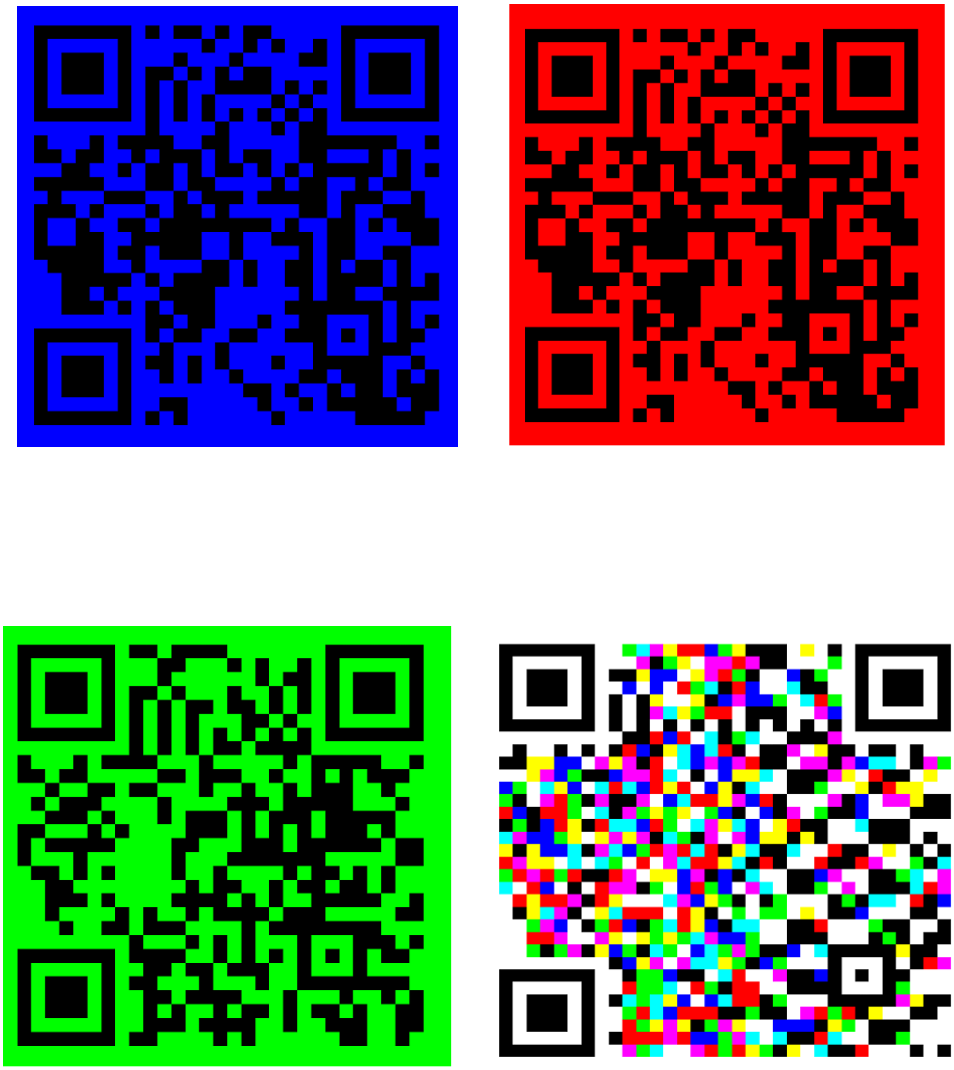


Figure 9: Colored QR codes and combined color QR code (bottom right)

3.1.12 Multiple Codes

If we need to transmit a file which length exceeds the capacity of the single QR-code, the file is divided into smaller blocks depending upon the capacity of single QR code. And these blocks are transmitted sequentially. Two different approaches have been implemented in this project:

1) Asynchronous Transmission

   In this approach, multiple QR codes are displayed on the screen of one cell phone at a specified rate. After short interval, codes are updated and transmitter is not synchronized with the receiver.

2) Synchronous Transmission

Synchronous transmission relies on the acknowledgement from the receiver. When the receiver decodes one code, it sends a flash in order to acknowledge correct reception. The transmitter, upon reading the flash, updates the next code on the screen.

A new protocol has been implemented for communication of multiple codes. Frame number information is put around the first finder block that aids the decoder in reading multiple frames. If the receiver has already decoded a particular frame then it ignores decoding of the same code and hence improving the performance of the system. Frame number information consists of 24 bits. 8 bits are used for frame number while remaining are used for error correction of frame number information.

Generate Frame Number Information

The Bose-Chaudhuri-Hocquenghem code was used for error correction of frame number information. The polynomial whose coefficient is the data bit string should be divided by the generator polynomial. The coefficient string of the remainder polynomial should be appended to the data bit string to form the (24, 6) BCH code string.

Placement of Frame Number Information in Matrix

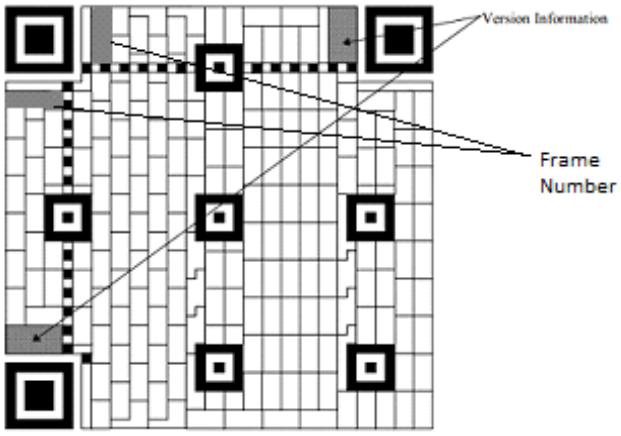


Figure 10: Frame number

The frame number information is divided into two blocks and placed around the first (top left) finder pattern. Each sub block contains 12 bits. 0 is the least significant while 23 is the most significant bit of the frame number information block. Table 9 and Table 10 illustrates these bits.

23	21	19	17	15	13
22	20	18	16	14	12

Table 9

0	1
2	3
4	5
6	7
8	9
10	11

Table 10

3.1.13 Multiple QR code data structure

The structure of the data message is visible in the figure below:

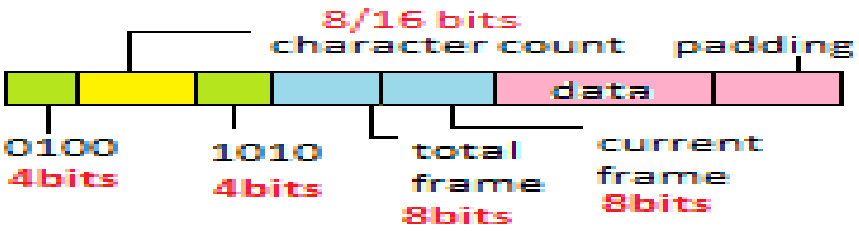


Figure 11: Data message structure

Now the data codeword and their corresponding error correction codeword should be known, as illustrated above. To transmit multiple QR Codes, a new protocol was implemented. The final message starts with 4 bits mode indicator which is followed by 8/16 bits character count that shows how many data bits there are in the message. Then there is a unique 4 bits frame indicator and a total frame number which shows how many frames that are used to transmit file followed them. The data part and padding part are also the same as single QR code data.

3.2 Decoding

Decoding of QR codes begin with the application of essential image processing techniques followed by the reading algorithm.

The task of the image processing includes location of the finder pattern, alignment patterns, and to do the affine transformation to the picture in the segments divided by the alignment pattern.

3.2.1 Binarization

The first step in decoding is to make the picture readable. In order to do is the picture has to be binarized, from different gray levels into black and white. We implemented the following methods for binarization and compared their performances. The results suggested Otsu Method as our preferred choice.

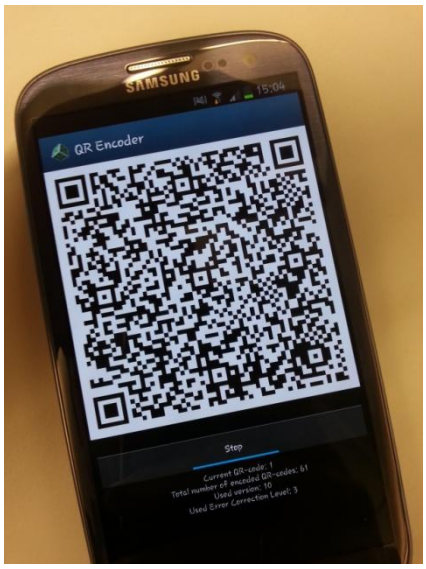


Figure 12: Original image

Figure 13: Binarized image

Static Thresholding

This is the basic method for converting grayscale images to black and white picture. In this method a threshold is chosen manually and pixels having grayscale value greater than the threshold are set to white while remaining are set to black. This method is very fast but it does not give good results especially when there is non-uniform light across the image.

Otsu Method

In this method, we iterate through all the possible values of thresholds and choose the one that minimizes intra-class variance i.e. for each threshold; we find the measure of spread for pixel levels each side of the threshold and then find the weighted sum of both calculated variances.

$$\sigma^2_{within-class} = w_F\sigma^2_F + w_B\sigma^2_B$$

Where, w_F is weight of the foreground,

w_B is the weight of the background

σ^2_F is the variance of foreground

σ^2_B is the variance of background

A faster approach for minimizing intra-class variance is to maximize inter-class (between-class) variance.

$$\sigma^2_{between-class} = \sigma^2 - \sigma^2_{within-class} = w_F w_B(\mu_F - \mu_B)^2$$

Where, μ_F is the mean value of foreground,

μ_B is the mean value of background

Barnes's Method

This algorithm was published in 1986 in [5]. In comparison to Otsu uses it local thresholds. This means that the algorithm runs each threshold calculation for a sub square of the original image. This sub-square is commonly referred to as b2 with b set to 31 as default. This sub-square is inspected for each pixel.
For this sub-square the ideal threshold is approximated by the mean value of the maximum and minimum pixel intensity within it. In short mathematical form it can be represented as follows:

$$\mathrm{T(x,y)} = \frac{F_{max} + F_{min}}{2}$$

$$b(x,y) = \begin{cases} 1 & if\ g(x,y) > T(x,y) \\ 0 & otherwise \end{cases}$$

where b(x,y) is the output binary image (or sub-image) and g(x,y) is the input image as compared with the threshold T(x,y). This method works well as long as the contrast difference in the sub divided window is large enough. An always associated problem with local thresholding is the calculation time. If it is increased b, it will give better results but also a higher computation time. In addition a high resolution photo from the camera is used and

the calculation is made on a mobile device. In the next section an algorithm is proposed which is adapted to working on QR codes.

Modified Sauvola's Algorithm by J. Zhou and Y. Liu

This adapted method [6] was proposed in 2010. It is one of the first methods trying to use the assumption that black:white should be 1:1 in the picture (due to masking). Sauvola's method is used for document processing and needs a huge amount of time for calculation. It estimates the threshold as:

$$T(x, y) = mean(x, y)[1 + k(1 - \frac{std.dev(x, y)}{R})]$$

where R by default is 128 and k is 0:5. The method is extremely effective but the default parameters are used for document processing.
Due to masking the image, only containing the QR code without the quiet zone, has the same amount of white and black pixels. Therefore k = 0:5 is proposed in the paper. For R, it is a value around D/3, where D is the maximum of image width and height. R can be a bit smaller for an image where the QR code won't fill up the whole image. The proposed value can be calculated using the QR features [7, 6].
With these values Savola's method is used. For each pixel the threshold is calculated by using the b-square around it. If the pixel(x,y) to be binarized is located at the border of the image, the square is filled up with black pixels.

Comparison

We compared the execution times of all of the above algorithms and drew the following conclusion:

- Static Thresholding -- Fast but unreliable (~40ms)

- Otsu Algorithm -- Fast and reliable (~200ms)

- Barnes's Algorithm -- Reliable and slows (~2sec)

- Modified Sauvola's Algorithm by J. Zhou and Y. Liu -- Excellent results but slow (over 10 sec)

3.2.2 Locate the Finder Pattern and the Affine Transformation

Rows Search

When the picture is readable the first step is to locate the finder patterns located in three of the four corners [5]. This position detection pattern consists of a black-white-black-white-black sequence with the ratio of 1:1:3:1:1, this is the distance between A and B in the Figure below. In order to find this pattern in the picture the rows are read and the value is put into an array of five. As an example if the first pattern is:

[3, 4, 5, 1, 16].

This means that we have three black pixels in a row and then four white pixels etc. To see if the pattern is found, the middle value is divided by three and compared to the remaining values (in the example 3, 4, 1, 16), to see if the pattern is found. Otherwise the procedure is repeated until there is a match. To save memory, instead of making a new array the current array is just shifted to the right, placing the new value to the left (in the example in the place of the 3).

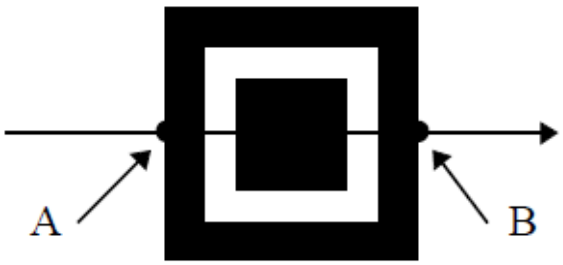


Figure 14

Research of the Center

When we have a match, we take the center of the matching sequence and we verify if ht sequence [1,1,3,1,1] is matched on the column. If it matches, we save the center of every matching sequence on the row in the black neighbor. And we do the same for the column. We have the result (red: centers of row matching, yellow: centers of column matching):

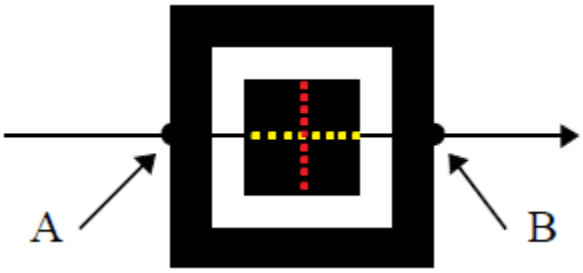


Figure 15

Now we can calculate the parameter of the line which goes through the red point and the line which goes through the yellow point and we are able to calculate the cross point of the two lines which is a very good estimation of the center of the finder pattern.
To finish we save the center of the pattern, the total number of point (red + yellow) and the estimate size of the finder pattern.

Avoid too Many Calculation

To avoid calculating the same pattern several times, we indicate the area where we have already researched a pattern so we can avoid these areas when the procedure of research of [1,1,3,1,1] is running.

Determine the Finder Pattern

We can determine more than 3 finder patterns in the picture, so we only have to keep the best one. For that we keep the finder patterns which have the biggest number of point (see above: red+yellow).

Determine the Orientation of the Picture

After finding the three best finder patterns, we need to put them in the good order to have the good orientation of the QR code. The first step is to find the top left finder pattern. For that we calculate the scalar product between the three vectors created by the three points that we have just found. Because the angle between the two vectors around the top left finder pattern is a multiple of 90º, the scalar product of the vectors should be the closest from zero. So we need now to put the two other points in the good order, for that we just need to verify the orientation of the orthogonal coordinate center in the top left point and set by the two other points.

Calculate the Length

We need now to evaluate the size of the QR code in number of module :

$$N = \frac{D + \frac{W_{UL}+W_{UR}}{2}}{\frac{W_{UL}+W_{UR}}{14}}$$

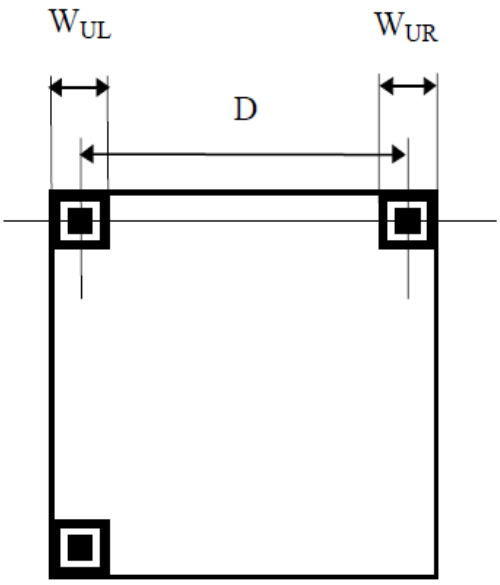


Figure 16

In addition, we know the length of a QR code has the form: $N = 4q + 1$, so we can select the better result in function of the previous calculation.

The Affine Transformation

When the three finder pattern points are obtained and the length of the QR code is known, we can determine an affine transformation to transform the relative coordinate in the QR code (x_1,y_1), into the real coordinate in the picture (x_2,y_2). The affine transformation is carried out according to the following equation.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

3.2.3. Decode the Version & Format Information

Case of a QR Length Inferior or Equal to 41

In this case, there is no information about the version in the QR code, so we determine it with:

$$Version = \frac{N - 17}{4}$$

The version will be between 1 and 6.

Case of a QR Length Superior or Equal to 45

In this case, we read the version information in the QR code:

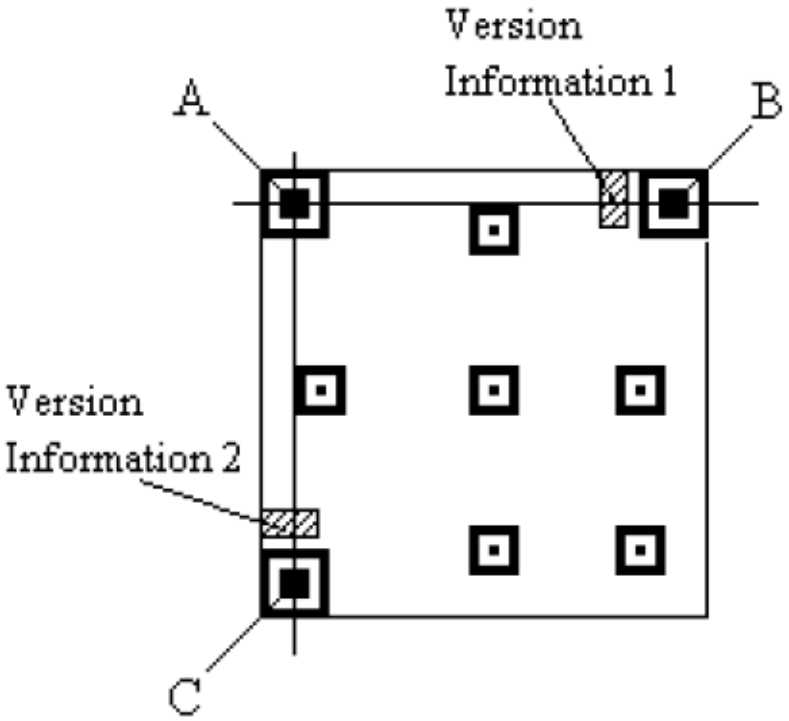


Figure 17

The version will be between 7 and 40.

Format Information

The format information is simply read around the finder patterns as explained in the encoding part.

3.2.3 Locate the Alignment Patterns

The Goal of the Alignment Patterns

After the finder pattern locations are found, the alignment patterns have to be located in the symbol to operate a better reading because a simple affine transformation is not enough to correct the perspective deformation of the QR code. Using the affine transformation is a good way to read the information around the finder patterns but for the rest of the QR code more precision will be needed.

Localization of the Alignment Patterns

For the top left, top right and bottom left positions, we will take the finder patterns. For the other, we approximate the location of the alignment patterns based on the standard and we are making a research in a local area of the two dimensional shape of the alignment pattern:

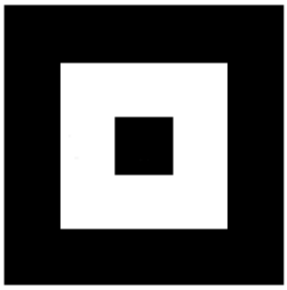


Figure 18: Alignment pattern

3.2.4 Reading of the Code

Next it is time to read the code. This is done in two steps:

First, we read all the information applying the mask and avoiding useless areas.

Second, we read the previous reading in the good order

Useless Area

First, we determine the area where the information is useless, like the different patterns, the version information, the timing line, etc. And the useful information will be in the rest of the QR code:

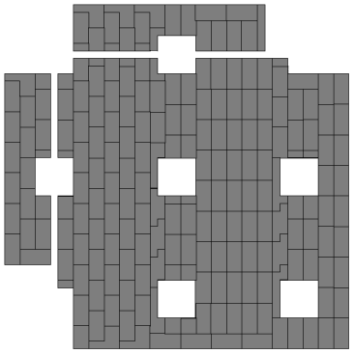


Figure 19

Reading and Applying the Mask

With the previous area, we can read all the information and apply the XOR mask at the same time using the Format information. We finally put it in a new array.

Snake Reading and Recombination of the Blocks

We can, now, aboard the second step by reading the previous array in the good order like in the encoding. After that, we recombine every block of byte in the good order in the goal to apply the Reed-Solomon algorithm for the decoding.

Reed-Solomon decoding

The Reed-Solomon algorithm for decoding operates in different steps:

- First Step: this algorithm is a linear encoding, so it checks if the message is in the alphabet. If yes, the message hasn't been deformed by the transmission.

- Second Step: in the case that the message is not in the alphabet, the algorithm tries to locate the errors in the message. For that, it resolves a linear system on the Galois Field to find a polynomial. The root of it gives it the position of errors.

- Third Step: the algorithm resolves a last linear system to find the correction link to the position of an error.

3.2.5 Colored Code

For decoding the colored QR codes, we just need to separate the three colored channel of the picture (red, green and blue) and we treat each channel separately like if it was a simple QR decode. We finally need to put the three decoded messages together to have the final message.
For the localization of the different patterns, we just need to do it for one of the channels and the two others have the same.

3.2.6 Multiple Decoding

For the multiple decoding, we need to decode several colored or standard QR code in real time. The emitter plots a code on its screen and updates it every second with the next code. The receptor needs to take a picture at least every second, to decode it and to put all the decoded messages together. For that we need to work with parallel computation using Threads. There are finally 5 threads.

Thread 1: the UI Thread

This is the main Thread. It manages the graphic interface and the callbacks on the messages.

Thread 2: the Preview Thread

This Thread manages the camera. It is saving a new picture when a one is called.

Thread 3: the PictureManager Thread

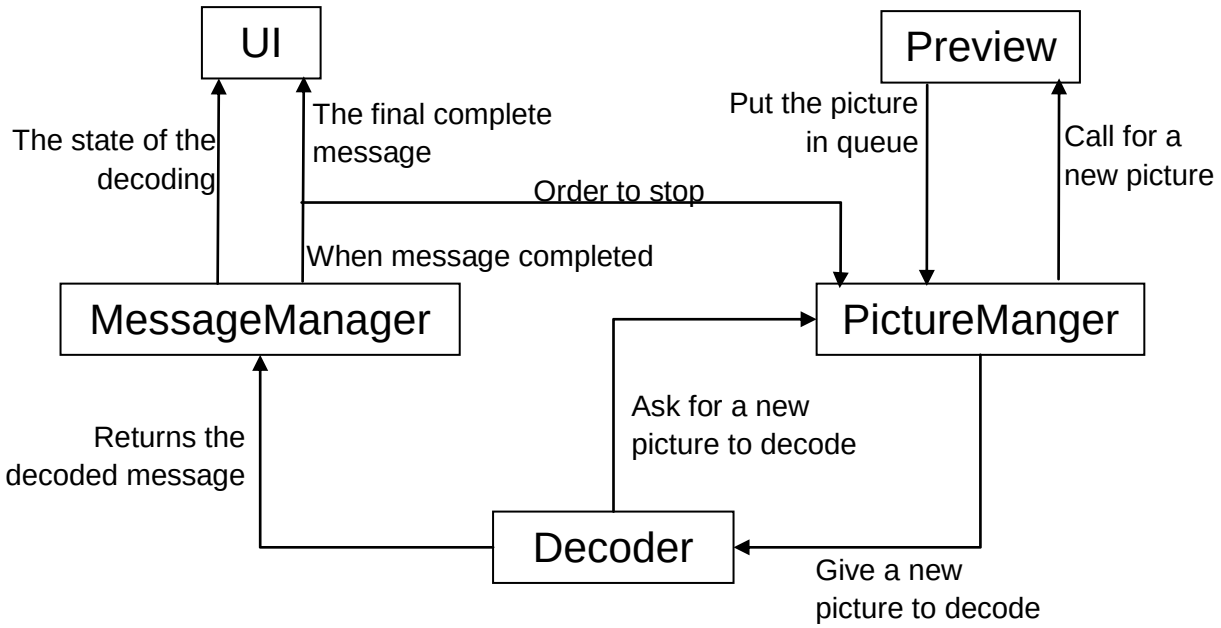
This Thread manages the queue of picture, it regulates the calling of picture to the Preview Thread and manages the states of the decoding.

Thread 4: the Decoder Thread

This Thread manages the decoding of a picture. It is called by the PictureManager and return the results to the last Thread (MessageManager).

Thread 5: the MessageManager Thread

This Thread manages the queue of decoded message and manages to put them in the good order. It also gives the order to end the decoding when all the codes have been decoded.



Combining of decoded messages

Each time the decoder finish to decode a code, it returns the decoded message to the message manager which verifies the markers, at the beginning of the message, to be sure that the message follows the protocol. It also checks the frame number, the total frame and the length of the message to put every message in the good order.

Frame indication

It is possible to check the frame number put in the QR code, before reading data, so if the code has already been decoded, we don't need to decode it to avoid useless calculation.

Synchronized Transmission

The transmission is not synchronized. The communication is only in one direction, so the encoder doesn't know what the situation of the decoder is.
But there is an option to synchronize the communication. With this option, the decoder emits a flash when it finished to decode the actual code and the encoder detects the flash with its light sensor, so it can change of code.
This solution is slower but avoids missing code.

Chapter 4

Implementation

4.1 Environment

4.1.1 Software

• **Matlab 2012b**. The initial tasks of encoding and decoding where implemented in Matlab. The message was encoded in Matlab and the uploaded to the Smartphone, and then another PC running Matlab encoded the message.

• **Eclipse SDK version 3.8.0.** The Android application was supposed to be written in Java. As Eclipse is a supported platform by Google it was used in this project.

• **Android Software Development Toolkit**. This toolkit contains some needed drivers and hardware platforms.

4.1.2 Hardware

• Smartphone's. The application have been run and tested on Samsung Galaxy S3 phones running Android Jelly Bean 4.2 as operating system.

• Computers. Four Dell laptops running Windows 7 have been used to write all the codes.

4.2 Encode

As there are a transmitter and receiver part, two phones have to be used, one for each purpose. Different graphical interfaces were implemented for each one.On the transmitter side the application gives the option to choose a text file that will be transferred and the user

can type in the name of the file. It also has a button to start the transmission. The transmission will begin only after a file has been selected.

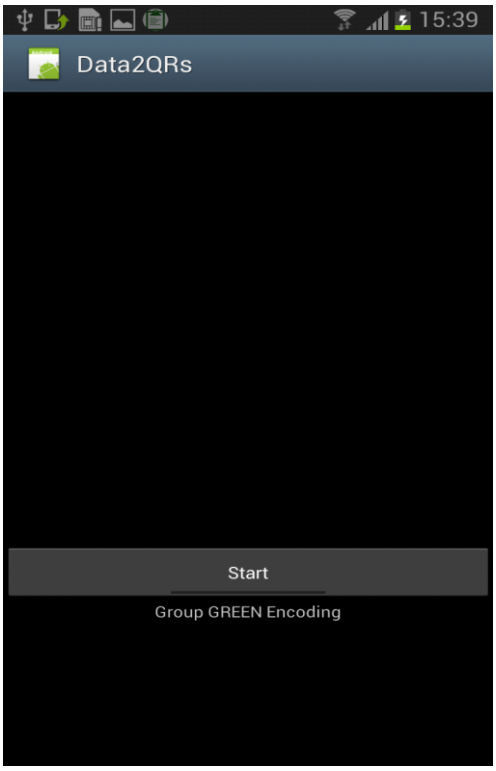


Figure 20. Launch the application, the GUI is displayed above. The file can be transmitted by pressing 'Start' .

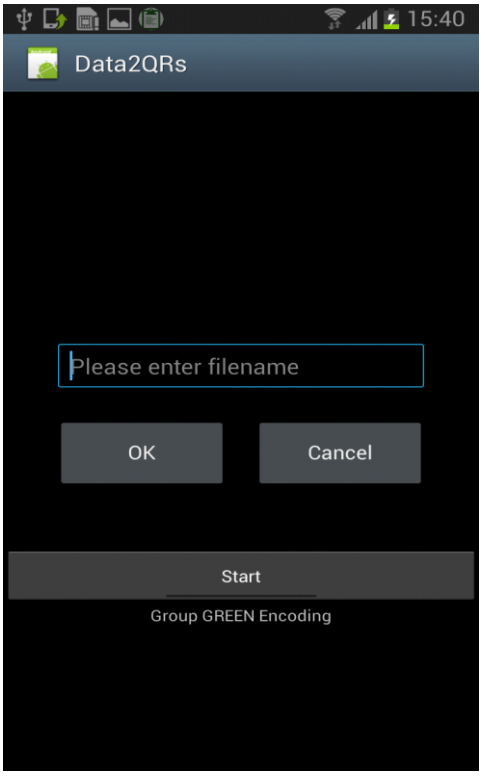


Figure 21. Input the file name by pressing Menu.

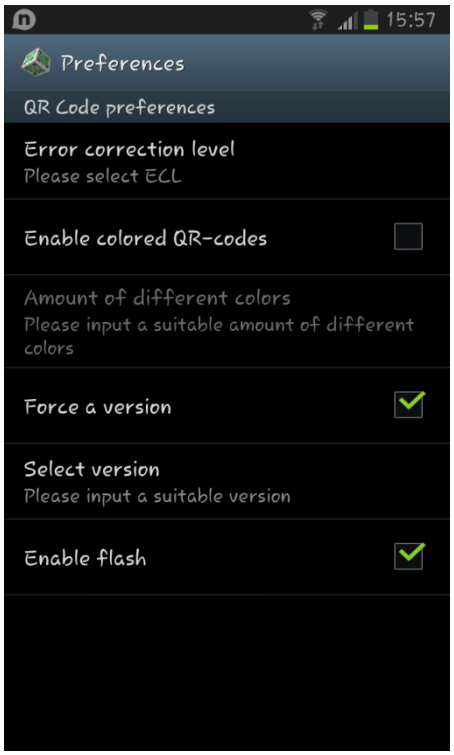


Figure 22

On the preference screen one can select version, Error correction level , colored QR-codes and flash. If nothing is selected a default values will be used.

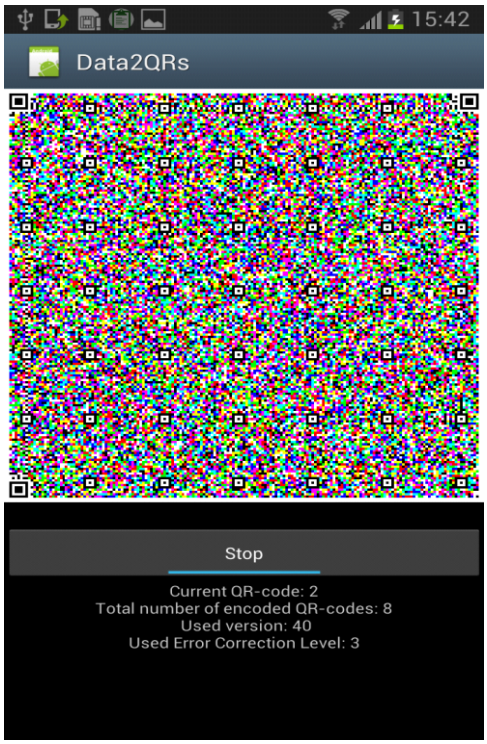
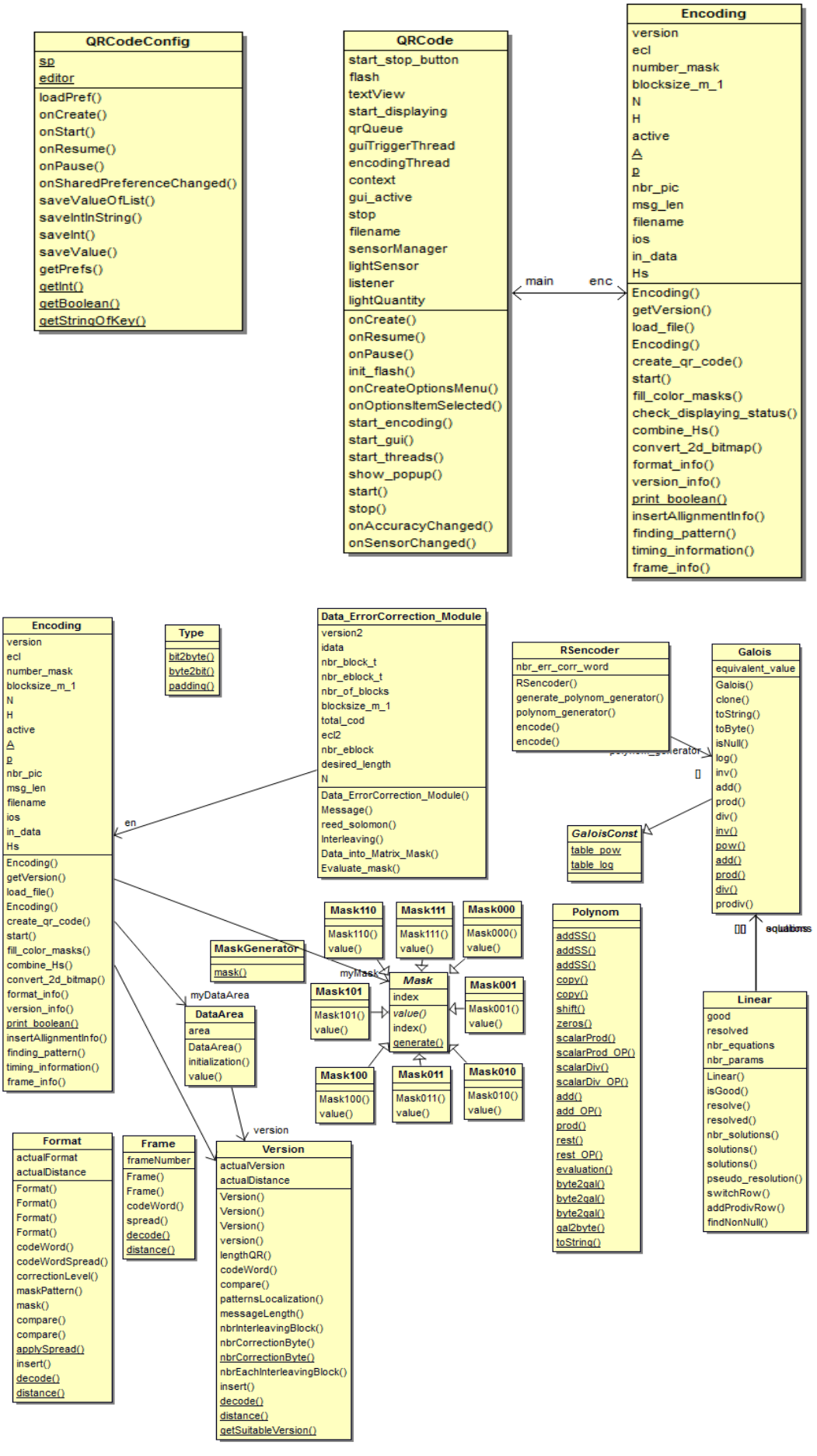


Figure 23

The application while transmitting codes

4.2.1 UML for Encoding



4.3 Decode

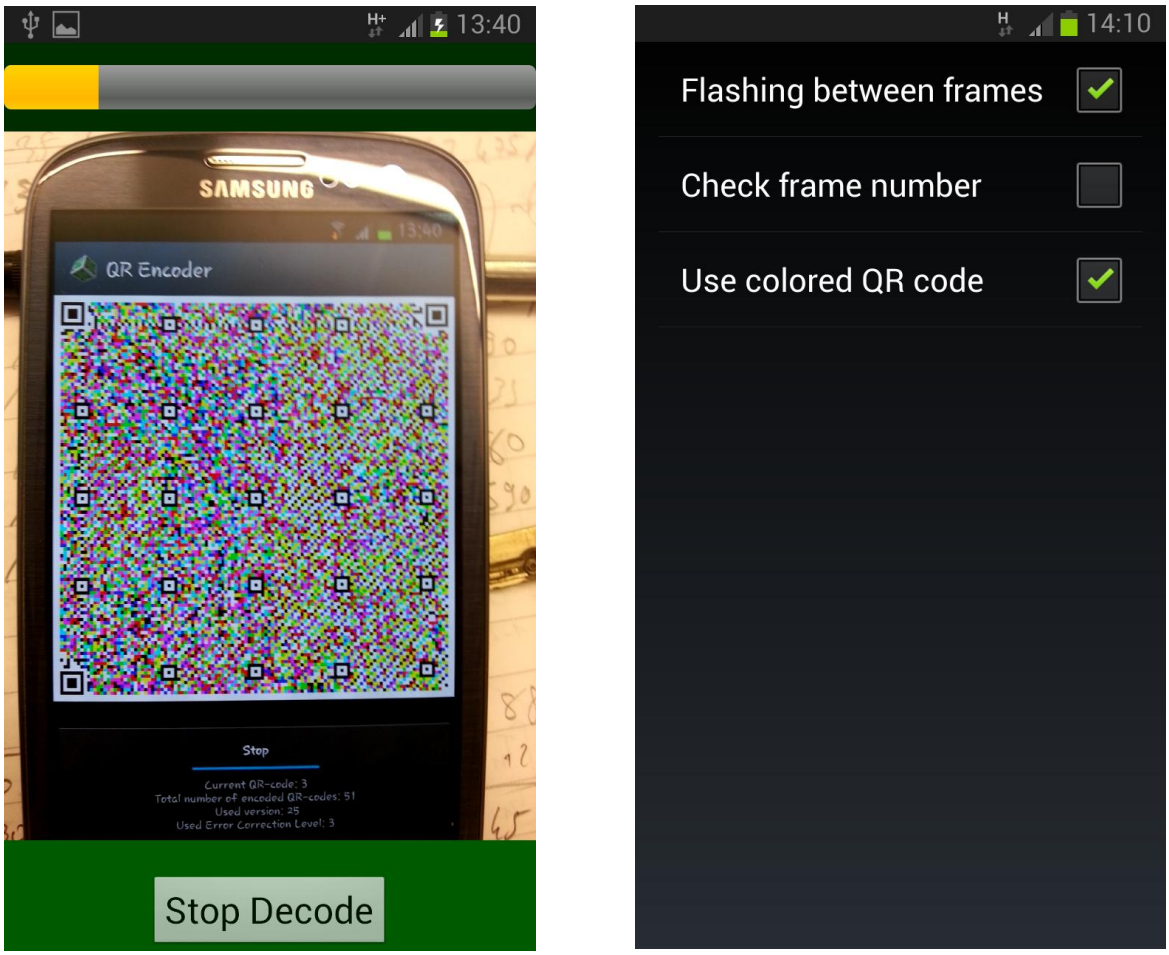


Figure 24 GUI of decoding application

Figure 24 is the graphical user interface of the decoding android application. The right part shows the options that can be selected. ‘Flashing between frames’ is used for synchronous mode. ‘Check frame number’ is used in asynchronous mode. The third option allows to decode black and white or colored codes.

4.3.1 UML for Decoding

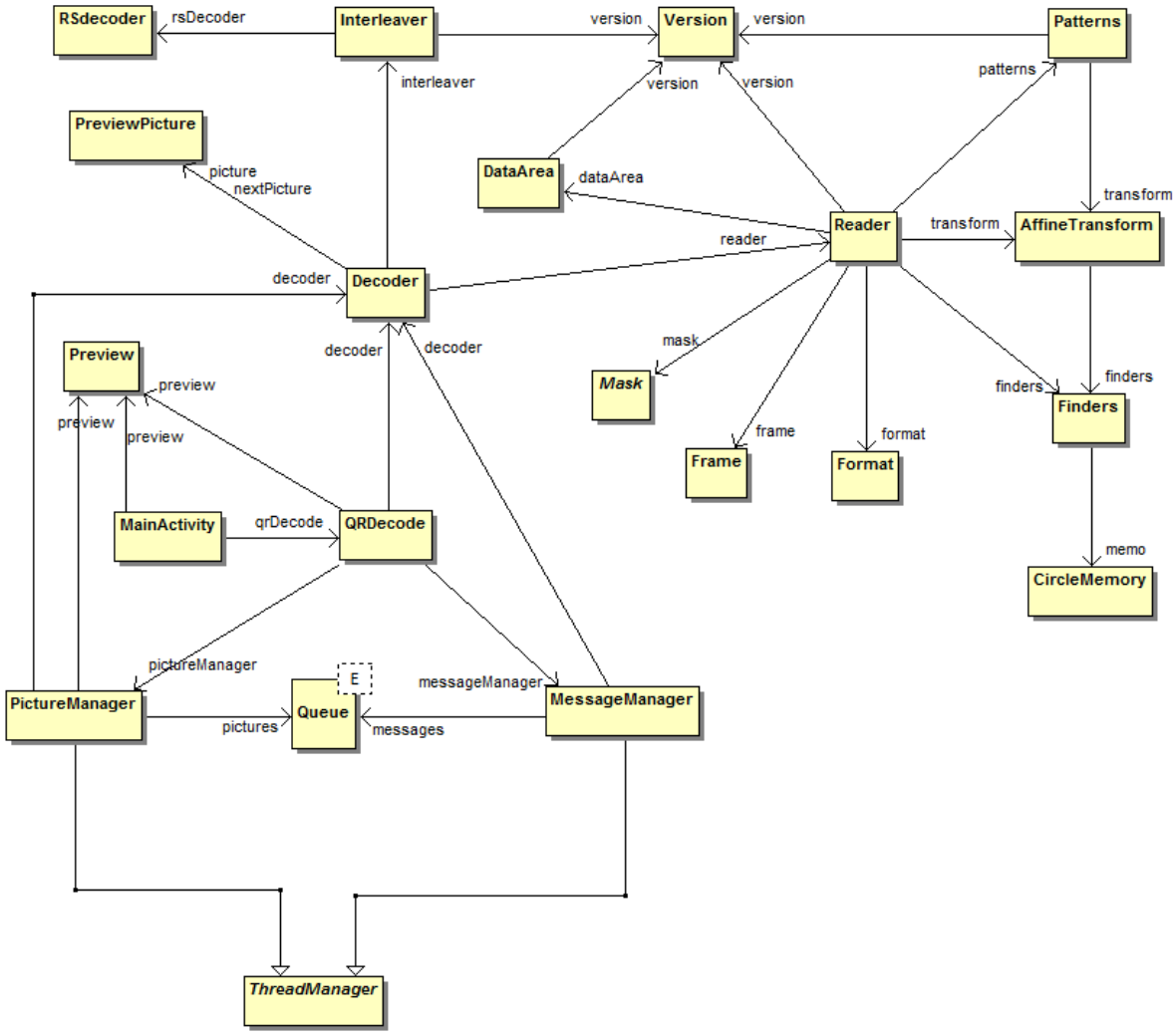


Figure25: UML of decoding

Principal Classes

MainActivity : Main class of the application, it initiates the activity and the other classes
Preview : Manages the phone's camera, the preview and creates **PreviewPicture** objects
QRDecode : Manages the all algorithm calling the other classes
Decoder : Manages the decoding of one **PreviewPicture**, apply the binarization
PictureManager : Manages the **Queue** of **PreviewPicture**, ask for new **PreviewPicture** and regulates the time between two **PreviewPicture**.
MessageManager : Manages the **Queue** of decoded messages put the messages together
PreviewPicture : Contains the values of a picture, can be black and white or colored

Reading QR code classes

Reader : Manages the all reading QR code in a binarized picture.
Finders : Research and sort the finding patterns
AffineTransform : Calculate the parameters of the affine transformation apply it.
Patterns : Research the alignment patterns.
DataArea : Gives the position, in the QR code, where is the message.
Mask : Gives the value of the mask in a given position in the QR code.

Information Classes

Version : Decodes and encodes the version and gives any information linked to the version.
Format : Decodes and encodes the format and gives any information linked to the format.
Frame : Decodes and encodes the frame number.

Message Treatment Classes

Interleaver : Apply the interleaving and the **RSDecoder** on a decoded message.
RSDecoder : Apply the Reed-Solomon algorithm decoding on a given message.

Utile Classes

Queue<E> : Manages a queue of any sort of elements.
ThreadManager : Abstract class creating and managing a **Thread** to run a loop action and control its start and stop.
CircleMemory : Manages an array of integer to be used like a fixed length queue

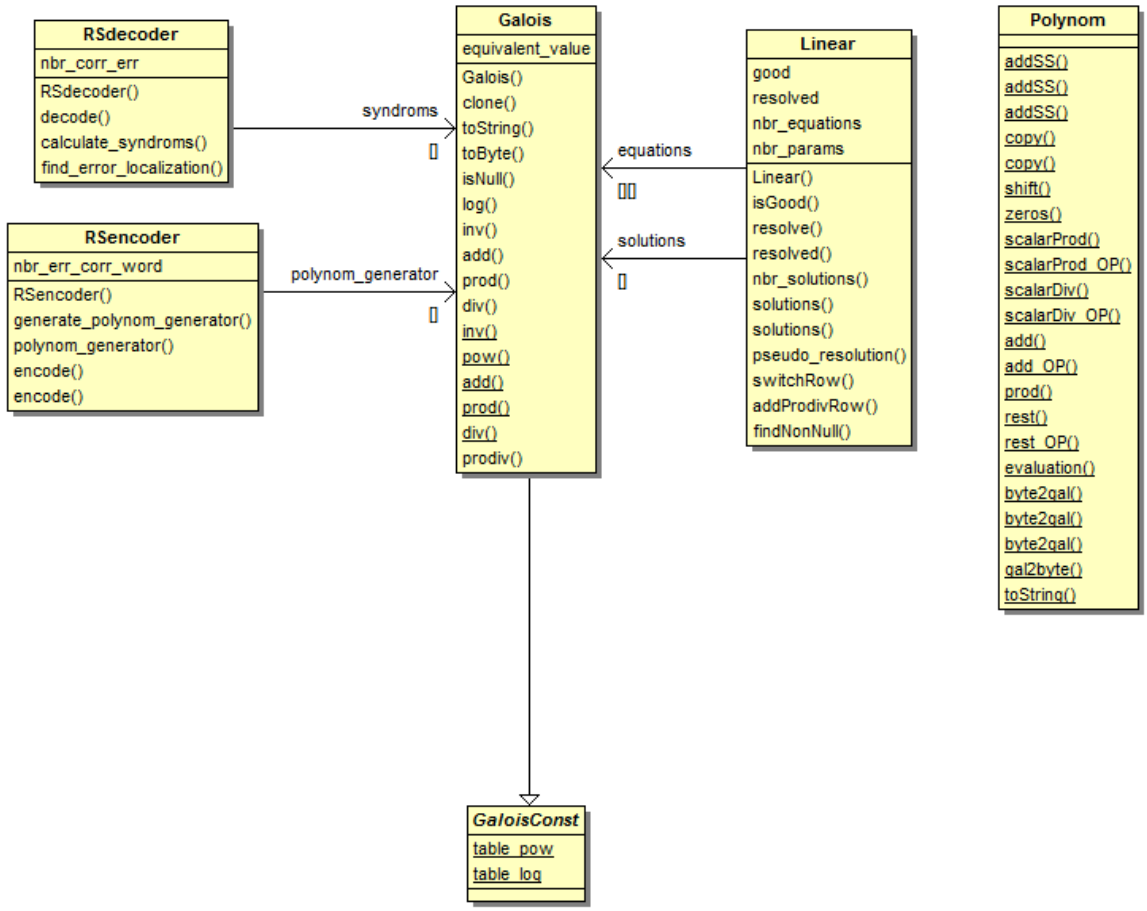


Figure 26: Reed Solomon UML

Classes

GaloisConst : Abstract class containing all the information on Galois field in static.
Galois : Class which represent a Galois number in the field Galois(256)
Polynom : Class of static function to manage **Galois** array like polynomial
Linear : Class to resolve a linear equation system on Galois field
RSDecoder : Manages the Reed-Solomon decoding on a given message
RSEncoder : Manages the Reed-Solomon encoding on a given message

Chapter 5

Results/Discussion

Data rate

- **Test the function using Matlab.**

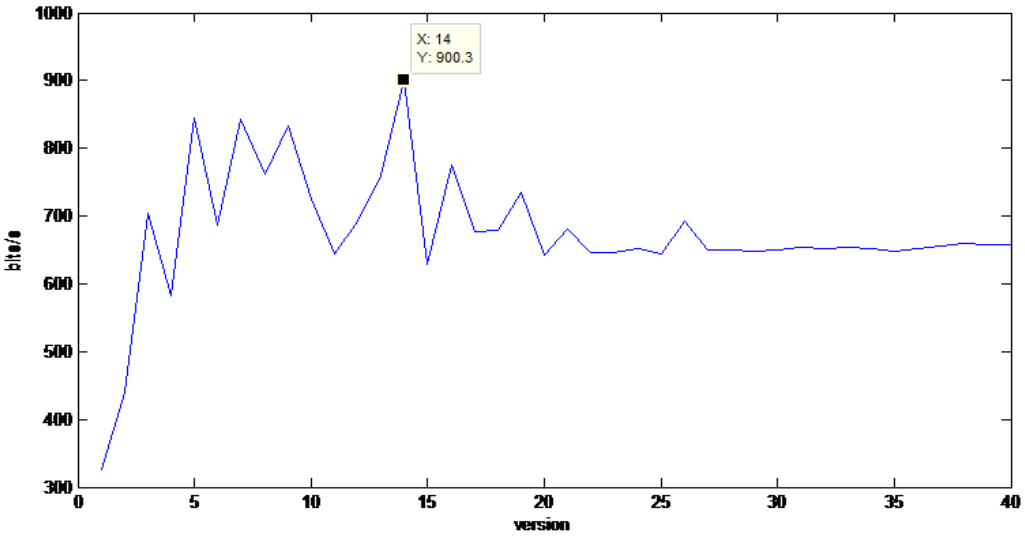


Figure 27: Encode rate for each version

By inputting different data 10 times, calculate the average time which is taken to generate the QR code for each version. From the figure above, we can see, the high version doesn't r enhance data rate that is due to the Matlab delay of R-S algorithm.

- The decode data rate for each version when the QR code is rotated.

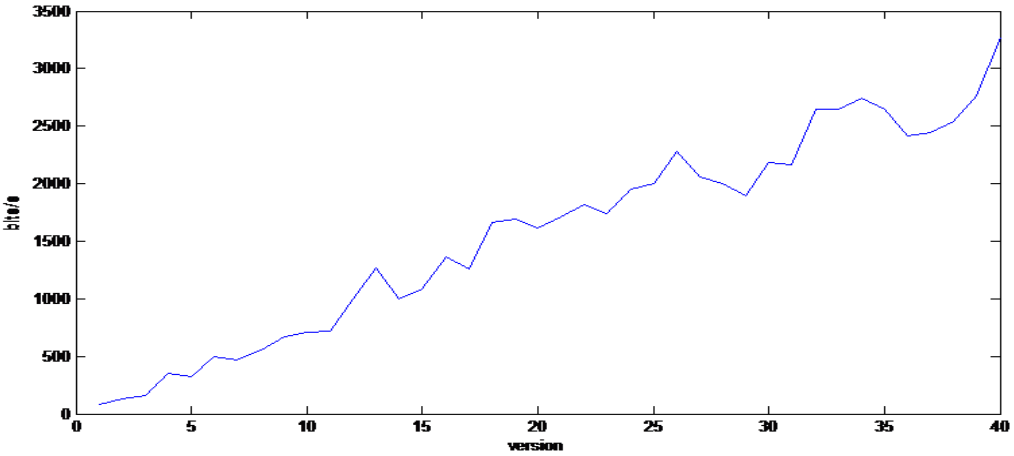


Figure 28: Decode data rate for each version with ideal images

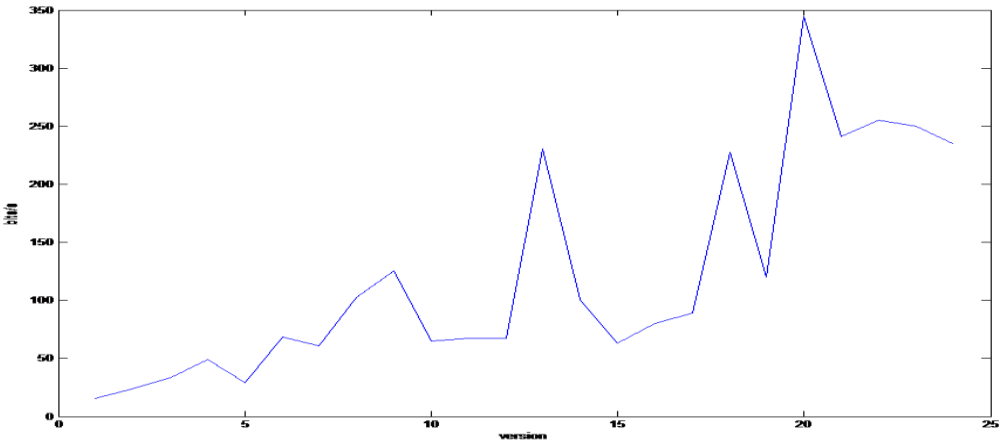


Figure 29: Decode data rate for each version with rotated images

- Compare the decode data rate of each version when the QR code is disturbed by the external inference with the **ideal curve**

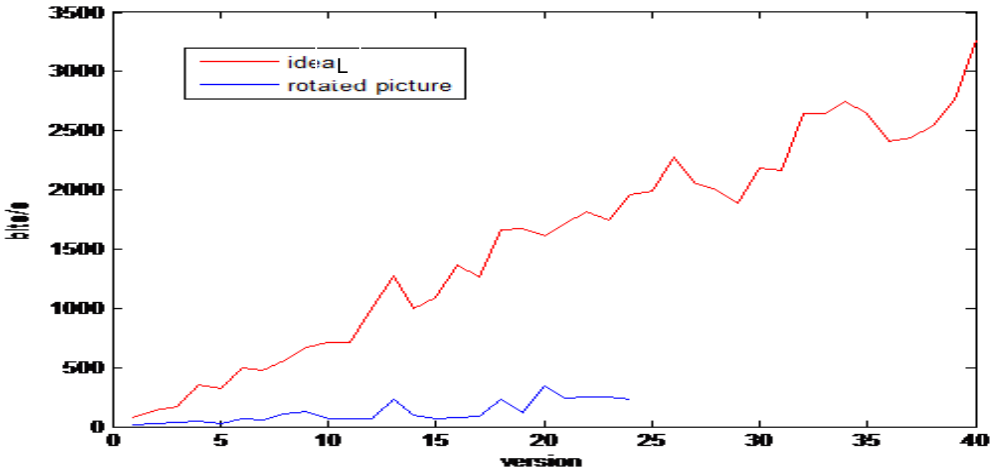


Figure 30: Decode data rate for each version

Test the function using Java.

- By inputting different data 10 times, calculate the average time which is taken to generate the QR code for each version.

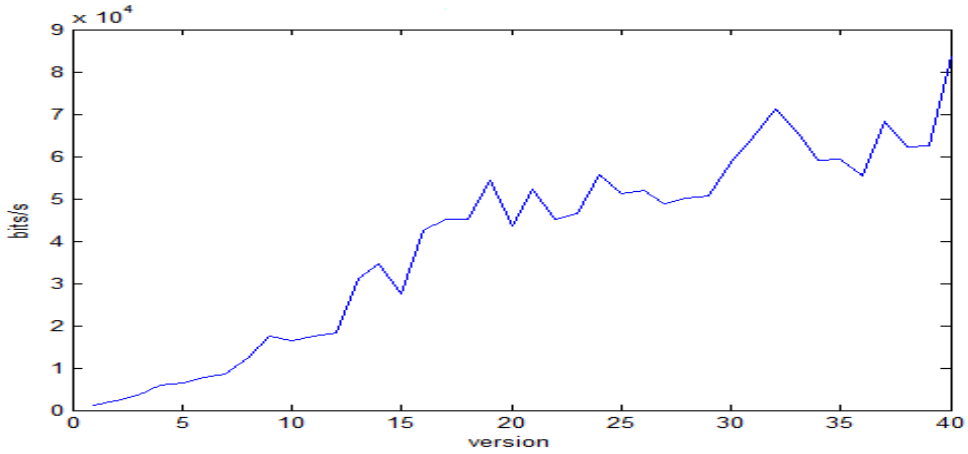


Figure 31: Encode data rate for each version

- The decode data rate for each version.

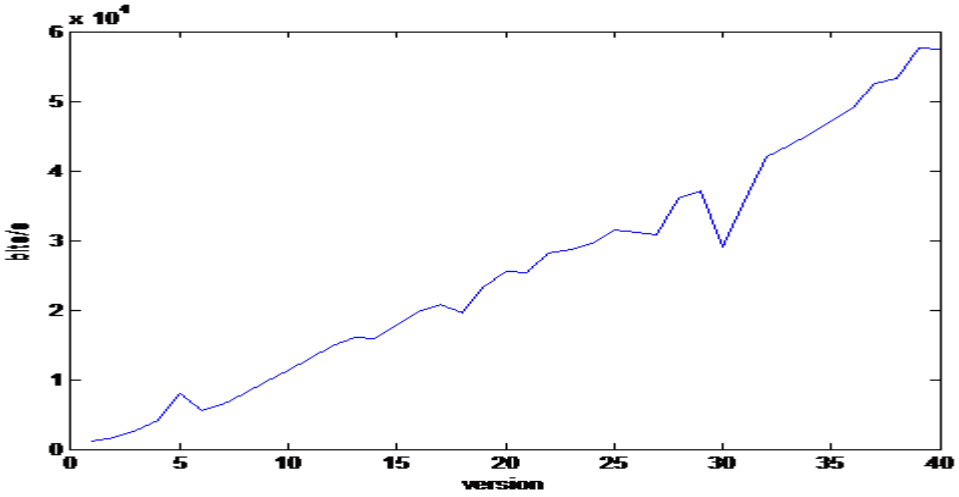


Figure 32: Decode data rate for each version

- The data rate of the total system (encode +decode+system channel (1secs)).

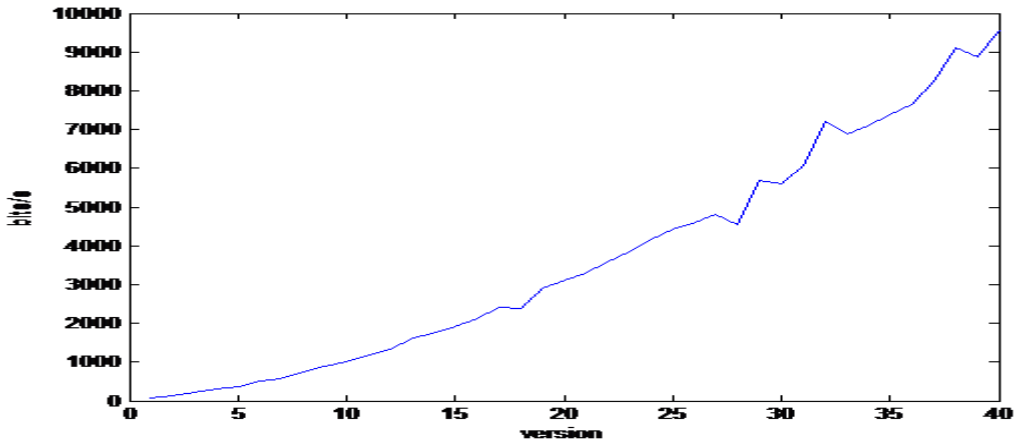


Figure 33: Data rate for each version

- The data bit can achieve 9-10 kbps when we use version 40.

.

- Due to the Matlab slow calculate speed, date rate is not that perfect when going to the high version. However, java could reach the high data rate.

- The qualities of the QR CODE image have influence on the bit rate of decoding part.

- To reach the best data rate, we need to use the version as high as possible.

Data rate of the android application

- Data rate of the application with different error correction level and asynchronous/synchronous mode.

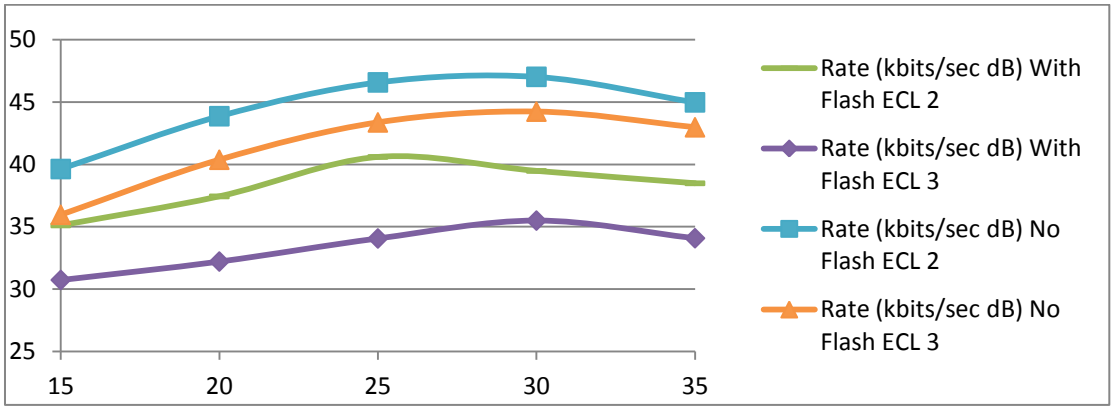


Figure 34: Data rate for each version

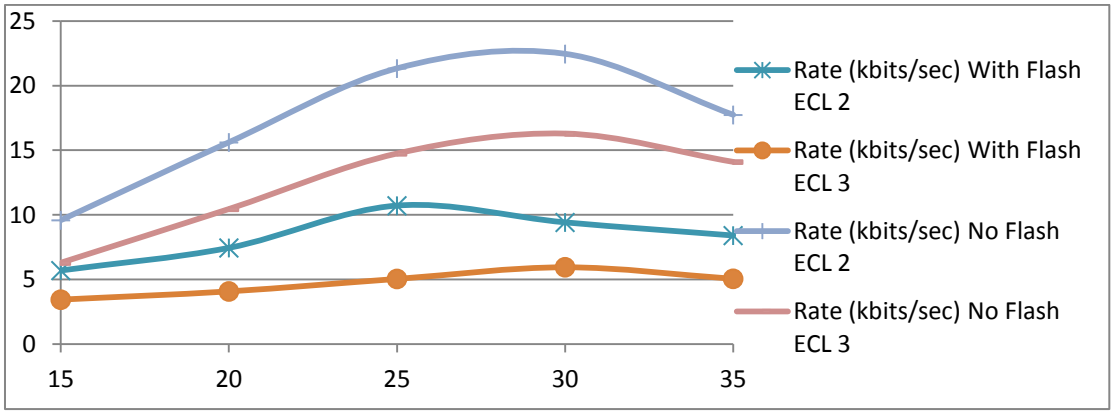


Figure 35: Data rate for each version

Version	Rate (kbits/sec)			
	With Flash		No Flash	
	ECL 2	ECL 3	ECL 2	ECL 3
15	5,693	3,440	9,583	6,281
20	7,443	4,078	15,609	10,442
25	10,713	5,047	21,328	14,738
30	9,417	5,952	22,453	16,282
35	8,399	5,058	17,736	14,092

Table 11: Data rate for each version

Chapter 6

Conclusion

A system that utilizes other means than usual radio frequencies has been implemented in this project. The method that has been developed uses modified black/white or color Quick Response codes.

A basic prototype was implemented in MATLAB, but the final system that runs on Android was developed in Java. The smart phones used in this project were Samsung Galaxy S3's.

The final model enables a file transfer of any type (image, audio, text and others). It is using 8 bits for frame number information which means maximum of 256 frames can be transmitted in one session. With this system, maximum possible size of the file that can be transmitted is approximately 754 Kb with the lowest error correction level. More frames can be transmitted by using more bits for frame number information. Data rate of up to 22.45kbits/sec is achieved with this system.

Two approaches were implemented for this visual based communication system i.e., asynchronous and synchronous. Synchronized transmission was more reliable but higher data rates were achieved with asynchronous mode.

Bibliography

[1] Federal Communication Commission “Using Wireless Devices on Airplanes”http

[2]L. Ahlin, J. Zander and B. Slimane.Principles of Wireless Communications.Studentlitteratur 2006

[3] http://www.norwegian.com/uk/travel-information/travel-services/wifi/

[4]“7 Benefits of QR Codes for Content Marketing or Inbound Marketing”http://seamlesssocial.com/7-benefits-of-qr-codes-for-content-marketing-or-inbound-marketing/

[5] J. Bernsen, “Dynamic thresholding of grey-level images,” in International Conferenceon Pattern Recognition, 1986, pp. 1251–1255.

[6] J. Zhou, Y. Liu, and P. Li, “Research on binarization of qr code image,” in MultimediaTechnology (ICMT), 2010 International Conference on. IEEE, 2010, pp.1–4.

[7] “ISO/IEC Information technology - Automatic identification and data capture techniques - Bar code symbology - QR Code,” 2000.

[8] QR Code Tutorial
“http://www.thonky.com/qr-code-tutorial/”

Appendix A [8]

Version	Number of Alignment Patterns	Row/Column coordinates of center module						
1	0	-						
2	1	6	18					
3	1	6	22					
4	1	6	26					
5	1	6	30					
6	1	6	34					
7	6	6	22	38				
8	6	6	24	42				
9	6	6	26	46				
10	6	6	28	50				
11	6	6	30	54				
12	6	6	32	58				
13	6	6	34	62				
14	13	6	26	46	66			
15	13	6	26	48	70			
16	13	6	26	50	74			
17	13	6	30	54	78			
18	13	6	30	56	82			
19	13	6	30	58	86			
20	13	6	34	62	90			
21	22	6	28	50	72	94		
22	22	6	26	50	74	98		
23	22	6	30	54	78	102		
24	22	6	28	54	80	106		
25	22	6	32	58	84	110		
26	22	6	30	58	86	114		
27	22	6	34	62	90	118		
28	33	6	26	50	74	98	122	
29	33	6	30	54	78	102	126	
30	33	6	26	52	78	104	130	
31	33	6	30	56	82	108	134	
32	33	6	34	60	86	112	138	
33	33	6	30	58	86	114	142	
34	33	6	34	62	90	118	146	
35	46	6	30	54	78	102	126	150
36	46	6	24	50	76	102	128	154
37	46	6	28	54	80	106	132	158
38	46	6	32	58	84	110	136	162
39	46	6	26	54	82	110	138	166
40	46	6	30	58	86	114	142	170

Appendix B

version	Total number of codeword	L	Error correction Blocks	M	Error correction Blocks	Q	Error correction Blocks	H	Error correction Blocks
1	20	7	1 0	10	1 0	13	1 0	17	1 0
2	38	10	1 0	16	1 0	22	1 0	28	1 0
3	64	15	1 0	26	1 0	36	2 0	44	2 0
4	94	20	1 0	36	2 0	52	2 0	64	4 0
5	128	26	1 0	48	2 0	72	2 2	88	2 2
6	166	36	2 0	64	4 0	96	4 0	112	4 0
7	190	40	2 0	72	4 0	108	4 2	130	1 4
8	236	48	2 0	88	2 2	132	2 4	156	2 4
9	286	60	2 0	110	2 3	160	4 4	192	4 4
10	340	72	2 2	130	1 4	192	2 6	224	2 6
11	398	80	2 2	150	4 1	224	4 4	264	8 3
12	460	96	2 2	176	2 6	260	6 4	308	4 7
13	526	104	4 0	198	1 8	288	4 8	352	3 12
14	575	120	1 3	216	5 4	320	5 11	384	5 11
15	649	132	1 5	240	5 5	360	7 5	432	7 11
16	727	144	1 5	280	3 7	408	2 15	480	13 3
17	809	168	5 1	308	1 10	448	15 1	532	17 2
18	895	180	1 5	338	4 9	504	1 17	588	19 2
19	985	196	4 3	364	11 3	546	4 17	650	16 9

20	1079	224	5 3	416	13 3	600	5 15	700	10 15
21	1150	224	4 4	442	17 0	644	6 17	750	6 19
22	1252	252	7 2	476	17 0	690	16 7	816	34 0
23	1358	270	5 4	504	14 4	750	14 11	900	14 16
24	1468	300	6 6	560	14 6	810	16 11	960	2 30
25	1582	312	4 8	588	13 8	870	22 7	1050	13 22
26	1700	336	2 10	644	4 19	952	6 28	1110	4 33
27	1822	360	4 8	700	3 22	1020	26 8	1200	28 12
28	1915	390	10 3	728	23 3	1050	31 4	1260	31 11
29	2045	420	7 7	784	7 21	1140	37 1	1350	26 19
30	2179	450	10 5	812	10 19	1200	25 15	1440	25 23
31	2317	480	3 13	868	29 2	1290	1 42	1530	28 23
32	2459	510	17 0	924	23 10	1350	35 10	1620	35 19
33	2605	540	1 17	980	21 14	1440	19 29	1710	46 11
34	2755	570	6 13	1036	23 14	1530	7 44	1800	1 59
35	2870	570	7 12	1064	26 12	1590	14 39	1890	41 22
36	3028	600	14 6	1120	34 6	1680	10 46	1980	64 2
37	3190	630	4 17	1204	14 29	1770	10 49	2100	46 24
38	3356	660	18 4	1260	32 13	1860	14 48	2220	32 42
39	3526	720	2 20	1316	7 40	1950	22 43	2310	67 10
40	3700	750	6 19	1372	31 18	2040	34 34	2430	61 20