

The copyright © of this thesis belongs to its rightful author and/or other copyright owner. Copies can be accessed and downloaded for non-commercial or learning purposes without any charge and permission. The thesis cannot be reproduced or quoted as a whole without the permission from its rightful owner. No alteration or changes in format is allowed without permission from its rightful owner.



**USING RGB COLOUR COMBINATION IN
COLOURED QUICK RESPONSE (QR) CODE
ALGORITHM TO ENHANCE QR CODE
CAPACITY**



AZIZI BIN ABAS

UUM
Universiti Utara Malaysia

**DOCTOR OF PHILOSOPHY
UNIVERSITI UTARA MALAYSIA
2018**

**USING RGB COLOUR COMBINATION IN
COLOURED QUICK RESPONSE (QR) CODE
ALGORITHM TO ENHANCE
QR CODE CAPACITY**



AZIZI BIN ABAS

UUM
Universiti Utara Malaysia

**Thesis Submitted to
Awang Had Salleh Graduate School of Arts and Sciences
Universiti Utara Malaysia,
In Fulfillment of the Requirement for the Degree of Doctor Philosophy**

Dissertation



Awang Had Salleh
Graduate School
of Arts And Sciences

Universiti Utara Malaysia

PERAKUAN KERJA TESIS / DISERTASI (Certification of thesis / dissertation)

Kami, yang bertandatangan, memperakukan bahawa
(We, the undersigned, certify that)

AZIZI ABAS

calon untuk Ijazah
(candidate for the degree of)

PhD

telah mengemukakan tesis / disertasi yang bertajuk:
(has presented his/her thesis / dissertation of the following title):

**"USING RGB COLOUR COMBINATION IN COLOURED QUICK RESPONSE (QR) CODE
ALGORITHM TO ENHANCE QR CODE CAPACITY"**

seperti yang tercatat di muka surat tajuk dan kulit tesis / disertasi.
(as it appears on the title page and front cover of the thesis / dissertation).

Bahawa tesis/disertasi tersebut boleh diterima dari segi bentuk serta kandungan dan meliputi bidang ilmu dengan memuaskan, sebagaimana yang ditunjukkan oleh calon dalam ujian lisan yang diadakan pada : **31 Mei 2018**.

*That the said thesis/dissertation is acceptable in form and content and displays a satisfactory knowledge of the field of study as demonstrated by the candidate through an oral examination held on:
May 31, 2018.*

Pengerusi Viva:
(Chairman for VIVA)

Assoc. Prof. Dr. Osman Ghazali

Tandatangan
(Signature)

Pemeriksa Luar:
(External Examiner)

Prof. Dr. Rusli Abdullah

Tandatangan
(Signature)

Pemeriksa Luar:
(External Examiner)

Assoc. Prof. Dr. Mohd Pouzi Hamzah

Tandatangan
(Signature)

Nama Penyelia/Penyelia-penyelia:
(Name of Supervisor/Supervisors)

Assoc. Prof. Dr. Yuhanis Yusof

Tandatangan
(Signature)

Nama Penyelia/Penyelia-penyelia:
(Name of Supervisor/Supervisors)

Dr. Farzana Kabir Ahmad

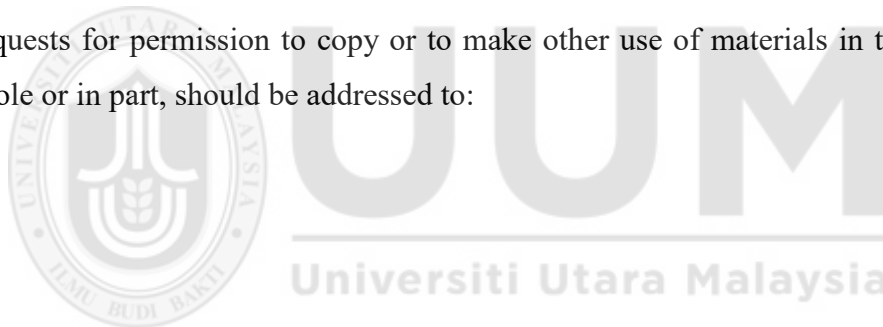
Tandatangan
(Signature)

Tarikh:
(Date) **May 31, 2018**

Permission to Use

In presenting this thesis in fulfilment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the Universiti Library may make it freely available for inspection. I further agree that permission for the copying of this thesis in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use that may be made of any material from my thesis.

Requests for permission to copy or to make other use of materials in this thesis, in whole or in part, should be addressed to:



Dean of Awang Had Salleh Graduate School of Arts and Sciences

UUM College of Arts and Sciences

Universiti Utara Malaysia

06010 UUM Sintok

Abstrak

Kod Respons Pantas (QR) ialah kod bar dua dimensi yang menyimpan aksara dan boleh dibaca oleh mana-mana kamera telefon pintar. Kod QR mempunyai keupayaan untuk mengekod pelbagai format data dan bahasa. Walau bagaimanapun, Kod QR hitam dan putih yang sedia ada menyediakan penyimpanan data yang terhad. Walaupun terdapat penyelidikan mengenai Kod QR berwarna untuk meningkatkan kapasiti penyimpanan, keperluan untuk kapasiti data yang lebih besar oleh pengguna terus meningkat. Oleh itu, tesis ini mencadangkan algoritma Kod QR berwarna yang menggunakan kombinasi warna merah, hijau dan biru (RGB) untuk membolehkan storan data yang lebih besar. Algoritma yang dicadangkan mengintegrasikan penggunaan teknik mampatan, pemultipleksan, dan pelbagai lapis dalam pengekodan dan penyahkodan Kod QR. Tambahan pula, ia juga memperkenalkan algoritma pengekodan/penyahkodan separa yang membolehkan pemanipulasi data. Algoritma yang merangkumi proses pengekodan dan penyahkodan adalah berdasarkan teknik warna RGB, yang digunakan untuk membuat Kod QR berwarna berkapasiti tinggi. Ini direalisasikan dalam eksperimen yang menyimpan aksara Kod Piawai Amerika bagi Saling Tukar Maklumat (ASCII). Aksara teks ASCII digunakan sebagai input dan prestasi diukur dengan bilangan aksara yang boleh disimpan di dalam Kod QR hitam dan putih versi 40 (iaitu tanda aras) dan juga Kod QR berwarna. Metrik eksperimen lain termasuk peratusan aksara yang hilang, bilangan Kod QR yang dihasilkan, dan masa berlalu untuk membuat Kod QR. Hasil simulasi menunjukkan bahawa algoritma yang dicadangkan menyimpan 29 kali lebih banyak aksara daripada Kod QR hitam dan putih dan 9 kali lebih banyak daripada Kod QR berwarna lain. Oleh itu, ini menunjukkan bahawa Kod QR yang berwarna mempunyai potensi untuk menjadi penyimpanan mini data kerana ia tidak bergantung kepada sambungan internet.

Kata kunci: Kod respons pantas, Kod bar, Pencapaian maklumat, Penyimpanan data, Warna RGB

Abstract

A Quick Response (QR) Code is a two-dimensional barcode that stores characters and can be read by any smartphone camera. The QR code has the capability to encode various data formats and languages; nevertheless, existing black and white QR code offers limited data storage. Even though there exist research on coloured QR Code to increase the storage capacity, requirement for larger data capacity by end user keep increasing. Hence, this thesis proposes a coloured QR Code algorithm which utilizes RGB colour combination to allow a larger data storage. The proposed algorithm integrates the use of compression, multiplexing, and multilayer techniques in encoding and decoding the QR code. Furthermore, it also introduces a partial encoding/decoding algorithm that allows the stored data to be manipulated. The algorithm that includes encoding and decoding processes is based on the red, green, and blue (RGB) colour techniques, which are used to create high capacity coloured QR code. This is realised in the experiments that store American Standard Code for Information Interchange (ASCII) characters. The ASCII text characters are used as an input and performance is measured by the number of characters that can be stored in a single black and white QR code version 40 (i.e. the benchmark) and also the coloured QR code. Other experiment metrics include percentage of missing characters, number of produced QR code, and elapsed time to create the QR code. Simulation results indicate that the proposed algorithm stores 29 times more characters than the black and white QR code and 9 times more than other coloured QR code. Hence, this shows that the coloured QR Code has the potential of becoming a useful mini-data storage as it does not rely on internet connection.

Keywords: Quick Response Code, Barcode, Information retrieval, Data storage; RGB colours.

Acknowledgement

First of all, I would like to thank my supervisor, Assoc. Prof. Dr. Yuhanis and Dr. Fauzana Kabir Ahmad for giving me the opportunity to pursue this long and rewarding journey, and for their help and guidance.

Most of all, I would like to thank my mother, Puteh Ismail, for her unconditional support in every way and for her trust and love. To my father, Abas Ismail, in which I undoubtedly see myself every day and who from heaven helped me to achieve what I once saw so far away.

To my wife Zuraida Saad, my daughter Alia Qistina and my son Adib Qayyum, thank you for your understanding, for loving me, and for being there for me all these years. This was a very special period in my life in which I have great successes and catastrophic failures, in which I learned about myself and the others, and in which I was reminded that what matters is always the journey and not the destination. None of these could have been possible without all of you, and for that I just would like to say thank you. Also, to all my friends in UUM who never let me forget.

Let us close this chapter today and start a new one in this story, without forgetting what I learned, what I am, and what I want to become.

Table of Contents

Dissertation	i
Permission to Use.....	ii
Abstrak	iii
Abstract	iv
Acknowledgement.....	v
Table of Contents	vi
List of Tables.....	ix
List of Figures	xii
List of Appendices	xvi
List of Abbreviations.....	xvii
CHAPTER ONE INTRODUCTION	1
1.1 Introduction.....	1
1.2 Problem Statement	7
1.3 Research Questions	10
1.4 Objectives.....	10
1.5 Significance of the Study	11
1.6 Research Scope	12
1.7 -Organisation of the Thesis	13
CHAPTER TWO LITERATURE REVIEW	16
2.1 QR Code.....	16
2.1.1 QR Codes Architecture Structure	23
2.1.2 Types of QR Code	25
2.2 Coloured Barcode.....	27
2.3 Coloured QR Code.....	31
2.3.1 Colour Depth.....	33
2.3.2 Colour Model	34

2.3.3 Pixelation	38
2.3.4 Multilayer Colour.....	39
2.3.5 Multiplexing and Demultiplexing.....	58
2.3.6 Compression	69
2.3.7 Hybrid Extension	74
2.3.8 Structured Append	79
2.4 Combination Techniques of QR Code Data Capacity	81
2.5 Summary	83
CHAPTER THREE RESEARCH FRAMEWORK	84
3.1 Research Methodology.....	84
3.1.1 Phase One.....	84
3.1.2 Phase Two	90
3.1.3 Phase Three	92
3.2 Summary	95
CHAPTER FOUR ARCHITECTURE OF PROPOSED COLOURED QR CODE	97
4.1 Encode Algorithmn	97
4.1.1 Encode Module.....	98
4.1.2 Encoding Steps.....	98
4.2 Decode Algorithmn.....	116
4.2.1 Decoding QR Code	116
4.2.2 Decoding Steps	117
4.3 Partial Extraction Algorithm	130
4.3.1 Level 1 Decoding Module.....	131
4.3.2 Level 1 Re-Encoding Module	135
4.3.3 Level 2 Decoding Module.....	137
4.3.4 Level 2 Re-Encoding Module	141
4.4 Summary	144
CHAPTER FIVE FINDING	145
5.1 Encode Experiment	145
5.2 Encode Modules Experiment Result.....	145

5.2.1 Overall Encode Experiment Result.....	152
5.3 Decode Experiment.....	156
5.3.1 Decode Modules Experiment Result	156
5.3.2 Calculation of Total Black and White QR Codes	168
5.4 Partial Extraction Levels	169
5.4.1 Partial Extraction Levels Experiment Result	170
5.5 Comparison With Existing QR code.....	182
5.6 Summary	185
CHAPTER SIX CONCLUSION	188
6.1 Summary of the Thesis	188
6.2 Encoding Design and Development Algorithmn	188
6.3 Decoding Design and Devopment Algorithm.....	191
6.4 Partial Extraction Decode and Re-encode Design and Development.....	193
6.5 Contribution	197
6.5.1 The Model.....	198
6.6 Limitation.....	201
6.7 Future Work.....	202
6.8 Summary	205
REFERENCES.....	207

List of Tables

Table 2.1:	Data density comparison between some 2D barcodes printed in 600 dpi (Courtesy: Melgar & Santander (2016)).....	18
Table 2.2:	The size and data capacity for different versions of QR code (Source: Garateguy, 2014).	26
Table 2.3:	A list of all the difference between colour depths.	33
Table 2.4:	Example of saturated green in different RGB notations.	36
Table 2.5:	The result of the scan process time in msec for QR code and HCC2D (Source: Grillo et al., 2010).	44
Table 2.6:	The identified information of QR code based on key elements.	45
Table 2.7:	The future research, advantages, and disadvantages.	47
Table 2.8:	The summary of multiplexing and demultiplexing methods of coloured QR code concepts.	61
Table 2.9:	The future research, advantages, and disadvantages.	62
Table 2.10:	The special symbols used for each pattern (Vongpradhip, 2013).	65
Table 2.11:	Example of distinct colour requirements for QR code multiplexing.	66
Table 2.12:	The normalised values of RGB combination for coloured QR.	67
Table 2.13:	The possibility problem experience if the priority exchange is implemented.	74
Table 2.14:	The processing time of encoding and decoding (Courtesy: Galiyawala & Pandya (2015)).	82
Table 3.1:	Maximum number of characters based on error correction level.	93
Table 4.1:	Module index number identification for detailed encoding process.	100
Table 4.2:	The complete character code map for ASCII printable characters.	102
Table 4.3:	The minimum character's total amount value from 20 times repeated experiment with error correction level H (Abas et al., 2017).	106
Table 4.4:	The amount of characters that can be stored in black and white QR code version 40 by character type (Courtesy: Wikipedia (2007)).	107
Table 4.5:	The maximum total characters stored in the QR code by error level (Abas et al., 2017).	108

Table 4.6:	The characters' file allocation.	111
Table 4.7:	The index number identification for decoding module.	120
Table 4.8:	The experiment of elapsed time order by error correction level.	122
Table 4.9:	Decimal to binary process.	124
Table 4.10:	The elapsed time of decoding demultiplexing process.....	127
Table 4.11:	The elapsed time of decompression process.	130
Table 4.12:	List of tasks for partial execution decoding level 1 module.....	134
Table 4.13:	List of tasks for partial extraction re-encoding level 1 module.....	137
Table 4.14:	List of tasks for partial execution decoding level 2 module.....	140
Table 4.15:	List of tasks for partial extraction re-encoding level 2 module.....	143
Table 5.1:	The maximum number of characters stored in each QR code version 40.	146
Table 5.2:	The size of the text file.	147
Table 5.3:	Amount of characters encoded based on the sequence of compression,multiplexing and multilayer.	148
Table 5.4:	The comparison of total characters in black and white QR code by type of characters.....	149
Table 5.5:	The result of total characters during Base64 encoding (before) and decoding (after) processes.	150
Table 5.6:	The elapsed time of encoding compression process.....	150
Table 5.7:	The elapsed time of encoding multiplexing process.	151
Table 5.8:	The result of multilayer process in second and millisecond.....	152
Table 5.9:	The elapsed time of encoding process.....	154
Table 5.10:	The difference of text capacity between QR code version 40 and proposed coloured QR code.	155
Table 5.11:	The compilation of elapsed time of overall decoding processes.....	157
Table 5.12:	The summary of processing time of decoding by Galiyawala and Pandya (Courtesy: Galiyawala & Pandya (2014)).	158
Table 5.13:	The normal QR code version 40 and compression tool (GZip) via binary to text encode/decode gap and percentage of compression order by error correction level.	160

Table 5.14:	The maximum total characters stored in QR code version 40 by error level with multiple compression tools without encoder/decoder.	160
Table 5.15:	The total character storage of 1, 8, 24, and N units of black and white QR codes after completion of compression process and binary to text decoding process.	161
Table 5.16:	The calculation or simulation of the outcome of total character order by error correction level from 24 and above units of black and white to 3 monocoloured QR codes (red, green, and blue).	163
Table 5.17:	The simulation in increment of channel using RGB model with 8-bit colour depth order by error correction level.	165
Table 5.18:	The simulation in increment of channel using RGB model with 10-bit colour depth order by error correction level.	166
Table 5.19:	The simulation in increment of channel using RGB model with 16-bit colour depth order by error correction level.	166
Table 5.20:	The simulation in increment of channel using RGB model with 24-bit colour depth order by error correction level.	167
Table 5.21:	The simulation in increment of channel using RGB model with 80-bit colour depth order by error correction level.	168
Table 5.22:	The comparison between benchmark and proposed techniques in level 1 of decoding process. Level 1(Decode).	178
Table 5.23:	The comparison between benchmark and proposed techniques in level 1 of re-encoding process. Level 1(Re-encode).	179
Table 5.24:	The comparison between benchmark and proposed techniques in level 2 of decoding process. Level 2 (Decode).	179
Table 5.25:	The comparison between benchmark and proposed technique in level 2 of re-encoding process. Level 2 (Re-encode).	180
Table 5.26:	The level 1 and level 2 time range difference.	182
Table 5.27:	The comparison text capacity between proposed coloured QR code and existing QR code (black-white and colour).	183
Table 6.1:	The module and sub-module upgrading plan.	201

List of Figures

Figure 1.1.	Examples of one-dimensional barcode and two-dimensional barcode (Source: Rinkalkumar (2014)).....	3
Figure 1.2.	An example of stacked and matrix symbologies images (Source: http://www.tec-it.com)	4
Figure 1.3.	An image of QR code (Source: www.qrcode.com).....	5
Figure 1.4.	Examples of QR version 1, 10, and 40.....	6
Figure 1.5.	Example of QR codes with metric columns.	8
Figure 2.1.	The mental model of RGB coloured QR code.	19
Figure 2.2.	The history of QR code.	22
Figure 2.3.	The structure of QR code version 2 (Galiyawala & Pandya, 2015; Kieseberg et al., 2010; Wakahara, Yamamoto, & Ochi, 2010).....	23
Figure 2.4.	The design of QR codes (Courtesy: www.qrcode.com).....	27
Figure 2.5.	Microsoft's High Capacity Colour Barcode (Courtesy: http://research.microsoft.com/en-us/projects/hccb/).	29
Figure 2.6.	The structures of standard and IP-based PM code technology (Source: Asia Global Technology Sdn. Bhd.).....	30
Figure 2.7.	The roadmap of PM code technology.	31
Figure 2.8.	The colour format and the calculation based on 24-bit format (0..23).	32
Figure 2.9.	The RGB model in a unit cube (Courtesy: Donald D. Hearn, M. Pauline Baker, 2010).	37
Figure 2.10.	The algorithm conversion from RGB to CMYK colour models.....	38
Figure 2.11.	The image zoomed out more closely.....	39
Figure 2.12.	The flow chart for encoding and decoding processes of coloured QR code (Nurwono & Kosala, 2009).....	41
Figure 2.13.	The layers in the image editor (Courtersy: Nurwono & Kosala (2009)).	50
Figure 2.14.	The result of combination of four layers (Courtesy: Nurwono & Kosala (2009)).	50

Figure 2.15.	The process of encoding the coloured QR Code (Courtesy: Ramya & Jayasheela (2014)).	52
Figure 2.16.	The process of encoding coloured QR code (Courtesy: Blasinski et al. (2013)).	53
Figure 2.17.	Coloured QR code produced (Courtesy: Melgar et al. (2012)).	54
Figure 2.18.	Values for data capacity for smaller version of HCC2D codes (Courtesy: Grillo et al. (2010)).	54
Figure 2.19.	Coloured QR code decoding algorithm (Courtesy: Nurwono & Kosala (2009)).	56
Figure 2.20.	Flow of decoding process (Courtesy: Blasinski et al. (2013)).	57
Figure 2.21.	Procedure of colour threshold. (Courtesy: Melgar et al. (2012)).	58
Figure 2.22.	The overview of multiplexing and demultiplexing methods. (Courtesy: Vongpradhip (2013)).	59
Figure 2.23.	The algorithms of multiplexing and demultiplexing (Courtesy: Vongpradhip (2013)).	64
Figure 2.24.	QR code with 8 special symbols (Vongpradhip, 2013).	65
Figure 2.25.	The process to produce coloured QR code (Pillai & Naresh, 2014).	66
Figure 2.26.	Flow of the multiplexing process of coloured QR code.	67
Figure 2.27.	Flow of the demultiplexing process of QR code with special symbols.	68
Figure 2.28.	Flow of the decoding process.	68
Figure 2.29.	Flow of the demultiplexing and decoding processes.	69
Figure 2.30.	The flow chart in generating a high capacity QR code (Courtesy: Victor, 2012).	72
Figure 2.31.	The steps to generate a large amount data for QR code.	72
Figure 2.32.	The hash map data can be encoded into a 2D barcode (Courtesy: Victor (2012)).	73
Figure 2.33.	The processes involved when the techniques of compression, multiplexing, and multilayer change positions.	77
Figure 2.34.	Single symbol and the structured append of symbols encoded with "ABCDEFGH IJKLMNOPQRSTUVWXYZ0123456789ABCDEFGHIH IJKLMNOPQRSTUVWXYZ".	80

Figure 2.35.	The methods of partial extraction.....	81
Figure 3.1.	The research framework.....	85
Figure 3.2.	The theoretical framework.....	87
Figure 3.3.	The testing activities.....	89
Figure 3.4.	The finalising and merging activities.....	90
Figure 3.5.	The proposed flow of the coloured QR code.....	91
Figure 3.6.	The proposed flow of the partial extraction process of coloured QR code.....	91
Figure 3.7.	The flow steps of the coding process.....	93
Figure 4.1.	The encoding flow process.....	99
Figure 4.2.	Coloured QR code encoding pseudocode.....	100
Figure 4.3.	The flow chart of character counting module.....	105
Figure 4.4.	The example of the first process in converting binary to decimal point number in the index location (0,0) for each black and white QR codes and assigning the value to the index location (0,0) at the red QR code.....	115
Figure 4.5.	The decoding flow process.....	118
Figure 4.6.	The pseudocode of main decoding programme.....	119
Figure 4.7.	The flow chart process of determining black or white pixels of black and white QR codes.....	126
Figure 4.8.	The flow chart of decompression method.....	129
Figure 4.9.	The abstract model of 8-bit colour depth and 3-channel RGB colour model.....	132
Figure 4.10.	The pseudocode of partial execution for decoding level 1.....	134
Figure 4.11.	The pseudocode of partial execution for re-encoding level 1.....	136
Figure 4.12.	The pseudocode of partial execution for decoding level 2.....	140
Figure 4.13.	The pseudocode of partial execution for re-encoding level 2.....	143
Figure 5.1.	A part of the employed Malay short story.....	146
Figure 5.2.	The flow processes of the encoding compression, multiplexing, and multilayer modules.....	153
Figure 5.3.	The diagram of RGB colour depth and colour channel.....	170
Figure 5.4.	Level 1 decoding abstract model.....	171

Figure 5.5.	Level 1 re-encoding abstract model.	172
Figure 5.6.	Level 2 decoding abstract model.	173
Figure 5.7.	Level 2 re-encoding abstract model.	174
Figure 5.8.	A part of input data text.	175
Figure 5.9.	The process flow results for QR code version 40.	176
Figure 5.10.	The process flow results for proposed technique level 1.	176
Figure 5.11.	The process flow results for proposed technique level 2.	177
Figure 6.1.	The complete model of compression, multiplexing, and multilayer for coloured QR code.	200
Figure 6.2.	The example of method implementation of parallel processing for partial extraction level 1.	203
Figure 6.3.	The combination of two coloured QR codes.	204
Figure 6.4.	The effect of light during decoding process.	205



List of Appendices

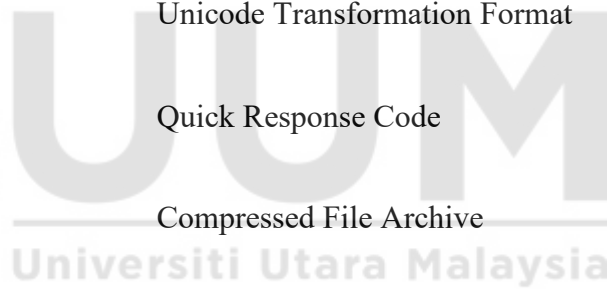
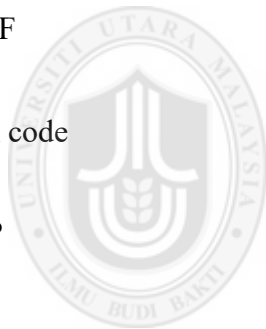
Appendix A : Result of Maximum Characters.....	227
Appendix B : Encode Level L.....	228
Appendix C : Decode Level L	233
Appendix D : Partial Extraction (Decode) Level 1	237
Appendix E : Partial Extraction (Re-encode) Level 1	241
Appendix F : Partial Extraction (Decode) Level 2	244
Appendix G : Partial Extraction (Re-encode) Level 2	248
Appendix H : Processing Time Module	252



List of Abbreviations

1D	One-dimensional
2D	Two-dimensional
3D	Three-dimensional
ANSI	American National Standard Institute
ASCII	American Standard Code for Information Interchange
CIAL	Content Idea Asia Limited
CMY	Cyan, Magenta, and Yellow
CMYK	Cyan Magenta Yellow and Key (Black)
CQR	Colour Quick Response
CQRC	Colour Quick Response Code
GZip	GNU Zip (Not Unix Zip)
HCC2D	High Capacity Coloured Two Dimensional
HCCB	High Capacity Colour Barcode
ISO	International Organization for Standardization
LED	Light Emitting Diode

LZW	Lempel–Ziv–Welch
MATLAB	Matrix Laboratory
PM	Paper Memory
RGB	Red Green Blue
RO	Research Objective
RQ	Research Question
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
QR code	Quick Response Code
ZIP	Compressed File Archive



CHAPTER ONE

INTRODUCTION

This research is on quick response code technology, which is one of the mechanisms to store information using two dimensional (2D) barcode images. Instead of using only white and black colour modules, this research proposes a coloured code that enables a larger data storage capacity.

1.1 Introduction

Currently, the use of digital media and communications technologies is growing rapidly from time to time. But in the same time, printed documents continue to form a convenient interface for people. A large number of important documents such as identity card, driving licence, passports, and other transaction data are still in printed form. Without exception, some of the printed items are used to tell information about the object or owner. Now in the digital era, one technique or mechanism is needed to interface with the information in the printed items or documents, which can be embedded inside printed objects. Thus, it can save more space in the printed document and it is secure. The data can subsequently be retrieved via a scanner or digital camera that can be aimed at the printed object (Bulan & Sharma, 2011b). In addition, it facilitates users to store data without using an electronic data storage device and saves the area of printed items or documents. The technique or mechanism used to embed digital information inside the printed object must be provided with additional operational features in the applications such as document authentication, meta-data embedding, and document tracking in workflows (Bulan & Sharma, 2011b). The information and methods as mentioned above refer to the use of barcode.

Barcode becomes a famous method of data storage because of its data retrieval accuracy, quick data retrieval, functionality process, and physical characteristic (Denso-Wave, 2015). The barcode is a graphical image representation, which is capable of storing digital information about an object such as ticketing information, tracking location, unified resource locator, contact list, and others related. The barcode can be divided into two categories, namely one-dimensional barcode or traditional barcode, and two-dimensional or matrix barcode (Feng & Zheng, 2010a). Technically, the differences between types of barcodes are based on the width of the bars, character set, method of encoding, checksum specifications, etc. (Purcaru & Roma, 2011). Based on previous research by Chuang, Hu, & Ko (2010), the one dimensional (1D) barcode is mostly known as “product identification”, while the 2D barcode emphasises on “product descriptions”. This is due to the limitation of 1D barcode storage as compared to 2D barcode. The 1D barcode contains various widths and parallel space lines. Meanwhile, the 2D barcode is typically a graphical image that stores information in both horizontal and vertical (Lin & Fuh, 2013). The major differences between 1D and 2D barcode include the capability to hold data per unit area, and the direction of accessing data whether vertical or horizontal. Figure 1.1 shows examples of one-dimensional barcode and two-dimensional barcode.

Based on the previous research from Feng (2010), 1D barcode information capacity is limited because it is capable to encode only in alphabets and figures. Meanwhile, the 2D barcode has high reliability and strong capability to resist interference. In addition, the 2D barcode has 100 times capability to hold information as compared to 1D barcode. Most 2D barcodes can store various information about a product such as product name, product details, web links, etc. (Chuang et al., 2010)

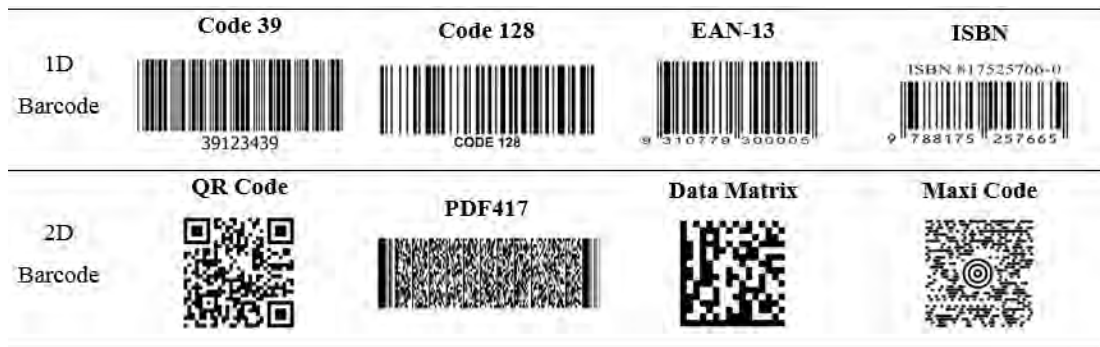


Figure 1.1. Examples of one-dimensional barcode and two-dimensional barcode

(Source: Rinkalkumar (2014))

The 2D barcode has two main categories, namely stacked and matrix symbologies (Intermec Technologies Corporation, 2007). The stacked symbology is developed with two or more small linear rows of barcode, which are stacked on the top of each other. Examples of stacked symbologies are Portable Data File with 4 bars and spaces and each pattern is 17 units long (PDF417), Code 16K, Code 49, and GS1 DataBar (Zhang & Yang, 2015). The matrix symbology, on the other hand, is mainly arranged in a grid with the geometric shapes of dark and light colours. It is commonly used in small item marking, unattended and high speed reading applications. Examples of matrix symbologies are Data Matrix, MaxiCode, Aztec Code, Code One, and QR Code (Sutheebanjard & Premchaiswadi, 2010). Figure 1.2 shows an example of stacked and matrix symbologies images, which are Global Standard 1 (GS1) DataBar Composite (a) and Data Matrix (b).

The quick response code (QR code) is a 2D barcode (Chang, 2014; Denso, 2011; Rawat, Sahu, & Puthran, 2015; Sarkar, Pu, Wu, Huang, & Wu, 2017), which is categorised under matrix symbology barcode. It was proposed in 1994 by a Japanese company, Denso Wave Incorporated and approved in 2000 as an AIM standard, JIS



Figure 1.2. An example of stacked and matrix symbologies images (Source: <http://www.tec-it.com>)

standard, and ISO standard (Commission & International Organization for Standardization., 2000; Rathod Rinkalkumar, 2014; Vizcarra Melgar, Zagherro, Macchiavello, & Nascimento, 2012; Wang, Yang, Li, Yao, & Zhang, 2015). Figure 1.3 shows an image of QR code.

As stated by Grillo, Lentini, Querini, & Italiano (2010), there are many benefits or advantages of QR code. The QR code has high capacity of encoded data because it can handle a large diversity of data, such as numeric, binary, and alphabetic characters. The area of space can be reduced as it can represent the data in a 1/30 space as compared to a 1D barcode. The QR code is able to carry data in both horizontal and vertical positions and offers high speed reading. Users can point the QR code in a position of 360° recognition (Shen, Lu, Qi, & Jiang, 2014) to read the contents of the QR code. Furthermore, the QR code has durability against soil, dirt, damage, distortion resistance or scratch (Grillo et al., 2010). The data can still be read if the condition of error correction level is set to the highest level. The storage can be clustered into many parts and can be appended if necessary (Denso, 2011). Denso has released the patent of QR code into the public domain so that anybody can use for free of charge (Boob, Shinde, Rathod, & Gaikwad, 2014).



Figure 1.3. An image of QR code (Source: www.qrcode.com).

The QR code can be used in various applications (Kumaraguru & Bormane, 2012), such as production, logistics, sales, and information to track (Shiang-yen, Foo, & Idrus, 2010; Szövetség & Várallyai, 2012) Nowadays, the QR code has been adopted in the areas where 1D barcode was utilised. These applications include retailing, healthcare, life sciences, transportation, office automation, marketing, and advertising (Denso, 2011; Pandya & Galiyawala, 2014).

The QR code is able to transmit information through a print-scan channel (Nikolaos & Kiyoshi, 2010; Magadam, 2017) and display the information in the forms of numeric and alphabetic characters, kanji, kana, hiragana, symbols, binary, and control (Bunma & Vongpradhip, 2014; Denso, 2011; Kan, Teng, & Chou, 2009; Qianyu, 2014). Conceptually, the idea behind this technology is not much different from the linear or matrix barcode (Dita, Ottesteanu, & Quint, 2011; Kato & Tan, 2005), nonetheless, its capability in data density allied with high speed reading made it popular. Currently, phone camera is used to scan the image of a QR code that contains contact information, short messages, authorisation to a wireless network, and opening a web page in the telephone's browser that is linked to the web server (Gutierrez, Abud, Vera, & Sanchez, 2013).

As reported by Nikolaos and Kiyoshi (2010) and Grillo et al. (2010) in their researches, each of the QR code module is represented as a single bit where a black square stores value 1 and a white square stores value 0. The capacity of a QR code depends on the number of modules allocated. Even though it can contain four data modes, numeric, alphanumeric, binary, or Japanese characters, it also depends on error correction levels and type of encoded data (Kieseberg et al., 2010).

In general, there are 1 to 40 versions of QR code that are different in the number of modules (Lyons, 2009). The storage capacity of QR code is determined by its version and version 40 has the highest encoding capacity among QR codes (Sangkwon et al., 2012). Figure 1.2 shows examples of QR code images for version 1, 10, and 40, respectively. The lowest module is version 1 that consists of 21 x 21 modules, while the largest, QR code version 40, consists of 177 x 177 modules (Denso-Wave, 2015; Jahagirdar & Borse, 2015; Liao Zhao-lai, Huang Ting-lei, Wang Rui, 2010; Luo, Wang, & Lin, 2016; Marktscheffel et al., 2016; Sun, Fang, Fu, & Zhao, 2009).



Figure 1.4. Examples of QR version 1, 10, and 40.

The QR code includes an error correction mechanism that helps to create redundant data, which facilitates the QR code reader to accurately read the code even if part of it is unreadable. The Reed-Solomon error correction code was implemented in QR code

to avoid data corruption and allows data recovery (Liu, Yang, & Liu, 2008; Skawattananon & Vongpradhip, 2013). The error correction consists of four levels of correction, namely L (Low), M (Medium), Q (Quartile), and H (High). Rawat et al. (2015) stated that if the level of error correction is high, then there will be less space for information storage. Hence, the maximum capacity of a QR code depends on the encoded content and error recovery level.

Harish and De (2014) encouraged studies on improving storage capacity of QR code even though the code can store more information than other conventional barcodes. Base on the survey done by Pandya and Galiyawala (2014), possible research areas to consider in QR code include data capacity, data recovery, security, and data allocation (Kajaree & Behera, 2017; Yadav & Dawande, 2016), whereby this can be achieved by using coloured QR code (Pandya & Galiyawala, 2014), multiplexing data (Umaria & Jethava, 2015; Vongpradhip, 2013), scratch removal (Thomas & Paul, 2013), and data hiding (Rungraungsilp, Ketcham, Wiputtikul, & Phonphak, 2012). The storage issues are the main important topic or problem for debate or discussion.

1.2 Problem Statement

To date, the QR code has the largest capacity storage among 2D barcodes, which is approximately 3 kilobytes (Grillo et al., 2010; Denso, 2011; Victor, 2012) and for the coloured barcodes, it has approximately 3 to 100 kilobytes (Galiyawala & Pandya, 2014). For all QR code versions, the size of the storage relies on the size of metric column (Zhang, Yao, & Zhou, 2012). The more metric columns exist in a QR code, the more data can be stored in it. Figure 1.5 shows different versions of QR code along with their character size.

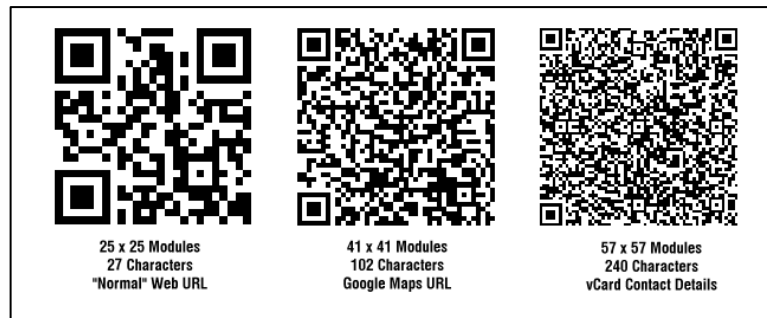


Figure 1.5. Example of QR codes with metric columns.

In detail, QR code has a limited size of storage whereby it can only store data up to 2,953 bytes (Victor, 2012; Vongpradhip, 2013), which is very small reported by Denso Wave Incorporated (Denso-Wave, 2014). The high capacity coloured QR code (HiQ) (3 layers) can obtain the data capacity only at 89 kilobytes and still can be considered as small capacity (Yang, Xu, Deng, Loy, & Lau, 2017). Furthermore, it can only store certain data (usually text or web address) (Chiang, Li, Hsia, Wu, & Hsieh, 2013). It is difficult to store large-scale images (Victor, 2012) or secured data that requires a large size of storage (Majumdar, Maiti, Bhattacharyya, & Nath, 2015).

In addition, the required number of QR code becomes larger if the amount of data to be stored increases (Denso, 2011; Warasart & Kuacharoen, 2012; Subpratatsavee & Kuacharoen, 2012). As a result, the QR codes are not suitable to be represented on limited sizes of printed media. Even though there exist various encode and decode application which employs coloured 2D barcodes such as Cobra, Strata and Focus (Yang et al., 2017), nevertheless, not all smart phones utilizes the same colour display. With limited colour display, the smart phones are not able to display actual colour QR code.

Generally, the common methods to improve data capacity of barcodes are by increasing the barcode size or barcode density (Feng & Zheng, 2010a). Nevertheless, in these circumstances, the two methods have several limitations. When the barcode size is enlarged, it will require a bigger printing area. On the other hand, enhancing the QR code density can distinctly decrease the capability to resist interference, and can increase the difficulty of barcode recognition. As a result, it will lead to high computational cost due to preparing relevant equipment (Victor, 2012).

To overcome the issue on QR code storage size, various studies have turned into coloured QR code (Liu, Zheng, & Jia, 2009; Yang et al., 2016). However, the experiment conducted by Nurwono and Kosala (2009) has only been able to store data up to nine kilobytes using three layers and eight colours. This is still considered insufficient as it could not store plain text e-book, news paper and journal. The red, green, and blue (RGB) colour combination is a technique used as a medium to increase the data capacity of QR code (Tank, Unde, Patel, & Raskar, 2016). Even though many researchers have used various colours to increase data capacity, their study consumes large computational effort (i.e. time and storage) to encode and decode (Galiyawala & Pandya, 2015; Grillo et al., 2010).

There is also work on using one QR code to build upon several QR codes or vice versa (Qianyu, 2014). The technique is known as structured append feature (Denso Wave, 2014; Grillo et al., 2010). Since one QR code contains several QR codes inside it, there will be issue of processing time during the decode and re-encode processes. Such an approach requires a complete reproduction of QR code if partial information (data in the QR code) is to be deleted, added or updated. Furthermore, it may require additional QR codes if the data capacity of the current QR code has exceeded.

1.3 Research Questions

Based on the problem statement highlight in the previous research, the related questions have been found:

1. What is the technique can be used to increase the data storage of a coloured QR code compared to the black and white QR code and existing coloured QR code data capacity (RQ1)?
2. How to append the compression, multiplexing and multilayer techniques with coloured QR codes in order to get the maximum data size (RQ2)?
3. How can the coloured QR code be manipulated for information extraction (RQ3)?

1.4 Objectives

The aim of this research is to enhance the data capacity of a coloured QR code by designing and developing the algorithms of compression, multiplexing, and multilayer. The research aims is achieved by the following objectives:

1. To design and develop a RGB coloured QR code encoding algorithm with a larger data storage (RO1).
2. To design and develop a RGB coloured QR code decoding algorithm that extracts all stored data (RO2).
3. To design and develop a RGB coloured QR code partial decoding and re-encoding algorithm that allows portion extraction of the required encoded data (RO3).
4. To evaluate the proposed RGB coloured QR code by comparing its performance against existing QR code (RO4).

The first research objective (RO1) was derived as response to the first research question (RQ1) as listed in Section 1.3. It is aimed to acquire the appropriate test adequacy criteria (see Section 3.1.3) to be include in Phase Three research framework. The second and third research objective will try to answer the second research question (RQ2). Generally, it concern the detail design and techniques that can be used to get more data capacity in a single coloured QR code. The third objective (RO3) is to manipulate the data in a single coloured QR code after encode and decode processes completed without involving many processes and saving the processing time. The last research objective is not related to research question because it is only involving some verification and validation after the whole system completed.

1.5 Significance of the Study

This research contributes in two areas, which are data storage and processing time. In the data storage area, this research contributes to those who are interested to store data in a QR code form. QR code can store data or information without using any electronic chips. It can be used as a mini secondary storage. The QR code can reduce cost (Charoensiriwath, Surasvadi, Pongnumkul, & Pholprasit, 2015) of storing data because it only uses a common printer to print the QR code image. This paper-based storage medium is cheap and easy to produce. It is also easy to distribute and carry and does not require any complex technology to use. On the other hand, users can easily retrieve relevant information by using QR code. It can improve business processes through faster access to information (Charoensiriwath et al., 2015).

1.6 Research Scope

The scope of this research consists of encoding, decoding, and data manipulation of a coloured QR code. This research does not include how the device reads the pattern inside a QR code. Furthermore, it does not consider the quality of the printed QR code. It also assumes that the stored data is text-based, with common symbols and not using other text symbols such as Arabic, Japanese, etc. The aim of this research is to increase the data storage capacity in a QR code by using a combination of methods from previous researches (compression, multiplexing, and multilayer) and the enhancement of the techniques. The techniques may use some colours such as red, green, blue, white, and black. QR code version 40 is used in this research as a benchmark as this version offers the largest capacity storage among QR codes.

The decoding process of coloured QR code does not use any QR code reader device, but it will be generated by using a coloured QR code decoding algorithm. The reason is that coloured QR code version 40 contains information that is translated into small pixels and the reader device is not able to capture the correct colour due to inconsistent brightness of light. Overall, the size of a QR code version 40 is 177 pixels x 177 pixels. In this case, to regain the small pixels, it needs to use a high-resolution camera smartphone in order to capture the small pixels of a QR code. This limitation is caused by the availability of the device and its high cost. As a solution, to overcome this problem, all the reading process is simulated by using computer programmes. The lighting environment is also not tested because all the reading processes are generated by computer programmes, and not in the actual environment. The input data is captured from a random Malay short story and it is measured in millisecond for processing time and total number of characters stored.

1.7 -Organisation of the Thesis

The QR code is one type of data storage media that can be utilised as content distribution and integrated with printed media. If this research is able to increase the data capacity and accuracy, the QR code can be used as a medium to convey a large amount of information. The data capacity of QR code can be improved by merging multiple researches in the data capacity of QR code. Based on this thesis, a lot of information will be explained starting from the background study until the existing QR code researches. Some reviews in certain researches were done to prove the problems that may occur if this structure is implemented. Nonetheless, from this research, it may contribute the solution to change the data capacity of QR code.

This thesis is organised in seven chapters. All the chapters are presented in chronological order, i.e. introduction, literature review, research framework, encoding, decoding, partial extraction, and conclusion. A brief explanation and organisation of the respective chapters are as follows:

Chapter One explains the main introduction of the research in this thesis. It contains a brief explanation on the preamble of knowledge terminologies involved in the research, which is related to the use, structure, development chronologies, and previous researches of QR code. Besides, this chapter also explains in detail the background of the problem, research questions, goal, and objectives as well as its scope and contributions to the domain research area.

Chapter Two reviews in detail about the QR code, which mainly concerns the terminologies of fact and information knowledge for the research in this thesis. The topics that will be picked out and emphasised in the discussion are QR code in detail,

review of coloured QR Code with its subordinate, comparison from previous techniques applied to increase the data storage, and the combination of selected techniques to be used in this research.

Chapter Three presents the research methodology that was undertaken in completing this research. It comprises the explanation of three phases, which are phases one, two, and three. The first phase reviews the preliminary studies that have been conducted to find the suitable algorithms and techniques. Meanwhile, the second phase discusses the development requirement and criteria. The last phase covers the tests and results based on the problem, limitations, and assumptions of the research.

Chapter Four provides the information of the design and implementation of the encoding process based on the suitable priority of encoding algorithms. Each algorithm is converted into the actual programme and fixed with the processing time for each selected algorithm. The input data is based on common American Standard Code for Information Interchange (ASCII) characters. The result is collected in several executions of the encoding programme. Some discussions on the result are guided from the quantitative experiment. The details on the decoding process after the encoding process is completed also discussed. The main objective is to evaluate the information of previous actual text back without any data loss after decoding. This chapter discusses the detailed steps of decoding processes based on reverse encoding processes. Another process that was discussed is to bring two types of execution level, namely levels one and two. The first level discusses the decoding and re-encoding of conversational QR code. Meanwhile, the second level comprises decoding and re-encoding monocolour QR code.

Chapter Five is aiming on the result from the finding in Chapter Four. Several executions are conducted as well to obtain the data capacity, processing time and total lost data. In addition, it includes some suggestions to gain data capacity based on each algorithm conducted. Meanwhile, in the partial extraction algorithmn, the result is collected and discussed as it may help the experiment of processing time for each level.

Chapter Six is the last chapter that concludes the thesis. It includes the summary of the thesis, highlights the contributions and limitations of the research and the possible future work.



CHAPTER TWO

LITERATURE REVIEW

This chapter offers the setting and some related researches conducted on QR code. The QR code became popular and its convenience is universally recognised; the market began to call for codes capable of storing more information, more character types, and that could be printed in a smaller space. The patterns of QR are square, dots, hexagons, and other geometric shapes inside the image; such a kind of barcode is referred to as matrix or 2D barcode. The QR is very significant because it carries information on it. The data storage is limited due to the structured design. The codes contain information in both upright and horizontal proportions. Many researchers have proposed various researches of QR code, which are planned to satisfy the storage capacity extension.

2.1 QR Code

The QR code has become widely popular due to its reading speed, accuracy, and functional characteristics (Coleman, 2011; Y. Zhang, Gao, Li, & Lin, 2012). Nowadays, there is an increasing amount of data, including emails, pictures, and videos, all of which must be accessible in a timely and dependable fashion. This data can be stored in personal computers, mobile phones or in data centres around the world (Asare & Asare, 2015). On account of the growing data requirements, storage is rapidly becoming an important factor in the data centre of information technology equipment. Presently, the overwhelming flow of data continuously increases in volume and detail, such as social media, internet of things, and multimedia (Hashem et al., 2015; Richard L. Villars, Olofson, & Eastwood, 2011). As a result, the data growth is the greatest challenge for larger enterprises. Still, storage not only demands

to scale in size, but also in performance, reliability, and power efficiency, among others; all these challenges must be met while minimising deployment and administration efforts (Prakash & Singh, 2010). Enterprises are actively calling for steps to mitigate the growing data problem. The recent years have witnessed substantial innovation in storage system extensions that provide critical improvements in understanding some of these storage system goals (Frost et al., 2007; Gunawi, Prabhakaran, Krishnan, Arpaci-Dusseau, & Arpaci-Dusseau, 2007; Z. Li, Chen, Sudarshan, & Yuanyuan, 2004; Meyer et al., 2008; Narayanan, Donnelly, & Rowstron, 2008; Sundaram, Wood, & Prashant, 2006; Zhu, Li, & Patterson, 2008).

The QR code is one medium used to keep information using paper that has a code inside it. Content Idea of Asia Co. Ltd. (2013), a Japanese mobile development company, has created a three dimensional (3D) barcode system called Paper Memory Code System or PM code. The technique is to stack the QR codes into one integrated 3D barcode. Microsoft Research has developed a 2D barcode called Microsoft's High Capacity Colour Barcode (HCCB), which is capable to store more than 84 bytes. As claim by Nurwono and Kosala (2009), there are less academic researches related to this system. Table 2.1 below shows the data density comparison between some 2D barcodes printed in 600 dpi, which contain QR code, HCCB, High Capacity Coloured Two Dimensional Code (HCCB2D), and CQR Code-5.

The primary goals of enhancing QR code are to increase the space available for the data and to preserve strong robustness and error correction properties similar to the original QR standard. In this respect, three parts are needed to consider when taking on the QR code data capacity. The parts are the techniques to increase, the standard of

Table 2.1

Data density comparison between some 2D barcodes printed in 600 dpi (Courtesy: Melgar & Santander (2016)).

Two Dimensional Barcode	Data Density [Bits per in²]
QR Code	5,136
HCCB	16,384
HCCB2D	15,409
CQR Code-5	17,163

the QR Code, and algorithms. The data capacity techniques are based on previous data capacity researches or its enhancement of storage capacity. Moreover, the standard of QR code follows the current version of QR code, which is version 40. Meanwhile, the algorithms concentrate on encoding, decoding, and partial extraction. Many algorithms have been divulged over internet environment now and the reason is to reveal the method of creating, decoding, and encoding the QR code in different ways. Some experimental studies have been conducted to achieve the best implementation. One of the popular QR code data capacity techniques is using the combination of red, green, and blue (RGB) colours, which is to make them united (Ahlawat & Rana, 2017; Chandran, 2014; Toh, Goh, & Yeo, 2016). Thus, this technique subsequently proved that data can be stored in the QR code in a large amount of data if RGB is combined. Figure 2.1 shows the aim of a QR code mental model is to increase data capacity. When considering the use of RGB colours in developing a coloured QR code, three aspects need to be considered; data capacity techniques, encode and decode algorithms and standard benchmarking. Relevant techniques need to be search for, extend and combined to produce RGB coloured QR code as an output. In addition, a standard

benchmark need to be determined (i.e black and white QR code) prior to the process of enhancing its storage capacity. When the input has been determined, algorithms to produce (i.e encode) QR code need to be designed. This is followed by designing the decoding algorithm that allows the extraction of the stored data. During the designing process, among of the concern is the computational time to encode and decode the QR code. The proposed QR code should be encoded and decoded in less than the time required by existing QR code. In practical, there is also a need to manipulate data stored in the QR code. This includes updating portion of the data such as a chapter in book or updating a broken link.

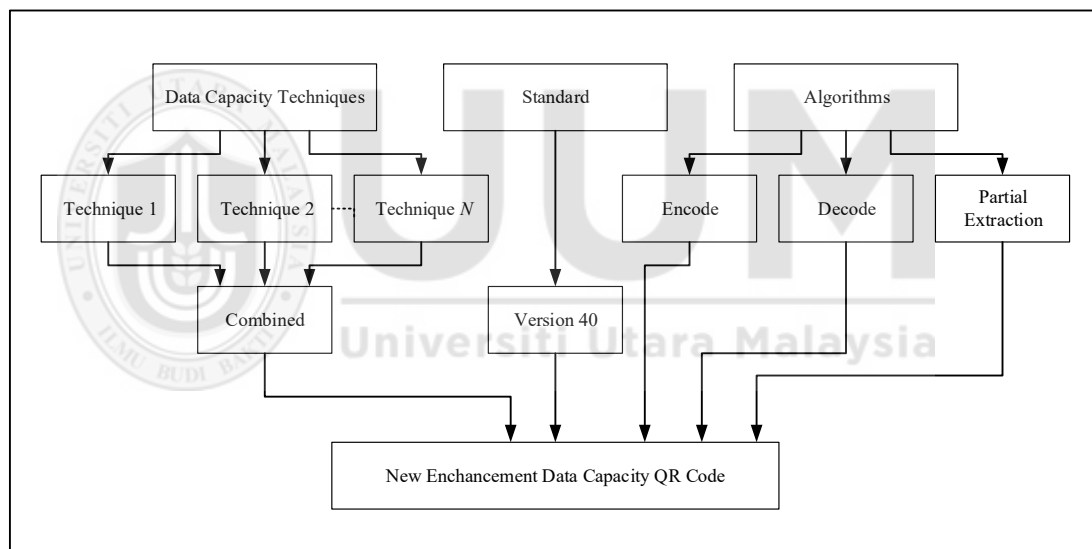


Figure 2.1. The mental model of RGB coloured QR code to increase data capacity.

The coloured QR code is beneficial in various ways but the main advantage is that it can act as a mini data storage by end users to store permanent or temporary data. As there exist information explosion in the current era, end users are overwhelmed with lots of data and knowledge. Some of the data need to be easily stored and accessed. Hence having a data storage that requires no additional device (such as thumb drive, portable hard disc) or internet connection (cloud storage) will benefit the end users. To

date, users move around with smartphones and the use of coloured QR code as a means of data storage will ease daily routines and fit into their lifestyle. This can be seen advertising and marketing, item tracking, data sharing, product description, airport boarding pass, web and mobile based authentication etc. (Nandhini, 2017; Singh, Verma, & Raj, 2017). In many marketing companies, QR code can be found on magazine pages, billboards, product boxes, beverages, advertisement papers, flyers, and other marketing mediums (Čović & Šimon, 2016). This is due to the fact that end users or customers like to access promotional information and discounted item (Okazaki, Navarro, & Campo, 2013). Most of users are sharing a large amount of information via QR code by visiting web sites that provide additional information (Demir, Kaynak, & Demir, 2015). As QR code became popular and their convenience universally recognized, the market began to call for codes capable of storing more information, more character types, and that could be printed in smaller space (Bhardwaj, Kumar, Verma, Jindal, & Bhondekar, 2016; Grillo et al., 2010). Hence, recently, researchers are focusing on speed reading and coding capacity of QR code (Bhardwaj et al., 2016).

The 1D barcodes are usually found in the purchasable items that are scanned at the purchasing register counter. The functionality of the traditional barcodes depends on the readability method that is from left to right. This is one of the main limit variances in 1D barcodes. Nonetheless, the QR Code has several uniqueness, such as the following:

1. **High capacity of encoding the data (Lin & Fuh, 2013).** The 1D barcode has limited storage capacity and stores mostly less than 20 characters (Winter, 2011). In addition, it can be scanned only in a horizontal direction. A QR code has the capability

to store hundred times than the capacity of 1D barcode. If the QR code uses one symbol, it is able to store a maximum of 7,089 characters. The QR code is capable of storing various types of data including numeric and alphabetic characters, kanji, kana, hiragana, symbols, binary, and control codes (Denso, 2011).

2. **Come out small printed data.** If the 1D barcode and QR Code have the same amount of data, the size among them has a 25% difference (Denso-Wave, 2014). It is clearly stated that QR code can save more printed area as compared to 1D barcode.

3. **Include Japanese fonts (kanji and katakana).** The QR code was developed in Japan for industrial purposes. Due to the reason above, the kanji and katakana setting are the first priority in the implementation of QR code in Japan (Denso-Wave, 2014).

4. **Dirt and damage robustness.** As stated by Ji (2014), when the image of QR code is tainted with dirt or damaged, the data can be recovered by a mechanism called error correction and restoration. The data can be restored even if a segment of the QR code is unreadable. There are four levels of error correction, namely level L (7% recovery), M (15% recovery), Q (20% recovery), and H (25% recovery).

5. **Can be read in many directions as compared to normal barcode.** The reader device is able to read the QR code in a 360° direction (Shen, Lu, Qi, & Jiang, 2013). The QR code was developed with the detection pattern located in the three corners of the symbols.

6. **Structured append feature.** The QR code is developed to store data in various QR code symbols. This means that a QR code can be divided into more than one QR code and all the QR codes can also be stored in one QR code.

The chronological history of QR code is shown below in Figure 2.2.

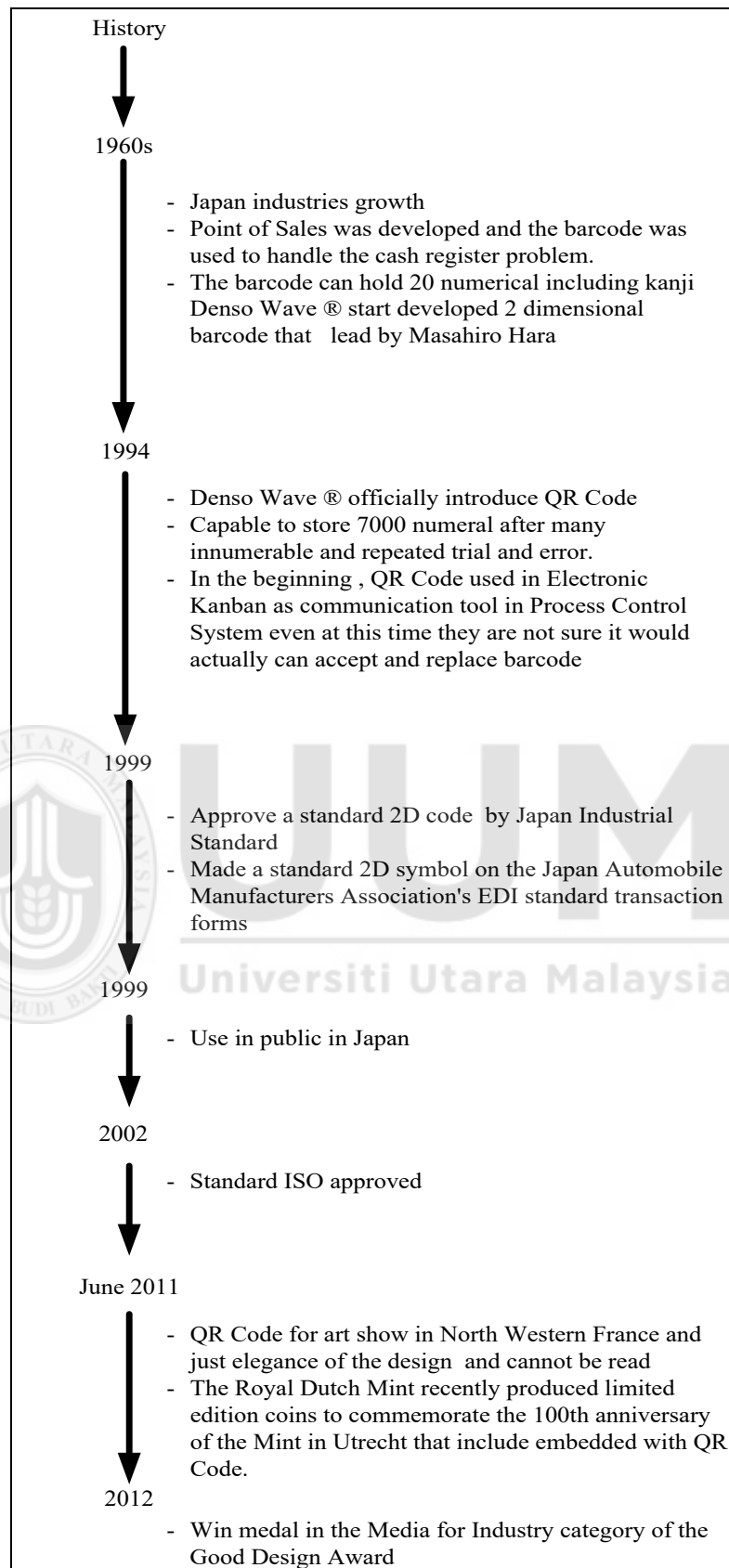


Figure 2.2. The history of QR code.

2.1.1 QR Codes Architecture Structure

QR codes consist of different segments that are reserved for specific purposes. In QR code version 2, the segment is divided into eight sections based on the numbers given in Figure 2.3 and each segment has its own specific tasks. The specific tasks include:

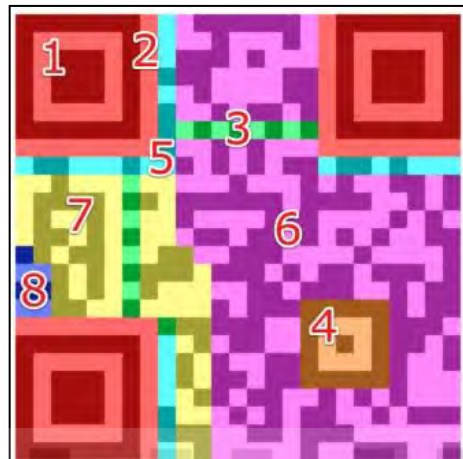


Figure 2.3. The structure of QR code version 2 (Galiyawala & Pandya, 2015; Kieseberg et al., 2010; Wakahara, Yamamoto, & Ochi, 2010).

1. **Finder Pattern (1).** It is used to detect a position of QR code in a decoder application. It is surrounded by two guard zones of the one QR module called the separator (Garateguy et al., 2014). The finder pattern consists of three identical structures that are placed in all corners of the QR code except the bottom right one. Each design is based on a 3x3 matrix of black modules surrounded by white modules that are again surrounded by black modules. The finder pattern enables the decoder software to identify the QR code and determine the correct orientation (Li et al., 2017).
2. **Separators (2).** The white separators improve the recognisability of the finder patterns as they separate them from the actual data. The separators have a width of one pixel (Galiyawala & Pandya, 2015).

3. **Timing Pattern (3).** In the decoder application, timing pattern is used to help determine a symbol's coordinate. Alternating black and white modules in the timing pattern enables the decoder software to determine the width of a single module and denote it as the timing zone, which is also located between the finder patterns (Garateguy et al., 2014).

4. **Alignment Patterns (4).** Alignment pattern is used to determine the sampling grids from which code words are extracted and to ensure the correct deformation of the pattern image (Garateguy et al., 2014). In version 2 and above, alignment patterns support the decoder software in compensating for moderate image interference. On the other hand, it is used to correct interruptions in the decoder application. The alignment patterns are not included in QR code version 1. More alignment patterns are added when the size of the QR code increases.

5. **Format Information (5).** The format information section stores information about the error correction level of the QR code and the chosen masking pattern. In addition, it consists of 15 bits next to the separators.

6. **Data (6).** Data is stored in 8-bit parts (called code words) (Farizshah & Abd Jalil, 2012) at the data section after it is converted into a bit stream. The size and data capacity for different versions of QR code are shown in Table 2.2.

7. **Error Correction (7).** Similar to the data section, error correction codes are stored in 8-bit long code words in the error correction section. As determined by Garateguy (2014), QR code has four types of error correction, namely L, M, Q, and H, which allow to correct up to 7%, 15%, 20%, and 30%, respectively of code words in error (Hajduk, Broda, Kováč, & Levický, 2016). The Reed-Solomon codes are used to correct and detect the capacity by the formula $e + 2t \leq k - p$ where k is the number of error correcting code words, p is the number of misses of decoded protection code

words, e is the number of erasures, and t is the number of errors. In versions 1 to 3, detection code words are used, which allow to identify a number of errors greater than the correct capacity and fail the decoding. The maximum data capacity is given by the size and correction level of the code. For example, a code of version 1 and 7% correction capacity has a total number of code words $n = 26$ and $k = 7$ correction code words. This code is capable of storing $n - k = 26 - 7 = 19$ code words and has an error correction capacity of 2 code words.

8. **Remainder Bits (8).** If the data and error correction bits cannot be divided into 8 bit code words without a remainder, it will consist of empty bits.

2.1.2 Types of QR Code

Various types of QR code were produced with certain patterns. As explain by Ji (2014) and Tiwari (2017), there are five types of QR code, which will be explained below:

1. **QR Code Model 1 / Model 2.** Model 1 is capable to store up to 1,167 numerals and it is the original QR code in the beginning. The highest version of the code is 14 with 73×73 modules. Model 2 is as important as model 1, which is capable to store up to 7,089 numeral characters. Version 40 with 177×177 modules is the largest module produced. This model is the type of QR code used nowadays.

2. **Micro QR Code.** It allows only one orientation to detect the pattern and can be printed in a smaller space than before. It can store up to 35 numerals and the largest module is 17×17 .

3. **iQR Code.** The code can be generated in two steps whether rectangle or square modules. The turned-over code, black and white inversion code or dot pattern code can be printed. It also can store more information in a code as compared to QR code

model 2. Version 61 can store 40,000 numeral characters and contains 422 x 422 modules.

Table 2.2

The size and data capacity for different versions of QR code (Source: Garateguy, 2014).

Version	Modules per side	Function pattern modules	Format and version modules	Data modules	Code word capacity	Reminder modules
1	21	202	31	208	26	0
2	25	235	31	359	44	7
3	29	243	31	567	70	7
4	33	251	31	807	100	7
5	37	259	31	1079	134	7
6	41	267	31	1383	172	7
7	45	390	67	1568	196	0
8	49	398	67	1936	242	0
9	53	406	67	2336	292	0
10	57	414	67	2768	346	0
11	61	422	67	3232	404	0
12	65	430	67	3728	466	0
13	69	438	67	4256	532	0
14	73	611	67	4651	581	3
15	77	619	67	5243	655	3
16	81	627	67	5867	733	3
17	85	635	67	6523	815	3
18	89	643	67	7211	901	3
19	93	651	67	7931	991	3
20	97	659	67	8683	1085	3
21	101	882	67	9252	1156	4
22	105	890	67	10068	1258	4
23	109	898	67	10916	1364	4
24	113	906	67	11796	1474	4
25	117	914	67	12708	1588	4
26	121	922	67	13652	1706	4
27	125	930	67	14628	1828	4
28	129	1203	67	15371	1921	3
29	133	1211	67	16411	2031	3

30	137	1219	67	17483	2185	3
31	141	1227	67	18587	2323	3
32	145	1235	67	19723	2465	3
33	149	1243	67	20891	2611	3
34	153	1251	67	22091	2761	3
35	157	1574	67	23008	2876	0
36	161	1582	67	24272	3034	0
37	165	1590	67	25568	3196	0
38	169	1598	67	26896	3362	0
39	173	1606	67	28256	3532	0
40	177	1614	67	29648	3706	0

4. **SQR Code.** It has a privacy function that can be used to store private information or manage company information. The feature also includes restricting reading function.

5. **Logo Q.** Logo Q is an incorporated level of design features such as illustrations, letters, and logos. The readability is not compromised since propriety logic is used in generating this type of code.

Figure 2.4 shows the design of various types of QR code as explained above.

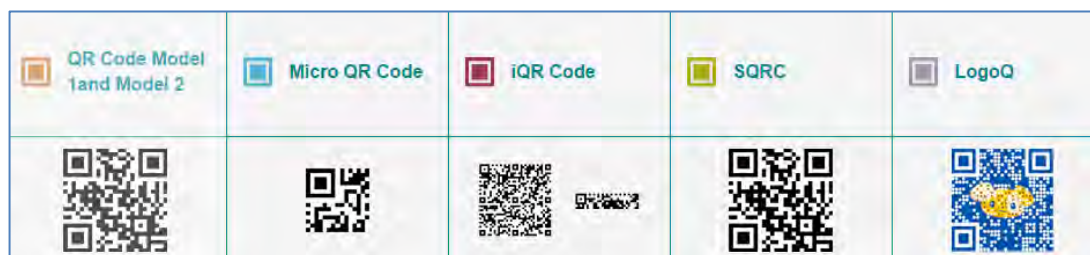


Figure 2.4. The design of QR codes (Courtesy: www.qrcode.com).

2.2 Coloured Barcode

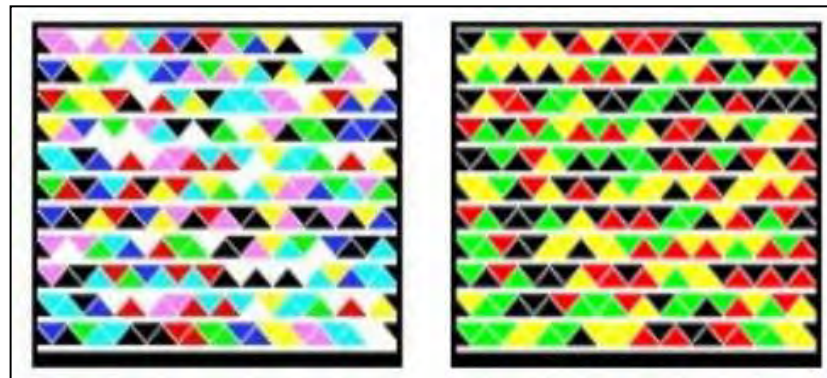
As stated by Grillo et al. (2010), Bishop (2007), and Fried (2007), some 2D barcodes are using colours to create more symbols, resulting in a larger data capacity within the

same size. Examples of such barcodes are the Colour Bar Code System of Imageid Ltd. and the more widely diffused Microsoft's High HCCB. HCCB, also known as Microsoft Tag, is used for a variety of applications such as information, entertainment, and interactive experiences on mobile phones (Jancke, 2015). The main features of HCCB in Figure 2.5 consist of rows of strings of symbols (triangles) of four different colours: black, red, green, and yellow, and the consecutive rows are separated by a white line. While the number of rows in a HCCB code may vary, the number of modules in each row is always a multiple of the number of rows. A module represents the basic entity for storing information in a 2D code. HCCB has a black boundary around it, further surrounded by a thick white band. These patterns are designed to act as visual landmarks in order to locate the barcode in an image. The black boundary at the bottom of HCCB is thicker than the boundaries on the other three sides: the bottom boundary acts as an orientation landmark, as barcodes may be at an arbitrary orientation in the image. HCCB uses a grid of coloured triangles with four or eight colours to encode data (Bagherinia & Manduchi, 2012). The last eight symbols on the last row are always in the fixed order of black, red, green, and yellow (two symbols per colour) and can be used as a colour palette during the scan.

This technique only considers certain primary colours such as black, red, green, and yellow. It does not include the combination colour of RGB that can make many types of colour. Furthermore, the storage for eight colours only can store 84 bytes as compared to QR version 40 that can store up to 3 kilobytes.

As maintained opinion by Grillo et al. (2010), the main limitation of the HCCB code is related to the fragility of the detection and alignment mechanisms. Indeed, the

detection process works as follows: it starts from a point that is supposed to be in the interior of the code



*8 colours barcode storing
84 RAW bytes*

*4 colours barcode storing 58
RAW bytes*

Figure 2.5. Microsoft's High Capacity Colour Barcode (Courtesy:

<http://research.microsoft.com/en-us/projects/hccb/>).

and proceeds on squares of larger sizes until it recognises the white border around the code. After the white border has been located, it starts the alignment process by looking for the thick bottom boundary. The fragility of the detection process derives from the fact that not all the images inside a white border are necessarily codes and this will bring the rise to delayed failures. Based on Prakash and Jancke (2008) researches, the weakness of the alignment process derives from the fact that different slopes in the scan phase might result in failure to properly recognise the thick bottom boundary.

Paper Memory Code (PM code) is known as a technology to convert a semi-huge data, such as text, images, sound, animation etc. into encoded data in the QR coloured matrix code (Dagan, Binyamin, & Eilam, 2016; Nurwono & Kosala, 2009). The raster generated code image can be decoded by the decoder and restored to the original data. PM code was developed by Content Idea Asia Limited (CIAL) in Japan and the

structure is based on a three-dimensional $x/y/z$ axis that allows more memory capacity in the QR code to be stored. The PM code is a fifth generation media that allows the fusion of all existing forms of media. Figure 2.6 shows the structures of standard and IP-based PM code technology. A PM code with eight scales of colour without IP access has been successfully developed, but the 24 scales of colour are still in progress of development.

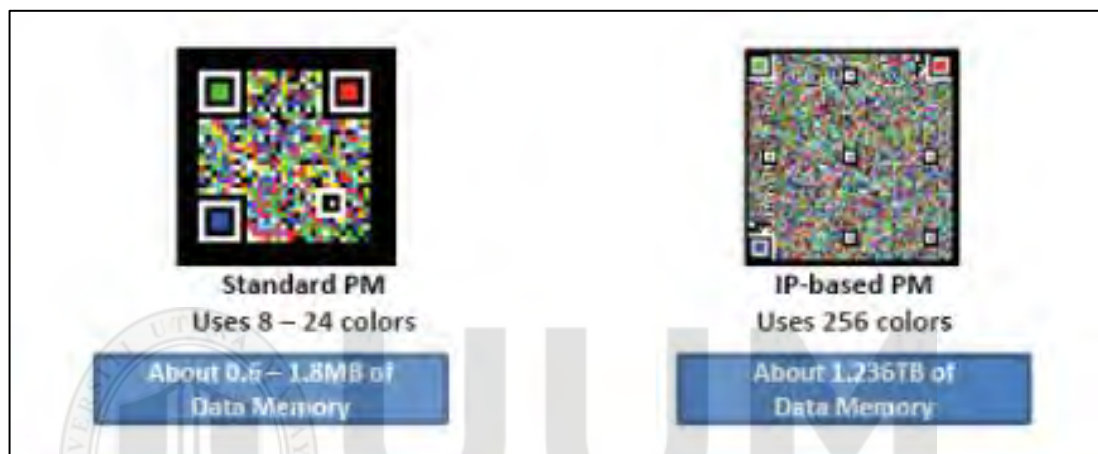


Figure 2.6. The structures of standard and IP-based PM code technology (Source: Asia Global Technology Sdn. Bhd.).

The roadmap of PM code technology is shown in Figure 2.7. CIAL started their project in 2009 and the largest amount of data that can be stored is 1.23 terabytes in 2003 by using the personal computer mode.

	2009	2010	2011	2012	2013
	Android OS 1.6, 2.0 PM-Code Reader	MAC OS 1.6 PM-Code Writer リリース予定	Win & MAC OS 4D PM-Code Writer リリース予定	Win & MAC OS 5D PM-Code Writer リリース予定	Win & MAC OS 6D PM-Code Writer リリース予定
	Feel Sketch Win&Mac PM-Code Writer		Android OS 2.1 PM-Code Reader リリース予定	Android OS X PM-Code Reader リリース予定	Android OS X PM-Code Reader リリース予定
	Feel Sketch Browser PM-Code Reader For Android	iPhone OS 4 PM-Code Reader リリース予定	Win Phone 7 PM-Code Reader リリース予定	iPhone & iPad OS X PM-Code Reader リリース予定	iPhone & iPad OS X PM-Code Reader リリース予定
Capacity		3D PM-Code	4D PM-Code	5D PM-Code	6D PM-Code
for Mobile		~ 10KB ~ 48KB	~ 1.5MB	~ 15MB	~ 380MB
for PC		~ 0.76MB	~ 768MB	~ 1.96GB	~ 1.23TB

Figure 2.7. The roadmap of PM code technology.

The advantages of PM code are include the following: (1) any type of digital data can be encoded; (2) it can be printed; (3) more data can be stored as compared to the normal QR Code; (4) and the RGB colours are used as the base colours that can produce more than 17,000 colours to store the data.

Even though the PM code has the advantage of large data storage, data manipulation is not considered with the idea to prevent outdated or rubbish data. Moreover, they used their own encoding and decoding algorithms, thus the Denso QR code generator cannot encode or decode the presented PM code.

2.3 Coloured QR Code

Most of the reproduction in printing uses cyan, yellow, and magenta colours, but in capturing an image, it uses red, green, and blue sensing channels (Ramya & Jayasheela, 2014). Many researches have proposed coloured QR code, which it can offer more data capacity as compared to black and white barcodes (Pandya & Galiyawala, 2014). The coloured QR code contains the same function patterns including finder patterns, separators, and quiet zone (Melgar et al., 2012). As usual, all

digital colours are extracted from RGB model parameters. Each RGB colour combination provides red, green, and blue colour codes, which can provide high embedding capacity for coloured QR code.

An RGB colour system was developed from the combination of red, green, and blue colours. There are 16,777,216 possible colours that use 8-bit colour. The colour format and the calculation based on 24-bit format (0..23) are shown in Figure 2.8 below.

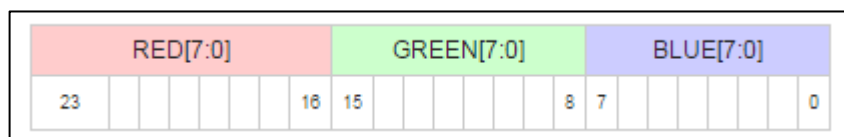


Figure 2.8. The colour format and the calculation based on 24-bit format (0..23).

As stated on the Online Reference and Tool website (2012), RGB colour space, or RGB colour system, constructs all the colours from the combination of red, green, and blue colours. The red, green, and blue use 8 bits each and have integer values from 0 to 255. The total of possible colour is $256 \times 256 \times 256 = 16,777,216$ possible colours. Each pixel on the LCD monitor displays colours this way, by a combination of red, green, and blue LEDs. When the red pixel is set to 0, the LED is turned off. When the red pixel is set to 255, the LED is turned on. Any value between them sets the LED to partial light emission.

Each pixel in the LCD monitor displays colour based on RGB LED (light-emitting diodes). The calculation of colour code is $RGB = (R \times 65536) + (G \times 256) + B$, where R is red, G is green, and B is blue. For example, the colour for red is $0 \times 65536 + 0 \times 256 + 255 = 255$ = #0000FF (in hexadecimal). RGB colours were selected because all computers use

this colour scheme to display multicolours in LED. It is easy to computerise the colour chosen using the given calculation.

2.3.1 Colour Depth

It is also known as bit depth or pixel depth that represents a number of bits used in a pixel and a single pixel contains its colour component (Kim, Kim, & Cho, 2004; Pascale, 2003; Süssstrunk, Buckley, & Swen, 1999). In the computer display, it refers to the number of bits per pixel that represents a specific colour. Currently, almost all computers support at least 32-bit colour, which allows up to 16.7 million of colour combination. The 48-bit colour can be supported by an operating system that is assuming the computer's video card supports this colour depth. When the technology and available system resources have increased, the evolution of colour depth also expanded. The evolution of colour depth starts with a two-colour combination known as monochrome display. Table 2.3 shows a list of all the differences between colour depths.

Colour depth refers to visual appeal. If the colour depth is set into high mode, then more visual appeals appear on the computer display. The image will come out more clear and vivid.

Table 2.3

A list of all the difference between colour depths.

Bit	Colours	Display name
-----	---------	--------------

1	2	Monochrome
2	4	CGA

4	16	EGA
8	256	VGA
16	65536	XGA
24	16,777,216	SVGA (True Colour)
32	4,294,967,296	Add Alpha Channel
48	281,474,976,710,656	Deep Colour

2.3.2 Colour Model

As explained by Donald D. Hearn and M. Pauline Baker Hearn (2010), any method for explaining the properties or behaviour of colour within some particular context and creating a full range of colours from a small set of primary colours is called a colour model. There are two types of colour models, which are additive colour models that use light to display colour, and subtractive colour models that are applied for printing by using inks. Currently, the colour model is simply a way to define colour and to appear on hardware such as colour monitors and printers.

In the RGB model, the primary spectral component for each colour appears in red, green, and blue colours (Sartor & Weeks, 1999; Farley, 2010). Additive colours use the colour of light. Generally, the images represented in the RGB colour model consist of three component images (red, green, blue) and when these components enter the RGB monitor, the three image components combine on the phosphor screen to produce a composite colour image that goes straight from a monitor to the eyes.

Basically, in the RGB image of 24 bits, each channel has 8 bits (red, green, blue), which are composed of three images (one for each channel). Each image can store discrete pixels with conventional brightness intensities between 0 and 255. The

combination of red, green, and blue may produce many colours to be displayed. The display adapter with 24 bits of information can contain 8 bits per component, then it is multiplied by three components that will produce 16,777,216 (256^3 or 224) discrete combinations of R, G, and B values or colours. The colours in the RGB colour model are described by indicating how much of each of red, green, and blue is included based on its combination. Each component is defined from zero to maximum values. If all the components are set up as zero, then the result will be black; meanwhile, if all components are set up at maximum, the result is representable as white. The range of these components can be indicated as follows:

1. From 0 to 1, with any fractional value in between. This representation is used in theoretical analyses and in systems that use floating point representations.
2. Each colour component value can also be written as a percentage, starting from 0% to 100%.
3. In computers, the component values are often stored as integer numbers in the range of 0 to 255, the range that a single 8-bit byte can offer. These are often represented as either decimal or hexadecimal numbers.
4. High-end digital image equipment are often able to deal with larger integer ranges for each primary colour, such as 0..1023 (10 bits), 0..65535 (16 bits) or even larger, by extending the 24-bit (three 8-bit values) to 32-bit, 48-bit, or 64-bit units (more or less independent from the particular computer's word size).

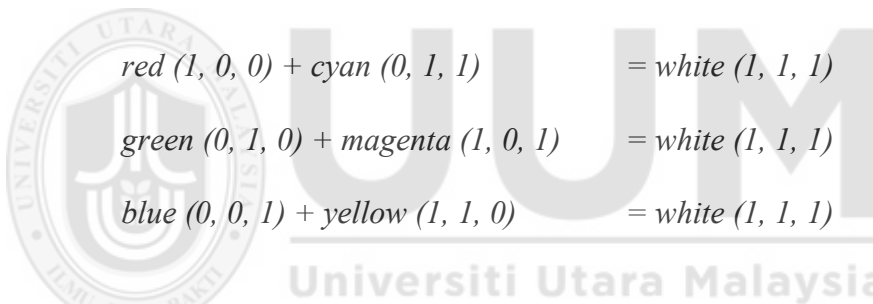
Table 2.4 shows the example of saturated green written in different RGB notations.

Table 2.4

Example of saturated green in different RGB notations.

Notation	RGB Triplet
Arithmetic	(0.0, 1.0, 0.0)
Percentage	(0%, 100%, 0%)
Digital 8-bit per channel	(0, 255, 0) or sometimes #00FF00 (hexadecimal)
Digital 16-bit per channel	(0, 65535, 0)

The RGB colour model can be represented in a unit cube. Each point in the cube or vector (where the other point is the origin) represents a specific colour. Figure 2.9 shows the RGB model by using a unit cube. For example if,



$$\begin{aligned}
 \text{red } (1, 0, 0) + \text{cyan } (0, 1, 1) &= \text{white } (1, 1, 1) \\
 \text{green } (0, 1, 0) + \text{magenta } (1, 0, 1) &= \text{white } (1, 1, 1) \\
 \text{blue } (0, 0, 1) + \text{yellow } (1, 1, 0) &= \text{white } (1, 1, 1)
 \end{aligned}$$

The RGB colour model can be converted to other colour models such Cyan Magenta Yellow Key (Black) (CMYK), Luminance (lightness) and two colour channels (a and b) (Lab), perceived luminance and colour/luminance information (YIQ), National Television System Committee (NTSC) colour model and others. Some colour model needs to be converted to produce more clear images such as

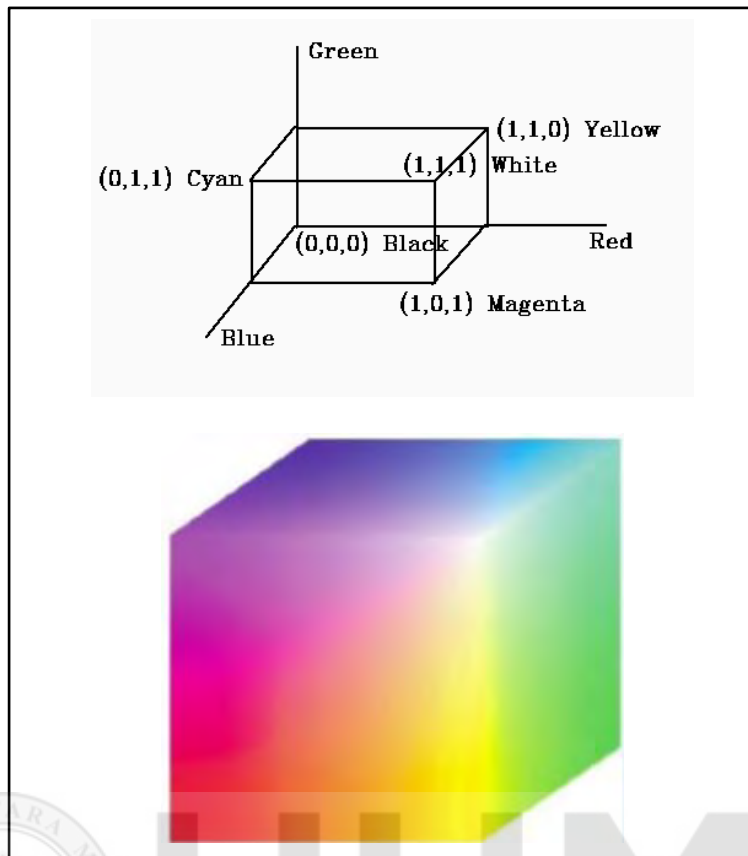


Figure 2.9. The RGB model in a unit cube (Courtesy: Donald D. Hearn, M. Pauline Baker, 2010).

from RGB to CMYK. For example, to convert from RGB to CMYK colour models, the algorithm that can be used is shown in Figure 2.10.

/* identify and calculate complementary colours with given Red, Green, and Blue colours*/

$c = 255 - R;$ */* R = Red */*

$m = 255 - G;$ */* G = Green */*

$y = 255 - B;$ */* B = Blue */*

/* find the black level k */

$K = \text{minimum}(c, m, y)$

/* correct complementary colour level based on k */

$$C = c - K$$

$$M = m - K$$

$$Y = y - K$$

/* The RGB values are in the range of [0:255] */

/* The CMYK values are in the range of [0:255] */

Figure 2.10. The algorithm conversion from RGB to CMYK colour models.

2.3.3 Pixelation

Pixelation occurs when the image is zoomed in bitmap images and the individual colour pixels (small colour square) are visible as elements that comprise the bitmap (Gerstner et al., 2013). In early graphic application, it is easily visible with sharp edges that gave curved objects and diagonal lines an unnatural appearance. Figure 2.11 shows the image that is zoomed out with a small section for a closer view, whereby the fur can be distinguished by individual pixels. It can happen accidentally when the image is designed to be displayed on a normal computer; when displayed at a large screen, each pixel can be seen separately. The pixels are used to detail out the information of an image. It can contribute the information regarding the type of colour displayed of each pixel. From the combination of pixels, it can display an image. The more pixels used in an image, the more details of the image are produced.



Figure 2.11. The image zoomed out more closely.

2.3.4 Multilayer Colour

Taveerad and Vongpradhip (2016) and Nurwono and Kosala (2009) mentioned in their research works that data capacity can be improved by adding more colour layers into the QR code. Based on this research also, they mention that the QR code with a capacity of nine kilobytes is not good enough according to the user standards and independent internet information. Even though the researchers have not created the auto generated coloured QR code application, they developed the pseudocode as a guideline. Based on a study by Melgar et al. (2012), the development of coloured QR code included a combination of layers. Each layer or depth was represented in one colour. The coloured QR code contained stacks of single coloured QR Codes. After that, each single coloured QR code was translated into binary data so that the coloured QR codes could be encoded in 8-bit data.

A research on coloured QR code was proposed by Nurwono and Kosala (2009). In the research entitled *Colour QR Code for Mobile Content Distribution*, they noted that the coloured QR code is a matrix code that can be extended into a 3D barcode system. The

3D barcode contains x, y, and depth axis and it is an extended version of the traditional QR code. Data capacity can be improved by adding more layers into the barcode system by stacking them. From the experiment,, the coloured QR code was limited to store only nine kilobytes with three layers, which had three colours and eight combinations of colours. The constraint of this research is the colours experimented were limited to red, green, and blue only. In addition, the experiment was only conducted in a small amount of characters, which was only 19 characters. The version of the coloured QR code used for testing was not version 40. Moreover, the nine kilobytes data capacity was not sufficient according to user standards and for the barcode to be internet independent (Nurwono & Kosala, 2009). The idea of this paper is similar to Paper Memory Code and the development of the encoder and decoder prototype was implemented on a mobile phone. Figure 2.12 illustrates the flow chart for the encoding and decoding processes of coloured QR Code.

Another research by Ramya and Jayasheela (2014), entitled *Improved Colour QR Codes for Real Time Applications*, claimed that the way to increase the QR code data capacity is by adding colours. In order to obtain maximum data storage, the characteristics of coloured QR code must be in a bigger size of pixels by reducing the number of pixels needed to convey information. Three layers of cyan, magenta, and yellow (CMY) colours were used and the combination of CMY resulted in eight colours including black and white.

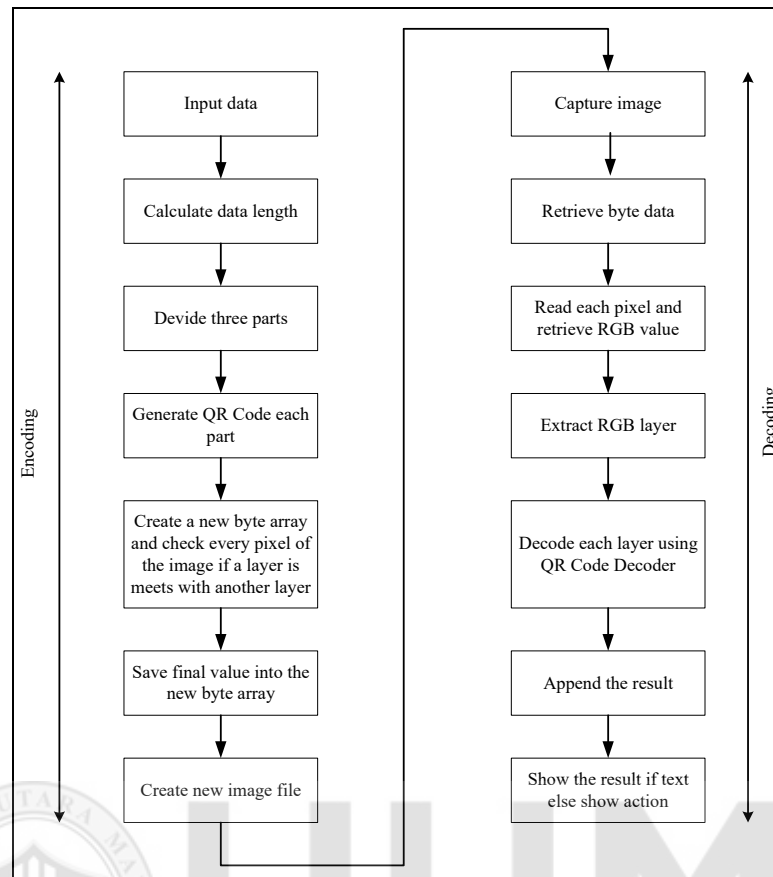


Figure 2.12. The flow chart for encoding and decoding processes of coloured QR code (Nurwono & Kosala, 2009)

The numbers of QR code input in this research were limited to three monocoloured QR codes. The RGB colours were experimented due to the code's capability to construct all the colours from its combination (Online Reference and Tool, 2012).

Blasinski etc al. (2013) exploited the spectral diversity afforded by CMY colours in generating the coloured QR using colour printing. The CMY colours were used to increase the data rate by encoding independent data of CMY channels and decoding the data from the RGB colour channels. The colours were limited to CMY colours. As a result, a coloured QR code was produced from a combination of CMY colours only. In addition, the coloured QR code was only limited to a three-colour combination and

not more than that. A small selection of words, i.e. “Hello World” (11 characters), was tested and a small version of QR code was created. QR code version 40 was not mentioned or experimented in this experiment in order to ascertain how many characters could be stored in a single coloured QR code. Furthermore, the coloured QR codes designed in their proposed framework could not be read by a monochrome QR code reader due to the unsuitable designed decoder. This is not an effective method for extending monochrome QR codes to the coloured QR code with low bit error rates that are readily handled by error correction. The experimental results show that the framework did not entirely accomplish to overcome the impact of the colour interference, providing a low bit error rate and a high decoding rate. The research title is *Per-Colourant-Channel Colour Barcodes for Mobile Applications: An Interference Cancellation Framework*.

In a research entitled *CQR Codes: Coloured Quick-Response Code*, red, green, blue, black, and white colours were used. Melgar et al. (2012) proposed the “coloured QR code structure”, which was used to store or transmit information. The limited RGB colours were used to deploy a new proposed code that enables twice the storage capacity as compared to traditional binary QR Codes. The research did not use any other colours except the RGB colours, black, and white. The research mentioned that some Colour Quick Reference (CQR) code performance was not as good and not correctly decoded. The experiment was limited to 1,024 bits of information only. The Reed-Solomon error-correcting code with a theoretical correction capability out of 38.41% was concerned. The coloured QR code for testing was printed and a digital camera with 3.2 megapixels was employed as input to decode the QR code in this experiment. The result showed that the coloured QR code consistently decoded 1,024

bits of information stored on a 1.3 cm × 1.3 cm printed area. Among its features, the structure had a remarkable transmission capacity and more complex aesthetic patterns. Further research may consider a different set of colours at a large number, an algorithm to counterbalance distortions caused by the inclination of the sensor, and the impact of compression algorithms used in a digital camera.

Grillo et al. (2010) developed a 2D barcode called High Capacity Coloured Two Dimensional (HCC2D) in order to increase the data capacity of QR code. The aims of HCC2D are to increase the data capacity, preserve the strong reliability and robustness properties of QR code, error correction, and to make sure of not losing any compatibility with the original QR standard (Singh et al., 2017). The experiment showed that HCC2D codes obtain data densities close to Microsoft's HCCB and strong robustness similar to QR code. The constraint between QR Code and HCC2D is that the overhead tends to increase when the number of colours and the code size increased. Table 2.5 illustrates the result of the scan process time in millisecond for QR code and HCC2D. This research was documented in a paper entitled *High Capacity Coloured Two Dimensional Codes*.

Five research work papers were reviewed in order to gain the ideas in multilayered colour on data capacity from the previous research results of a QR code. On the other hand, a brief table of five papers has been summarised in order by the key element as discussed in the papers previously. Among the key elements that have been identified are colour used, increment data, error correction, transmission capacity, processing time and others. This information can assist in making decisions concerning the best structure for generating maximum data capacity on the coloured QR code mode. The summary of the five research papers is shown in Table 2.6.

Table 2.5

The result of the scan process time in msec for QR code and HCC2D (Source: Grillo et al., 2010).

Version	QR	HCC2D 4 color	HCC2D 16 color
1L	122	132 (7.57%)	135 (9.62%)
1M	123	136 (9.55%)	138 (10.87%)
1Q	129	133 (3.01%)	135 (4.45%)
1H	131	135 (2.97%)	136 (3.68%)
5L	143	171 (16.38%)	206 (30.59%)
5M	151	174 (13.21%)	208 (27.40%)
5Q	163	181 (9.95%)	208 (21.63%)
5H	161	182 (11.54%)	209 (22.97%)
10L	176	209 (15.79%)	246 (28.45%)
10M	188	208 (9.61%)	249 (24.49%)
10Q	189	210 (10.0%)	273 (30.77%)
10H	190	212 (10.37%)	282 (32.62%)
20L	240	349 (31.23%)	387 (37.98%)
20M	253	334 (24.25%)	398 (36.43%)
20Q	250	337 (25.81%)	389 (35.73%)
20H	257	323 (20.43%)	395 (34.93%)
30L	333	447 (25.50%)	474 (29.75%)
30M	350	430 (18.60%)	460 (23.91%)
30Q	347	437 (20.59%)	478 (27.40%)
30H	338	416 (18.75%)	506 (33.20%)
40L	430	483 (10.97%)	550 (21.81%)
40M	415	481 (13.72%)	540 (23.15%)
40Q	420	500 (16.0%)	566 (25.79%)
40H	373	485 (23.09%)	552 (32.42%)

From the result of these papers, some researchers have come out with the ideas for future research and some have identified the advantages and disadvantages of their research work. The purpose of it is to identify the hole to improve their research in a way to give ideas to other researchers. Hereby, Table 2.7 can be referred as the summary of future research, advantages, and disadvantages from the five research papers.

Table 2.6

The identified information of QR code based on key elements.

No.	Researchers and year	Colours	Increments	Error Correction	Transmission Capacity	Processing Time	Others
1	Kris Antoni Hadiputra Nurwono and Raymondus Kosala (2009)	<ul style="list-style-type: none"> • Red • Green • Blue 	<ul style="list-style-type: none"> • 3 times based on layers combined 	<ul style="list-style-type: none"> • Reed-Solomon 			
2	M. Ramya and M. Jayasheela (2014)	<ul style="list-style-type: none"> • Cyan • Magenta • Yellow 	<ul style="list-style-type: none"> • 3 times based on layers combined 	<ul style="list-style-type: none"> • Reed-Solomon 			
3	Henryk Blasinski, Orhan Bulan, and Gaurav Sharma (2013)	<ul style="list-style-type: none"> • Cyan • Magenta • Yellow 	<ul style="list-style-type: none"> • 3 times 	<ul style="list-style-type: none"> • Reed-Solomon 			<ul style="list-style-type: none"> • Otsu Algorithm - the reduction of a grey level image to a binary image
4	Max E. Vizcarra Melgar, Alexandre Zaghetto, Bruno Macchiavello, and	<ul style="list-style-type: none"> • Red • Green • Blue • Black • White 	<ul style="list-style-type: none"> • 1,024 bits of information 	<ul style="list-style-type: none"> • Reed-Solomon 	<ul style="list-style-type: none"> • Size of 1.3 cm × 1.3 cm, resulting in a transmission capacity of 		

	Anderson C. A. Nascimento (2012)		approximately 605 bits/cm ²
5	Antonio Grillo, Alessandro Lentini, Marco Querini, and Giuseppe F. Italiano (2010)	<ul style="list-style-type: none"> • 4 and 16 colours • 1.881 kilobytes/inch (600 dpi) • Reed-Solomon 	<ul style="list-style-type: none"> • The average overhead introduced by HCC2D with 16 colours is about 25%



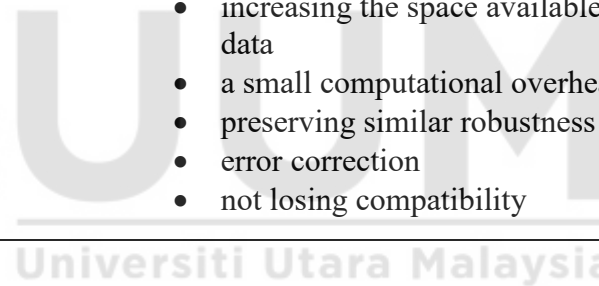
UUM
Universiti Utara Malaysia

Table 2.7

The future research, advantages, and disadvantages.

No.	Researchers	Future Research	Advantage	Disadvantage
1	Kris Antoni Hadiputra Nurwono and Raymondus Kosala (2009)	<ul style="list-style-type: none"> • layers in the barcode system • barcode reader accuracy • colour correction algorithm 	<ul style="list-style-type: none"> • suitable for outdoor environment under daylight 	<ul style="list-style-type: none"> • the discolouration and colour changes of the colour barcode by external factors • three colours and three layers only • limited to a maximum of nine kilobytes; not more than that • QR code version 40 not tested
2	M. Ramya and M. Jayasheela (2014)	<ul style="list-style-type: none"> • creating other colour codes such as red, green, and blue, and it also deals with implementing the coloured QR code in real-time applications by using field programmable gate array 	<ul style="list-style-type: none"> • reads the colour code in 360° • expand barcode capacity by adding colour • utilises existing capabilities of mobile devices 	<ul style="list-style-type: none"> • CMY colours only
3	Henryk Blasinski, Orhan Bulan, and Gaurav Sharma (2013)		<ul style="list-style-type: none"> • yellow channel has the highest variability (lacking) and error rates across commercial printer 	<ul style="list-style-type: none"> • CMY colours only • 11 characters tested • QR code version 40 not tested • can only be decoded with suitable decoder

4	<p>Melgar, Alexandre Zaghetto, Bruno Macchiavello, and Anderson C. A. Nascimento (2012)</p>	<ul style="list-style-type: none"> • a larger number or a different set of colours • an algorithm to counterbalance distortions caused by the inclination of the sensor • the impact of compression algorithms, commonly used in digital cameras 	<ul style="list-style-type: none"> • greater transmission capacity and the use of more complex aesthetic patterns 	<ul style="list-style-type: none"> • RGB colours only • limited to two times expandable • high performance overhead • decoder cannot perform correctly
5	<p>Antonio Grillo, Alessandro Lentini, Marco Querini, and Giuseppe F. Italiano (2010)</p>		<ul style="list-style-type: none"> • increasing the space available for data • a small computational overhead • preserving similar robustness • error correction • not losing compatibility 	<ul style="list-style-type: none"> • 180dpi cannot be read • Microsoft HCCB leads bigger on data capacity as compared to HCCB2D • consume more overhead



2.3.4.1 Encoding

Nurwono and Kosala (2009) proposed the steps to encode multilayer coloured QR code. The first step to start this process is by preparing to generate the single coloured QR code. It can be generated via online (e.g. Kaywa Website, Swetake's QR code Encoder). Meanwhile, the Image Editing Software (e.g. Adobe Photoshop) is used to combine the colours. Below are the steps to combine multicolour layers using the image editing software by stacking three layers. The steps are as follows:

1. Firstly, the data is divided into three parts for ease of encoding. Each part will be encoded into a different layer of the coloured QR code. If the user wants to encode a file or a binary data, they need to obtain its binary data and divide them into three parts. QR codes can only encode 8-bit data so the user needs to divide them proportionately.
2. Then the user encodes each part into a QR code. The user can choose whatever size and error correction. The chosen size must be the same size for all three QR codes. The codes and error correction level can be different in each QR code.
3. After all three QR codes have been generated, they must be assigned and stacked with different colours. The first portion of the data or QR code should be assigned with the red colour. The second layer should be green, and the last should be blue.
4. Once all layers have been assigned a colour, the colours must be combined on points where the layers meet by using the image editing software for this purpose.
5. Four separate layers in the image editor need to be created and each should contain a QR Code with one layer filled with black colour. Figure 2.13 shows the layers in the image editor.



Figure 2.13. The layers in the image editor (Courtesy: Nurwono & Kosala (2009)).

6. The layers need to combine the colours on points where the layers meet. This can be achieved by adding “Difference” effect on the layers. Figure 2.14 shows the result of a combination of 4 layers.



Figure 2.14. The result of combination of four layers (Courtesy: Nurwono & Kosala (2009)).

The encoding process is not implemented in the programming structure, so the experiment cannot be performed such as performance and validation of the colour. All combinations entirely depend on the image editor. The auto generator of coloured QR code cannot be performed due to the limitations of the image editor’s capability. Furthermore, the combination of colours for this experiment is limited to three colours only as the benchmark.

A pseudocode has been proposed to build a coloured QR code auto generator. The steps are as follows:

1. Get input data. Data can be numeric, alphanumeric, binary, or Japanese characters.
2. Calculate the data length and divide them into three parts.
3. Generate QR codes for each data part.
4. Assign colour for each layer. Layer 1 is Red, so every black point of the generated QR code for Layer 1 is changed to Red. The RGB value of Black is (0,0,0) or #000000, changed to (255,0,0) or #FF0000 for Red. Repeat the process to all layers and colours.
5. Create a new byte array and check every pixel of the image and examine whether a layer meets with another layer. If yes, add their RGB values. For example: Layers Red and Green meet at pixel position (10,25). Add Red RGB value #FF0000 and Green RGB value #00FF00, so the new RGB value for that pixel is (255,255,0) or #FFFF00. Then, invert this value using a logical NOT, so the final value is #0000FF. Afterwards, store the final value in the new byte array.
6. Save the byte data and create a new image file. With the limitation of this method, the encoded QR code needs to divide them proportionately. The size of QR code must be similar for each layer, but the error correction level is not compulsory. In this case, the limitation of this method cannot be applied for different sizes of QR codes. The stack of layers needs to be applied with a black background layer and the combination of the four layers needs to be merged.

A research by Ramya and Jayasheela (2014) has encoded the coloured QR Code by converting all information to ASCII in the beginning. Three sets of data were used and encoded into 8 bits of data, which contained 0 and 1 (Yeap, Cheong, Nisar, & Teh, 2014). The input element is divided into two elements. The ASCII value of the first

element should be added to the ASCII value of the next element and so on. All these values are grouped together in the block of data. After the grouping conversion, these data are divided into 8 bits of data. Suppose the data does not contain 8 bits, then zero padding is carried out. The matrix laboratory (MATLAB) is used to carry out the mask pattern. The steps above are repeated and the entire bit that has pixel value is converted into square blocks. As a result, the colours such as cyan, yellow, and magenta are assigned to all the bits and pixel values. Figure 2.15 illustrates the process of encoding the coloured QR code.

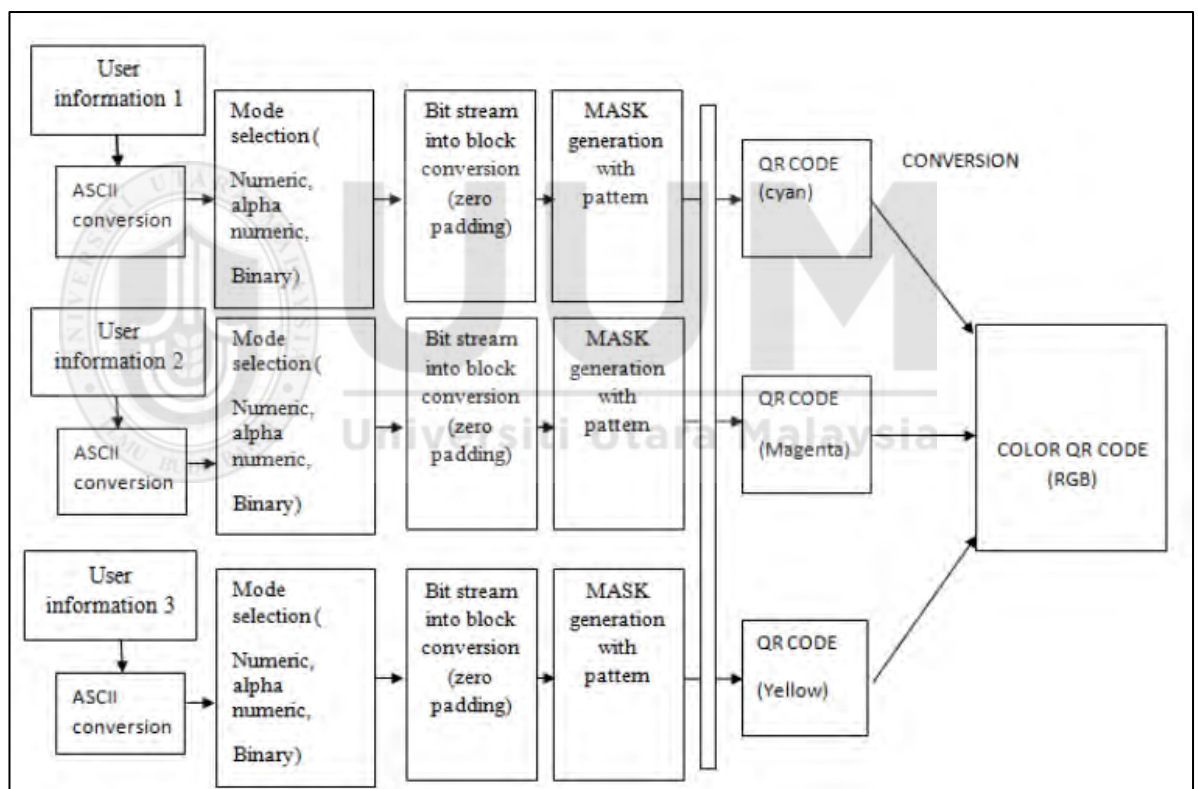


Figure 2.15. The process of encoding the coloured QR Code (Courtesy: Ramya & Jayasheela (2014)).

The data encoding process by Blasinski et al. (2013) starts with encoding three independent data to produce QR codes. Then, the QR codes are transformed into three different coloured QR codes, which are CMY colours. The CMY coloured QR codes

are combined by overlay printing to generate the coloured QR code. Figure 2.16 illustrates the process of encoding coloured QR code.

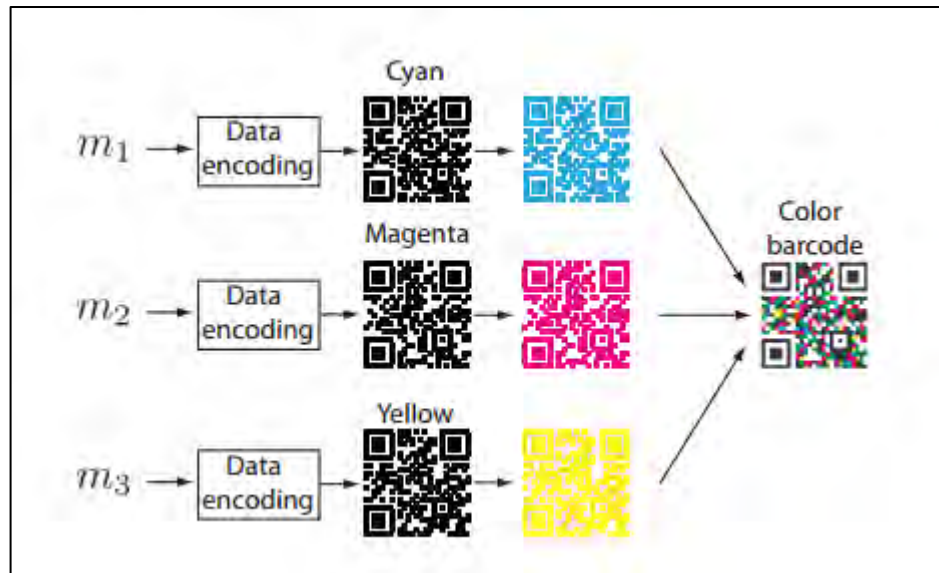


Figure 2.16. The process of encoding coloured QR code (Courtesy: Blasinski et al. (2013)).

Melgar et al. (2012) proposed the encoding process starting with binarising the data by considering each group of 16 bits representing one symbol. The Reed-Solomon algorithm is used to generate the redundancy symbols and they will be merged with the original data symbols. The purpose of the process above is for recovering data. Two bits are used to be mapped into one of the four colours. Once the encoding processes are completed, the coloured QR code is printed. This process will perform noise during the printing process of coloured QR code if the resolution of the printer is low. Figure 2.17 shows the coloured QR code produced from this research.



Figure 2.17. Coloured QR code produced (Courtesy: Melgar et al. (2012)).

Grillo et al. (2010) noted in their research that in order to increase the data in HCC2D (referring to the storage space), it should be generated in each module of the data area with a colour selected from a colour palette. HCC2D can be composed of at least four colours for each module, which is able to store more than one bit. The Bits per Module (BpM) can be defined as the number of bits that a single module is able to store.

$$BpM = \log_2(\text{number of colours})$$

If more colours are used, more data can be stored into the code. Figure 2.18 shows the values for data capacity for a smaller version of HCC2D codes.

Version	Data Capacity codewords		
	4 colors 2bits/module	8 colors 3bits/module	16 colors 4bits/module
1	52	78	104
2	88	132	176
3	140	210	280
4	200	300	400
5	268	402	536
6	344	516	688
7	392	588	784

Figure 2.18. Values for data capacity for smaller version of HCC2D codes (Courtesy: Grillo et al. (2010))

The HCC2D code encoder was developed by using the *libqrencode* library, whereby it is a C library for encoding data in standard QR code symbols (Fukuchi, 2010).

2.3.4.2 Decoding

As usual, the first activity involved in decoding a coloured QR code is capturing the image. The image is extracted into byte data and each of the byte data needs to be read and translated into RGB colour values. Each RGB colour pixels will be separated into monocolour layers and the process is performed by a QR code decoder. The layers must have an appended structure because they use a combination of monocolour concepts to store the information. The problems during decoding are the accuracy problem due to lighting source, hardware, discolouration, image resolution etc. (Nurwono & Kosala, 2009).

As specified by Nurwono and Kosala (2009), there are two main processes involved when decoding the coloured QR code. The processes are separating the layers and reading each layer. The processes to decode each QR code layer are basically similar to the QR code encoding process; in contrast, it starts from bottom to top. The coloured QR code decoding algorithm is shown in Figure 2.19.

From Figure 2.19, the algorithm only explains the detail about decoding the coloured QR code, which is limited to the URL. Furthermore, the method to extract the RGB colour should be explained in detail because it will reflect the performance and validation of the coloured QR code. The flow of the decoding processes is illustrated in Figure 2.12.

-
- 1 *Get captured image*
 - 2 *Retrieve byte data*
 - 3 *Read each pixel and retrieve RGB value*
 - 4 *Extract RGB layers*
 - 5 *Decode each layer using QR Code Decoder*
 - 6 *Append the result*
 - 7 *Detect result type*
 - 8 *If result is text then Show result*
 - 9 *Else, show result and show available actions (example: result contains URL, available action is to access URL)*
-

Figure 2.19. Coloured QR code decoding algorithm (Courtesy: Nurwono & Kosala (2009)).

The coloured QR code generation proposed by Ramya and Jayasheela (2014) in Figure 2.15 illustrates the ways to encode the coloured QR code generation. Even though the decoding processes are not shown in Figure 2.15, the encoding processes can be diverted to the decoding process if the flow processes are generated starting from the end result of encoding coloured QR code. The processes start with converting the coloured QR code into the separated CMY colours, followed by decoding the data from the CMY QR code.

Blasinski et al. (2013) noted in their research that the decoding process is a data recovery process because there is no cross-interaction between the camera channels (typically RGB colour channels used in capture devices) and CMY colourant layers when capturing the image (Bulan & Sharma, 2011a). Nonetheless, CMY print colourants can be extracted from RGB capture channels (Blasinski et al., 2013). After capturing the images, the encoded data is extracted by using the localisation and geometry correction algorithms (Parikh & Jancke, 2008) for monochrome barcodes to produce an RGB QR code. After the RGB QR code is produced, the process continues with overcoming the cross channel colour-interference by using the colour interference

cancellation algorithm. This process will result in estimated CMY QR codes. From the result achieved, the estimated CMY QR will be decoded to extract information stored in the previous CMY QR code. Figure 2.20 illustrates the flow of the decoding process.

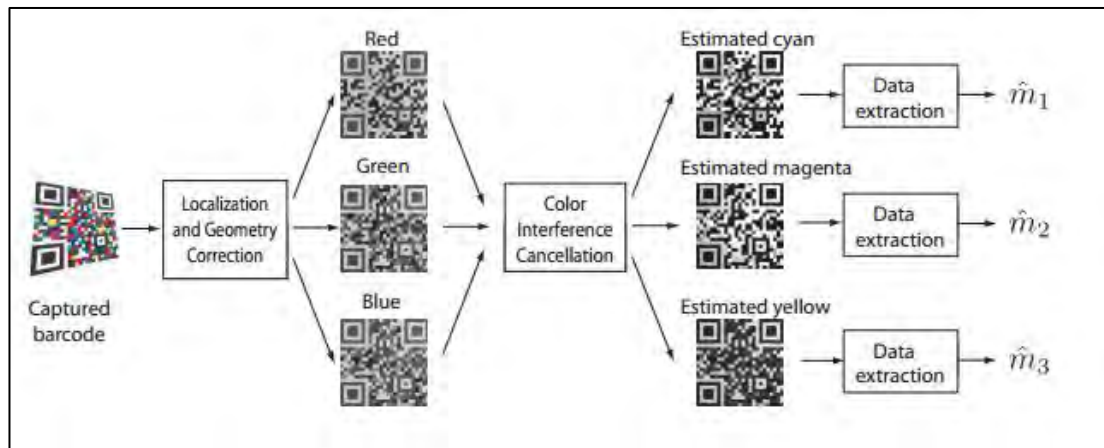


Figure 2.20. Flow of decoding process (Courtesy: Blasinski et al. (2013)).

Based on the research by Melgar et al. (2012), the decoding process starts with capturing the images. The issue of positioning the coloured QR code during the decoding process is to avoid the distortion of colours and symbol error. As a solution, the image is converted to greyscale and a threshold is determined by the following equation:

$$threshold = (lm^1 + lm^2) / 2$$

Where the value lm^1 is a grey level corresponding to a first local maximum of grey level and lm^2 is two times of grey level of lm^1 .

Once the *threshold* is identified, the procedure of *colour threshold* is performed to estimate the coloured QR code as shown in Figure 2.21.

```

COLOR-THRESHOLD(C, th)
1  for i = 1 to C.height
2    for j = 1 to C.width
3      MaxC = max(C(i, j, :))
4      MinC = min(C(i, j, :))
5      D = MaxC - MinC
6      if D > th
7        if MaxC == C(i, j, R) // red
8          C(i, j, RGB) = {255, 0, 0}
9        elseif MaxC == C(i, j, G) // green
10         C(i, j, RGB) = {0, 255, 0}
11        else // blue
12         C(i, j, RGB) = {0, 0, 255}
13      else
14        if C(i, j, RGB) < {th, th, th} // black
15         C(i, j, RGB) = {0, 0, 0}
16        else // white
17         C(i, j, RGB) = {255, 255, 255}
18  return C

```

Figure 2.21. Procedure of colour threshold. (Courtesy: Melgar et al. (2012)).

After the process is completed, the estimated coloured QR code is ready to be decoded with minimal symbol error percentage. The decoding process can be performed by using any coloured QR code algorithm because the coloured QR code has been improved from the noise of the code word. This research aims to improve the decoding process for coloured QR code, especially in the distortion of colours and positioning of the camera.

2.3.5 Multiplexing and Demultiplexing

The multiplexing technique refers to combining multiple similar types of item for transmission over a single line. If multiplexing is used, only one QR code is required (Ryu, 2015). This leads to the reduction in the line cost and assists in keeping track of single QR codes. Meanwhile, the demultiplexing technique performs the reverse

function, which is to split the combined similar types of item into the original items (Rouse, 2015).

In a research paper by Vongpradhip (2013), he mentioned about an algorithm to increase the capacity of QR code by using multiplexing and demultiplexing. The special symbol code concept is used to increase the data capacity from the traditional QR code. The original data needs to be encoded and divided into smaller parts. The smaller parts are generated in the standard form of the QR code and will be multiplexed to produce a black and white QR code with special symbols inside. In the decoding process, the QR code with special symbols will be dedicated in the same amount of patterned QR code before it is multiplexed. These patterns can be read by a QR code reader of a general smartphone and the data can be converted to the original form. The processes of multiplexing and demultiplexing can be used in generating high capacity QR code. However, at the same time, they generate overhead during the encoding and decoding processes if they involve a large amount of layers. It is because the amount of code symbols are dependent on the formula 2^n (n = numbers of layers). A large amount of n will influence the amount of code symbols. The overview of the multiplexing and demultiplexing methods can be referred to Figure 2.22.

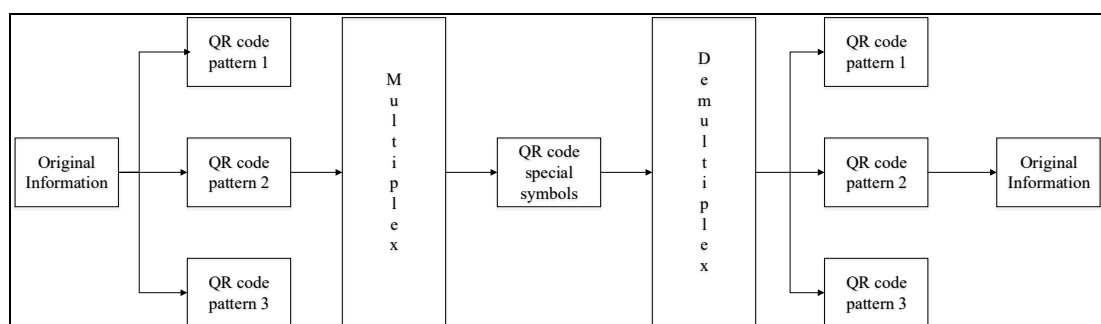


Figure 2.22. The overview of multiplexing and demultiplexing methods. (Courtesy: Vongpradhip (2013)).

In the research paper entitled *Improving the Capacity of QR Code by Using Colour Technique* proposed by Pillai and Naresh (2014), it was mentioned that data capacity can be further improved by considering more colours along with black, white, red, blue, green etc., and mixing with the multiplexing technique. This algorithm is capable of increasing data capacity up to three times from the traditional QR code version 40 which is total 12888 bits. The code word of Reed-Solomon code is generated by using finite fields and a damaged QR code can be recovered up to 30%.

The technique from a research entitled *To Increase Data Capacity of QR Code Using Multiplexing with Colour Coding in Embedding Speech Signal in QR Code* has made an effort to increase data capacity by multiplexing QR codes to produce a coloured QR code. This idea has been proposed by Galiyawala and Pandya (2014). This technique is similar with the paper done by Vongpradhip (2013); nonetheless, the difference is a decoding method will generate a coloured QR code. The result showed that this technique can increase data capacity up to 24 times from traditional QR code version 40 which is total 103104 bits. In the demultiplexing process, it is recommended to multiplex 12 or less QR codes because it will generate faster results. The speech signal can be embedded into the proposed technique due to its capability of offering a large capacity. This process involves dividing the information into several parts and then multiplexing the parts using a colour coding technique. For demultiplexing, it is the reverse task of that of the multiplexing method.

Table 2.8 shows a summary of the multiplexing and demultiplexing methods of coloured QR code concepts as discussed above. Table 2.9 shows a list of future research, advantages, and disadvantages by researchers.

Table 2.8

The summary of multiplexing and demultiplexing methods of coloured QR code concepts.

No.	Researchers and year	Colours	Increments	Error Correction	Transmission Capacity	Processing Time	Others
1	Sartid Vongpradhip (2013)	<ul style="list-style-type: none"> • Black • White 	<ul style="list-style-type: none"> • 3 times 	<ul style="list-style-type: none"> • Reed-Solomon 			
2	Prathibha N. Pillai and K. Naresh (2014)	<ul style="list-style-type: none"> • Black • White • Red • Green • Blue 	<ul style="list-style-type: none"> • 3 times 	<ul style="list-style-type: none"> • Reed-Solomon – finite field 			
3	Hiren J. Galiyawala and Kinjal H. Pandya (2014)	<ul style="list-style-type: none"> • 2^n colours $n = total$ QR code 	<ul style="list-style-type: none"> • 24 times 	<ul style="list-style-type: none"> • Reed-Solomon 		<ul style="list-style-type: none"> • processing time very high during multiplexing 	

Table 2.9

The future research, advantages, and disadvantages.

No.	Researchers	Future Research	Advantage	Disadvantage
1	Sartid Vongpradhip	<ul style="list-style-type: none"> • capacity improvement • security • information hiding 	<ul style="list-style-type: none"> • able to increase data capacity 3 times 	<ul style="list-style-type: none"> • limited to ASCII special symbols • Overhead of performance due to comparison activities
2	Prathibha N. Pillai and K. Naresh		<ul style="list-style-type: none"> • damage up to 30% • able to increase data capacity 3 times • high recovery level 	<ul style="list-style-type: none"> • limit 3 colours • version 40 not tested
3	Hiren J. Galiyawala and Kinjal H. Pandya	<ul style="list-style-type: none"> • the proposed technique is only used for digital transmission • same colour across devices without some kind of colour management • quality of camera to capture images • code optimisation 	<ul style="list-style-type: none"> • data capacity 24 times as compared to basic QR code, which is greater than any other techniques due to availability of huge amounts of possible distinct colours • the multiplexing technique does not distort the QR code visually 	<ul style="list-style-type: none"> • use MATLAB as test simulation as compared to actual environment. • not consider brightness and illumination condition matters • the processing time requirement for demultiplexing is high • the printing device's dependency on generating colours

2.3.5.1 Encoding

Sartid Vongpradhip (2013) purposed the QR code using multiplexing and demultiplex methods to increase the information of QR code. In the beginning, the original messages are divided into small parts and performed into a QR code pattern. The process of multiplexing starts with changing the module of each part to special symbols and regenerating the QR code with special symbols inside. The patterns are encoded and represented for each module in the QR code with special symbols that are in black and white. Figure 2.23 shows the flow chart of the multiplexing and demultiplexing processes.

The special symbols used are based on the number module pattern. Table 2.10 shows the special symbols used in the module in each pattern. The value 0 represents the black module and 1 represents the white module. When the corresponding module in the same position of each pattern of QR code is *black, black, black (0 0 0)*, the symbol \ is used in the QR code. However, the symbols are tested to 10 characters only (\, /, ^, V, >, <, ~, and !) in the multiplexing and demultiplexing processes and there are many symbols that can be introduced. Even though the experiment mentioned that any characters on the keyboard or any special character patterns or any special symbols can be used, there is no information of common symbols used in another language keyboard. For example, the common symbols of Japanese were not mentioned in this experiment. It is better for the compatibility process between two languages or more. Furthermore, it cannot be read by the common QR code reader after it is multiplexed. Figure 2.24 shows the QR code with eight special symbols.

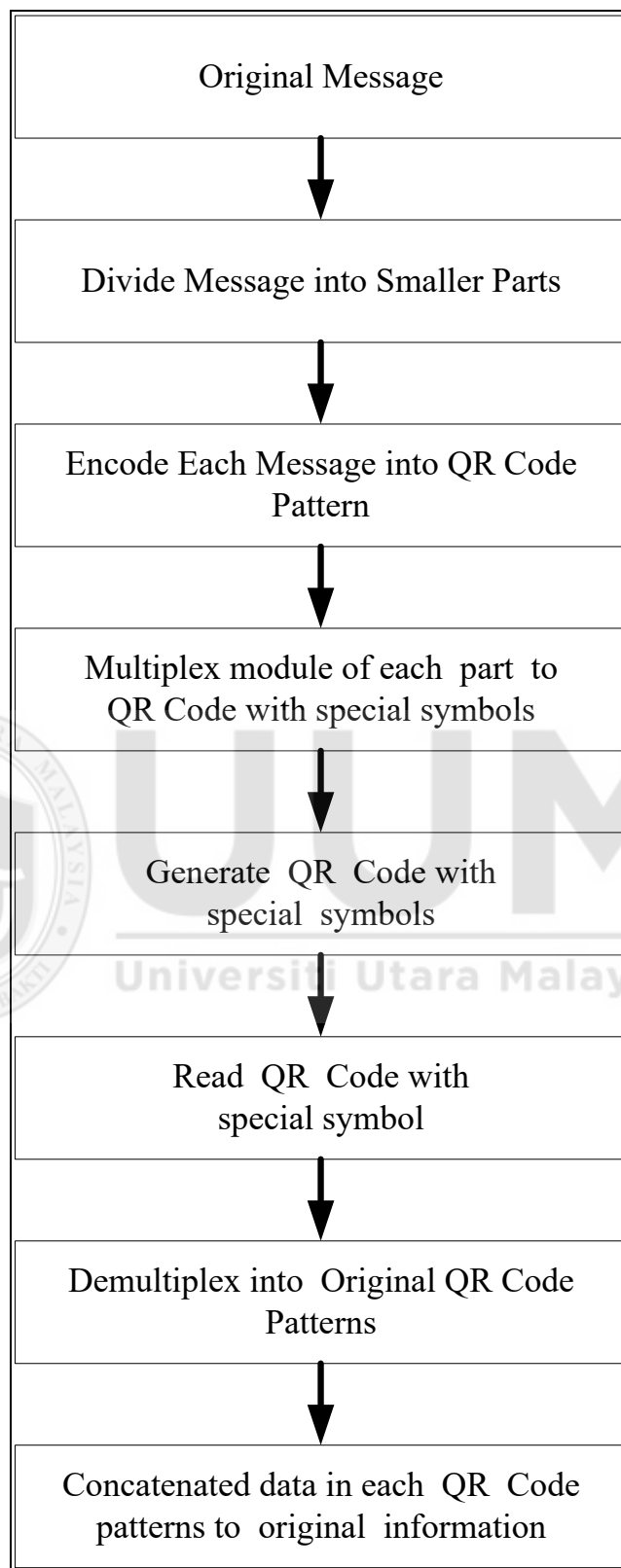


Figure 2.23. The algorithms of multiplexing and demultiplexing (Courtesy:Vongpradhip (2013)).

Table 2.10

The special symbols used for each pattern (Vongpradhip, 2013).

Module in each pattern	Symbols
0 0 0	\
0 0 1	/
0 1 0	v
0 1 1	^
1 0 0	>
1 0 1	<
1 1 0	L
1 1 1	└

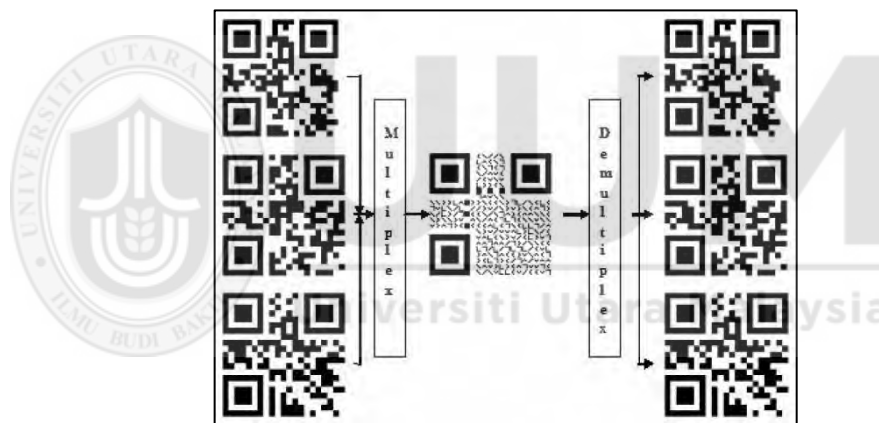


Figure 2.24. QR code with 8 special symbols (Vongpradhip, 2013).

Prathibha N. Pillai and K. Naresh (2014) started the encoding process by entering the data and converting it into ASCII code equivalents. The primitive polynomial is used to generate the ASCII equivalent finite field numbers. The Reed-Solomon encoder algorithm is used to generate code words by using finite field numbers. After that, the code word is converted into its binary and placed according to the QR code pattern. The QR code pattern is considered as the colour layers and combined to generate a coloured QR code. Figure 2.25 shows the process to produce a coloured QR code.

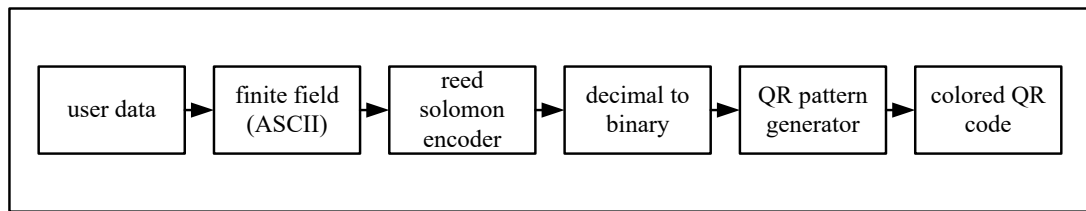


Figure 2.25. The process to produce coloured QR code (Pillai & Naresh, 2014).

Hiren J. Galiyawala and Kinjal H. Pandya (2014) divided the original information into n parts. The n part is the number of QR code used for the multiplexing process and also to be used as the required distinct colours based on the 2^n equation. Table 2.11 shows the example of distinct colour requirements for QR code multiplexing.

Table 2.11

Example of distinct colour requirements for QR code multiplexing.

Number of QR codes for multiplexing (n)	Required distinct colours
2	4
3	8
4	16
5	32
6	64
...	...
14	16384

The QR codes will be generated based on n parts. The multiplexing process starts with identifying the distinct colour by using the colouring code technique. The distinct colour will be assigned for each possible pattern after the QR code is combined. The distinct colour generated from the RGB combination colours is performed from the colour map matrix in MATLAB. The normalised values of the RGB combination of

coloured QR code are listed in Table 2.12. A colourful QR code will be generated after the distinct colour has been identified during the multiplexing process.

Table 2.12

The normalised values of RGB combination for coloured QR.

QR Code 1	QR Code 2	QR Code 3	Index	Red	Green	Blue	Colour
0	0	0	0	0	0	0	Black
0	0	1	1	0	0	1	Blue
0	1	0	2	0	1	0	Green
0	1	1	3	1	0	0	Red
1	0	0	4	0	1	1	Cyan
1	0	1	5	1	0	1	Magenta
1	1	0	6	1	1	0	Yellow
1	1	1	7	1	1	1	White

Figure 2.26 illustrates the flow of the multiplexing process of coloured QR code.

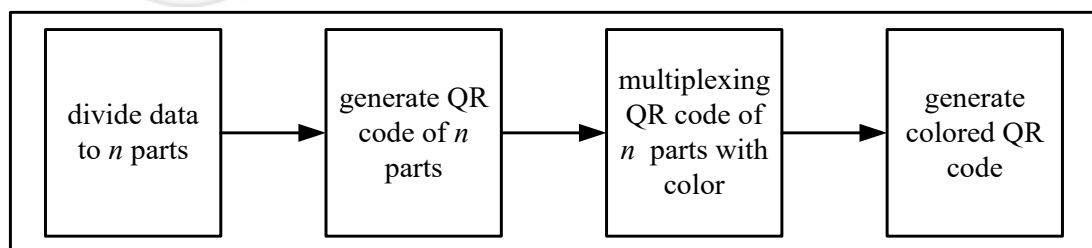


Figure 2.26. Flow of the multiplexing process of coloured QR code.

2.3.5.2 Decode

As reported by Sartid Vongpradhip (2013) in the research, the decoding phase involves reading a QR code with special symbols and demultiplexing it into the original pattern. The symbol in each module (code word) will be extracted back to each layer

(pattern). The value 0 that represents the black module and 1 that represents the white module will be restored back to each layer. After the module restoration is completed, the original QR code is ready to be read as normal. Figure 2.27 illustrates the flow of the demultiplexing process of QR code with special symbols.

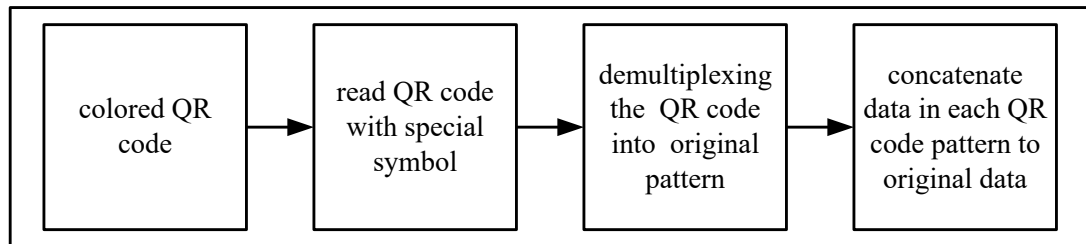


Figure 2.27. Flow of the demultiplexing process of QR code with special symbols.

Meanwhile, Pillai and Naresh (2014) started the decoding process by reading the coloured QR code image and splitting it into RGB colour layers. The split RGB colour layers contain binary information and these layers will be converted to a decimal value for each byte. The decimal values are used as an input for the Reed-Solomon decoder. The outputs of the Reed-Solomon decoder are ASCII equivalents and they will be converted into the original data. Figure 2.28 shows the flow of the decoding process.

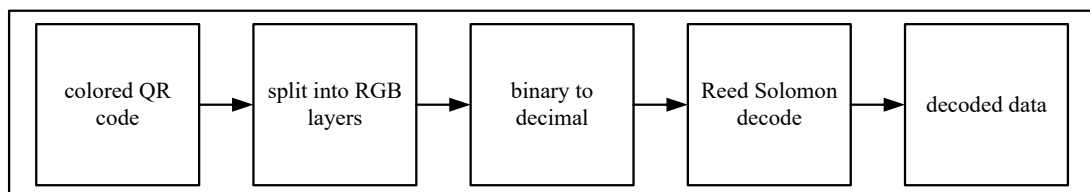


Figure 2.28. Flow of the decoding process.

The demultiplexing process developed by Hiren J. Galiyawala and Kinjal H. Pandya (2014) involves the reverse task of the multiplexing method. The table of assigned colour combination is referred to obtain the RGB values from the multiplexed coloured

QR code. The original pixel value of QR code will be restored by comparing it with all the combinations of multiplexing tables. After the original pixel value of QR code is restored, it is compatible to be scanned by reader devices. The extracted information can be concatenated to form an original long message sequence. Figure 2.29 shows the flow of the demultiplexing and decoding processes.

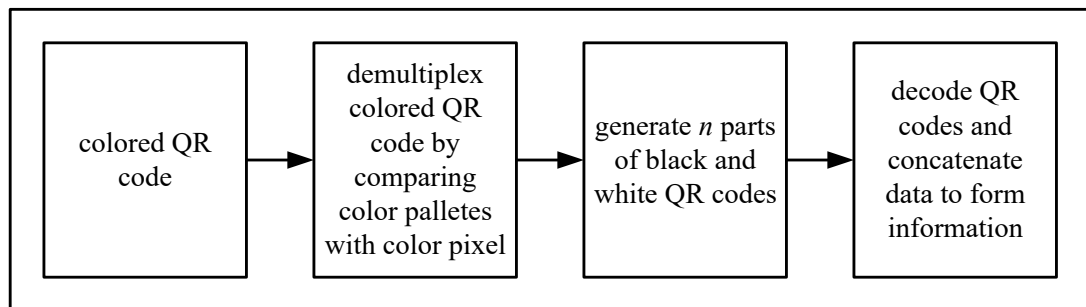


Figure 2.29. Flow of the demultiplexing and decoding processes.

2.3.6 Compression

Nancy Victor (2012) mentioned in her research entitled *Enhancing the Data Capacity of QR Codes by Compressing the Data Before Generation* that to increase the data capacity in the QR code, it needs to be compressed first by using any type of data compression techniques before the encoding process is performed. This research suggests a technique for data compression by employing two steps: (1) the first step is converting the text data into a binary form; (2) and the second step is generating the hash map data from this binary data. As a result, the data compression can exceed more than four megabytes of data in a QR code as compared to four kilobytes as before.

The compression idea is useful because it can reduce the consumption of resources (Umaria & Jethava, 2015). Some extra computational processes need to be performed

when the process of decompressing is implemented. The factors of concern when a compression process is implemented are the degree or percentage of compression, the amount of distortion, and the computational resources (Donoho, Vetterli, Devore, & Daubechies, 1998).

As in the general objective, the main idea of this research is to increase data capacity by using multiple combination techniques on the coloured QR code. Based on research by Victor (2012), normally, the data can be compressed up to four kilobytes in a normal QR code at the error correction level L (maximum); nonetheless, by using the technique proposed by the researcher, it can store more than four kilobytes. Victor (2012) also noted that it is more efficient if the data compression technique used in a QR code is capable to store more than four megabytes of data.

Error detection and colour distortion can be used as the metric measurement of effectiveness of the proposed coloured QR code. Error detection level can be looked through at the level of error correction. Meanwhile, colour distortion is not suitable to be used in this research because it is related to the ability of the peripheral/device to translate the colour when reading the image (QR code). In addition, the distortion is caused by light environment, printed colour etc.

Even though the data capacity of QR code can increase drastically beyond four kilobytes, some enhancements of QR code encoding and compression algorithm techniques can be used to add more data capacity capability as compared to the normal QR code. Data capacity can be improved by mixing the features in compression and

coloured QR code generation. There are three criteria that can be experimented, namely data storage capacity, reading speed, and accuracy of the data (Victor, 2012).

2.3.6.1 Encoding

The idea to generate a high capacity QR code is by starting with a simple compression of the data before the encoding process is performed. The implementation of encoding is done through the normal processes of encoding the QR code, which are as follows (Feng & Zheng, 2010b):

1. Analyse the data and convert the data to symbol characters. Identify the error correction and detection level.
2. Encode the data.
3. Initiate error correction coding.
4. Add remainder bits and data masking patterns.
5. Generate the format information and version information (Hahn & Joung, 2002).

Figure 2.30 illustrates a flow chart in generating a high capacity QR code.

As what was proposed by Victor (2012), a coloured QR code can be developed from the hash map data. By using the RGB colour coding technique, every letter from 'A' to 'P' will be assigned with different colours. Whenever such a character is encountered, the corresponding colour will be placed in the QR code. The total bit for colour allocation is four bits.

Another remainder of four bits is assigned to an allocation of sixteen characters in the

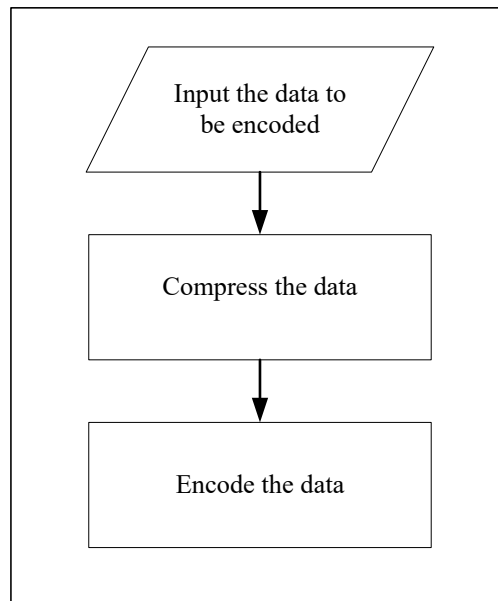


Figure 2.30. The flow chart in generating a high capacity QR code (Courtesy: Victor, 2012).

hash map. For example: ‘a’ to ‘o’ and ‘p’ to ‘z’. Every sixteen character allocation map will be represented as a group of colour codes. The steps to generate a large amount data for QR code is shown in Figure 2.31.

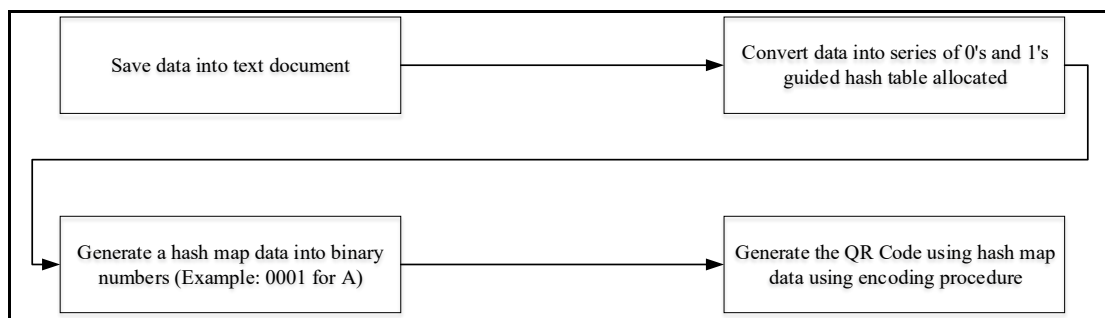


Figure 2.31. The steps to generate a large amount data for QR code.

When the above activities are completed, the coloured QR code generation can be developed by combining both four bits. Figure 2.32 shows the example of how the

hash map data can be encoded into a 2D barcode. The colour mapping is limited to 16 colours because the four bits can have only 16 binaries.

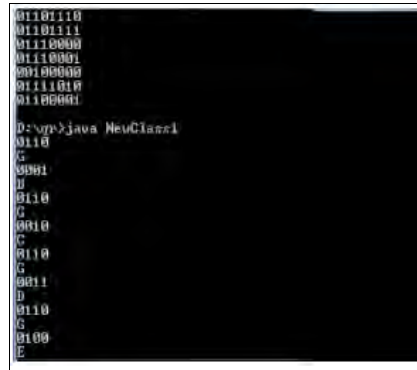


Figure 2.32. The hash map data can be encoded into a 2D barcode (Courtesy: Victor (2012)).

2.3.6.2 Decoding

The decoding process is the reverse of the encoding procedure (Victor, 2012). The quiet zone needs to be identified in order to decode the correct data. In the decoding procedure, the distortion of symbols can be corrected by alignment patterns. Two techniques are recommended including the “edge to similar edge” estimation method to check whether the detected bar-space pattern is correct (Hahn & Joung, 2002) and the preprocesses that are conducted before performing the pattern finder. The preprocesses include greyscaling of the captured colour image, histogram stretching of the image, local adaptive thresholding, noise filtering, cropping, rotation correction, and tilt correction (Falas & Kashani, 1994). The decoding process is performed when the preprocesses and pattern finder are performed. Victor (2012) claimed that the decoding process can be done from its encoded coloured QR code, but there is no experiment result attached. It is just a theoretical concept.

2.3.7 Hybrid Extension

Many techniques were presented by the researchers in regard to increasing the data capacity of QR code. Several of the techniques are compression (Victor, 2012), multiplexing (Ferreira & André, 2014; Vongpradhip, 2013), and multilayer (Nurwono & Kosala, 2009). These techniques can be incorporated if the suitable method is implemented together.

Three techniques were chosen to extend the data capacity. The techniques are compression, multilayer, and multiplexing. At the same time, there will be several possibility problems during the implementation. Table 2.13 shows the possibility problem experience if the priority exchange is implemented.

Table 2.13:

The possibility problem experience if the priority exchange is implemented.

Priority 1	Priority 2	Priority 3	Remarks
Compression	Multiplexing	Multilayer	It is suitable because it needs to compress the data first, then run the black and white QR code, followed by generating the coloured QR code through the multilayer technique.

			<p>Multilayer involves the coloured QR code. Meanwhile, the multiplexing technique covers the black and white QR code. As a result, it consumes more processing time when converting from coloured to black and white QR code. It is hard to perform the conversion.</p>
Compression	Multilayer	Multiplexing	
			<p>Not suitable because the text file needs to be compressed first.</p>
Multiplexing	Compression	Multilayer	<p>It consumes more processing time when decoding the QR code back for text compression.</p>
			<p>Not suitable because the text file needs to be compressed first.</p>
Multiplexing	Multilayer	Compression	<p>It consumes more processing time</p>

			when decoding the QR code back for compression.
			Not suitable because the text file needs to be compressed first.
Multilayer	Multiplexing	Compression	It consumes more processing time when decoding the QR code back for compression.
			Not suitable because the text file needs to be compressed first.
Multilayer	Compression	Multiplexing	It consumes more processing time when decoding the QR code back for compression.



UUM
Universiti Utara Malaysia

The compression must be generated first in order to save more characters. As explained by Victor (2012), the text must be compressed first before the QR code is generated. Figure 2.33 below shows the processes involved when the techniques of compression, multiplexing, and multilayer change their positions. From the abovementioned figure, the first technique shows the least processes as compared to the second and third techniques. The main reason is when the compression technique is moved to another position after the text, it needs extra processing time in order to obtain a high capacity

QR code. Even if the compression technique is put in the last position, the QR code cannot be generated. As a conclusion, the first technique is the best solution to gain more data capacity in the proposed QR code.

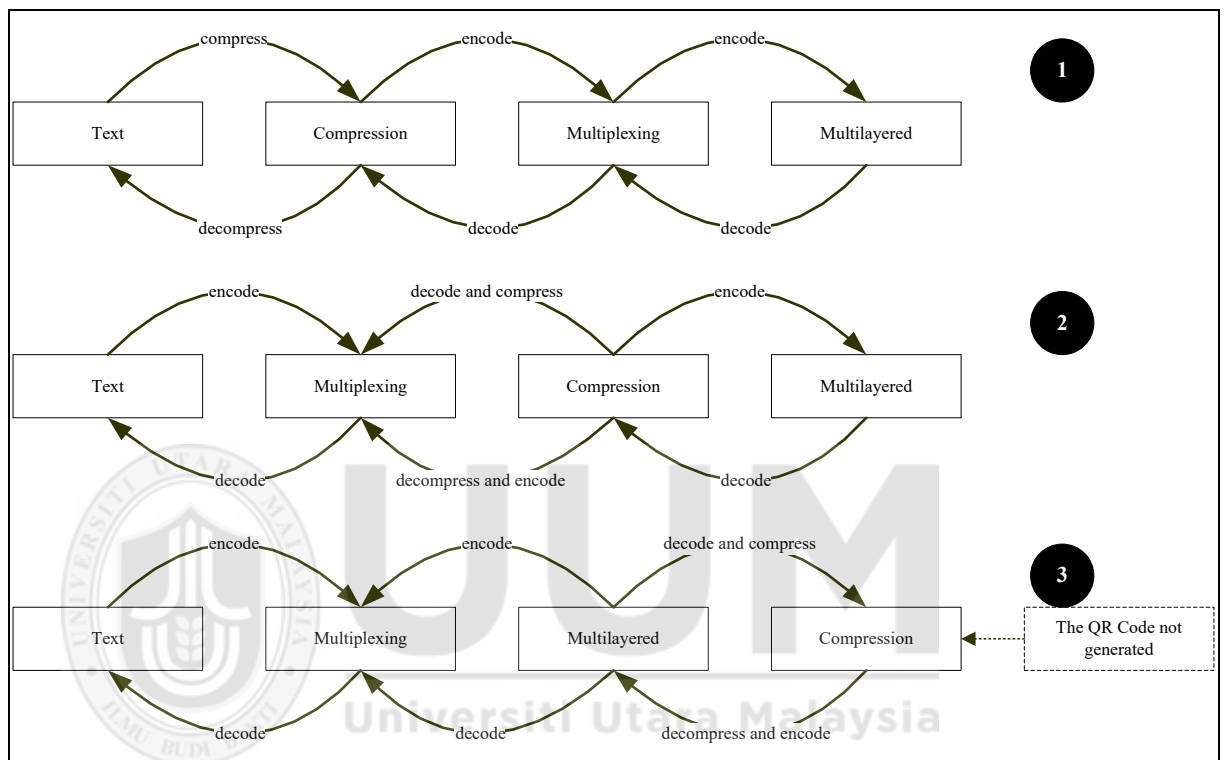


Figure 2.33. The processes involved when the techniques of compression, multiplexing, and multilayer change positions.

There are three proposed techniques that were based on the previous studies of several researchers with some modification. The technical breakdown is as follows:

1. **Compression technique.** Based on a preliminary test performed by Azizi Abas, Yuhanis Yusof, and Farzana Kabir Ahmad (2015), it showed that the GNU Zip (GZip) compression was the best compression utility to compress text by using QR code. In addition, it must be added with Base64 encoder/decoder.

2. **Multiplexing technique.** Vongpradhip's (2013) research was chosen because the QR code was still generated based on black and white colours. Moreover, Vongpradhip (2013) mentioned that the QR code needs to be separated into small parts before it can be multiplexed. With this idea, the compressed data will be separated based on the maximum capacity storage of QR code version 40. The symbols are not used to multiplex because there are limited symbols offered by characters in the operating system. To solve this problem, another technique is utilised by using pixel modules in QR code version 40. In this case, 177 x 177 modules will be plotted with black representing 1 and white representing 0.

3. **Multilayer technique.** A coloured QR code will be generated based on a combination of red, green, and blue QR code layers from the multiplexing process. Nurwono and Kosala (2009) proposed a pseudocode to generate a coloured QR code auto generator. The coloured QR code is built based on the combination of colours from two layers. For example: the red and green layers meet at pixel position (10,25). Then, the Red RGB value `#FF0000` and Green RGB value `#00FF00` are added, so the new RGB value for that pixel is (255,255,0) or `#FFFF00`. After that, the inversion of this value uses a logical NOT, so the final value is `#0000FF`. Then, the final value is stored in the new byte array. For some reason, this technique will encounter a problem when the White RGB value `#FFFFFF` from layer 1 and White RGB value `#FFFFFF` from layer 2 at the same pixel position meet. The new value for that pixel cannot be assigned because the result of new value is `#IFE` for each segment, which is not valid in the RGB colour system. The changes must be made by using multiplexing first to obtain 8 bits colour identification for each RGB segment.

2.3.8 Structured Append

The main feature of structured append in QR Code 2005 is the capability to divide a maximum of 16 symbols structure format in a single QR code. The 16 symbols can be divided into multiple data areas and QR symbols. The benefit of structured append in QR code is that it allows printing in a narrow area (British Standards., 2009).

There are two types of mode in the structured append 8-bit code word that is placed in a header block. They are:

1. **The position of the particular symbol (4 bits).** Located in the first four bits of the header code word and identified as the position of the particular symbol. If the indicator is 0011, then it refers to four symbols.
2. **The total number of symbols to be concatenated (4 bits).** Located in the last four bits of the header code word and identified as the total number of symbols to be concatenated in the structured append format. If the indicator is 0110, then it refers to the total amount of symbols.

The bit pattern in a header block is the combination of position and total number of symbols. For example: if the position is *0111* and total number of symbol is *0100*, the bit pattern is *01110100*. Figure 2.34 illustrates a single symbol and the structured append of symbols encoded with *"ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"*

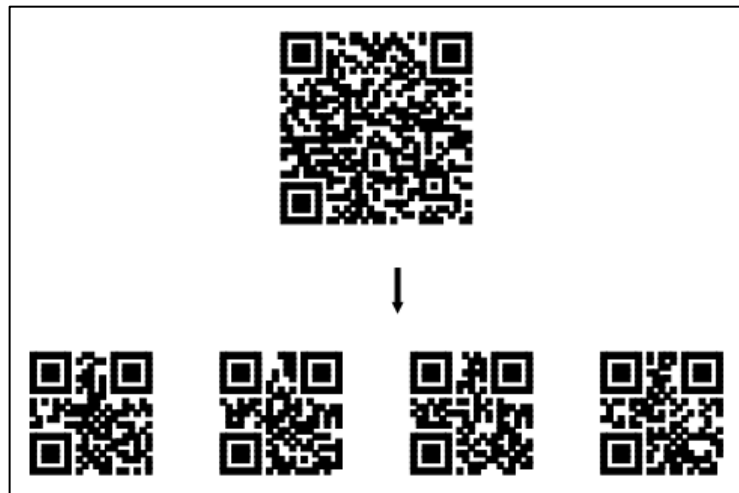


Figure 2.34. Single symbol and the structured append of symbols encoded with "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ RSTUVWXYZ".

2.3.8.1 Implementation of partial extraction

Partial extraction is a method to manage the data inside the QR codes. The activities include displaying, adding, updating, and deleting an operation of a decoded QR code. This process is similar to the structured append format; nonetheless, it involves a selected single QR code and not the symbols inside a QR code. This process will improve the management of data inside the QR code. The structured append in QR Code 2005 is limited to 16 symbols, but partial extraction is unlimited based on how many QR codes are combined. Figure 2.35 illustrates the methods of partial extraction. Usually, the process starts with decoding the new combined QR code until the original QR code is produced. The data in the original QR Code will be decoded, followed by managing the data (displaying, adding, updating, and deleting operations). After the process of managing the data is completed, the set of original QR codes will be re-encoded.

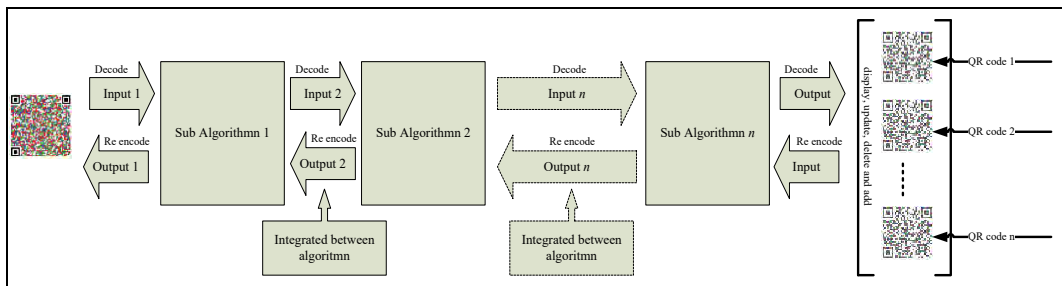


Figure 2.35. The methods of partial extraction.

2.4 Combination Techniques of QR Code Data Capacity

Based on an analysis performed by Feng and Zheng (2010), the performance of 2D barcode is divided into three areas, which are as follows:

1. Information capacity. By adding colours, the 2D barcode can improve its capability to store more information. The coded rules need to be modified in order to enhance the information capacity.
2. Information compression ratio. The high compression ratio will extend the total amount of characters that can be stored. In addition, the QR code can store binary values that can be used in the compression file.
3. Correcting capability. The high percentage of recovery value is good for QR code because it will improve the QR code to be more robust and accurate in data accessibility.

Based on Galiyawala and Pandya's (2014) research in Table 2.14, the processing time to encode and decode 14 QR codes took approximately 53.153 and 1236.105 seconds, which is time-consuming to complete the task. It uses 16,384 colours to expand the data capacity in QR code with an amount of 14 QR codes.

Table 2.14.

The processing time of encoding and decoding (Courtesy: Galiyawala & Pandya (2015)).

Sr. No	Number of QR codes multiplexed	Assigned distinct colours (2*)	Processing time (seconds)	
			During encoding	During decoding
1	2	4	2922	2.892
2	3	8	3194	2.988
3	4	16	3364	2.982
4	5	32	3131	3.084
5	6	64	3297	3.381
6	7	128	3489	3.624
7	8	256	3911	4.359
8	9	512	4811	6.128
9	10	1024	6229	11.406
10	11	2048	9453	28.398
11	12	4096	15787	91.393
12	13	8192	27940	323.198
13	14	16384	53157	1236.11

The models and algorithms developed by Vongpradhip (2013), Nurwono and Kosala (2009), and Victor (2012) are chosen based on several reasons. Among the reasons are:

1. **Data capacity.** The new enhanced QR code is able to generate four megabytes of data storage (Victor, 2012) and the data increment can achieve three times in a single QR code (Vongpradhip, 2013).

2. **Colours.** Red, green, and blue are the colours tested by researchers regularly. The colour combination of layers are only tested using a photo image application (Adobe Photoshop). This technique is not a suitable method to experiment with the colours and to combine them between the layers.

3. **Compression.** The compression technique is a value-added function to this research because Victor (2012) mentioned in her research work that the data storage capacity of QR codes can increase drastically if the compression technique is implemented.

Even though three research works are chosen, modifications must be made to combine the input and output between the three methods. As in objective three, partial information can be created using the index display technique. It means that the data will be displayed using the selected index.

2.5 Summary

This chapter provided the background studies of the subjects tracked in this proposition. The first items considered are the organisational process of the QR code and the overview as support knowledge. The research on QR code was explained in detail, especially on encoding, decoding, and partial extraction processes in QR code. Partial extraction was explained in the way of its functionality and capability. As a conclusion, several works from other researchers will be denoted as the necessary foundation knowledge of the whole inquiry.

CHAPTER THREE

RESEARCH FRAMEWORK

This chapter describes the methodology used to achieve the research objectives. The research methodology contains the methods and techniques that are used by the researchers in a systematic way to solve a problem in bringing out the research outcome; for example: data collection techniques, data processing techniques, and instruments (Rajasekar, Philominathan, & Chinnathambi, 2013). This research applies a deductive approach (Trochim, 2015), which begins with a general idea (such as theory, principles, and concepts) and need to commented to the other chapters. As a strategy, each phase have its own method and outcome.

3.1 Research Methodology

The process for this research is as illustrated in Figure 3.1.

3.1.1 Phase One

The first phase is called the preliminary study as shown in Figure 3.1. The tasks involve searching for information related to various sources and selecting the suitable algorithms, which can be executed successfully before any modification is made on the selected algorithm. Three components are to be considered when this phase is implemented, namely creating the theoretical framework, testing existing algorithms, and determining the suitable algorithm to be merged.

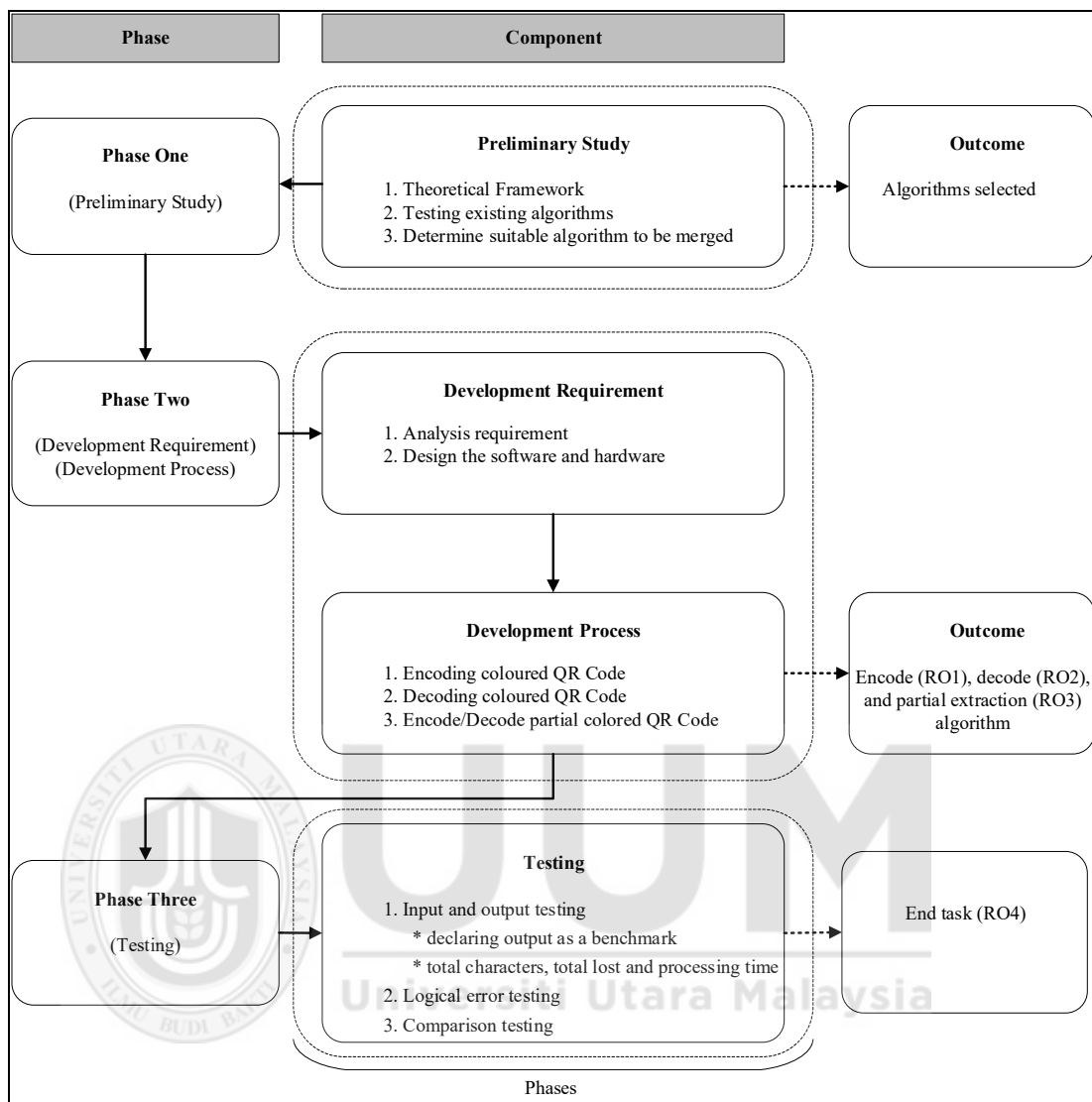


Figure 3.1. The research framework.

3.1.1.1 Theoretical Framework

The theoretical framework (Figure 3.1) focuses on a research question that tries to identify, analyse, select, and synthesise research evidence relevant to that question. In this research, the steps include the following:

1. Investigating storage increment methods in QR codes. This information can be found in the literature of previous researches and commercial resources (Section 2.2 and Sections 2.3.4 until 2.3.8).
2. Identifying and collecting information from one or more databases to search the algorithm used in QR code and coloured QR code, specifically in increasing data capacity and compression. Online/offline databases such as e-journals, e-papers, books, documents, proceedings and others are reviewed. This research aims to identify the state-of-the-art method in QR code related to data capacity and compression algorithm.
3. Developing an explicit search strategy to select the suitable algorithms produced from previous researches. The reason is to determine the most relevant algorithms for research work. The criteria based on successful execution, processing time and total characters stored.
4. Selecting titles, abstracts, and implementation based on explicit inclusion after they are identified from step (3).

The output discussion of the theoretical framework can be found in Sections 2.3.7 and 2.3.8.

Figure 3.2 shows the theoretical study of the first item in the first phase of the research framework strategy.

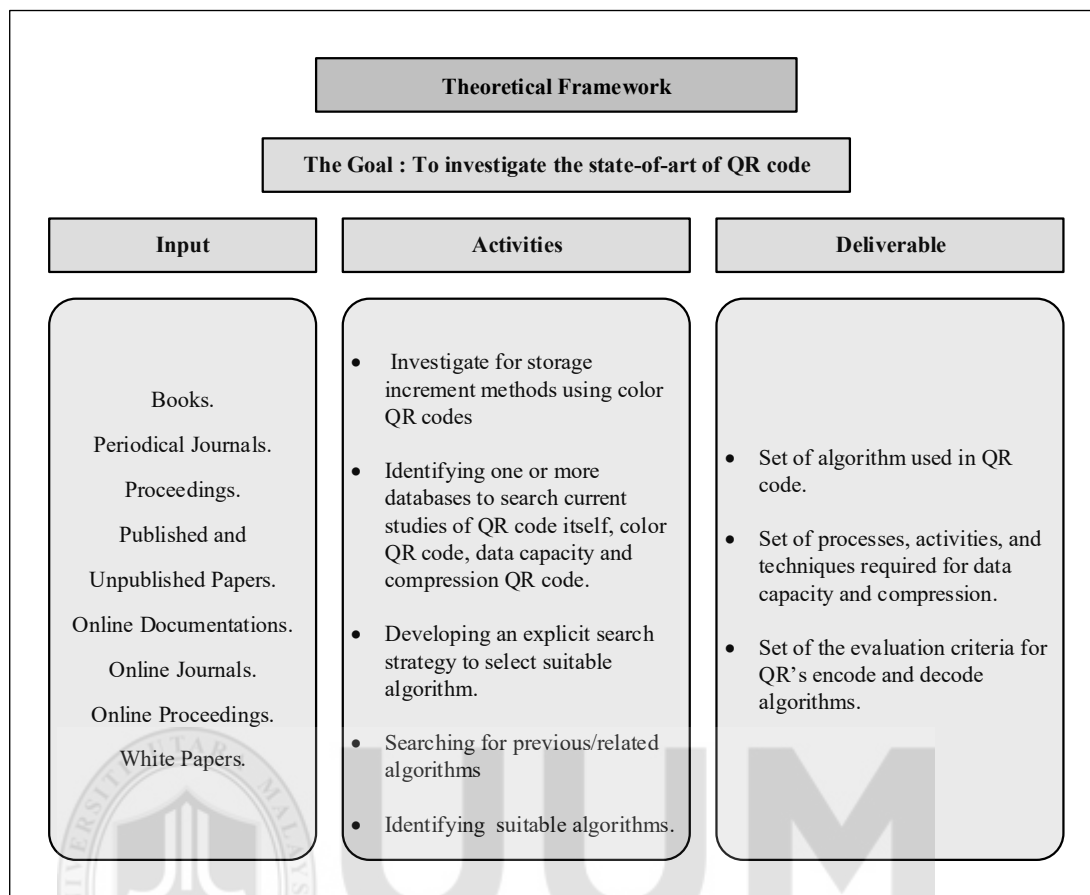


Figure 3.2. The theoretical framework.

3.1.1.2 Testing Selected Existing Algorithms

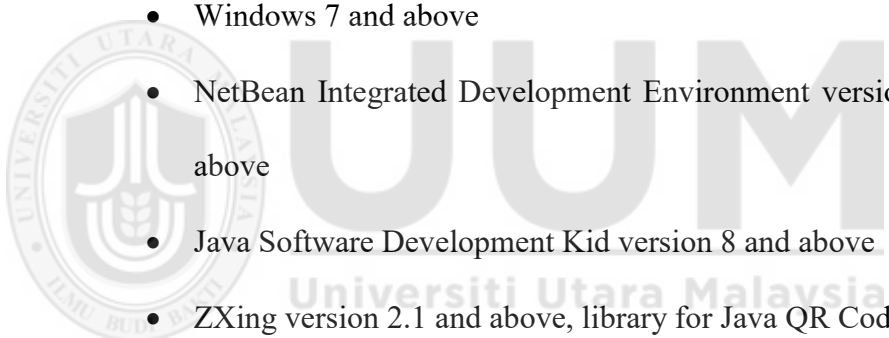
The second component is more on testing the selected existing algorithms, which include the following:

1. Finding the flow charts, pseudocodes or programming codes that were developed to increase the data capacity of coloured QR code (Sections 2.3.4 until 2.3.6 and Section 2.3.8).
2. Assembling the selected algorithm as a preparation for the coding process. Reorganising the pseudocode, flow chart, and codes into a readable flow to ease the process of coding them back afterwards.

3. Identifying the hardware. A personal computer will be used to develop the application. Among the hardware are as follows:

- Intel i7 processor
- 8GB DDR4 memory
- 700GB storage allocation
- 1440 dpi printer

4. Identifying the software. The selection of software includes system, application, and programming language software.

- 
- Windows 7 and above
 - NetBean Integrated Development Environment version 7.2.1 and above
 - Java Software Development Kit version 8 and above
 - ZXing version 2.1 and above, library for Java QR Code
 - Apache Commons Compress library version 1.9 and above
 - Graphic display application such as Microsoft Paint etc.

5. Developing the encoding and decoding processes. The developing process will follow the steps in the algorithm created by the researcher.

6. Testing each algorithm based on data capacity (Grillo et al., 2010; Victor, 2012), processing time (Galiyawala & Pandya, 2015; Grillo et al., 2010), and error correction level (Pillai & Naresh, 2014) only. These three testing methods are the most evaluated methods by the researchers in their research (Refer Tables 2.6 and 2.8)

Figure 3.3 shows the testing activities and outcomes towards choosing the best algorithm for data capacity in QR code.

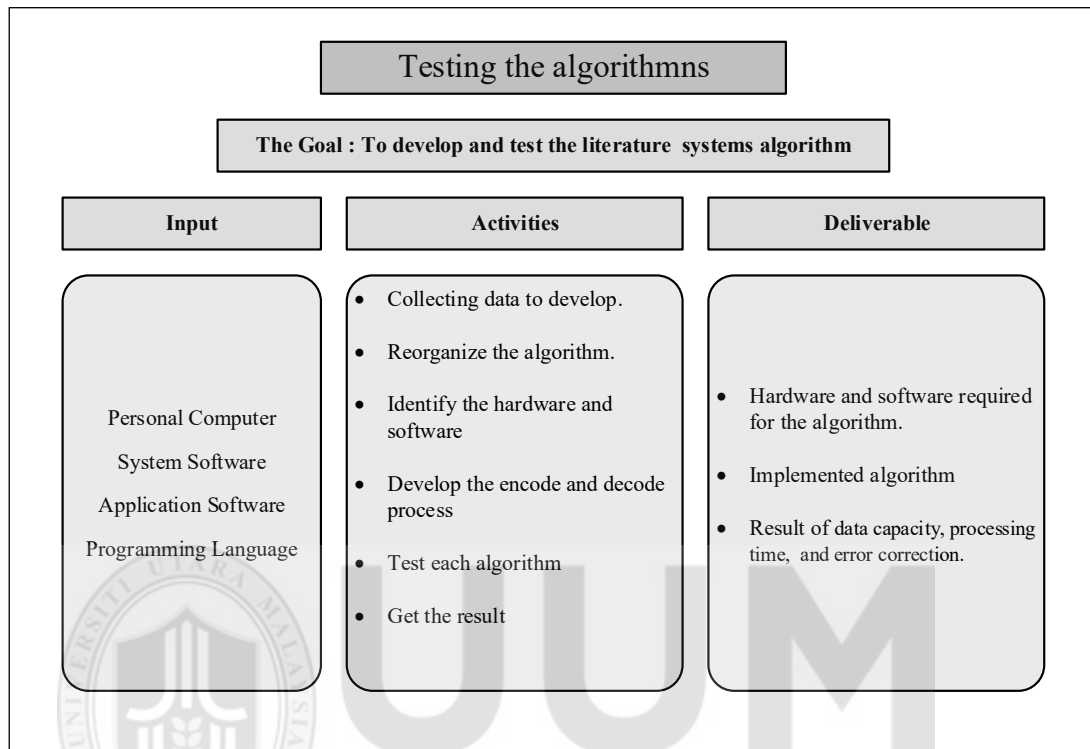


Figure 3.3. The testing activities.

3.1.1.3 Merging the Algorithms

The third component focuses on merging the three identified algorithms.

1. Using the test results to gain the compatibility among the three algorithms when the merging process is implemented and to gain the maximum data capacity that can be stored.
2. Determining the priority of the algorithm based on the test.
3. Synchronising the input and output among the selected algorithms.

Figure 3.4 illustrates the finalising and merging activities as a guideline to develop a new enhanced QR code.

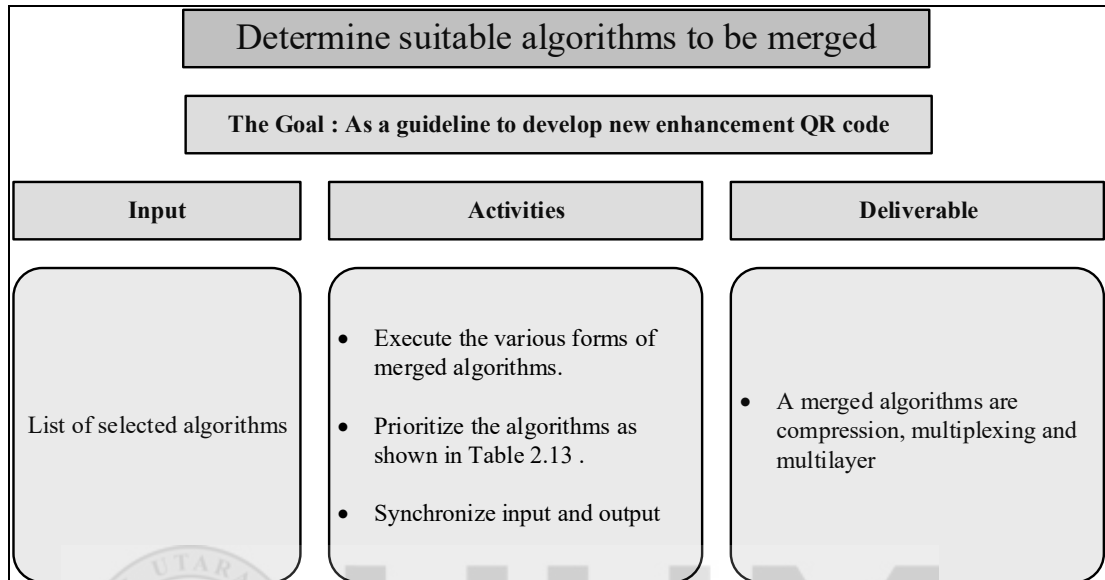


Figure 3.4. The finalising and merging activities.

3.1.2 Phase Two

This phase is concerned with the design and development of a coloured QR code. It is divided into two parts: development requirement and development process.

3.1.2.1 Development Requirement

The development requirement phase investigates the processes involved in developing a coloured QR code. This is achieved by analysing the required information on hardware, software, and steps to develop a coloured QR code. The coloured QR code algorithm consists of several algorithms, namely compression, multiplexing, and multilayer. These algorithms will be designed for integration based on the required input and output. Figure 3.5 shows the proposed flow of the encoded and decoded

coloured QR code. The partial extraction processes of a coloured QR code are illustrated in Figure 3.6.

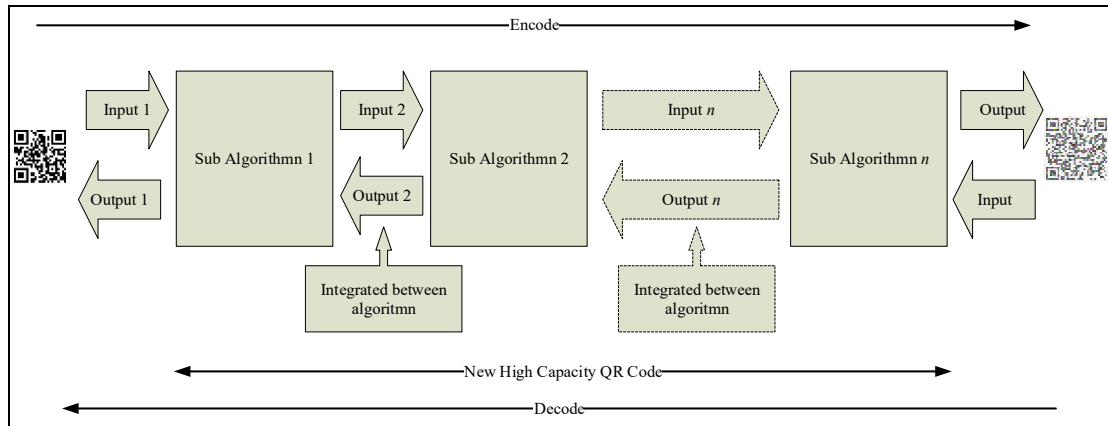


Figure 3.5. The proposed flow of the coloured QR code.

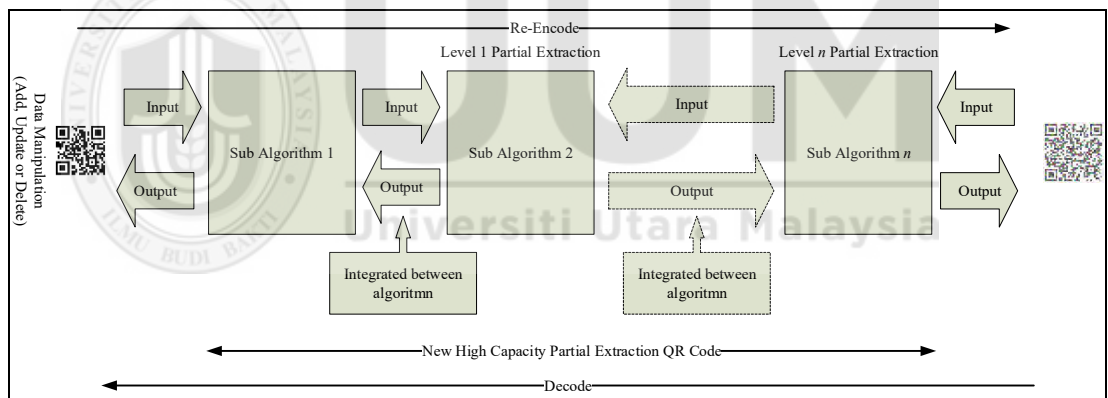


Figure 3.6. The proposed flow of the partial extraction process of coloured QR code.

3.1.2.2 Development Process

This phase involves three main processes, which are encoding, decoding a coloured QR code, and partial encoding/decoding a QR code. The processes include writing separate encoding or decoding programmes by using Java core libraries and following the *Javadoc* comment style. This is followed by merging the encoding and decoding

programmes. Next is the development of partial encoding/decoding algorithm based on the required portion of information. It will be based on the proposed encoding and decoding implementation of coloured QR code.

The illustration in Figure 3.7 shows the flow steps of the coding development process. The algorithm from 1 to n consists of a combination of selected algorithms chosen from Phase One. Algorithm 1 is the first priority to be executed in this development process. Meanwhile, the algorithm n is the last portion to be executed and it will produce the output, which is the coloured QR code. Before the integration of algorithms is implemented, each algorithm is developed based on the encoding, decoding, and partial extraction processes. Each algorithm is combined or merged to build a completed model of high density coloured QR code after completing the test task. Since the design and development of encode, decode and partial extraction are completed in Phase Two then research objective one, two and three are fulfilled simultaneously.

3.1.3 Phase Three

After the development processes are completed, the testing phase is performed. The input starts with the ability of QR code to store the characters inside, according to its version and error correction level. The character allocation in the QR codes for version 40 has been set with a limited amount of characters according to the error correction level, which is shown in Table 3.1. The character allocation is based on the preliminary test (Abas et al., 2017).

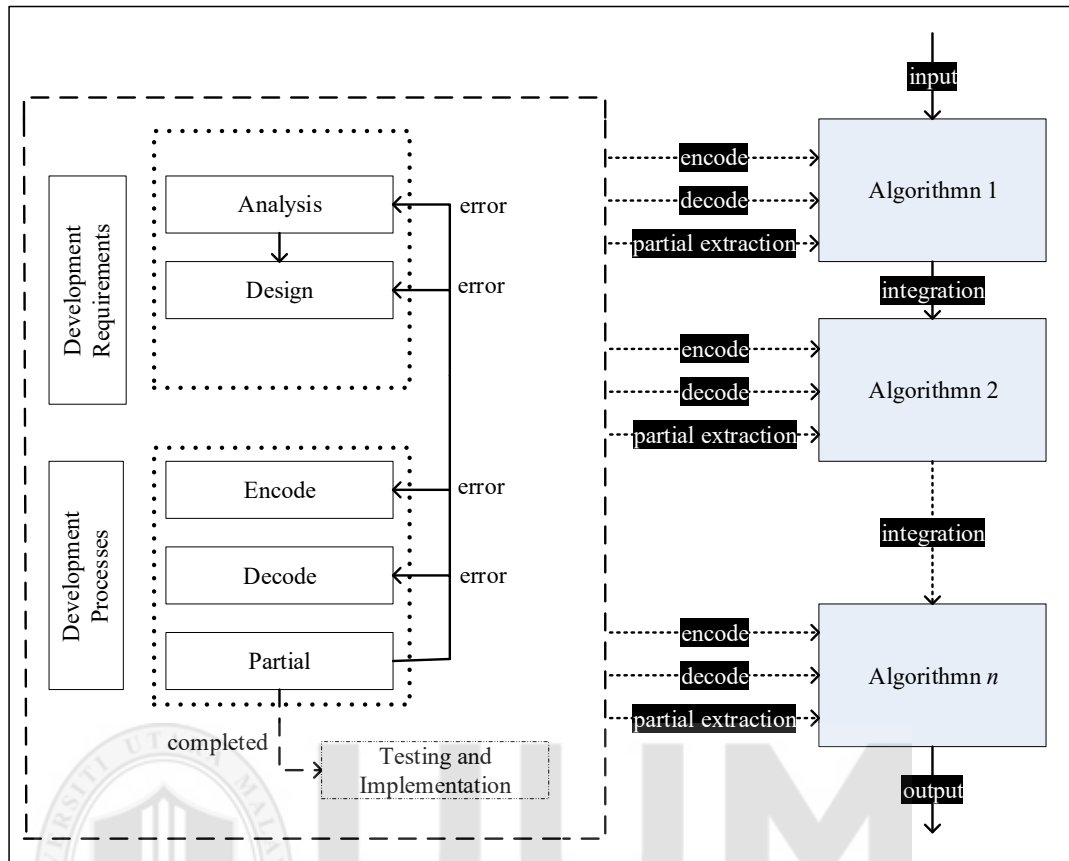


Figure 3.7. The flow steps of the coding process.

Table 3.1

Maximum number of characters based on error correction level.

Error Correction Level	Total Characters for each QR Code
L	2952
M	2330
Q	1662
H	1270

Error detection and colour distortion are not discussed in this research. Error correction level, data density, and processing time are used to test the newly proposed data capacity of QR code.

3.1.3.1 Testing

The testing involves the utilisation of several input and output data. It is to ensure that the proposed algorithm is implemented accordingly. The testing is to identify that the programme execution is running well, efficient, robust, and reliable.

The experiment metric employed in this research is as follows:

1. **Data density.** To validate the total amount of ASCII codes that can be stored in the coloured QR code. The larger the value, the better the algorithm is. It is formulated based on how many characters are successfully encoded. The unit of measurement used in this testing is the total amount of characters saved in the coloured QR code.

2. **Accuracy order by error correction level.** To validate the data restoration rate.

The higher the value, the better the algorithm is. The unit of measurement used in this accuracy test is the percentage of character lost, which is:

$$(Total\ unit\ character\ decode / Total\ unit\ character\ encode) * 100$$

3. **Processing time.** To verify the total amount of time taken to complete the process.

The smaller the value, the better the algorithm is. This processing time is formulated based on:

$$End\ task\ time - Start\ task\ time$$

The metric unit used in this processing time measurement is millisecond.

The testing includes:

1. **Input and output testing.** Several numbers of characters are used as input. The reason is to validate the maximum total number of characters that can be held by the QR code. The output needs to be validated as well because it will help to confirm the stored total number of characters. The characters include alphanumeric, numeric, binary, and several common symbols in the ASCII. The total characters need to be identified during decode processes and the total processing time will be calculated during encode, decode and partial extraction. This result is used as a benchmark to compare with other system.
2. **Logical errors.** This testing is to check the unintended or undesired output or other behaviour in the algorithms that will cause them to operate incorrectly.
3. **Comparison testing.** The comparison testing involves examining the result of the proposed algorithms against QR version 40 and others high capacity coloured QR code. The QR version 40 is used as a benchmark for all comparison testings including high capacity coloured QR code.

After the testing part in Phase Three that has been implemented, the research objective four is fulfilled after this process completed.

3.2 Summary

This chapter has highlighted the research procedures, which gave detailed description of the steps taken in completing the research in this thesis. The procedures include the phases of preliminary study, development requirement, development process, and testing. Besides, this chapter also discussed the design of the proposed algorithm which

is the coloured QR code that utilizes the RGB colour combination. Such contribution (refer to Chapter Four) includes details on encoding, decoding, and partial extraction processes. The encoding process provides a proposed approach that could significantly improve data density and processing time of the coloured QR code as compared to the conventional QR code. Meanwhile, the decoding process retrieves the total characters that have been successfully encoded. Lastly, the partial extraction process is able to improve the processing time to regenerate the coloured QR code for data manipulation. On the other hand, this research also investigates the merging of algorithms to generate a coloured QR code with a larger capacity storage. It is later tested based on three measurements: data density, accuracy, and processing time. This research framework is a guideline to complete all the research objectives that have been stated.



CHAPTER FOUR

ARCHITECTURE OF PROPOSED COLOURED QR CODE

This chapter explains the design and development of coloured QR code starting from encode, decode and partial extraction. The design and development are based on the methodology from Chapter 3. The process starts with designing the encode algorithm and follow by development of encode algorithm. After that, it will proceed with the design and development for decode and partial extraction. The processes contain initialization, counting characters, creating black and white QR code, placing the pixel and etc. The algorithm, pseudo code and flow chart are displayed which can bring more understanding about the processes flow. After all the process are completed, the empirical analyst will be implemented in order to give more impact on the next research finding.

4.1 Encode Algorithm

The encoding algorithm is a step-by-step process to produce a coloured QR code. This chapter presents the results gathered from the experiment concentrating on encoding modules based on the algorithm developed. In the beginning, all modules are explained in the pseudocode mode because it is easier to explain. Then, all the pseudocodes will be converted into the algorithms as shown in Appendix B. The findings are based on processing time, total characters successfully encoded by type of error correction level, total text file allocation after compression, and total bytes of coloured QR code image. The following sections describe the findings of the experiment in detail.

4.1.1 Encode Module

In general, the encoding modules used in this research are employed to generate a complete coloured QR code that can store a large amount of data (focusing on text characters as input). In this research, the encoding modules can be referred to in Figures 3.5 and 3.6, which contain the development requirement and development processes in Phase Two. Technically, Phase Two contains the preparation to obtain a proposed model to generate a high density coloured QR code. The process to encode a coloured QR code involves compression, multiplexing, and multilayer sub-modules.

4.1.2 Encoding Steps

There are some steps to be considered when encoding the coloured QR code. The steps involve the following three sub-modules:

1. **Compression.** The steps from compression are made by providing the prerequisite text preparation and execution of compression.
2. **Multiplexing.** When compression succeeds, the multiplexing process will be prepared by converting binary to text, converting the American National Standards Institute (ANSI) code to Unicode Transformation Format 8 bit (UTF-8) and multiplexing it.
3. **Multilayer.** After multiplexing is completed, the multilayer process will combine the red, green, and blue QR codes into a coloured QR code. The encoding flow process is shown in Figure 4.1.

In the previous researches (see Sections 2.3.2, 2.3.3, and 2.3.4), the encoding processes are executed based on a single module as explained in Chapter 2. The researchers did

not concentrate on merging other modules so as to gain extra storage for the QR code. Some of the methods used in each module are different from this research. The names of the modules are similar to this research and the output concentrates on producing the coloured QR code.

Figure 4.2 shows the pseudocode to encode a coloured QR code.

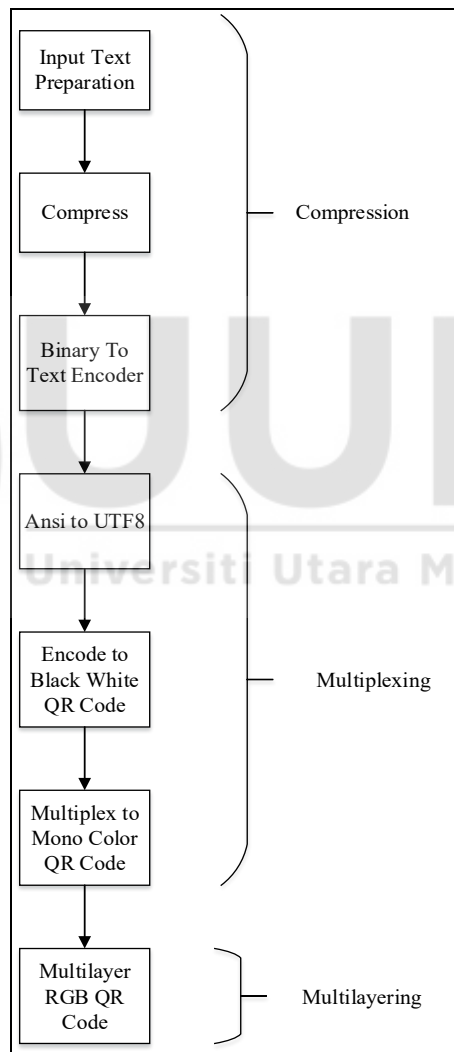


Figure 4.1. The encoding flow process.

1	: Initialize possible initialisation for file type, file input, file output, file location, and count text (Module Index E1)
2	: Compress using compression module (Module Index E2)
3	: Encode binary to text using encoder (Module Index E3)
4	: Convert ANSI to UTF-8 (Module Index E4)
5	: Produce black and white QR code (Module Indices E5, E6, and E7)
6	: Convert black and white QR code to Monocolour QR code (Module Index E8)
7	: Convert monocolour QR code to coloured QR code (Module Index E9)

Figure 4.2. Coloured QR code encoding pseudocode.

There are several modules involved in this algorithm that are represented by index numbers (See Appendix B). The index number identification for encoding modules are shown in Table 4.1

Table 4.1

Module index number identification for detailed encoding process.

Module Index	Description
E1	To count all ASCII printable and control characters, such as line feed and carriage return. The text file is used as input file and the total characters are used as a division of text for each black and white QR code.
E2	To compress a text file using an identified compression tool or algorithm.

- E3 An encoder is used to convert from binary to text. It will minimise the storage capacity of text file after compression is completed.
- E4 Conversion of ANSI to UTF-8 character encoding due to compatibility of QR code to store the data inside it.
- E5 To create N value of blank files and each can contain a maximum of 2,952 characters inside. As a default, the 2952 space characters will be allocated inside each N value file.
- E6 Divide the characters from UTF-8 file into 2,952 characters each and append them to the N value files. If the total of divided characters is more than N value files, the system will reject.
- E7 Creation of N value of black and white QR codes.
- E8 Creation of red, green, and blue QR codes.
- E9 Creation of coloured QR code.

Note: N value consists of total amount of text files or black and white QR codes.

4.1.2.1 Encoding Compression Modules

The encoding compress algorithm consists of two modules, namely count character module and compression module. These modules are separated due to their different tasks. The first module focuses on counting the module to identify the amount of ASCII printable and control characters, while the second module focuses on compressing the file.

4.1.2.1.1 Count Character Module

This is the first step to fulfil the compression procedure known as preparation input text. The compression process started by counting the total amount of characters

involved before the actual compression process began. The characters were counted first to avoid a surplus of characters that would cause the failure of creating the QR code. The character codes map in this research is referred to ASCII printable characters (alphabets, numeric, and symbols) and two control characters, namely carriage return and line feed. The other types of characters except as mentioned above are not supported; for example, Arabic, Japanese, Chinese etc. Table 4.2 shows the complete character code map in ASCII printable characters.

Table 4.2.

The complete character code map for ASCII printable characters.

Dec	Character	Description	Dec	Character	Description
32	Space	space	80	P	uppercase
33	!	exclamation mark	81	Q	uppercase
34	"	double quote	82	R	uppercase
35	#	number	83	S	uppercase
36	\$	dollar	84	T	uppercase
37	%	percent	85	U	uppercase
38	&	ampersand	86	V	uppercase
39	'	single quote	87	W	uppercase
40	(left parenthesis	88	X	uppercase
41)	right parenthesis	89	Y	uppercase
42	*	asterisk	90	Z	uppercase
43	+	plus	91	[left square bracket
44	,	comma	92	\	backslash
45	-	minus	93]	right square bracket
46	.	period	94	^	caret / circumflex
47	/	slash	95	_	underscore
48	0	zero	96	`	grave / accent
49	1	one	97	a	lowercase
50	2	two	98	b	lowercase
51	3	three	99	c	lowercase
52	4	four	100	d	lowercase

53	5	five	101	e	lowercase
54	6	six	102	f	lowercase
55	7	seven	103	g	lowercase
56	8	eight	104	h	lowercase
57	9	nine	105	i	lowercase
58	:	colon	106	j	lowercase
59	;	semicolon	107	k	lowercase
60	<	less than	108	l	lowercase
61	=	equality sign	109	m	lowercase
62	>	greater than	110	n	lowercase
63	?	question mark	111	o	lowercase
64	@	at sign	112	p	lowercase
65	A	uppercase	113	q	lowercase
66	B	uppercase	114	r	lowercase
67	C	uppercase	115	s	lowercase
68	D	uppercase	116	t	lowercase
69	E	uppercase	117	u	lowercase
70	F	uppercase	118	v	lowercase
71	G	uppercase	119	w	lowercase
72	H	uppercase	120	x	lowercase
73	I	uppercase	121	y	lowercase
74	J	uppercase	122	z	lowercase
75	K	uppercase	123	{	left curly bracket
76	L	uppercase	124	 	vertical bar
77	M	uppercase	125	}	right curly bracket
78	N	uppercase	126	~	tilde
79	O	uppercase	127	DEL	delete

The module initiated by setting the counter to count the characters in the file and initialising them with integer zero. The input file was identified and initialized. Additionally, the tree character map table is used as reference to count the group of unique characters. The characters were to be received in UTF-8 format. After the initialisation was completed, the process began by reading the first row of line inside the input file and each character in the row line was separated into a group of unique characters. If the character that was read was equal to the character in the tree character map table, the counter of that character would increase by one. This process was

repeated by reading a row line in sequence until the end of the file. Since the characters were read in an ASCII printed mode, the ASCII control characters were counted too. The steps of this process started by reading the first row line inside the input file. If the row line contained carriage return or line feed, then the counter would increase by one based on what was found in the row line. Finally, all information regarding the amount of unique characters were sent to the main programme. The flow processes of counting the characters as shown in Figure 4.3

4.1.2.1.2 Compression Module

This module compresses the text file by using compression tools. Compression utility was used to reduce the capacity of text file. The text file was converted to a binary file by a compression utility. The GZip compression tool was chosen based on an experiment to obtain the best compression tool (Abas et al., 2017). The experiment was divided into two phases. In the first phase, several compression utilities were used as an experiment to compress the text file. The compress utilities are Lempel–Ziv–Welch (LZW) (Victor, 2012), Zip (Kattan & Poli, 2008), GZip (Husain, Bakhtiari, & Zainal, 2014), Huffman Coding (Shahbahrami, Bahrampour, Rostami, & Mostafa Ayoubi, 2011; M. Sharma, 2010; S. Sharma & Sejwar, 2016), Huffman Coding merged with GZip (Oswal, Singh, & Kumari, 2016) , and Huffman Coding merged with Zip (Pathak et al., 2011). The input data was taken from ASCII printed mode characters and it was repeated 20 times to attain the minimal value compressed. After the compression was completed, all the compressed items were kept in the QR code with error correction level H. The compressed data inside the QR code is binary file

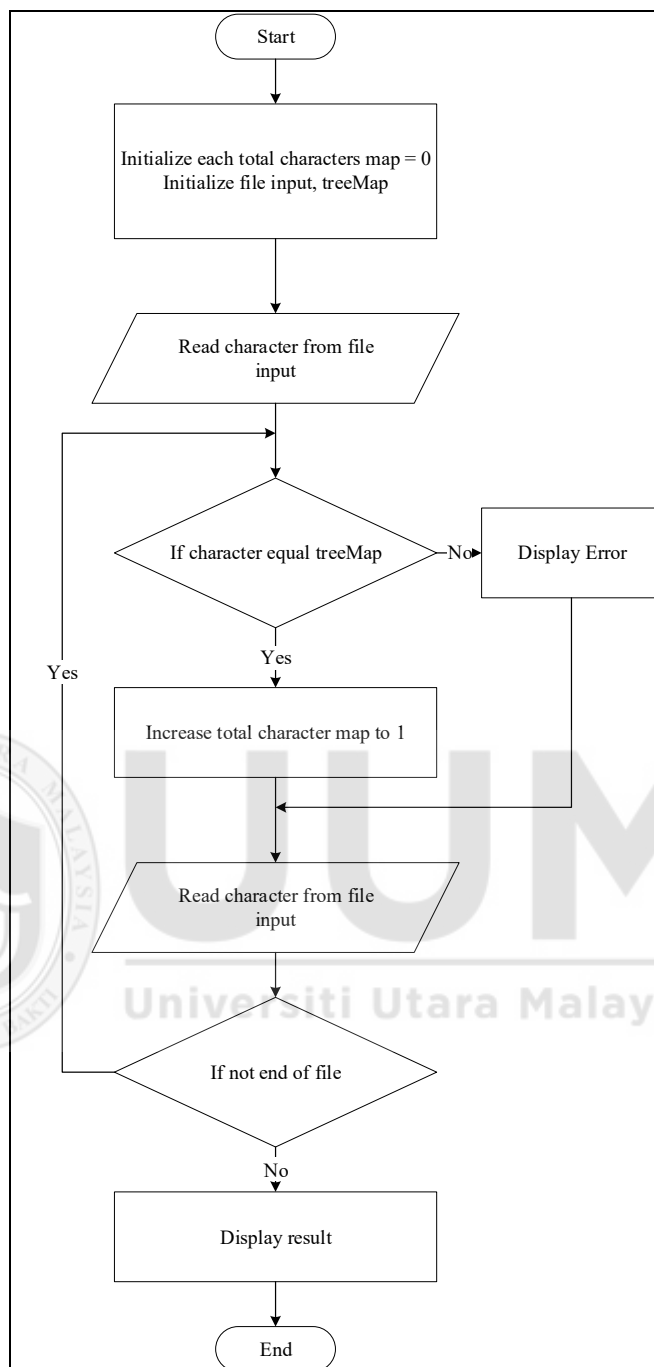


Figure 4.3. The flow chart of character counting module.

type. The results from this phase are shown in Table 4.3 and Appendix A. Appendix A shows the result of 20 times repeated experiment and Table 4.3 shows the minimum characters' total amount value from the 20 times repeated experiment with error correction level H.

The equation can be formulated in 4.1 as shown below:

$$A = \{a_i\}_{i=1}^N \quad (4.1)$$

A = set of elements
a = element
i = index
N = total of elements

which is denoted *min A* or *min_i a_i*, and is equal to the first element of a sorted character (i.e. in order) from a set of *A*.

Table 4.3

The minimum character's total amount value from 20 times repeated experiment with error correction level H (Abas et al., 2017).

Normal	Zip	GZip	LZW	Huffmann Coding	Huffman and GZip
1,271	469	632	433	109	466

From the data in Table 4.3, the normal method without compression is the best way to store data in a QR code. It can store up to 1,271 characters as compared to the compressed file. The reason why this result cannot provide more characters to be stored in the QR code is because the QR code has more capability to store data in alphanumeric as compared to binary. Table 4.4 shows the amount of characters that can be stored in black and white QR code version 40 by character type.

The pseudocode to compress text data using compression utilities as mentioned in Section 4.3.1.2.2 was designed as a guideline to execute the actual action. Each data

output result produced from each compression utility was used as an input in the second phase task.

Table 4.4

The amount of characters that can be stored in black and white QR code version 40 by character type (Courtesy: Wikipedia (2007)).

Input mode	Max. characters	Possible characters, default encoding
Numeric only	7,089	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Alphanumeric	4,296	0–9, A–Z (upper-case only), space, \$, %, *, +, -, ., /, :
Binary/byte	2,953	ISO 8859-1
Kanji/kana	1,817	Shift JIS X 0208

In the second phase, the compressed files were imposed with the Base64 encoder/decoder. The reason is to obtain more characters to be compressed so that the QR code can store more data. As mention by Guwalani, Kala, Chandrashekar, Shinde, and Mane (2014) in their research, the Base64 encoding schemes can be used to encode binary data to textual data that needs be stored and transferred over media. There are various encoders that can be used in this experiment, but the Base64 encoder/decoder was chosen because the time for encoding and decoding is faster than other decoders (He et al., 2010).

The fixed character composition was embedded as a test data. The results were separated by error correction level as shown in Table 4.5. Each experiment was only

performed once as it used fixed composition characters in the input file, where the compression algorithm generated the same size files. From the results in Table 4.5, it is noted that the highest total character is obtained by GZip compression. For example, the QR code can hold up to 1,784 characters at the H level in version 40. The GZip compression can exceed more than 40% of data compression as compared to the normal text.

Table 4.5

The maximum total characters stored in the QR code by error level (Abas et al., 2017).

Error Level	Normal	Zip	Gzip	LZW	Huffman Coding	Huffman And Gzip	Huffman And Zip
H	1270	1560	1784	1167	212	1364	1166
Q	1662	2114	2405	1627	282	1827	1639
M	2330	3188	3470	2441	392	2607	2425
L	2952	4226	4480	3253	503	3323	3095

4.1.2.2 Encoding Multiplexing Modules

In this phase, the experiment starts with dividing the characters into appropriate files according to the amount of characters specified before generating the black and white QR codes. This process end with the development of red, green, and blue QR codes. The sequence of process includes converting ANSI to UTF-8, creating blanks files, dividing characters into blanks files, creating black and white QR codes, and creating the red, green, and blue QR codes.

4.1.2.2.1 Converting ANSI to UTF-8 Module

The conversion needed to be done because the Base64 encoder would produce the ANSI format during the conversion of binary to text. As usual, when creating the black and white QR Code, the text data needed to be in UTF-8 format. The reasons of using the UTF-8 format are as follows:

1. Most scanner devices use the JIS8 (QR 2000) format to scan the data that is compatible with UTF-8 format. The JIS is Japanese Industrial Standards for encoding the Japanese language and it provides code set conversion support for UTF-8 (Stinner, 2017).
2. The *ZXing library* that was used to create the QR code is only able to receive the UTF-8 format. This is a constraint for this experiment and the only solution is to convert all outputs from the encoder received into the UTF-8 format.

The conversion of ANSI to UTF-8 used a class provided by Java library called *Charset* with method *encode*.

4.1.2.2.2 Creating Blank Files module

After the encoding process was completed, the next step was to create blank files. This process is categorised as one of the preprocesses before the actual multiplexing process execution. The blank files had to be created because the multiplexing process had to be prepared with a certain total of blank files before the input data could be embedded inside the related files. In this experiment, N value blank files were created and divided into three groups that represented eight files each.

The reason why the file must be created in eight files each will be mentioned in the next paragraph. Three things were considered when initialising the blank files, which were the total maximum characters, total files used, and file location to be written. Basically, the allocation of maximum characters were based on the previous result of the second column of Table 4.5. The error correction level L consumed 2,952 maximum characters and it was used in the experiment for multiplexing. Meanwhile, the total files used in this experiment were N value blank files and they were located at the current directory of the programme execution. During the process of creating blank files, there was an instruction to add the 'space' symbols to all the blank files. In other words, there were 2,952 'space' symbols embedded in each file. The reason why this action was taken is to ensure that all QR codes become version 40 after the process of creating black and white QR codes was completed even if the information was not fulfilled for complete N value of black and white QR codes. Essentially, the version of QR code from this experiment will be created based on the total of characters embedded.

4.1.2.2.3 Dividing Characters Module

For the next part, the characters from input files were separated with certain total characters allocated. In this case, 2,952 characters were allocated for each file. Table 4.6 briefly shows the characters' file allocation.

Several conditions were made to avoid logic error during execution. The conditions include if the total character embedded exceeded more than N value files, the programme would automatically fail to proceed. An error message would be displayed. Moreover, if the characters embedded did not exceed (less) than N value files, then the

programme would continue to execute but it would keep the ‘space’ symbols balanced inside the file that was embedded previously. All ‘space’ symbols were replaced when inserting the encoded characters.

Table 4.6

The characters’ file allocation.

	A File	Eight Files	Twenty Four Files	N Files
Characters	2,952	8 x 2,952 23,616	24 x 2,952 70,848	$N \times 2,952$ $2,952N$

Two calculations were made before the process began to execute. They were used to identify the exact total files to be used. The first calculation is:

$$\text{Total Files} = \text{Total Characters} / \text{Maximum Characters in Each File}$$

If the value had a remainder, it would take only the integer value. Afterwards, the next calculation was to obtain the mod value, which is:

$$\text{Total Mod Value} = \text{Total Characters} \% \text{Maximum Characters in Each File}$$

When these calculations were completed, it would increase the total file by one, if the total mod value had a remainder. The remainder was the balance characters for a last file. If the characters were less than 2,952 characters, the remainder of ‘space’ symbols in the last file would stay permanently together with the balance characters, as to complete the 2,952 total characters in a single file.

When the files were ready to receive the characters, all the characters were separated into 2,952 characters each and put into a file until the characters were moved to all related files completely. If the characters did not reach the sufficient amount of 2,952 characters at the end of the process, the balance would be replaced with 'space' symbols permanently and stored in the last file. At the end, the process sent the total files completed for the next process.

4.1.2.2.4 Creating Black and White QR Code Module

In this part, it would concentrate on creating the N value of black and white QR codes. It used the file that contained the separated characters as the input data. The first step was to initialize the text input and graphic output file. The graphic output files were assigned names because it would ease the next session of execution. After the initialisation was completed, the process extracted the data from input files by reading it and putting it into the data array at the module of creating the black and white QR codes. The process recalled (object-oriented) the same module as a part to create the black and white QR code because the amount of input files was more than one.

Some criteria had to be applied before creating the black and white QR code. These criteria are initialising the height, width, colour, character set, error correction level, and margin. After these parameters were completed, the black and white QR code were created by using *ZXing* library that contained *QRCodeWriter* class. This class produced a byte matrix that contained the graphic pixels of QR code. This byte matrix was used to generate the image of QR code with execution of buffer image and graphic image. The design of black and white QR code was generated by pointing the location of x-axis and y-axis pixel by pixel.

4.1.2.2.5 Creating Red, Green, and Blue QR Code Module

This method is the last method for the multiplexing process. The process task was divided into three sub-processes. The sub-processes included reading the image and converting it to 8-bit structure, gathering information, and creating the red, green, and blue QR codes. In the first step, all information regarding the colour pixel from the black and white QR code created previously was captured.

The images were read after the initialisation of the image file names, total image files, and size of the images (height and width) had been identified. In this experiment, the N value of black and white QR codes were used to create red, green, and blue QR codes. The array of buffered image was assigned and information regarding the pixel colours of the N value of black and white QR codes were kept. In other words, the array of buffered image kept the information of pixel colours based on the total black and white QR codes. The array of buffered image kept the information of the colour type using the RGB scheme at the point of x-axis and y-axis of the black and white QR code. Only two types of colour in the RGB scheme were used, which were black and white.

After this information was collected, the next process is to convert the colour into a digit. In this case, the black colour was assigned as 1 and the white colour was 0. The code in the RGB scheme for black was RGB(000,000,000) and white was RGB(255,255,255). For example, eight black and white QR codes were chosen and combined via x-axis and y-axis index location in order to obtain a red QR code. The x-axis and y-axis indices from eight black and white QR codes at location (0,0) were merged as 8 bits of binary number because there were only 0 and 1. Afterwards, the

binary numbers were converted into a decimal number, which was between the range of 0 and 254. These numbers were located at the first column of the RGB scheme. The second and third columns were represented with 0 value. This information was represented at pixel location $(0,0)$. For another location, it will follow the same method until the end of the pixel location, which was pixel location $(width - 1, height - 1)$. For the blue and green QR codes, the processes were still the same as generating the red QR code. In this case, for the multiplexing, it generated the three QR codes, namely red, green, and blue QR codes. Figure 4.4 shows an example of the first process in converting binary to decimal point number in the index location $(0,0)$ for each black and white QR code and assigning the value to the index location $(0,0)$ at the red QR code. The creation of red, green, and blue QR codes came from the array of graphic images that contained the colour of each pixel. The multiplexing process used eight images and produced a single image by combining the binary numbers into 8 bits.

4.1.2.3 Encoding Multilayer Modules

After the multiplexing process was completed, the multilayer process would take over to combine the red, green, and blue QR codes. This process read the whole colour code from each index location starting from $(0,0)$ until $(width - 1, height - 1)$ of the images from red, green, and blue QR codes. The same index location of red, green, and blue QR codes was blended in the RGB colour scheme as *Colour* (x, y, z) , which indicated x as the colour code of the red QR code, y for the green QR code, and z represented the blue QR code. From this experiment, the multilayered QR codes consisted of 93,987 processes, which were calculated as $(177 \text{ (size of the pixel } x\text{-axis)} \times 177 \text{ (size of the pixel } y\text{-axis)}) * 3 \text{ (red, green, and blue QR codes)}$.

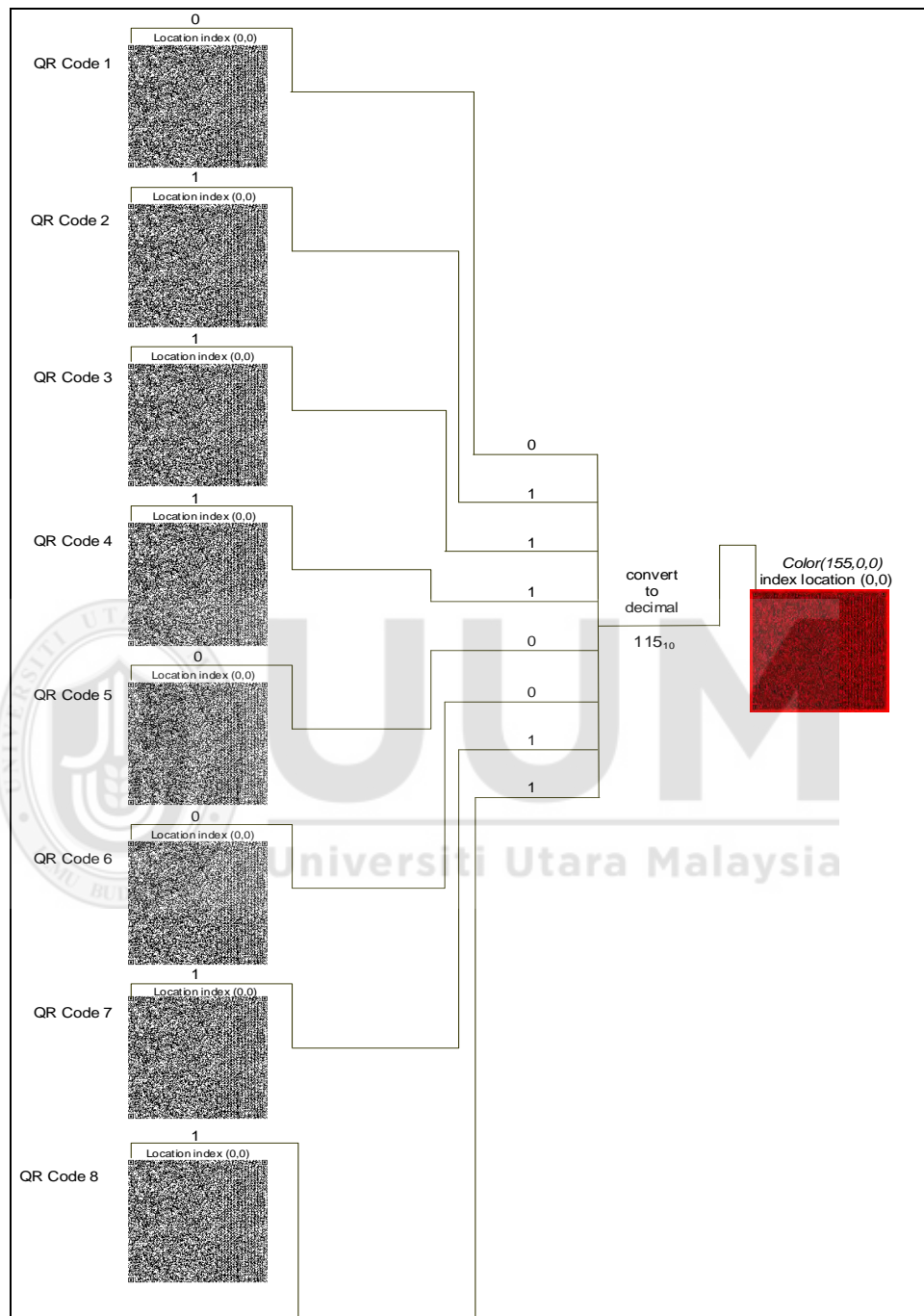


Figure 4.4. The example of the first process in converting binary to decimal point number in the index location (0,0) for each black and white QR codes and assigning the value to the index location (0,0) at the red QR code.

In the beginning, the total image files, image file names (red, green, and blue QR codes), image location, and final multilayered image file were initialized. However, the total image files, image file names (red, green, and blue QR codes), and image location were set as parameters for the multilayer algorithm. The buffer image for the final multilayered image file initialized the size of width and height, plus the colour type. All the pixel information from red, green, and blue QR codes were merged (as mentioned in the previous paragraph) and assigned to the graphic image. After the processes were finished, the graphic image assigned the colour value to the graphic file so as to create the image.

4.2 Decode Algorithmn

This chapter describes the process of decoding a coloured QR code. All modules will be explained in pseudocode form and all modules will be converted into algorithms. All algorithms are shown in Appendix C. The experiment of the undertaken experiment is based on time consumed, total of QR codes extracted, and error correction level. The results are based on the experiment conducted according to the methods used in this research. The pseudocodes are developed as a guideline before developing a real programme. This chapter also provides some result of simulation if the colour channel and colour depth are implemented. The following sections describe the findings of the decoding experiment in detail.

4.2.1 Decoding QR Code

The decoding process is to obtain the actual data that was encoded before. Basically, the coloured QR code is a final physical result of the encoding process. It contains

some information embedded inside it. The information is represented by the coloured QR code. When a coloured QR code is successfully encoded, it needs to be decoded in order to retrieve the data stored in it. This is because when the users received a coloured QR code, they need to know what is the information inside. The users cannot translate the image of coloured QR code in physical shape, due of that, it needs to be translated by using the decoding method.

In general, the decoding starts with the demultilayer process from the coloured QR code to red, green, and blue QR codes. After that, the demultiplexing process will take over to convert the red, green, and blue QR codes into 24 black and white QR codes. Lastly, the decompression process will convert the content inside the black and white QR codes into the original text.

4.2.2 Decoding Steps

The decoding process is the reverse process of the encoding method. There are several steps to be treated when maintaining the decoding processes, which are:

1. **Demultilayer.** This step is the first step for the decoding process. Generally, the coloured QR code will be converted to red, green, and blue QR codes. This process breaks the RGB colour code into separate unique red, green, and blue colour codes.
2. **Demultiplexing.** After the demultilayer process is completed, the demultiplexing process will be prepared by converting the red, green, and blue QR codes into black and white QR codes. The total of black and white QR codes is 24.

3. **Decompression.** The last process is to extract the contents inside the black and white QR codes. Firstly, it needs to decode the black and white QR codes. As a result, the GZip binary file is produced and ready to be extracted.

The decoding flow is presented in Figure 4.5.

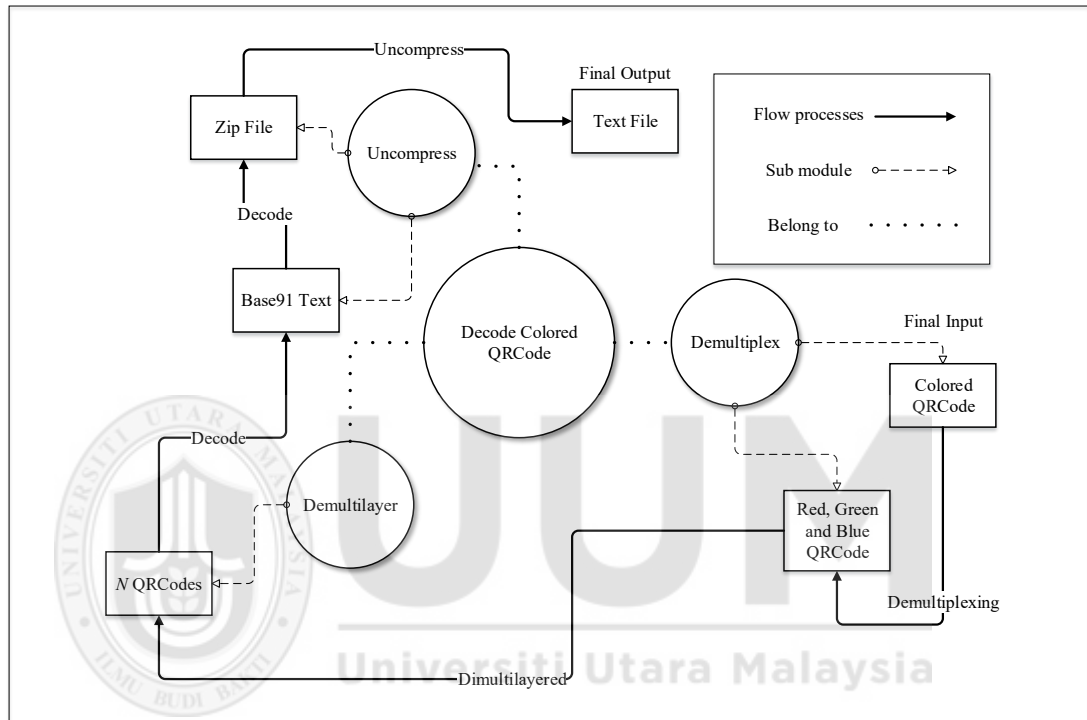


Figure 4.5. The decoding flow process.

The main algorithm was designed with some modification on the encoding process. Even though the decoding process is an inverse of the encoding process, some processes need to be reengineered. The main algorithm contains details of demultilayer, demultiplexing, and decompression. Figure 4.6 shows the main pseudocode of decoding a coloured QR code.

```

1 : Initialize possible initialisation for file type, file input, file output, and file
   : location. Initialize resultColouredQR Code Decode as two-dimensional
   : array of colour (Module Index D1).
2 : Demultilayer process (Module Index D2)
3 : groupQRCode = {"red", "green", "blue"}
4 : groupFiles = {M1, M2, M3... Mn}
5 : for each file in groupFile
6 :     Demultiplexing processes (Module Index D3)
7 : end for
8 : if total files less than N value from group files
9 :     Convert black and white QR code into textfile (Module Index D4)
10 : end if
11 : Text reconstruction (Module Index D5)
12 : Decompress (Module Index D6)
13 : /* Mn is a group of files */

```

Figure 4.6. The pseudocode of main decoding programme.

There are several modules in this decoding pseudocode and these are indexed by numbers (See Appendix C). The index number identification for the decoding module is shown in Table 4.7.

4.2.2.1 Initialisation Module

This is the first step that needs to be fulfilled before completing the whole processes. The decoding demultilayer pseudocode should be initialized first before it could proceed to bring the red, green, and blue QR codes as its output. The values consisted

of file type, file input, file output, and file location of coloured QR code information. The information above was used to break out the coloured QR code into three monocoloured QR codes. Then, it obtained the pixel colour information from the red, green, and blue QR codes as an output. All input information above were to be used in the next demultiplexing module.

Table 4.7

The index number identification for decoding module.

Index	Descriptions
D1	The coloured QR code is used as input file. Initialize all related file type, file input, file output, and file location of coloured QR code information.
D2	The demultilayer process is used to produce the red, green, and blue QR codes. The generation of it is based on breaking the RGB colour into single red, green, and blue colour codes.
D3	The demultiplexing process is used to break the red, green, and blue colour codes into groups of M_n black and white QR codes.
D4	The decoding process handles the conversion of N value of black and white QR codes into Base64 text file.
D5	The decoding process needs to handle the Base64 text reconstruction. The Base64 text will be changed into GZip binary file.
D6	The decompression process normally extracts the compress file into the original text file as used during the encoding process.

Note: N value is the total of black and white QR codes

4.2.2.2 Decoding Demultilayer Module

In detail, the coloured QR code was generated by the red, green, and blue QR code file image combination. The file images were created based on the unique colours for red, green, and blue. The RGB colours in each pixel location for red, green, and blue QR codes are as follows:

RGB (x, 0, 0) – Red QR Code

RGB (0, x, 0) – Green QR Code

RGB (0, 0, x) – Blue QR Code

which is

x between 0 until 255.

For example, if the pixel location at (0,0) contains RGB(233,6,34), then the division of RGB from red, green, and blue QR codes are:

RGB (233, 0, 0) – Red QR Code

RGB (0, 6, 0) – Green QR Code

RGB (0, 0, 34) – Blue QR Code

Each of the images were transferred into the colour that looked like red, green, and blue colour code images on each of it.

The tasks started by assigning the contents of pixel colour information inside the coloured QR code. From this information, the output image files were created first for ease of allocating the related pixel colours to the related red, green, and blue QR codes. All contents of pixel colour information inside the coloured QR were converted to the image buffer before transferred to the related red, green, and blue QR codes. The way

of transferring the pixel colour information was by separating each element of the RGB colour code scheme to red, green, and blue elements as discussed previously. The total of colour pixels was based on the size of the coloured QR code image. As usual, the image size of this coloured QR Code was 177 x 177 pixels, which as 31,329 pixels.

An experiment of elapsed time order by error correction level during decoding demultilayer was performed successfully and the result is shown in Table 4.8. From the results, the elapsed times were considered not much different in time range among the error correction levels. Error correction level M completed its task in 0s 899ms.

Table 4.8

The experiment of elapsed time order by error correction level.

Error Correction Level	Demultilayer
L	0s 925ms
M	0s 899ms
Q	0s 902ms
H	0s 900ms

4.2.2.3 Decoding Demultiplexing Module

The next process is demultiplexing, which consisted of demultiplexing from red, green, and blue QR codes into N values of black and white QR codes. This process extracted each pixel value, which was in decimal value, of each red, green, and blue QR codes of RGB colour into binary numbers of 8 bits. After the binary numbers were determined, they were broken up into eight separate units of 8-bit binary numbers. Each separate digit of binary numbers served as white (0) or black (1) pixel colours of

each black and white QR code. For example, say that the pixel location (0,0) of red QR code has RGB(45,0,0). Then, the 8-bit binary number is 00101101. The first digit of the binary is 0, which represents the white colour of the first black and white QR code at pixel location (0,0). It is also the same with the second and eighth black and white QR Codes. This method shows the one to many concepts that were implemented in the demultiplexing process.

The flow process started by identifying the red, green, and blue QR code image information and creating the group of M_n black and white QR codes of each red, green, and blue QR codes. The demultiplexing process started with extracting the decimal value of the red element of RGB red QR colour code for each pixel location starting from location (0,0) until (177,177). The last pixel location was determined by the size of width and height of image. The next process continued with green and blue QR codes by using conditional statement as they have to be executed one by one. The decimal values only took place to the related red, green, and blue colour elements. For example, if the RGB for red QR code is RGB(46,0,0), it will assign the decimal value of 46 to the temporary storage location. All colour information of red QR code was collected and kept into the temporary memory storage location. The temporary storage collected all information of the green and blue QR codes in decimal value. When all decimal values were collected, the next process was to convert the decimal value into binary number. The decimal values were assigned as integer values and converted into a string mode so as to easily manipulate the characters received. The calculation of conversion was identified by a normal mathematical calculation. The decimal value was divided by two and then the remainder was written down. This process was

repeated until it could not be divided by two anymore. For example, take the decimal value of 157 as shown in Table 4.9.

Table 4.9

Decimal to binary process.

Divide value (÷)	Modulus value (%)
$157 \div 2 = 78$	with a remainder of 1
$78 \div 2 = 39$	with a remainder of 0
$39 \div 2 = 19$	with a remainder of 1
$19 \div 2 = 9$	with a remainder of 1
$9 \div 2 = 4$	with a remainder of 1
$4 \div 2 = 2$	with a remainder of 0
$2 \div 2 = 1$	with a remainder of 0
$1 \div 2 = 0$	with a remainder of 1

Next, the value of the remainders from bottom to top represented the value of binary number, which was 10011101.

After the binary numbers were collected from each red, green, and blue QR code, the next step was breaking them into a single bit and marking the index location of each bit. When the single bit was identified, the colour of pixels were labelled either black or white. The colour pixel was recognised when the single bit gave a value of 0 or 1, whereby 0 was for white and 1 was for black. Three items were considered during this process in determining the colour, which were colour type of QR code, pixel location, and index of breaking 8 bits of binary number to single bit 0 (white) or 1 (black) digit combination. The processes of combination are as shown below:

Type of QR code $x = red; y = green; z = blue$

Pixel location = $a(0,0) \sim a(177,177)$

Break out index 8 bits location into single bit = $p(0) \sim p(7)$

The determination of black or white colour can be shown as a set of {type of QR Code} {pixel location} {single bit}, which is in detail referred to as $\{x_i\} \{a_{ij}\} \{p_i\}$. Figure 4.7 below shows the flow chart in determining the black or white pixels of black and white QR codes.

4.2.2.4 Decoding Black and White QR Codes

When the 24 black and white QR codes were fully generated, the next step was to decode them into the text file. The decoding process consisted of decoding each black and white QR code, merging the decoded texts, and saving to the logical file. The decoding process of QR code began by calling the first black and white QR code from the red QR code and decoding it into text. The text contained in each black and white QR code was a part of a full text during the previous encoding process. This process continued until the last black and white QR code of the green QR code. The suitable decoding module of black and white QR code is *MultiFormatReader*.

After the decoding process to text was completed, the next step was to merge the texts. The texts were merged by using a string class, which could reserve about 2,147,483,647 ($2^{31} - 1$) characters overall. Each text information stored in each black QR code was assigned by using operator '+'. Lastly, all text information stored in the buffer string class were stored in the file directory by using *File* class. The text file in the file directory contained Base64 text format, which has to be decoded by its decoder.

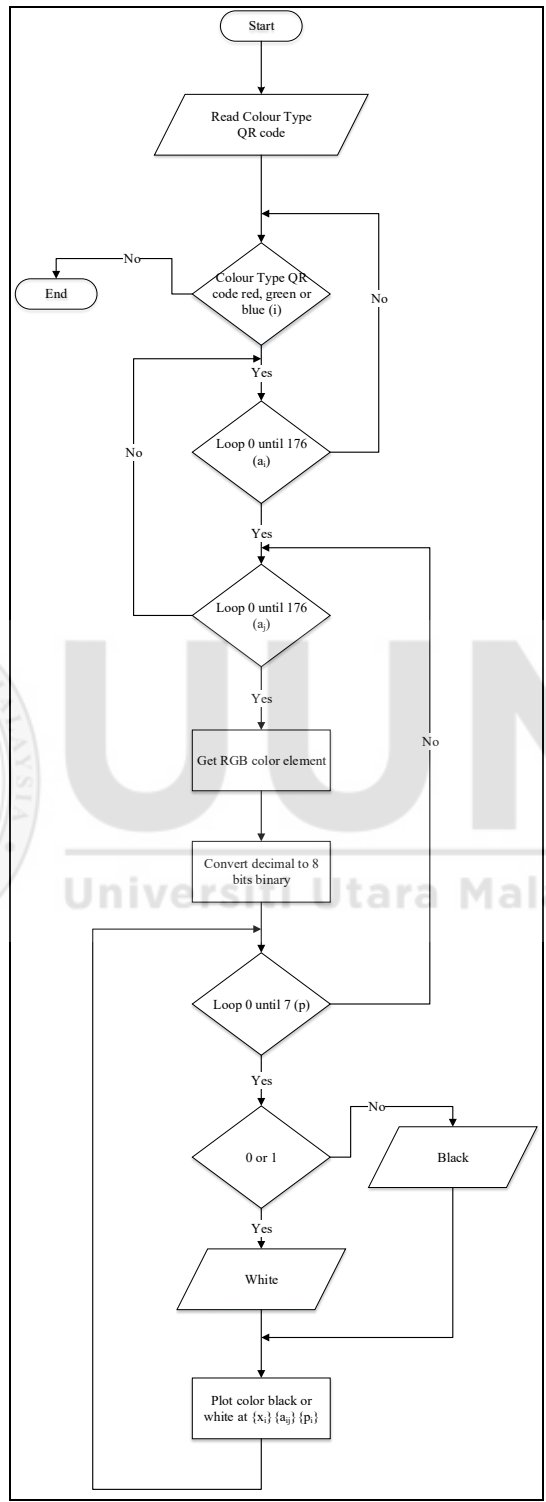


Figure 4.7. The flow chart process of determining black or white pixels of black and white QR codes.

An experiment of elapsed time order of error correction level as shown in Table 4.10 was tested and it concentrated on the demultiplexing process only. From the tabulated data, error correction level Q has the minimum total decoding demultiplexing elapsed time, which is 1 second 614 milliseconds. Meanwhile, error correction level L has the maximum decoding demultiplexing elapsed time with 1 second 656 milliseconds. From the result, the time consumed to decode the coloured QR code to Base64 text file was not a big issue because the range time between them was not too long. It took approximately 1.65 seconds to complete the process.

Table 4.10

The elapsed time of decoding demultiplexing process.

Error Correction Level	RGB Demultiplexing	Black and White Demultiplexing	Total Multiplexing
L	0s 402ms	0s 470ms	1s 656ms
M	0s 437ms	0s 465ms	1s 650ms
Q	0s 411ms	0s 450ms	1s 614ms
H	0s 421ms	0s 473ms	1s 655ms

4.2.2.5 Decoding Text Encoder/Decoder for Decompression Module

The Base64 text file format, i.e. the ASCII text decoder file that was generated from the demultiplexing process, will be used as the first process in the decompression method. In this process, the Base64 text needs to be converted to the compress file, which is GZip file. The GZip file is a binary file that can decompress and produce the

original text file. There are two processes involved in this method, which are ASCII text decoding and decompression tool.

The method started with reading the Base64 text file until the end of the file. The reading function should be capable of reading the printed and control characters inside the Base64 text file. After the reading process was completed, the characters were converted to the binary file by using *Base64.decode* method. The binary compress file was completely generated when the decoding of Base64 process successfully converted it.

4.2.2.6 Decoding Compression Tool for Decompression Module

The last step is to decompress the binary compress file into the original text file. The binary compress file in this case refers to the GZip file, which was used as the compression tool. The suitable decompression class used in this research is *GZip* class. Normally, this class was developed by the provider of compression tools. The decompression step started with preparing the output stream file. Then, the data was decompressed inside the binary compress file and put it into the output text file inside the suitable internal directory. Figure 4.8 shows the flow chart of the decompression method.

Table 4.11 shows the elapsed time of the decompression process. The decompression process started from text decoder (Base64) until decompression (GZip). Error correction level H had the capability to complete the decompression process in the fastest time as compared to the other error correction levels, which was 0 second and 8 milliseconds.

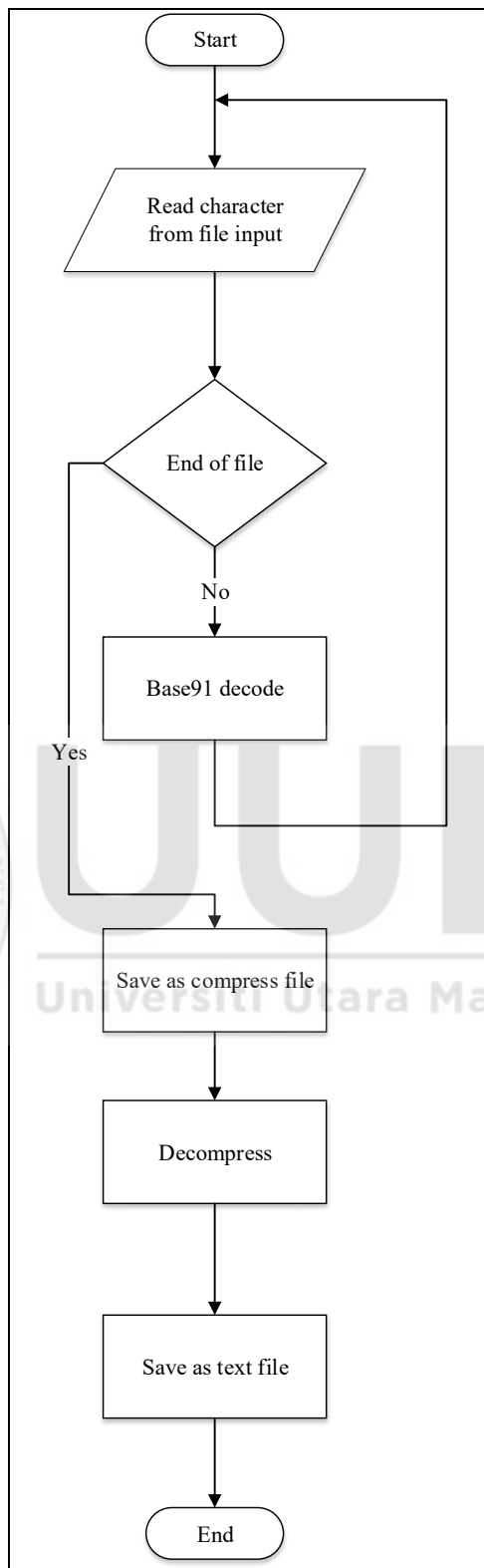


Figure 4.8. The flow chart of decompression method.

From this perspective, it can be concluded that the time consumed to complete this process did not contribute to the file size of QR code for this research.

Table 4.11

The elapsed time of decompression process.

Error Correction Level	Decompression
L	0s 12ms
M	0s 11ms
Q	0s 9ms
H	0s 8ms

4.3 Partial Extraction Algorithm

This chapter describes the technique of decoding and re-encoding the coloured QR code during half way of the process. It is called partial extraction. The need for it is for data manipulation purposes. The manipulation includes update, insertion, and deletion. All modules are written in pseudocode mode and these are arranged in the form of module indices. Then, the algorithms are developed and shown in Appendices D, E, F, and G. From the proposed partial extraction module, it can be proved that the processing time can be reduced. The module is based on the current decoding module but with some modification on the flow processes. The findings are based on how fast the process can be manipulated and the amount of information.

The coloured QR code was developed based on the abstract model as shown in Figure 4.9. The QR codes that were produced in this model are monocolour (red, green, blue), black, and white, then they are divided into two levels when partial extraction is executed. The first level includes the manipulation of black and white QR code and

the second level is monocoloured QR code. Each level contains decoding and re-encoding processes and each of them will be involved in both levels.

4.3.1 Level 1 Decoding Module

This process started by identifying which black QR code index would be manipulated. Moreover, the value bits of colour depth and the total of colour channels from the colour model were to be described. When the index location of black and white QR code, colour depth, and colour channels were classified, the next process was to decode the coloured QR code by using the demultilayer process. For example, if the index of black and white QR code is 24, the colour depth is 8 bits and uses the RGB colour channel, the next process must be referred to the blue monocoloured QR code after the demultilayer process. The formula to identify which index location of monocoloured QR code will be referred from the index of black and white QR code is shown below:

Index location of monocolour = index location of black and white QR code / total colour depth

which is by avoiding the remainder and take only the integer value. From the example discussed previously, the index location of monocoloured QR code is $23 / 8 = 2.875$, in which the index location is 2.

When the monocoloured QR code has been created, the next process was to identify the index location of monocoloured QR code that has to be extracted. The calculated index location of monocoloured QR code was used as a guideline to extract the eight

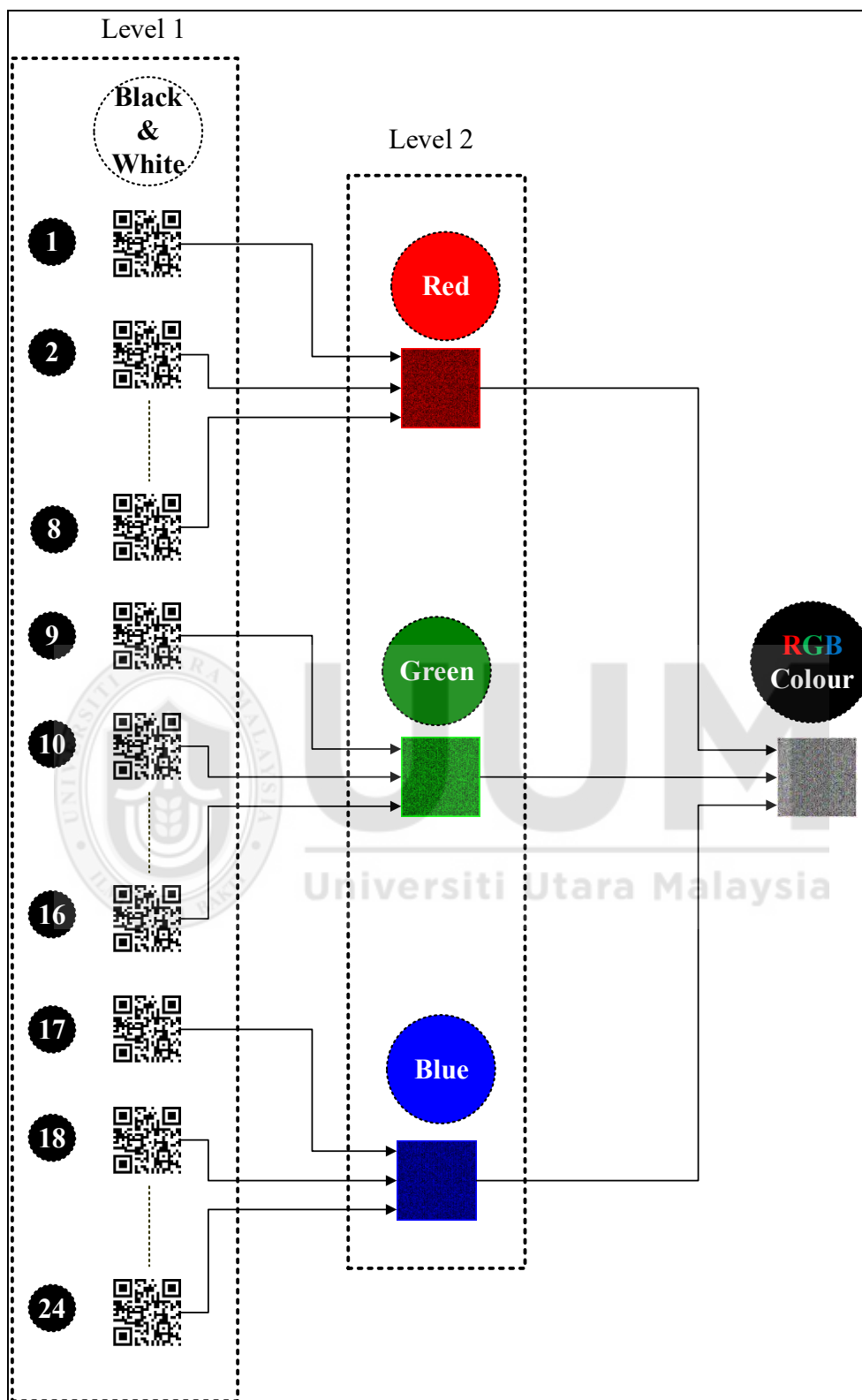


Figure 4.9. The abstract model of 8-bit colour depth and 3-channel RGB colour model.

black and white QR codes. After the index location of the monocoloured QR code was identified, the next process extracted the monocoloured QR code into eight black and white QR codes. The index location of black and white QR code that was described or inputted were to be used to identify which black and white QR code has to be extracted into characters. The comparison process was used in order to find the selected index location of black and white QR code. This part is known as the demultiplexing process.

When a black and white QR code was selected, the next process as to extract it into character text by using the QR code decoder. The characters from the black and white QR code that was produced were in a non-readable text format that has to be decoded. In the decompression process phase, the characters were decoded into a binary file, which was a compression type format. Since the decompression process had two processes, the next step was to decompress them into readable characters, which were the original characters. The decompression process used decompression tools. The text produced from the decompression tools could be manipulated such as add, update, or delete.

The details of the decoding pseudocode for level 1 implementation started with requesting the index location of black and white QR code as shown in Figure 4.10, while Table 4.12 shows the index identification task from Figure 4.10. The pseudocode in this section was converted into the algorithm as shown in Appendix D.

<p>1 <i>Input possible initialisation of index location of black and white QR code (colour depth and colour channel if applicable) (Module Index P1)</i></p>
--

```

    Calculate the possible initialisation input of black and white QR code index
2   location (colour depth and colour channel if applicable) and location of
    monocolour from index of black and white QR code
3   Produce solution (Module Index P2)
4   if (image of monocoloured QR code based on index location equals to
    calculated index location)
5       Perform solution (Module Index P3)
6   end if
7   Initialize the image of black and white QR code based on index location
    if (image of black and white QR code equals to index location initialized by
    input
8       Perform solution (Module Index P4)
9   end if
10  Perform solution (Module Index P5)
11  Perform solution (Module Index P6)
12  Perform solution (Module Index P7)

```

Figure 4.10. The pseudocode of partial execution for decoding level 1.

Table 4.12

List of tasks for partial execution decoding level 1 module.

Index	Description
P1	Initialize the related variable value before executing the whole process such as file directory, file name, file type, and temporary storage variable (array).
P2	The demultilayer process is used to produce the red, green, and blue QR codes. The generation of it is based on breaking the RGB colour into single red, green, and blue colour codes.

- P3 The demultiplexing process is used to break the monocoloured QR code based on the index location initialized into eight black and white QR codes.
- P4 The selected index location of black and white QR codes will be decoded into printed and controlled characters.
- P5 The character file is decoded by using a text to binary decoder (Base64). The binary compression file type is developed.
- P6 The uncompressed process extracts the compress file into parts of the original text file as selected from the index location in black and white QR codes.
- P7 The text data is manipulated (add, update, delete)

4.3.2 Level 1 Re-Encoding Module

When all the texts have been manipulated, the next process was to re-encode them to the coloured QR code. The re-encoding process started by compressing the new manipulated characters. It only involved the identified index location of black and white QR codes and the remaining black and white QR codes stayed as images in the computer directory file. The remaining codes would be used during the multiplexing process. As usual, the new manipulated characters were compressed by compression tools (GZip) and then from the binary compression file that was brought up, they were converted from binary to text by using a decoder (Base64). The mode of the new text was changed from ANSI to UTF-8. Then, the text from the decoder was encoded into black and white QR codes.

The remaining and new black and white QR codes were combined again in the multiplexing process and used as an input to develop monocoloured QR Codes. After the monocoloured QR codes were created, the next step was to create a coloured QR code with the multilayer process. The re-encoding methods of multiplexing and multilayer were the same methods used in the previous chapter.

The details of the re-encoding pseudocode for level 1 implementation started with the compression of the new manipulated text. It was used as the first input and was stored in a text file at a specific location. Figure 4.11 and Table 4.13 show the method of re-encoding pseudocode based on the identified index location of black and white QR codes. Appendix E shows the detailed algorithm of level 1 re-encoding processes after conversion from the pseudocode.

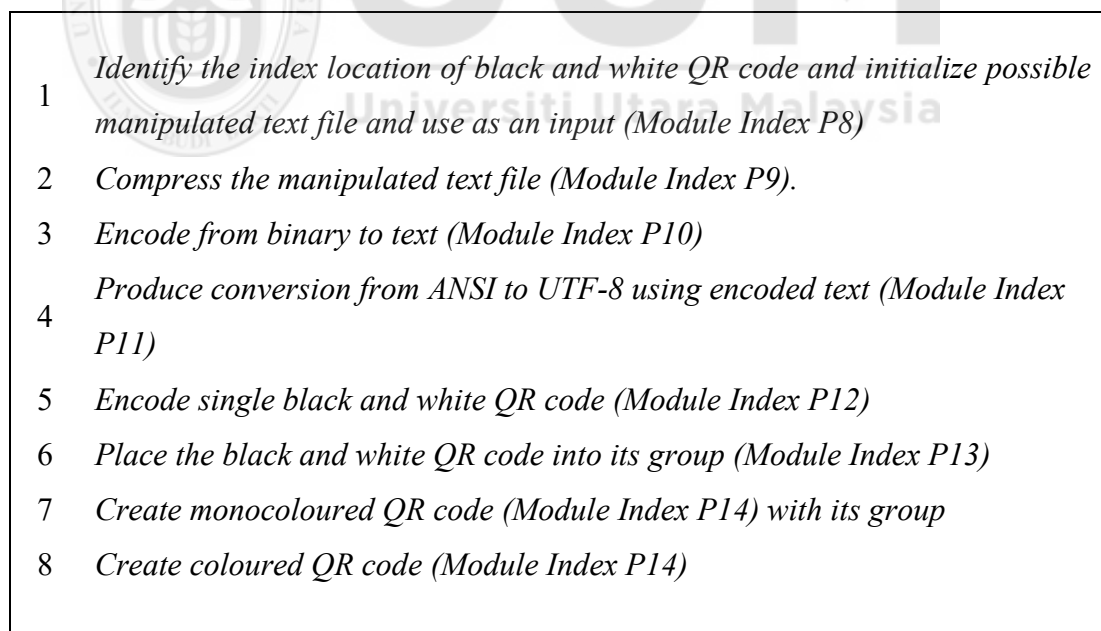
- 
- 1 *Identify the index location of black and white QR code and initialize possible manipulated text file and use as an input (Module Index P8)*
 - 2 *Compress the manipulated text file (Module Index P9).*
 - 3 *Encode from binary to text (Module Index P10)*
 - 4 *Produce conversion from ANSI to UTF-8 using encoded text (Module Index P11)*
 - 5 *Encode single black and white QR code (Module Index P12)*
 - 6 *Place the black and white QR code into its group (Module Index P13)*
 - 7 *Create monocoloured QR code (Module Index P14) with its group*
 - 8 *Create coloured QR code (Module Index P14)*

Figure 4.11. The pseudocode of partial execution for re-encoding level 1.

4.3.3 Level 2 Decoding Module

The level 2 decoding pseudocode involves only the partial extraction of monocoloured QR code, which needs to be identified first. The levels 1 and 2 processes have a similar flow of task; nonetheless, level 2 extracts monocoloured QR codes, whereas level 1 extracts black and white QR codes. The benefit of the extraction of monocoloured QR codes is that the extraction of data in the black and white QR codes can be made for more than one black and white QR code. It depends on the total colour depth that was used before.

Table 4.13

List of tasks for partial extraction re-encoding level 1 module.

Index	Description
P8	Identify the index location of black and white QR code and initialize possible manipulated text file and use as an input.
P9	The manipulated text file is used as an input and produces the compression task by using compression tools. It will produce the binary compress file.
P10	The binary compress file is converted back to text by using the binary to text encoder (Base64).
P11	Changing the mode of text file from ANSI to UTF-8
P12	The encoded text file is used as an input and the creation and execution tasks of the black and white QR code image use this input.
P13	Place the image of black and white QR code at the index location that has been identified.

- P14 Convert the black and white QR codes into monocoloured QR codes by using new and remaining black and white QR codes as an input.
- P15 Produce coloured QR codes by using new and remaining monocoloured QR codes as an input.
-

The process began with the identification of which index location of monocoloured QR code has to be manipulated. The next step was to initialize the text files' index location based on the monocoloured QR code index location. By using the function, the total of black and white QR code was based on the type of colour depth of a single colour channel used. For example, if the colour depth is 8 bits for a colour channel, then the total of black and white QR code is 8 for a monocoloured QR code. To allocate the index location of black and white QR codes, the formula is as follows:

First index location of black and white QR code = index location of monocoloured QR code x type of colour depth of single colour channel

Then, the index location of black and white QR codes was increased by 1 and this increment was repeated + 1 times based on the size of colour depth of the single colour channel. The process continued with generating (demultilayer) the group of monocoloured QR codes from coloured QR codes. The total of monocoloured QR codes was based on the colour model used. When the monocoloured QR codes were created, the condition statement was made to identify which monocolour would be used to extract them. At that moment, when the monocoloured QR Code was identified, it was then decoded (demultiplex) into black and white QR codes based on colour depth. The image of black and white QR codes was initialized and located into

the specific location directory. All the images of black and white QR codes were decoded into a text file, which contained the decoded characters (binary to text). After the decoding process from black and white QR code to text file was completed, the decoded text files were converted into binary compress files. The decoder tool was used for this process. The binary files that were produced from the previous process were decompressed. This was the last process to gain a part of the original texts or characters. The part of original texts were manipulated with add, update or deletion processes. Overall, the steps of decoding and re-encoding in level 2 were the same process as discussed in the previous chapter, but with a difference in implementation.

The details of the decoding pseudocode for level 2 implementation started with demultiplexing the coloured QR code until a part of the decoded characters were decompressed into the original characters. Figure 4.12 and Table 4.14 show the method of decoding pseudocode based on the identified index location of monocoloured QR code. The details of the algorithm is shown in Appendix F.

```
1  Input possible initialisation of index location of monocoloured QR code  
   (colour depth and colour channel if applicable). Identify the index location of  
   monocoloured QR code and set the index of black and white QR code  
   (Module Index P16).  
2  Execute the demultilayer process (Module Index P17)  
3  if (image of monocoloured QR code equals to the indexed location  
   initialized by input)  
4     Execute the demultiplexing process (Module Index P18)  
5  else  
6     exit system  
7  end if
```


	<i>Initialize the image of black and white QR code based on index location</i>
8	<i>Decode selected group of black and white QR codes (Module Index P19)</i>
11	<i>Decode the text using a text decoder (Module Index P20)</i>
12	<i>Uncompress the binary file (Module Index P21)</i>
13	<i>Manipulate data (add, update, delete) (Module Index P22)</i>

Figure 4.12. The pseudocode of partial execution for decoding level 2.

Table 4.14

List of tasks for partial execution decoding level 2 module.

Index	Description
P16	Input possible initialisation of the index location of monocoloured QR code (colour depth and colour channel if applicable). Identify the index location of monocoloured QR code and set the index of black and white QR code.
P17	The demultilayer process is used to produce the red, green, and blue QR codes. The generation of it is based on breaking the RGB colour into single red, green, and blue colour codes.
P18	The demultiplexing process is used to break the monocoloured QR code based on the index location initialized into eight black and white QR codes.
P19	The group of selected index location of black and white QR codes (based on the previous calculation) will be decoded into printed and controlled characters.
P20	The character files are decoded by using a text to binary decoder (Base64). The binary compression file type is developed.

- P21 The uncompressed process extracts the compress files into parts of the original text file as selected from the index location of black and white QR codes.
- P22 The text data is manipulated (add, update, delete)
-

4.3.4 Level 2 Re-Encoding Module

After a group of characters has been manipulated, the next process is to re-encode them into the coloured QR code. The re-encoding process involves a group of texts from selected monocoloured QR codes. The benefit from this process is that a large amount of characters can be manipulated as compared to the level 1 implementation.

The first step was managing the characters into the updated text file. The size of the text file should be similar to the previous decompression process. The text file was named based on the index location of the first black and white QR code and it was put at the current specific directory, which was used in the programme to get these files.

The next process was to compress a group of updated text files by using a compression tool. The updated text file that has to be compressed depended on which characters were updated. If the text file was not updated, the compression and binary to text decoding processes were not compulsory or necessary to be performed. It could be used with the old version of black and white QR code. Assume that a group of text file is updated, the compression tool will compress the text file. The process continued with encoding the compress file type into encoded characters by using an encoder tool. As a result, a bunch of characters were produced and were ready to be divided and put into suitable black and white QR code version 40 maximum character capacity as well

as error correction level. The characters were converted from ANSI to UTF8 format due to the restriction of embedding characters into the black and white QR codes. The encoded characters in each file contained the maximum total characters of black and white QR code version 40 and were represented in the form of images of black and white QR codes. The total amount of files that was divided had to be similar to the total amount of black and white QR code images that were decoded previously. If not, the system would not be able to proceed for the next step. When all the criteria were fulfilled, the encoded character files were encoded into images of black and white QR code. The name of the images of black and white QR codes were based on which monocoloured QR code was used and they were located in the same location directory of the original black and white QR code. The next step was the multiplexing process, which converted a group of updated black and white QR codes into a single updated monocoloured QR code. The type of monocolour model was identified based on the decoding process that was done previously. The old and current monocoloured QR codes were combined to produce coloured QR code via the multilayer process. This was the last process of level 2 re-encoding pseudocode.

The details of the re-encoding pseudocode for level 2 implementation started with gathering updated information in a text file until producing the updated coloured QR code. Figure 4.13 and Table 4.15 show the method of re-encoding pseudocode based on the identified index location of monocoloured QR code. Appendix G shows the detailed algorithm for the re-encoding pseudocode of level 2.

	<i>Identify the index location of black and white QR code and monocoloured</i>
1	<i>QR code and initialize possible manipulated text file and use as an input (Module Index P23)</i>
2	<i>Compress the manipulated text file (Module Index P24).</i>
3	<i>Encode from binary to text (Module Index P25)</i>
4	<i>Produce conversion from ANSI to UTF-8 using encoded text (Module Index P26)</i>
5	<i>Place and divide text characters into suitable maximum file container (Module Index P27)</i>
5	<i>Encode a group of black and white QR codes and place the them to their groups (Module Index P28)</i>
6	<i>Create monocoloured QR code (Module Index P14) with its group (Module Index P29)</i>
7	<i>Create coloured QR code (Module Index P30)</i>

Figure 4.13. The pseudocode of partial execution for re-encoding level 2.

Table 4.15

List of tasks for partial extraction re-encoding level 2 module.

Index	Description
P23	Identify the index location of black and white QR code and monocoloured QR code and initialize possible manipulated text file.
P24	The manipulated text file is used as an input and produces the compression task by using compression tools. It will produce the binary compress file.
P25	The binary compress files are converted back to text by using a binary to text encoder (Base64).

P26 The mode of text file is changed from ANSI to UTF-8.

P27 The encoded characters are divided into a file based on the maximum values of black and white QR code version 40. If it has exceeded the maximum capacity, another file will be created until all the characters are fulfilled. Nonetheless, at the same time, the file quantity must be the same as the previous quantity of decoded characters.

P28 Encode the text files into the updated black and white QR codes. Place the image of black and white QR code at the index location that has been identified.

P29 Convert the black and white QR codes into the updated monocoloured QR codes by using new and remaining black and white QR codes as the input.

P30 Produce the coloured QR code by using new and remaining monocoloured QR codes as an input.

4.4 Summary

This chapter has provide a detail up about flow processes of encode, decode and partial extraction. The main ground of this thesis is to provide and prove the algorithms that have been design and developed can be increase the data capacity of coloured QR code. These algorithms contribute to the proposed coloured QR code which is in Figure 2.1 of Section 2.1. As the research in this thesis includes both pseudo code and the flow chart to be presented, hence it is a complete detail how the encode, decode and partial extraction were developed. From Chapter Four, the development of this system is based on the algorithm developed. It will continue the process in getting the finding in Chapter Five after the development of the system completed.

CHAPTER FIVE

FINDING

This Chapter covers the detail experiment result of encode, decode and partial extraction modules that was also includes in the Phase Three testing methodology. Each modules contain compression, multiplexing and multilayer processes. The following sections describe findings of the experiment in detail.

5.1 Encode Experiment

The experiments cover in getting the total characters, total characters lost and processing time during conversion from text file to colour QR code. Every module will be tested and comparison will be made among black and white QR code. The results are divided into different error correction levels which are L, M, Q and H.

5.2 Encode Modules Experiment Result

The experiment of the proposed techniques in generating a coloured QR code is performed based on three criteria, namely error correction level, data density, and computational time. The input data is employed in this experiment and a text based on a short story is stored in twenty-four black and white QR codes. Figure 5.1 shows a part of the employed input text that has various types of characters such as numeric, alphabets, and several symbols.

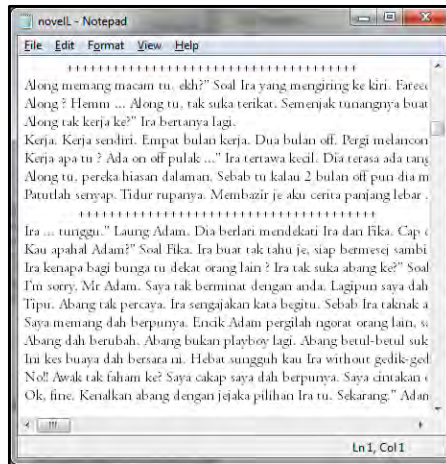


Figure 5.1. A part of the employed Malay short story.

The utilised benchmark method is QR code version 40. In this experiment, the character's allocation on each QR code has been set with a limited amount of characters according to error correction level. Hence, the maximum number of characters stored in each black and white QR code version 40 is shown in Table 5.1. It is shown that error correction level L has an advantage or highest on total amount characters among others error correction level. It can store 2952 characters for black and white QR code version 40

Table 5.1

The maximum number of characters stored in each QR code version 40.

Error Correction Level	Total Characters of each QR Code
L	2,952
M	2,330
Q	1,662
H	1,270

Meanwhile, the size of the text file is shown in Table 5.2. The text files are used as an input to this experiment and the amount of characters for each black and white QR code is divided based on error correction level.

Table 5.2

The size of the text file.

Error Correction Level	Text file size
L	126 KB (129,512 bytes)
M	94.2 KB (96,477 bytes)
Q	68.8 KB (70,545 bytes)
H	51.7 KB (52,996 bytes)

The reason why error correction level needs to be tested is because it needs to obtain some information about the amount of characters that can be stored by using different levels within this model. Normally, if the error correction level is set with a lower recovery mode such as L, then the total characters that can be stored in the QR code increase as compared to level H. In this experiment, the error correction level is placed with four levels, which are L, M, Q, and H. From the data depicted in Table 5.3 below, it can be studied that the amount of characters will increase if the error correction level is lower, which is level L. This is due to the feature of recovery function that was embedded in the QR code. The minimum amount of characters that can be stored in the new coloured QR code is 51,240 characters excluding new line and carriage return in error correction level H. Meanwhile, the maximum amount of characters that can be stored is 125,114 characters in error correction level L. There are no missing

characters during the encoding processes from the experiment conducted. All the characters can be recovered in all error correction levels.

Table 5.3

Amount of characters encoded based on the sequence of compression, multiplexing and multilayer.

Characters	Error Correction Level L		Error Correction Level M		Error Correction Level Q		Error Correction Level H	
	Encode	Decode	Encode	Decode	Encode	Decode	Encode	Decode
Total characters	125,114	125,114	93,295	93,295	68,201	68,201	51,240	51,240
Total characters including new line and carriage return	129,512	129,512	96,477	96,477	70,545	70,545	52,996	52,996

The benefit of compression includes reducing the size of data, hence utilising less time to transmit the data. The lossless compression technique was chosen because it uses less bandwidth (Kumar, Sharma, & Singh, 2012) and storage space (Goel & Singh, 2014). Furthermore, the lossless compression technique is suitable for text data as every single bit of data that was in the original file remains the same after the file is extracted. The lossless compression technique can avoid the loss of information and requires less storage (Goel & Singh, 2014). The name of the application used is remained from the experiment in Section 4.1.2.1.2. The compression process continues with the Base64 encoder, which is used to convert the binary file to the text file. An experiment was conducted to check the availability of characters during the encoding

processes. Table 5.5 shows the result of total characters during Base64 encoding (before) and decoding (after) processes. The tabulated data shows that no missing characters were found during the encoding and decoding processes for all error correction level.

It seems that Base64 encodings can be used to reduce the amount of characters and this result can be compared in Table 5.5. When the Base64 encoder is applied to convert binary to text, the total characters in text mode that can be embedded into the QR code become larger than the binary mode. Table 5.4 shows the comparison of total characters that can be embedded into QR code order by type of characters. The numeric character type can hold more characters in the QR code version 40 compared with others character type.

Table 5.4

The comparison of total characters in black and white QR code by type of characters.

Character type	Characters	Maximum characters
Numeric	0,1,2,3,4,5,6,7,8,9	7089
Alphanumeric	(0-9), (a-z), (A-Z), space, \$, %, *, +, -, ., /, ;, .	4296
Binary / Byte	8 bits	2953
Kanji / Kana	Japanese symbol	1817

Table 5.6 shows the elapsed time of the encoding compression process. The encoding process starts from GZip compression until Base64 encoder. Error correction level H has the capability to complete the encoding compression process faster than the other

Table 5.5

The result of total characters during Base64 encoding (before) and decoding (after) processes.

Characters	Level L		Level M		Level Q		Level H	
	Before	After	Before	After	Before	After	Before	After
Total characters	70,845	70,848	53,275	53,280	39,887	39,888	30,479	30,480
Total characters including new line and carriage return	70,845	70,848	53,275	53,280	39,887	39,888	30,479	30,480

error correction levels, which consumed 21 milliseconds to complete the processes. The reason why error correction level H consumed less processing time because it hold less characters compare with others which is less to process time for central processing unit (CPU) to process. From this perspective, it can be concluded that the time consumed to complete the process contributes to the total characters of QR code for this research.

Table 5.6

The elapsed time of encoding compression process.

Error Correction Level	Compression
L	0s 64ms
M	0s 29ms
Q	0s 24ms
H	0s 21ms

The encoding multiplexing process starts when the compression process is completed. This process will complete when the red, green, and blue QR codes are created. The time consumed for this process has been tested in order to gain information on which error correction level consumes the longest time during this process. Table 5.7 clearly shows that error correction level H has the capability to complete the process immediately as compared to others. This refers to the elapsed time displayed. It can be concluded that error correction levels H, Q, and M are able to complete over 50% less than error correction level L.

Table 5.7

The elapsed time of encoding multiplexing process.

Error Correction Level	Multiplexing Elapsed Time
L	13s 238ms
M	6s 531ms
Q	6s 276ms
H	6s 254ms

Finally, the last process in this model is finding the time employed in the multilayer process by using different error correction levels. This process starts after the red, green, and blue QR codes are created. These QR codes will be merged to produce the coloured QR code. From the experiment conducted, the multilayer process is measured based on time in the computer system. Table 5.8 tabulates the result of the multilayer process in second and millisecond. The time range between error correction levels M, Q, and H are not so much different, but error correction level L has a much different

time consumption.. It can be concluded that error correction level L is not suitable to be used if the user needs to complete the process immediately (multilayer).

Table 5.8

The result of multilayer process in second and millisecond.

Error Correction Level	Multilayer Elapsed Time
L	0s 925ms
M	0s 899ms
Q	0s 902ms
H	0s 900ms

5.2.1 Overall Encode Experiment Result

Overall, the compression, multiplexing, and multilayer encoding processes are the best way to increase the capacity of QR code. These processes can contribute extra data or information to be embarked into an image, which is the QR code. To achieve the extra capacity of the storage, a procedure needs to be followed in order to reach the goal. Figure 5.2 shows the flow process of the encoding processes of compression, multiplexing, and multilayer. It starts from a plain text file until a coloured QR Code is produced. These step processes are the summaries from the chapters discussed previously.

The encoding process starts from compression (GZip and Base64 encode), multiplexing, until multilayer. Table 5.9 shows the elapsed time of the encoding process. Error correction level H has the capability to complete the encoding process

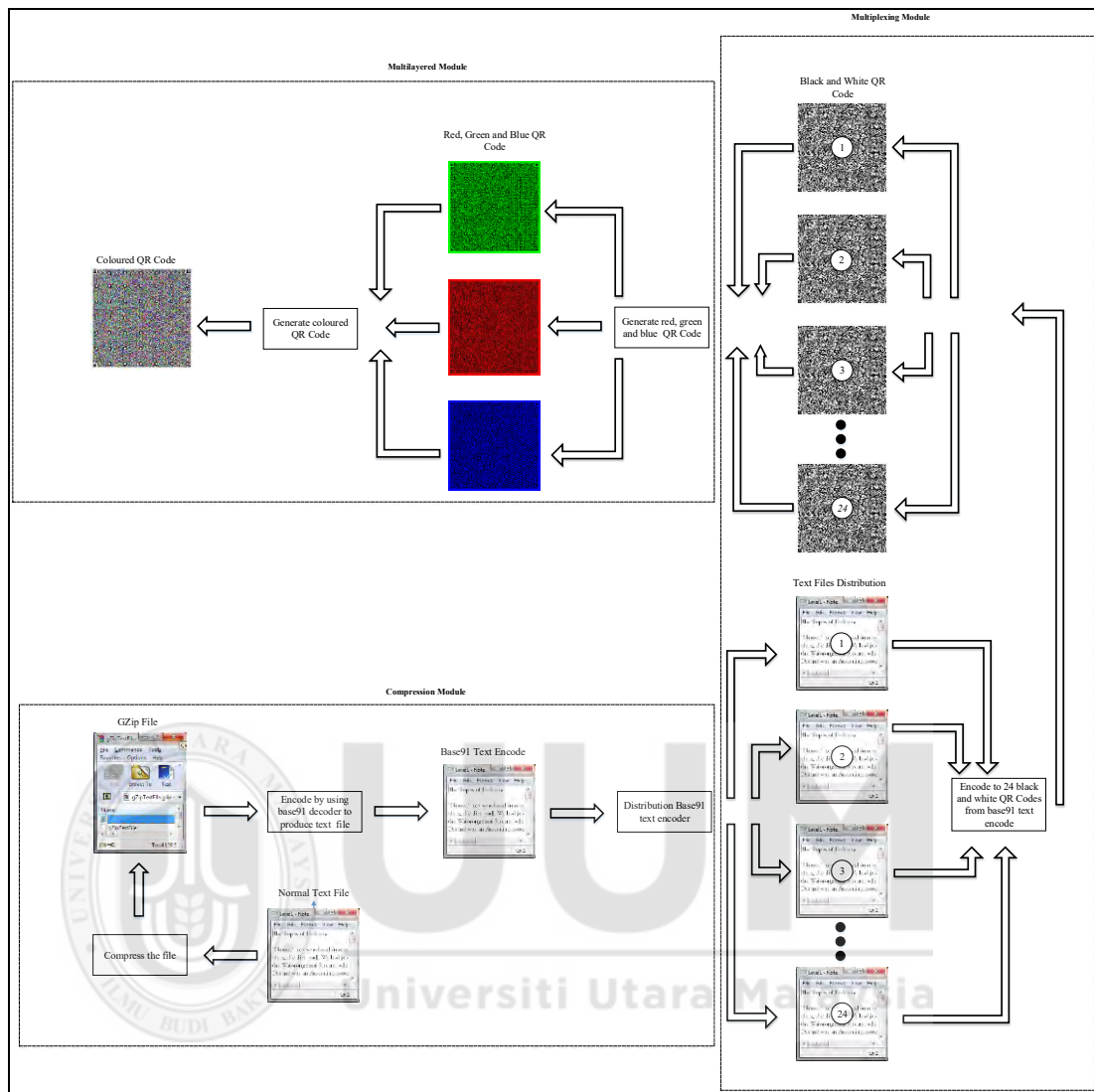


Figure 5.2. The flow processes of the encoding compression, multiplexing, and multilayer modules.

in the fastest time as compared to the other error correction levels, which consumed 6 seconds and 254 milliseconds to complete the processes. Nonetheless, at the same time, the total bytes of the image (coloured QR code) with error correction level L have the minimum size of only 105 kilobytes (108,339 bytes). From this perspective, it can be concluded that the time consumed to complete the process does not contribute to the file size of QR code for this research.

Table 5.9

The elapsed time of encoding process.

Correction Level	Error	Compression	Multiplexing	Multilayer	Multiplexing and Multilayer	Total elapse time	Coloured QR Code image	Total Bytes of
L		0s	13s	0s	13s	16s	105 KB	
		64ms	238ms	169ms	406ms	683ms	(108,339 bytes)	
M		0s	6s	0s	6s	7s	105 KB	
		29ms	531ms	97ms	629ms	551ms	(107,997 bytes)	
Q		0s	6s	0s	6s	6s	107 KB	
		24ms	276ms	92ms	638ms	494ms	(110,241 bytes)	
H		0s	6s	0s	6s	6s	108 KB	
		21ms	254ms	98ms	353ms	964ms	(110,679 bytes)	

The traditional QR code version 40 has been employed and used to compare with the new proposed coloured QR code in the form of data capacity. Table 5.10 shows the difference of text capacity between QR code version 40 and proposed coloured QR code. From this table, error correction level L has a large difference between the two QR codes. It consists of 29.12337 times of extended characters (for coloured QR code) as compared to the traditional QR code version 40. Certainly, the new proposed QR code has a large capability to extend the characters with a maximum of 125,114 characters.

Table 5.10

The difference of text capacity between QR code version 40 and proposed coloured QR code.

Error Correction Level	Version 40 (Maximum)	Proposed Coloured QR Code (Maximum)	Time difference expanded
L	4,296	125,114	29.12337
M	3,391	93,295	27.51253
Q	2,420	68,201	28.18223
H	1,852	51,240	27.66739

Recently, there are many studies and researches on extending data capacity by using a QR code. From Table 2.6 in Section 2.3.4, the research by Galiyawala and Pandya (2014) was able to extend the data capacity of QR code up to 24 times, which is the highest increment as compared to the traditional QR code and other researchers' experiment. In addition, their research as shown in Table 2.14 in Section 2.4 revealed that the total processing time to encode and decode 14 QR codes took about 53.153 and 1236.105 seconds, which indicates too much processing time consumed to complete the task. This clearly shows that Galiyawala and Pandya (2014) need to improve their processing time of decoding and encoding processes. Nonetheless, Table 5.10 shows an improvement of increment in the storage of coloured QR code, which were 29.12337 (error correction level L), 27.51253 (error correction level M), 28.18223 (error correction level Q), and 27.66739 (error correction level H) times. Furthermore, the time to complete the encoding processes is 16 seconds 683 milliseconds (error correction level L), 7 seconds 551 milliseconds (error correction

level M), 6 seconds 494 milliseconds (error correction level Q), and 6 seconds 964 milliseconds (error correction level H).

The proposed encoded QR code that employed the compression, multiplexing, and multilayer techniques has shown an increase in QR code data storage. The experiment was realised on the text based on a short story, which contained not more than 125,114 characters for error correction level L. As the undertaken experiment produced good results, it is suitable to store or embedding product description for advertisement purposes would also be a successful implementation.

5.3 Decode Experiment

The decode experiment can be done after the encode process completed. The experiment results are suggestive to expendable colour depth and colour channel. Each modules also tested such as compression and decompression, division of black and white QR code and comparing processing time and compression tool with other researcher.

5.3.1 Decode Modules Experiment Result

The decompression process is the last step to regain the original text. From the beginning until the end of the decoding processes, the elapsed time was calculated to obtain the actual time to complete the process (See Appendix H). Table 5.11 shows the compilation of elapsed time of overall decoding processes.

From Table 5.11, the overall processing time is nearly 6 seconds to complete the decoding process. In detail, error correction level Q was able to complete the process

Table 5.11

The compilation of elapsed time of overall decoding processes.

Error Correction Level	Demultilayer	Demultiplexing	Decompression	Overall Total
L	0s 925ms	1s 656ms	0s 12ms	5s 959ms
M	0s 899ms	1s 650ms	0s 11ms	5s 929ms
Q	0s 902ms	1s 614ms	0s 9ms	5s 858ms
H	0s 900ms	1s 655ms	0s 8ms	6s 11ms

within 5 seconds 858 milliseconds, which is the fastest among the other three error correction levels. The overall total column contains all process information starting from the first command until the end line of the programme. The demultilayer, demultiplexing, and decompression module columns contain the time consumed to complete the specific task of the module. The decoding process in this model experiment is still acceptable because the total of black and white QR codes from this model is 24 units as compared to the number of maximum QR codes from the experiment in Table 5.12, which is only 14 units. In addition, the processing time is still the lowest as compared to the previous research from Table 5.12 given below. It shows that this model is able to save more data or information as compared to the experiment in Table 5.12. In detail, Table 5.12 shows 14 black and white QR codes that were used to decode black and white QR codes with 1236.105 seconds to complete the task. Meanwhile, the current experiment is able to decode 24 black and white QR codes with completion time of approximately 6 seconds. The time difference between them is 1230.105 seconds.

This method can extend the data storage of QR code if a minor modification is made in the compression, multiplexing, and multilayer modules. The modification involves choosing the best compression technology, developing a detailed colour depth, and extending the colour channel.

Table 5.12

The summary of processing time of decoding by Galiyawala and Pandya (Courtesy: Galiyawala & Pandya (2014)).

Sr No.	Number of QR Codes Multiplexed	Assigned Distinct Colour (2*)	Decoding Processing Time (second)
1	2	4	2.892
2	3	8	2.988
3	4	16	2.98
4	5	32	3.084
5	6	64	3.381
6	7	128	3.624
7	8	256	4.359
8	9	512	6.128
9	10	1024	11.406
10	11	2048	28.398
11	12	4096	91.393
12	13	8192	323.198
13	14	16384	1236.105

5.3.1.1 Compression and Decompression

GZip was chosen and used in this experiment as it is the most popular data compression tool (Morse Jr., 2005). The choice of compression tool is based on the experiment of many compression tools. The compression model for the QR code in this research is merged with the binary to/from text encoder/decoder. The reason is that the text mode allows more data to be stored in the QR code. Nevertheless, the text file must be compressed first before the binary to text process is performed. This method will increase the percentage of compression. Table 5.13 shows the normal QR code version 40 and compression tool (GZip) via binary to text encoding/decoding gap and percentage of compression order by error correction level. The total character result is based on one unit of black and white QR code only.

The percentages in the QR code are different among them because the error correction level has fixed the level of recovery of the QR code. The result of this experiment emphasised the percentage that the QR Code can embed after compression. The GZip algorithm can exceed within 40% to 52% of data compression of this model. If another compression tool can exceed more than 52% and is compatible with the binary to text encoder, it is better to use that compression tool as a goal to compress more data. This compression model is flexible with other compression tools because all the experiments were performed by using multiple compression tools such as Zip, Huffman Coding, LZW etc. The best compression tool can be selected if the rate of compression of text is high. Table 5.14 shows an experiment to obtain the maximum total characters that can be stored in QR code version 40 by error level with multiple compression tools without encoder/decoder.

Table 5.13

The normal QR code version 40 and compression tool (GZip) via binary to text encode/decode gap and percentage of compression order by error correction level.

Error Correction Level	Total Normal	GZip and Binary to Text	GZip and Binary to Text (%)	Total Reduction (%)
L	2,952	1,528	52	48
M	2,330	1,140	49	51
Q	1,662	743	45	55
H	1,270	514	40	60

Table 5.14

The maximum total characters stored in QR code version 40 by error level with multiple compression tools without encoder/decoder.

Error Level	Normal	Zip	Gzip	LZW	Huffman Coding	Huffman & Gzip	Huffman & Zip
H	1270	1560	1784	1167	212	1364	1166
Q	1662	2114	2405	1627	282	1827	1639
M	2330	3188	3470	2441	392	2607	2425
L	2952	4226	4480	3253	503	3323	3095

The total normal characters that can be loaded into a black and white QR code version 40 after compression and binary to text conversion is formulated as follows:

$$\text{Total normal character after decompression} = (a * b) / c$$

Where:

$a =$ total maximum characters of a normal QR code version 40 by error correction level

$b =$ total characters after compression and encoding of that same value of maximum character storage of a normal QR Code version 40 by error correction level

$c =$ total character after compression and encoding of normal characters of QR code version 40 by error correction level

If the 24 units of black and white QR code are used to embed the text, more text characters can be installed and can be referred to the next step, which is the multiplexing process. It can be shown in Table 5.15.

Table 5.15

The total character storage of 1, 8, 24, and N units of black and white QR codes after completion of compression process and binary to text decoding process.

Error Correction Level	1 Black and White QR Code	8 Black and White QR Codes	24 Black and White QR Codes	N Black and White QR Codes
L	5703.079	45624.63	136873.9	$N * 5703.079$
M	4762.193	38097.54	114292.6	$N * 4762.193$
Q	3717.69	29741.52	89224.57	$N * 3717.69$
H	3137.938	25103.5	75310.51	$N * 3137.938$

5.3.1.2 Multiplexing Using Colour Depth

The multiplexing process depends on how many bits of colour are allocated in each colour channel to produce a monocoloured QR code. The total bit allocated in each colour channel represents a type of colour that will be displayed in the monocoloured QR code in a single pixel. The total colour in the monocoloured QR code is known as colour depth. Colour depth is a bit depth of the bit number used for each colour component of a single pixel. If the total value of bit is less, it means the colour

component of the single pixel can contain less colour pixels. Nonetheless, it is vice versa if the total value of bit contains many bits. The experiment result shows that the increasing total colour bit depth contributes to the outcome of total characters. This process can be done during the multiplexing process. In addition, the value of total colour bit depth is equivalent to the total of black and white QR codes used for the multiplexing process, which is eight units for each monocoloured QR code. It can be formulated by using the initialisation value from the experiment of 24-bit colours. Table 5.16 shows the calculation or simulation of the outcome of total character order by error correction level from 24 and above units of black and white to 3 monocoloured QR codes (red, green, and blue). If the 24-bit colours are used, the total characters are 136873.9 for error correction level L, 114292.6 for error correction level M, 89224.57 for error correction level Q, and 75310.51 for error correction level H. These total characters' values can be used as a constant order by error correction level, if the calculation to expand the colour depth is implemented. The formula is as follows:

$$\text{Total characters} = (x \text{ new bit RGB colour depth of error correction level} * (\text{total character of 24-bit colour depth of error correction level}) / 24 \text{ bits of RGB colour depth of error correction level})$$

where:

$x =$ the value of new colour bit RGB

From the data tabulated, it is shown that if the colour bit of RGB increases, the total characters will be raised. The increase of total characters is applied to all error correction levels. For example, error correction level L with 240 bits RGB colour have

Table 5.16

The calculation or simulation of the outcome of total character order by error correction level from 24 and above units of black and white to 3 monocoloured QR codes (red, green, and blue).

Colour Bit Single Channel	Colour Bit RGB**	Error Correction Level			
		L	M	Q	H
8	24	136874	114293	89225	75311
10	30	171093	142866	111531	94138
16	48	273748	228586	178449	150621
24	72	410622	342879	267674	225932
32	96	547496	457172	356898	301242
40	120	684370	571465	446123	376553
48	144	821244	685758	535348	451863
56	168	958118	800051	624572	527174
64	192	1094992	914344	713797	602484
72	216	1231866	1028637	803021	677795
80	240	1368740	1142930	892246	753105

*Note: ** also referred to as the total black and white QR codes used*

the highest bit colours among the data tabulated, which is 1368740 million characters. If the average of 7 characters is equivalent to a word, then the total words are approximately 195534.29 words or 434.5 pages (Arial font, size 12, and single spacing) (“Convert Words to Pages,” 2016). It is shown that in the future, if the 240 bits RGB colour exists, then the coloured QR code is able to save the content of a research thesis or book in a single coloured QR code.

5.3.1.3 Multilayer Using Colour Channel

The multilayer process in this model is a process of combining monocoloured QR codes into a single coloured QR code. In this experiment, the total colour channels used are three colours, which are red, green, and blue. A colour channel contains the colour information of the dominant colour element in a colour model. For example, the RGB colour model consists of three major colours, which are red, green, and blue. The colour channel can be added with more three main colours. This technique can be formulated based on the following:

*Total characters = colour channel * (total character of 24 (RGB of 8 bits) bits colour depth of error correction level) / 24 bits of normal RGB colour depth of error correction level*

where:

Colour channel = colour model(3^{x bit}, 4^{x bit}, 5^{x bit}, ..., n^{x bit})

where:

n = total main colour channel

x bit = the colour depth {8 bits, 16 bits, 24 bits, ... x bits}

From the model created in this research experiment, a simulated colour channel has been added with more than three colour channels. All data is based on simulation by adding more than three colours in a single colour model. This experiment is suitable for the multilayer/demultilayer process, which changes from monocolour to coloured QR code. The colour depth depends on bit colours allocated. The first experiment starts with an 8-bit colour depth and increment by 8 bits of a new colour component. Table

5.17 shows the increment of colour component of the RGB model with 8-bit colour depth order by error correction level.

Table 5.17

The simulation in increment of channel using RGB model with 8-bit colour depth order by error correction level.

Channel (8 bits)**	Colour Bit RGB	Error Correction Level			
		L	M	Q	H
3	24	136874	114293	89225	75311
4	32	182499	152391	118966	100414
5	40	228123	190488	148708	125518
6	48	273748	228586	178449	150621
7	56	319373	266684	208191	175725
8	64	364997	304781	237932	200828
9	72	410622	342879	267674	225932
10	80	456247	380977	297415	251035
11	88	501871	419074	327157	276139
12	96	547496	457172	356898	301242

*Note: ** also referred to as the total monocoloured QR codes used*

Even though the total characters are not so many as for colour depth technique, this technique can also gain the characters that can be embeded into the coloured QR code. Other colour depths of colour channels that can be implemented are shown in Tables 5.18, 5.19, 5.20, and 5.21. All the tables given below are increased by 10 bits, 16bits, 24 bits, and 80 bits.

Table 5.18

The simulation in increment of channel using RGB model with 10-bit colour depth order by error correction level.

Channel (10 bits)**	Colour Bit RGB	Error Correction Level			
		L	M	Q	H
3	30	171093	142866	111531	94138
4	40	228123	190488	148708	125518
5	50	285154	238110	185885	156897
6	60	342185	285733	223062	188276
7	70	399216	333355	260238	219656
8	80	456247	380977	297415	251035
9	90	513278	428599	334592	282414
10	100	570308	476221	371769	313794
11	110	627339	523843	408946	345173
12	120	684370	571465	446123	376553

Note: ** also referred to as the total monocoloured QR codes used

Table 5.19

The simulation in increment of channel using RGB model with 16-bit colour depth order by error correction level.

Channel (16 bits)**	Colour Bit RGB	Error Correction Level			
		L	M	Q	H
3	48	273748	228586	178449	150621
4	64	364997	304781	237932	200828
5	80	456247	380977	297415	251035
6	96	547496	457172	356898	301242
7	112	638745	533367	416382	351449

8	128	729995	609563	475865	401656
9	144	821244	685758	535348	451863
10	160	912493	761953	594831	502070
11	176	1003743	838149	654314	552277
12	192	1094992	914344	713797	602484

*Note: ** also referred to as the total monocoloured QR codes used*

Table 5.20

The simulation in increment of channel using RGB model with 24-bit colour depth order by error correction level.

Channel (24 bits) **	Colour Bit RGB	Error Correction Level			
		L	M	Q	H
3	72	410622	342879	267674	225932
4	96	547496	457172	356898	301242
5	120	684370	571465	446123	376553
6	144	821244	685758	535348	451863
7	168	958118	800051	624572	527174
8	192	1094992	914344	713797	602484
9	216	1231866	1028637	803021	677795
10	240	1368740	1142930	892246	753105
11	264	1505614	1257223	981471	828416
12	288	1642488	1371516	1070695	903726

*Note: ** also referred to as the total monocoloured QR codes used*

From the tables given above, it can be concluded that if the colour depth and the channel in a colour model are added, the total characters will be increased dramatically. From this situation, the model can contribute to the increase in total

Table 5.21

The simulation in increment of channel using RGB model with 80-bit colour depth order by error correction level.

Channel (80 bits) **	Colour Bit RGB	Error Correction Level			
		L	M	Q	H
3	240	1368740	1142930	892246	753105
4	320	1824987	1523907	1189661	1004140
5	400	2281233	1904883	1487077	1255175
6	480	2737480	2285860	1784492	1506210
7	560	3193727	2666837	2081907	1757245
8	640	3649973	3047813	2379323	2008280
9	720	4106220	3428790	2676738	2259315
10	800	4562467	3809767	2974153	2510350
11	880	5018713	4190743	3271569	2761385
12	960	5474960	4571720	3568984	3012420

*Note: ** also referred to as the total monocoloured QR codes used*

characters if the colour depth and channel change the selection of the best compression tool.

5.3.2 Calculation of Total Black and White QR Codes

The models of compression, multiplexing, and multilayer need to be merged in order to obtain more characters stored in the coloured QR code. Nonetheless, at the same time, the total number of black and white QR codes relies on colour depth and colour channel. For example, if the colour depth is 16 bits and the channel is 4, then the black and white QR code would be 64 units. It can be formulated by using the following:

*Total black and white QR code = Total bits colour depth * Total channel of colour model.*

The main reason to determine the number of black and white QR code is to estimate the suitable total characters that suit with the total black and white QR codes. If the text characters are not so many, it is not suitable to use 80-bit colour depth with 12 channels because it will consume more elapsed time to complete the process. Figure 5.3 illustrates the diagram of RGB colour depth and colour model based on compression, multiplexing and multilayer processes. The calculation formula can be used in the opposite way. For example:

Total bits colour depth = Total black and white QR code / Total channel of colour model

or

Total channel of colour model = Total black and white QR code / Total bits colour depth

From statement above, it is shown the way to get optimal total bits colour depth or total channel of colour model that can be used to avoid wastage of colour depth and colour channel when users have certain amount of black and white QR code.

5.4 Partial Extraction Levels

This partial experiment consists 2 levels of extraction. The first one is called partial extraction level 1 and another one called called partial extraction level 2. Each level have their own purpose which are level 1 extracting only a single black and white QR

code and level 2 is extracting only a group of black and white QR code. Each level has decode and re-encode processes.

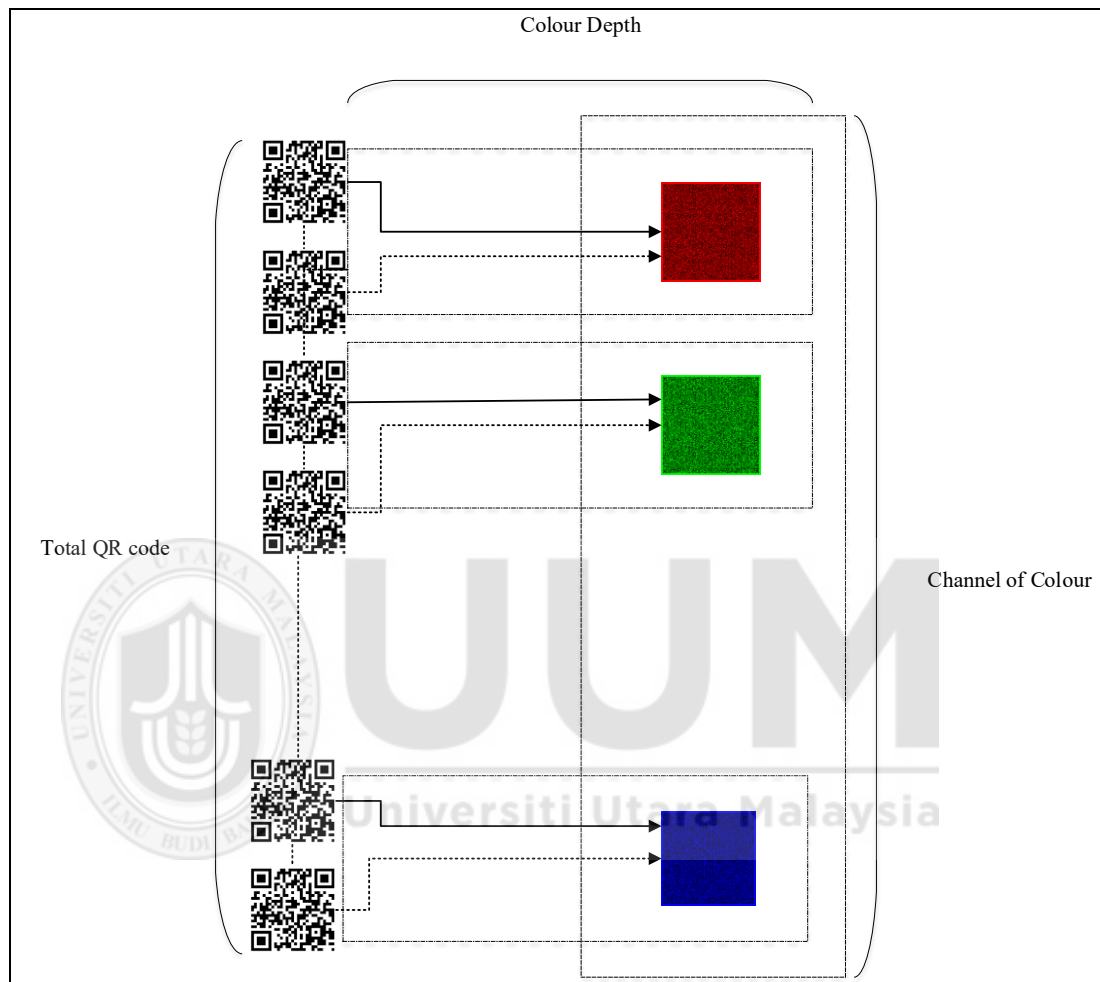


Figure 5.3. The diagram of RGB colour depth and colour channel.

5.4.1 Partial Extraction Levels Experiment Result

The level 1 decoding abstract model that was used to update the information in black and white QR codes is shown in Figure 5.4. The task of this model is updating the black and white QR code information at the suggested index location. Meanwhile, the re-encoding abstract model is shown in Figure 5.5 and this process will begin with updating information in the text file at the previous index location in level 1.

In addition, the level 2 abstract model consists of the operation to update the mono QR codes (red, green, and blue). The process will generate eight text files and the provider can manipulate a larger amount of information. Figures 5.6 and 5.7 show the example of decoding and re-encoding of red monocoloured QR code.

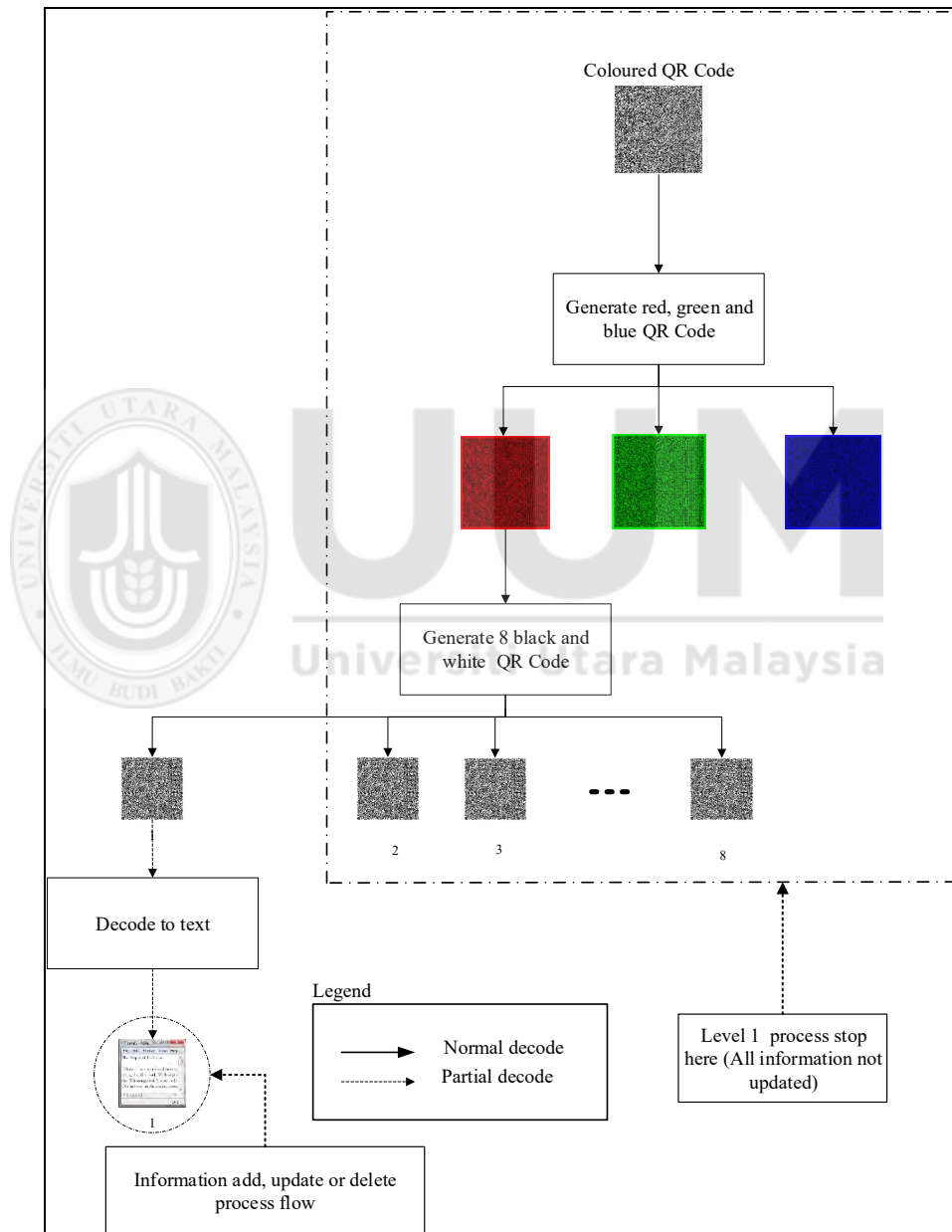


Figure 5.4. Level 1 decoding abstract model.

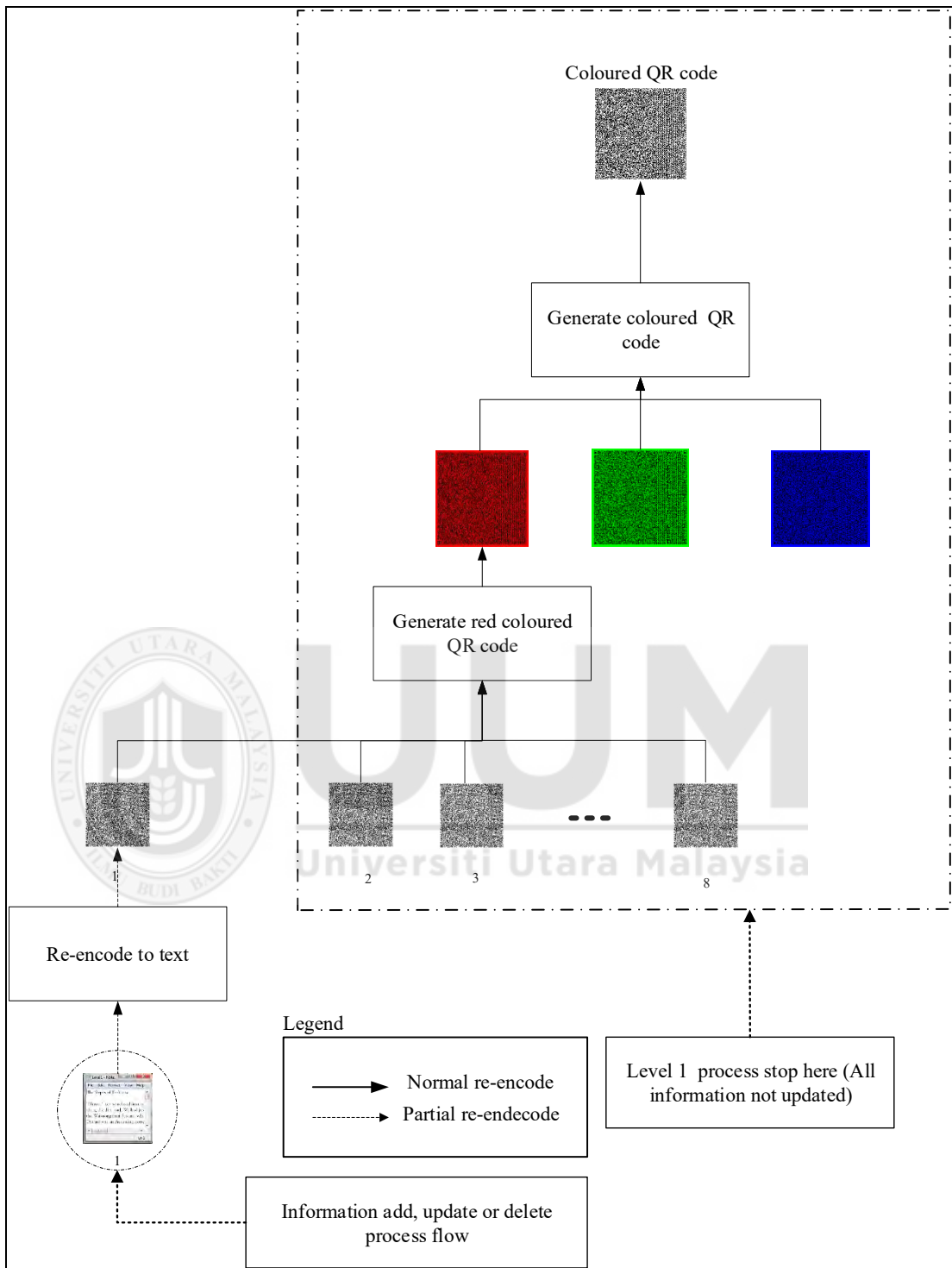


Figure 5.5. Level 1 re-encoding abstract model.

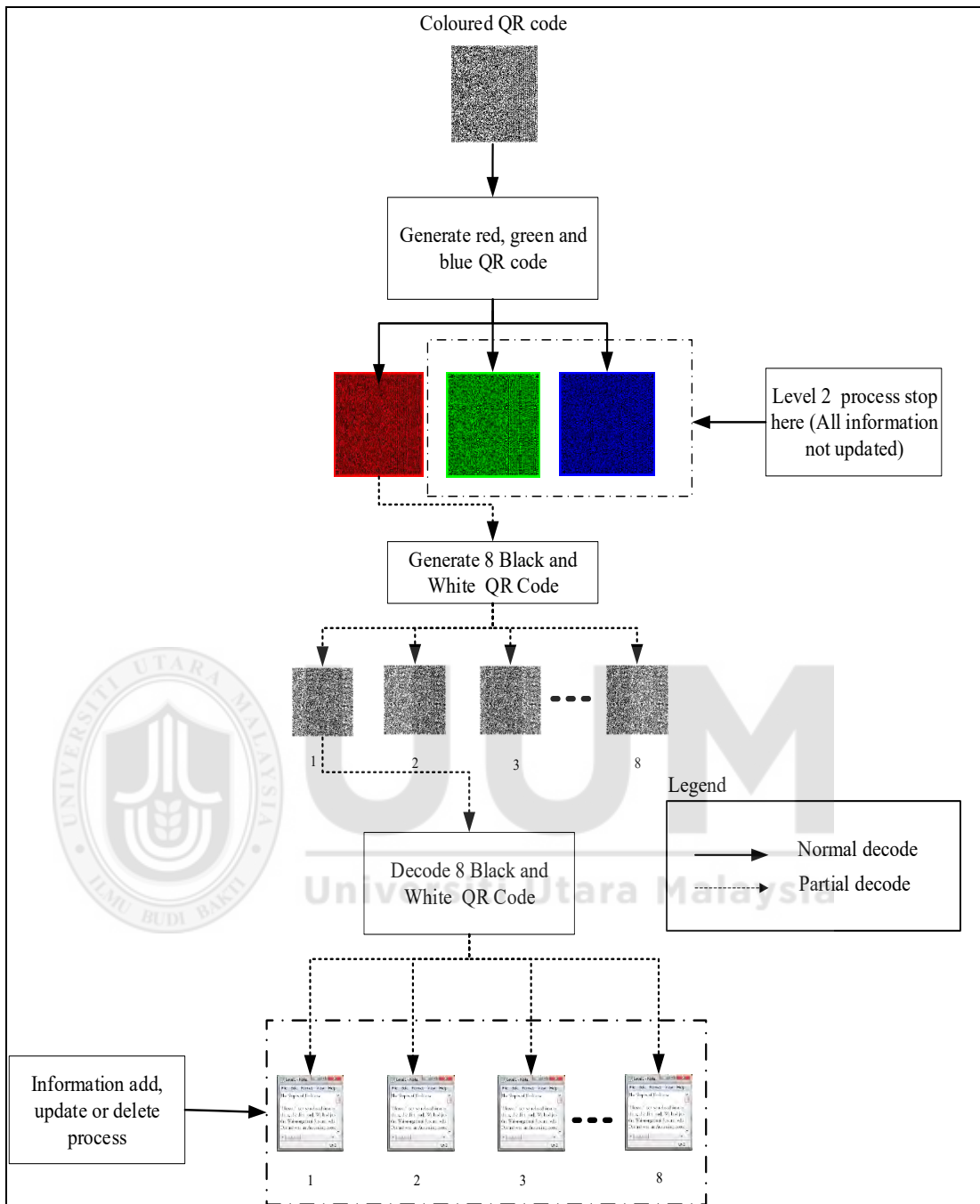


Figure 5.6. Level 2 decoding abstract model.

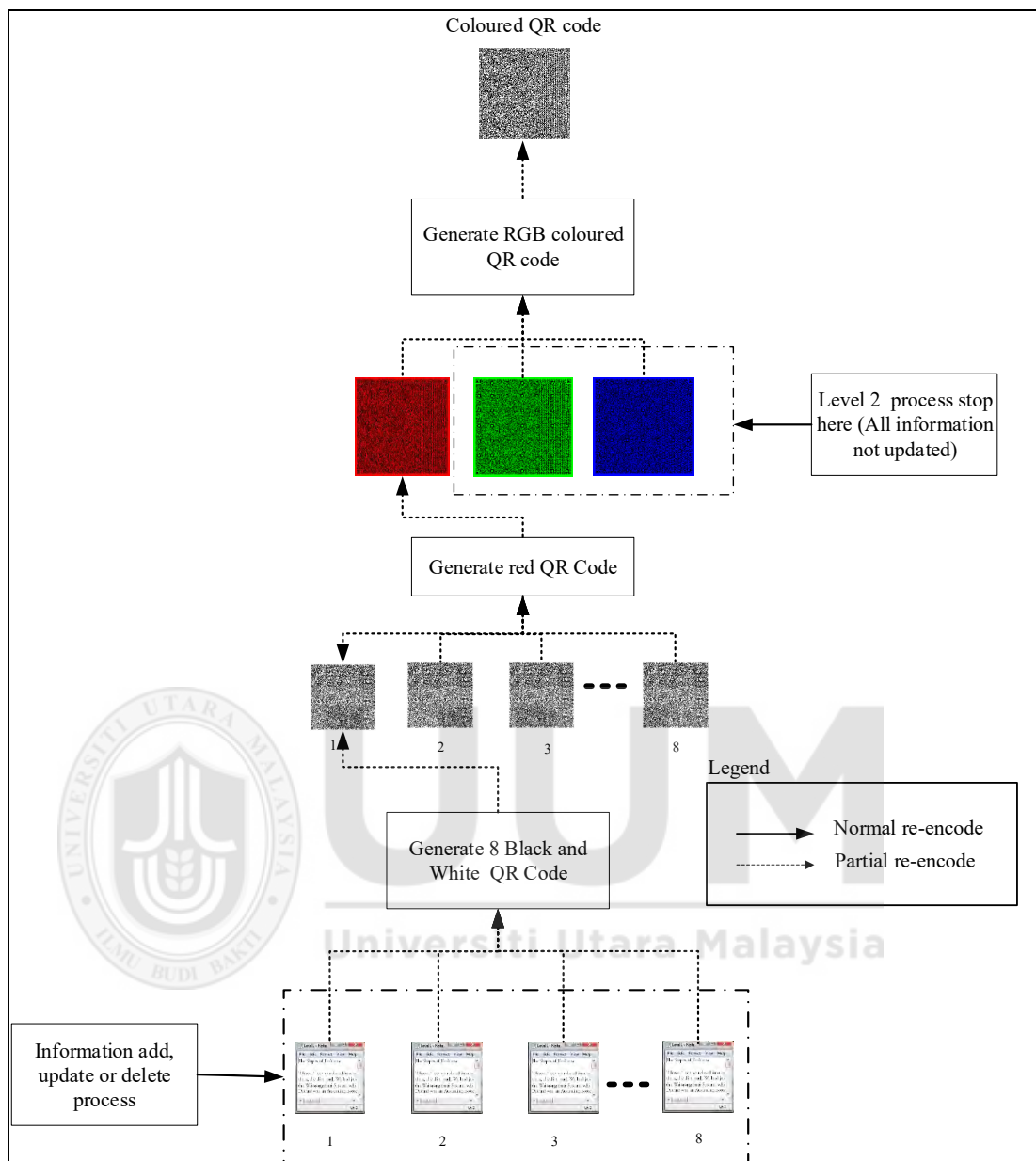


Figure 5.7. Level 2 re-encoding abstract model.

5.4.1.1 Partial Input Data

The input data is a compilation of texts based on short stories, which are stored in black and white QR codes. The amount of characters depends on the error correction level. error correction level L will be used in this experiment. The reason why other error correction levels are not tested is that the target of this experiment is to save processing

time. Figure 5.8 illustrates a part of the employed input text, which includes various types of characters such as numeric, alphabets, and several symbols. The utilised benchmark method is QR code version 40 with error correction level L. The maximum number of characters stored in each QR code version 40 is shown in Table 4.4. The experiment of the undertaken experiment is based on computational time.

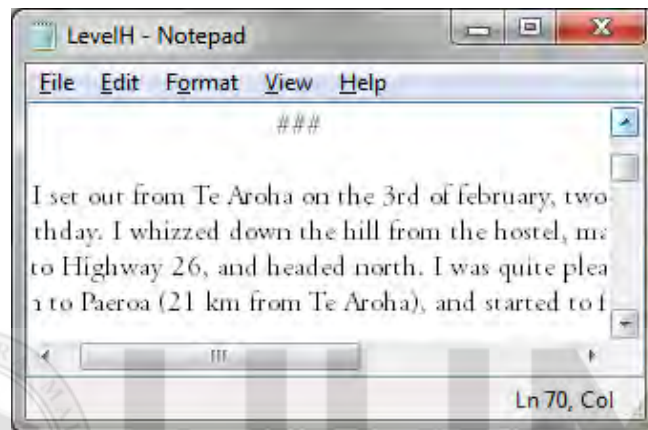


Figure 5.8. A part of input data text.

5.4.1.2 Result of Extraction Levels

Several experiments were conducted and some figures and tables were tabulated as the results of the experiments. The process flow results for the benchmark method (i.e. QR code version 40) and the proposed technique are shown in Figures 5.9, 5.10 and 5.11.

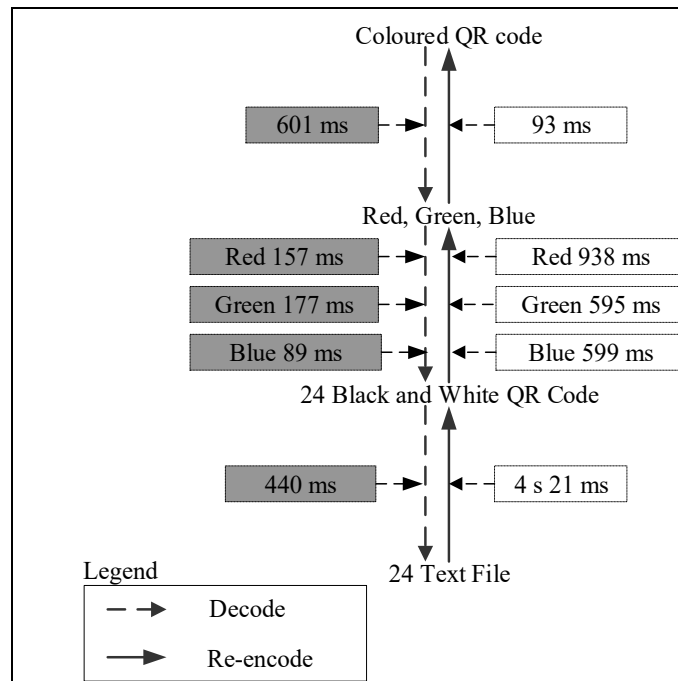


Figure 5.9. The process flow results for QR code version 40.

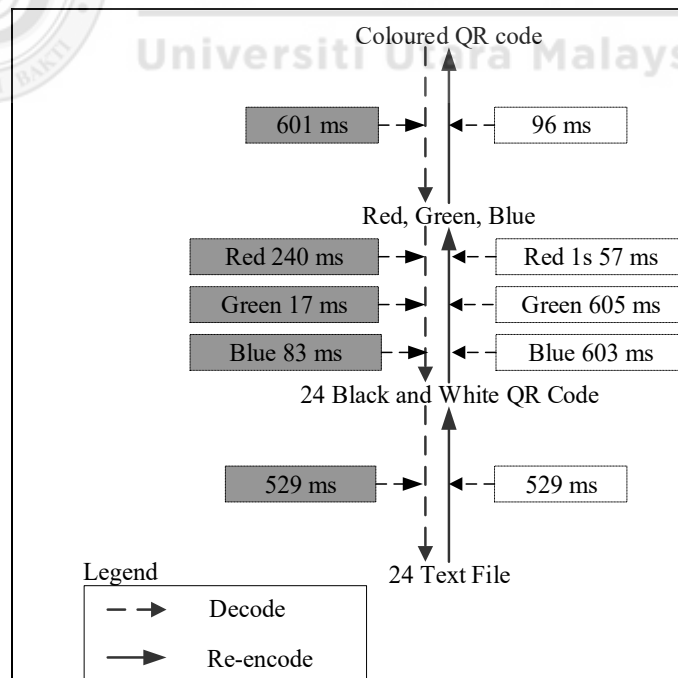


Figure 5.10. The process flow results for proposed technique level 1.

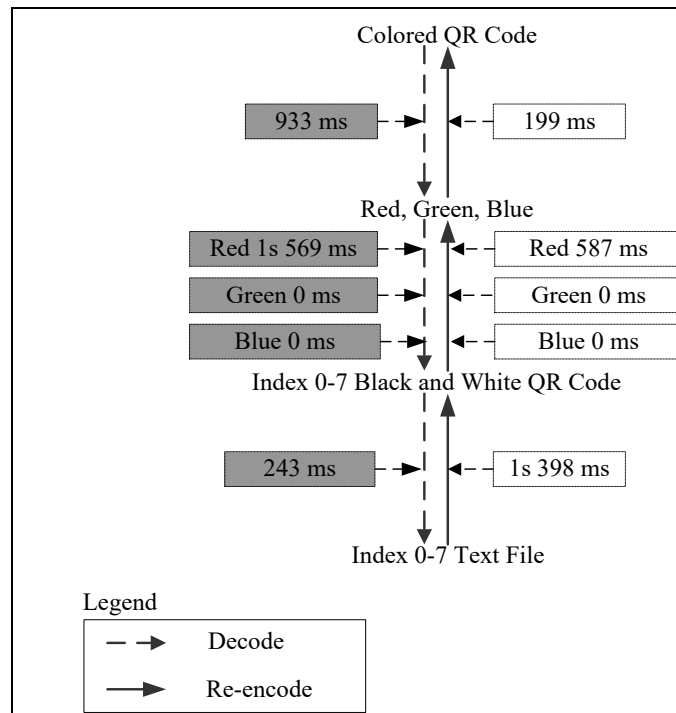


Figure 5.11. The process flow results for proposed technique level 2.

Based on Figures 5.9, 5.10, and 5.11, the data was collected and tabulated into four tables ordered by levels 1 and 2. Comparisons were made between each level and the benchmark method. At the same time, each level is divided into decoding and re-encoding processes. Tables 5.22, 5.23, 5.24, and 5.25 show the comparison between levels and processes in terms of different processing time.

From Table 5.22, the overall result in level 1 (decode) shows that the proposed technique is able to complete the task within 1 second 251 milliseconds as compared to the benchmark method, which was 1 second 464 milliseconds. The time difference between these techniques is 231 milliseconds. During the process to generate the red, green, and blue QR codes, the time consumed for the proposed technique is slower than the benchmark technique due to the implemented task of collecting information

of pixels in the red QR code. This process is not available in the benchmark technique. As a result, this will consume more processing time.

Table 5.22

The comparison between benchmark and proposed techniques in level 1 of decoding process. Level 1(Decode).

From	To	Benchmark	Proposed Level 1	Difference
Coloured QR Code	Red, Green, Blue	601 ms	978 ms	-377 ms
Red, Green, Blue QR Codes	24 / Index 0 Black and White QR Code	423 ms	187 ms	236 ms
24 / Index 0 Black and White QR Code	24 / Index 0 File Text	440 ms	84 ms	356 ms
Total		1s 464 ms	1s 251 ms	231 ms

When the decoding process in Table 5.22 is completed, the re-encoding process will take over after the process of updating, deleting or adding information is done. From Table 5.23, the total time consumed has a large time difference between the benchmark and proposed techniques, which is 5 seconds 199 milliseconds. The benchmark technique has to generate 24 black and white QR code, meanwhile, the proposed technique only needs to generate one black and white QR code. This is why the benchmark technique consumes more time as compared to the proposed technique.

Table 5.23

The comparison between benchmark and proposed techniques in level 1 of re-encoding process. Level 1(Re-encode).

From	To	Benchmark	Proposed Level 1	Difference
24 / Index 0 File Text	24 / Index 0 Black and White QR Code	4s 21 ms	231 ms	3s 790ms
24 / Index 0 Black and White QR Code	Red, Green, Blue	2s 132 ms	621 ms	1s 511ms
Red, Green, Blue QR Codes	Coloured QR Code	93 ms	195 ms	-102 ms
Total		6s 246 ms	1s 47ms	5s 199ms

Table 5.24

The comparison between benchmark and proposed techniques in level 2 of decoding process. Level 2 (Decode).

From	To	Benchmark	Proposed Level 2	Difference
Coloured QR Code	Red Green, Blue	601 ms	933 ms	-332 ms
Red, Green, Blue QR Codes	24 / Index 0-7 Black and White QR Code	423 ms	1s 569 ms	- 1s 146ms

24 / Index 0-7 Black and White QR Code	24 / Index 0-7 File Text	440 ms	243 ms	197 ms
Total		1s 464 ms	2s 745 ms	-1s 281ms

Table 5.25

The comparison between benchmark and proposed technique in level 2 of re-encoding process. Level 2 (Re-encode).

From	To	Benchmark	Proposed Level 2	Difference
24 / Index 0-7 File Text	24 / Index 0-7 Black and White QR Code	4s 21 ms	1s 398 ms	2s 623ms
24 / Index 0-7 Black and White QR Code	Red, Green, Blue	2s 132 ms	587 ms	1s 545 ms
Red, Green, Blue QR Codes	Coloured QR Code	93 ms	199 ms	-106 ms
Total		6s 246 ms	2s 184ms	4s 62ms

Overall, during the generating of coloured QR code in level 2, the proposed technique consumed nearly twice the total time consumed in the benchmark technique. This is because the proposed technique has to collect pixel information in the green and blue QR codes. This can be referred in Table 5.24. But after that, the data in Table 5.25 shows the time consumption process has been conquered back by the proposed

technique. The proposed technique took only 2 seconds 184 milliseconds to complete the process while the benchmark technique completed it in 6 seconds 246 milliseconds.

In level 2, the decoding and re-encoding processes are executed in order to recover the red QR code. Table 5.24 shows the same result of the decoding process in level 1 at Table 5.22. The proposed technique took 933 milliseconds to complete the decoding process from coloured QR code to red, green, and blue QR codes, whereas the benchmark technique only took 601 milliseconds. For the next process, the proposed technique consumed 1 second 569 milliseconds, but the benchmark technique only took 423 milliseconds. The range difference is 1second 146 milliseconds. The reason is the red QR code needs to separate the image file into a group of black and white QR codes, which start from indices 0 until 7. This process needs extra time to process. Overall, the time range difference between the benchmark and proposed techniques is 1 second 281 milliseconds.

During the level 2 re-encoding process in Table 5.25, the proposed technique led the time process with a time range difference of 4 seconds 62 milliseconds. However, in generating the coloured QR code, the benchmark technique led a time range difference of 106 milliseconds. This result is similar to the level 1 re-encoding process and the reason is the same as mentioned before.

Table 5.26 below shows the information on the overall results of level 1, which has a time range difference of 5 seconds 430 milliseconds and level 2, which has a time range difference of 2 seconds 781 milliseconds. From this result, the proposed technique has a good performance as compared to the benchmark techniques.

Table 5.26

The level 1 and level 2 time range difference.

Level		Benchmark	Proposed	Total Time (Level)	Total Time
1	Decode	1s 464 ms	1s 251 ms	231 ms	5s 430ms
	Re-encode	6s 246 ms	1s 47ms	5s 199ms	
2	Decode	1s 464 ms	2s 745 ms	-1s 281ms	2s 781ms
	Re-encode	6s 246 ms	2s 184ms	4s 62ms	

5.5 Comparison With Existing QR code

All data for validation are taken from the previous researches of coloured QR code and some commercial products. An analysis of comparison between the proposed coloured QR code and existing coloured QR code is based on the data capacity for each of them. The results are compared in term of total text characters stored in UTF-8 string length or bytes saved. All of the QRcode data capacity is based on data in Table 2.1, Table 2.6, Table 2.8 and Section 2.3.6. The proposed coloured QR code (24 bits colour) will employ error correction level L because it can store more data compared to other error correct level. Table 5.27 shows the text capacity comparison between proposed coloured QR code and existing QR code (black-white and colour).

Table 5.27

The comparison text capacity between proposed coloured QR code and existing QR code (black-white and colour)

	Type of QR code/ Researchers	Type	Total Characters (UTF-8 String Length)	File size (Kilobytes)
1	Proposed QR code (24 bits colour)	Colour	125,114	122.181640625
2	QR code version 40	Black and White	4296	4.1953125
3	Nancy Victor (2012)	Black and White	4096	4
4	Kris Antoni Hadiputra Nurwono and Raymondus (2009)	Colour	12888	12.5859375
5	M. Ramya and M. Jayasheela (2014)	Colour	12888	12.5859375
6	Henryk Blasinski, Orhan Bulan and Gaura Sharma (2013)	Colour	12888	12.5859375
7	Max E. Vizearra, Alexandre Zaghetto, Bruno Macchiavello and Anderson C. A. Nascimento (2012)	Colour	128	0.125

8	Antonio Grillo, Alessandro Lentini, Marco Querinni, and Giuseppe F. Italiano (2012)	Colour	1926.144	1.881
9	Sartid Vongpradhip (2013)	Colour	12888	12.5859375
10	Prathibha N. Pillai and K. Naresh (2014)	Colour	12888	12.5859375
11	Hiren J. Galiyawala and Kinjal H. Pandya (2014)	Colour	103104	100.6875
12	Microsoft High Capacity Color Barcode	Colour	84	0.08203125
13	Zhibo Yang, Huanle Xu and Jianyuan Deng. (2017)	Colour	8859	8.651367188
14	High Capacity Colour Barcode	Colour	13704	13.3828125
15	High Capacity Colour Barcode Two Dimensional	Colour	12888	12.5859375
16	Colour QR Code-5	Colour	14356	14.01953125

From data in the Table 5.26, it can be seen that the proposed coloured QR code stores the largest total characters compared to the others. It can store 125,114 characters which is equivalent to 122.181640625 kilobytes. The second largest is the QR code proposed by Hiren J. Galiyawala and Kinjal H. Pandya which is can store 103104 characters. Prior to deriving the findings, it was learned that most of the coloured QR codes offer more storage capacity compared to black and white QR codes. From these findings, it can be concluded that the method of compression, multiplexing and multilayer used in this research is the best way to achieve largest text data capacity in coloured QR code.

5.6 Summary

The encoding process involved three major processes, namely compression, multiplexing, and multilayer. The end of this process is to produce a coloured QR code that can store more data inside it as compared to the conventional QR code. The compression process uses two type of tools, which are compression tool and text encoder. The current compression tool can be changed to another compression tool if it is able to compress more data and has extra capabilities such as more percentage of text compression or fast transaction as compared to the previous one. Likewise, the text encoder also can be changed to another text encoder that is capable to encode more characters but with less character representation. From the information above, it is shown that the compression module is a flexible module because the tools inside it can be changed according to the capability of the tools. The multiplexing process is used to join many QR codes into one QR code. This technique helps to minimise the n total of QR codes into a single QR code representation. The QR code representation uses monocolour, which can be expanded to other types of monocolour such as red, green,

and blue QR code monocolour. The n total of QR codes can also be represented by each monocolour representation. It means that more QR codes can be produced. The last process is the multilayer process, which stacks each monocoloured QR code and mixes the colours among them. This technique can save more monocoloured QR code allocation so that the QR code can be generated from monocolour into coloured presentation.

The decoding process is the inverse of the encoding process and it consists of demultiplexing, demultilayer, and decompression processes. The target is to regain the encoded data into data input. From the result, it is shown that the data can be captured back as it was inputted before without any missing characters. The maximum stored data is up to 136,874 characters as shown in Table 5.16 with error correction level L and 8 bits of each colour channel. On the whole, the characters can be increased if the colour depth and colour channel are implemented in this model. As simulated, a single coloured QR code can contain characters similar to a book or research thesis. Some equations were published as the benchmark to obtain the suitable total black and white QR code, colour depth, and colour channel that can be used as a reference as shown in Sections 5.2.1.2, 5.2.1.3 and 5.2.2

The experiment in this thesis was conducted in two parts. The first part involved the experiment to measure the time consumption of extracting a single black and white QR code in terms of whether it can consume less time than the normal method. The technique is called level 1 decoding and re-encoding. The second part was to evaluate the time consumption of extracting the monocoloured QR code. The contents of data

were bigger than the previous parts because the monocoloured QR code could contain many black and white QR codes. It is known as level 2 decoding and re-encoding.

As a result, it is shown that updating information in the QR code is suitable to be performed by this method as compared to the existing method (i.e. QR code version 40) because it consumed less computational time. With such achievement, the QR code can further be expanded in managing the updated data inside the code. Information stored in the code can always be updated according to the provider's needs.

In general, the comparison was made in the finding chapter due to prove that the proposed coloured QR code has capability to store more characters among the others QR code for a time being. The proposed QR code is using error correction L which is not considering the recovery mode criteria. The research and commercial QR codes were used as to compare with the proposed coloured QR codes. The finding in Section 5.5 has showed that highest data storage belong to the proposed coloured QR code as compared with others QR code. Thus, in overall it can be concluded that the proposed coloured QR code provides the best data storage for coloured QR code.

CHAPTER SIX

CONCLUSION

6.1 Summary of the Thesis

This thesis has explained in detail the model conducted to uncover the solution of a problem in expanding the storage of a QR code. From the past studies, many techniques were implemented to increase the storage of QR code such as compression, symbol converter, multiplexing etc. Based on the ideas from the past (which focused on expanding data), several ways have been revealed to expand more text characters to be embedded into the QR code. The proposed model can assist users to increase the storage in a faster way by using the QR code as a storage medium in a single image of QR code. Prior to this, a literature review was conducted to examine the research problem and decide the solution to the problem. The process of reviewing literature involves continuous activities that take part starting from the beginning until the final phase of the research.

6.2 Encoding Design and Development Algorithmn

The encoding algorithm process is starts with designing the encode process which is come from the preliminary study processes. The preliminary study incudes the theoretical framework, testing existing algorithm and determining the suitable algorithms. The outcome from this process is toward to select the suitable algorithms.

The selected algorithm was divided into three modules. Since three modules have been used, namely compression, multilayer, and multiplexing, the encoding throughput had to follow the sequence of the process for each module. The development includes

analyst requirement and designing the suitable hardware and software. The first method development is to compress the text data and utilise the binary to text encoder for the QR code. The compression tool is flexible to any type of compression software such as GZip, WinZip, WinRar etc., nonetheless, it is better if the compression tool is able to compress more data. The binary to text encoder is also adjustable to any kind of encoder tool such as Base8, Base16, Base91 etc., but it must also be compatible with common printed and controlled characters used with the encoder. The encoded binary to text encoder produces non-human readable text and is able to transmit data from binary into (most common) ASCII characters. In addition, the encoder is used in transmitting data or image over networking by using the email application. From the result of the compression module, the capability to increase the data into more than 52% as a maximum result comes from the GZip compression algorithm as tested before.

The multiplexing process consists of changing black and white QR codes into monocoloured QR codes. Colour depth is used for adding more black and white QR codes into single monocoloured QR codes. From the experiment, the 8-bit colour scheme was used, which produced eight black and white QR codes. If the colour depth is extended, more black and white QR codes can be produced for yielding monocoloured QR codes by this throughput. Based on the experiment, the multiplexing module provides an increment in data capacity up to 24 times as compared to the existing research module. In the undertaken experiment, 24 black and white QR codes were employed, meaning that the produced coloured QR code can increase up to 24 times.

The multilayer process brings the conversion from monocoloured QR code to coloured QR code. This is the last process from the encoding throughput model. The total monocolour QR codes that were produced from the previous process depended on the colour model used in this model. As the undertaken experiment produced positive results, it is shown that after updating information in the QR code until the multilayer process, less computational time was consumed and more data capacity was extended as compared to the existing method (i.e. QR Code version 40). In addition, the multilayer process complied with multiple error correction levels in the QR code.

From the flow process at Figure 4.1 in Section 4.1.2 of Chapter 4, the process starts with the compression technique at first, which involves compressing the data using a compression tool, then converting it from binary (compression file) to text using an encoder. The main reason to convert from binary to text is because the black QR code is not able to hold a large amount of binary data as compared to text data as shown at Table 4.4 in Section 4.1.2.1.2 of Chapter 4. At Figure 4.2 in Section 4.1.2 of Chapter 4, all information regarding total character, input file, output file, and file type were identified before the process began. The compression module ended up with producing black and white QR codes. All black and white QR codes were arranged according to the structure of text file. As to avoid compatibility of character type, the conversion to UTF-8 must be made as mentioned in index E4 at Table 4.1 in Section 4.1.2 of Chapter 4 and also in Section 4.1.2.2.1 of Chapter 4. Each black and white QR code must contain 2,952 characters due to the fixed minimum content of black and white QR code version 40 as tested in Appendix A for 20 times. The result is shown at Table 4.3 in Section 4.1.2.1.2 of Chapter 4. All QR codes were blended together by using the multiplexing technique, which produced red, green, and blue coloured QR codes.

The multiplexing technique consists of a combination of white as 0 and black as 1 for each QR Code. Each combination must refer to the eight black and white QR codes and as a result, they will produce a monocoloured QR code either red, green or blue QR code as shown at Figure 4.4 in Section 4.1.2.3 of Chapter 4. The size of the monocoloured QR codes used was 177 x 177 modules and they were stacked together via the multilayer process. The red monocoloured QR code was placed at the top, followed by green and blue. The multilayer technique is based on the combination or blended values of red, green, and blue monocoloured RGB colour scheme as mentioned in Section 4.1.2.3. The experiment involved all types of error correction level of the QR codes. As a result, if a higher level of error correction level is implemented, it will reduce the size of data density of the QR code as shown at Table 5.10 in Section 5.2.1 of Chapter 5. Furthermore, the percentage expanded among the black and white QR codes and coloured QR codes is approximately 27.8% as at Table 5.10 in Section 5.2.1 of Chapter 5. The fastest elapsed time to complete the encoding process is 6 seconds and 964 milliseconds at error correction level H as shown at Table 5.10 in Section 5.2.1 of Chapter 5. Therefore, the ROI was achieved as mention in Section 1.4 of Chapter 1.

6.3 Decoding Design and Devopment Algorithm

After the encoding process, the next process is to regain the original text. This process is called decoding, which is the reverse of encoding. The process follows the task sequence, which is demultilayer, demultiplexing, and decompression. The demultilayer process only involves the conversion from coloured QR code into monocoloured QR code. The decoding process for a coloured QR code requires

separating the layers. It can be done by reading the coloured QR codes and breaking them into monocoloured QR codes.

The demultiplexing throughput includes separating each one of the monocoloured QR codes into black and white QR codes. The demultiplexing process is a one-to-many concept, which associates single values of products with two or more value products. The total value of product refers to the number of bit allocation of colour depth in the black and white QR codes. The demultiplexing process took approximately 3 seconds to be completed in the experiment conducted.

The decompression process contains two parts of task, namely text to binary and decompression. Since the blacks and whites contain the encoded text from the binary to text encoder, they need to be converted back to the compression file type. A compression tool was used to extract all information from the compression file type into the original text. Based on the experiment in the decoding process, there was no character and control character missing at the end of the result.

The demultiplexing process involves demultilayer, demultiplex, and decompression, which are the reverse processes of encoding or known as decoding. As shown at Figure 4.6 in Section 4.2.2 at Chapter 4 , the coloured QR codes will be separated into three monocoloured QR codes (red, green, and blue) and can be expanded to other colours of the colour channel implemented as shown at Table 5.16 in Section 5.3.1.2 of Chapter 5. The next process in the algorithm is to break the monocoloured QR codes into a group of black and white QR codes. At Table 4.9 in Section 4.2.2.3 of Chapter 4, the decimal to binary technique is used to regain the eight black and white coloured

QR codes. The conversion in this experiment is based on the 8-bit colour code for each monocoloured QR code. Nevertheless, it can be expanded into other large bit colours of the colour depth implemented. As a conclusion, when more bit colours of the white coloured QR code are applied, then more black and white coloured QR codes can be produced. The experiment results are shown at Tables 5.17 until 5.21 in Section 5.3.1.3 of Chapter 5. All black and white QR codes will be decoded into text that contains a non-meaningful text file. The last process is the decompression process and it involves decoding the text (non meaningful) to binary conversion (Base 64) and decompressing data using the compression tool. From the result, the total character that can be stored in the black and white QR code depends on the type of compression tool used. From the experiment, the GZip compression tool is suitable to implement the compression module as compared to Zip, LZW, and Huffman Coding, and the comparison among the compression tools is as shown at Table 4.3 in Section 4.1.2.1.2 of Chapter 4. The total black and white QR codes can be identified if the total bit colour depth and total channel of the colour model have been selected. As explained in this Section, the RO2 was successfully fulfill as stated in Section 1.4 of Chapter 2.

6.4 Partial Extraction Decode and Re-encode Design and Development

Since the encoding and decoding processes consumed throughput time, the other way to minimise the throughput time is by using partial extraction. The partial extraction process involves certain parts of the process in order to save processing time. The content of black and white and monocoloured QR codes can be done partially, which does not involve others not related to the QR code. The content management is divided

into two parts, which are level 1 for black and white QR code, and level 2 for monocoloured QR code as shown at Figure 4.9 in Section 4.3.1 of Chapter 4 .

The level 1 process only involves a single selected black and white QR code as shown at Figures 5.6 and 5.7 in Section 5.4.1 of Chapter 5. The processes start from multilayer until multiplexing processes. Then, the selected black and white QR code will be chosen. The decompression process only involves a single black and white QR code until the part of original text is produced. After the text is manipulated, the text needs to be re-encoded into a single QR code. The process starts with compressing and encoding the black and white QR code image. Since the previous black and white QR code images were not updated, they can be used again for the multiplexing process together with the updated QR Code image. The process continues with multiplexing and multilayer until the coloured QR code image is developed. The experiment shows that partial extraction consumes less throughput time as compared to the normal method.

The pseudocode for partial extraction level 1 at Figure 4.10 in Section 4.3.1 of Chapter 4 starts with justifying the index location of black and white QR code that needs to be extracted. The extraction process (decode) will execute along the path to the selected index location of black and white QR code. When the selected black and white QR code index location is successfully decoded and the information is updated, then the re-encoding process is ready to begin. The steps of re-encoding as at Figure 4.11 in Section 4.3.2 of Chapter 4 start with identifying the index location of black and white QR code and then compressing it. Afterwards, the steps are similar to the encoding processes, but they only run the modified data. The rest will use the uninterrupted data

as an input to execute the whole process until the end. This will reduce the elapsed time of processing as shown at Tables 5.22 and 5.23 in Section 5.4.1.2 of Chapter 5.

Level 2 is an extracting process of monocoloured QR code into multiple black and white QR codes. The benefit of level 2 is the text amount can be manipulated into a large amount as compared to level 1. This technique can be referred to at Figures 5.6 and 5.7 in Section 5.4.1 of Chapter 5. The demultilayer process will extract a coloured QR code into monocoloured QR codes. The selected monocoloured QR code will be used for extracting into black and white QR codes. Only the selected monocoloured QR code will proceed to the next step, which is the demultiplexing process. The demultiplexing process continues only with the selected monocoloured QR code until parts of the original text are produced. After the text is manipulated, the text is ready to be re-encoded. The compression will take part to generate a decoded binary to text. From that, the generation of QR code pattern can be done until it is ready for the multiplexing process. The multiplexing process will generate the monocoloured QR code based on the type of monocoloured QR code previously selected. The combination of the not updated version of monocoloured QR codes and updated monocoloured QR codes is used to generate the multilayer process just before the coloured QR code is generated.

The pseudocode of partial extraction level 2 involves the modification of monocoloured QR code as shown at Figure 4.12 in Section 4.3.3 of Chapter 4. The pseudocode starts with identifying the index location of monocoloured QR code. The total monocoloured QR code depends on the total of colour channel as shown at Tables 5.17 until 5.21 in Section 5.3.1.3 of Chapter 5. If the total of colour channel increases,

the total monocoloured QR code will also increase. This process is a normal decoding process, which has been discussed in Chapter 5. However, contrastly, not all monocoloured QR codes are processed, only selected monocoloured QR codes will be decoded and processed as shown at Figure 4.10 in Section 4.3.1 of Section 4. After the monocoloured QR code has been identified and decoded, the index location of black and white QR code in the monocoloured QR code must be calculated of its index location. The process is shown in lines 4 until 8 at Figure 4.12 in Section 4.3.3 of Chapter 4. When a set of black and coloured QR code are successfully produced from the previous process, they will be decoded into subcomplete not meaningful text data (text encoded from the Base64 decoder). Based on the algorithm procedure at Table 4.14 in Section 4.3.3 of Chapter 4 at index P16, the binary file will be produced after the Base64 decoder is successful implemented. Another decoding process using a decompression tool is the last compression module process to be performed until the actual data is produced. The actual data can be edited in terms of adding, updating or deleting information. After the data has been manipulated, the process to generate a complete coloured QR code can be created. This process is called the re-encoding process. The re-encoding process starts with the identification of the index location of black and white QR codes as shown at Figure 4.13 in Section 4.3.4 of Chapter 4. The compression and Base64 encoder are executed based on the updated data only at the compression module. This process is illustrated at Figure 5.7 in Section 5.4.1 of Chapter 5. The new black and white QR codes will be generated by using the updated data compression only.

The path to generate coloured QR codes continues with generating monocoloured QR codes by using a combination of new and remaining black and white QR codes as data

input (Table 4.15 in Section 4.3.4 of Chapter 4 at index P23). Moreover, the generation of coloured QR codes uses the new and remaining monocoloured QR codes after the new monocoloured QR code is generated. The whole process can be referred to at Table 4.15 in Section 4.3.4 of Chapter 4, starting from indices 6.19 until 6.24. From the result, the decoding process at level 2 took 2 seconds 745 milliseconds, whereas the normal process only took 1 second 464 milliseconds. This can be referred to at Table 5.24 in Section 5.4.1.2 of Chapter 5. The re-encoding process shows better processing time for re-encoding level 2, which is 2 seconds 184 milliseconds as compared to the benchmark technique with 6 seconds 246 milliseconds. Overall, the combination of decoding and re-encoding processes is better than the benchmark technique based on the processing time as shown at Table 5.26 in Section 5.4.1.2 of Chapter 5. The RO3 and RO4 have been performed as required in the Section 1.4 of Chapter 2.

6.5 Contribution

As a conclusion, the model used in extending data capacity in the QR code is successful. This is the main contribution of this research because the model is able to extend the data capacity of the QR code by a combination of compression, multiplexing, and multilayer modules. This model is expected to assist users who tend to increase the data capacity in a single QR code in terms of processing time efficiency and data capacity in various error correction levels. In addition, this model benefits the users in terms of reducing the number of QR codes.

The encoding and decoding algorithms follow the sequence of process model to obtain the coloured QR code and to regain the original text. From the encoding algorithm

process, colour depth and colour model help to increase the capacity of QR code. The more colour depth and colour model used, the more data capacity can be embedded through the coloured QR code. In the decoding process, the algorithm is able to prevent any missing character due to its precision in calculations of decoding.

From decoding and re-encoding partial extraction, the processes implemented parts of the full data extraction so as to save the throughput time. The process includes the extraction of a single and group of black and white QR codes. This model helps the data to be manipulated without involving other black and white QR codes. Partial extraction gives two options, which are level 1 (single) and level 2 (group) of black and white QR codes. Figure 6.1 illustrates the complete model of compression, multiplexing, and multilayer for coloured QR code.

6.5.1 The Model

The model in Figure 6.1 was developed based on a combination of encoding, decoding, and partial extraction methods. The methods were implemented first by developing the algorithms followed by the programme. Most of the researchers as shown at Tables 2.6 and 2.8 in Section 2.3.5 of Chapter 2 and Section 2.3.6 of Chapter 2 proposed several methods based on a single method of multilayer, multiplexing, and compression without combine them in a suitable order. In order to obtain more data capacity in the QR code with a comprehensively quick processing time, the combination of methods based on the researchers above has been proposed. This combination was tested and the model was built as a guideline. This model will also guide users to upgrade the sub-module. Table 6.1 shows the module and sub-module that can be upgraded and it is guided from the model given in Figure 6.1. For example,

in the compression module, users are able to change or upgrade the compression utility to another type of compression utility. If the input contains different types of ASCII characters such as Japanese or Arabic characters, they can use another encoder/decoder to reduce the storage capacity. Partial extraction can reduce the processing time as discussed in Chapter 6. The error correction level follows the current black and white QR code.

The method used in this model must be followed in sequence starting from compression until multilayer processes. The sequence cannot be changed as it will cause errors in gaining more data capacity. The model is divided into two parts, namely the current method and partial extraction method. The current method consists of the encoding and decoding processes as discussed in Chapters 4 and 5. Meanwhile, the partial extraction method that was discussed in Chapter 6 contains two levels. Level 1 involves the extraction of black and white QR codes, while level 2 extracts monocoloured QR codes. All processes contribute in producing coloured QR codes without any data lost.

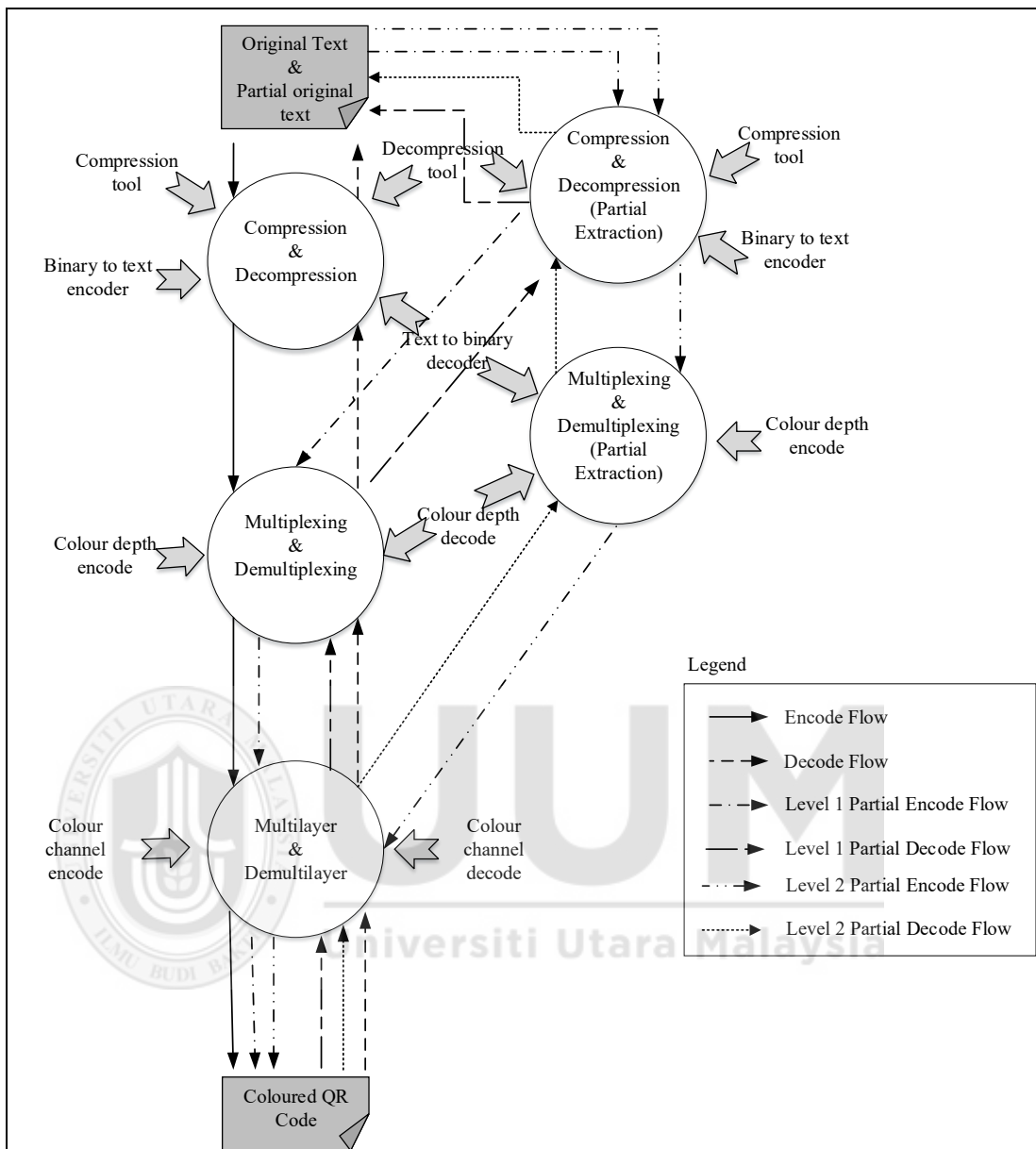


Figure 6.1. The complete model of compression, multiplexing, and multilayer for coloured QR code.

Table 6.1

The module and sub-module upgrading plan.

Module	Sub-Module	Current	Upgrade
Compression	Compression	GZip	Any compression utility
	Encoder/Decoder	Base64	Base91, Base128 etc. Depends on type of characters
Multiplexing	Multiplexing	24-bit colours	128-bit colour, 256-bit colours etc.
Multilayer	Multilayer	RGB	Any kind of extra colour channels in RGB

6.6 Limitation

The limitations of the research in this thesis are as follows:

1. In the compression process, the data needs to be compress and translated from binary to text files by using the encoder. The reason is the data can only be embedded into the black QR code with high storage capacity when the data is in a text mode as compared to a binary mode. It can cut the throughput time if the encoder discarded from this model. The black and white QR code is unable to receive a binary mode in a big amount of capacity as compared to text mode.
2. In the multiplexing process, the black and white QR codes only accept two types of colour, which are black and white. The model cannot accept more than two types of colour due to the restriction of 0 and 1 digit method. The 0 and 1 digits are used to create the monocoloured QR colour. If more than two colours are carried out, then the

binary digit inside the RGB colour cannot be implemented. This will give incompatible methods for this model.

3. In the multilayer process, the combination of layers are based on the combination of monocoloured QR codes that are generated from the multiplexing process. When the coloured QR codes are generated, the model cannot extend the combination layer with other coloured QR codes. If the combination occurs, the data capacity can be extended more than two times.

6.7 Future Work

The possible recommendations for future work are as follows:

1. The implementation of parallel processing could be implemented from this model, especially in the partial extraction module. The reason of using the parallel processing is to speed up the throughput time. The decoding and re-encoding processes can run together and stop for a while at the black and white QR code image. Figure 6.2 shows the example of method implementation of parallel processing for partial extraction level 1 with 24 bit RGB.

2. The combination with the same model can contribute to extra data capacity of coloured QR code. This method can be performed if two or more complete coloured QR codes are combined and as a result, it will give another output image of the combination. The technique that may be used is the multilayer process, which combines the images. If the combination is successful, the data capacity can increase up to two times or more from the current storage capacity. Figure 6.3 illustrates the combination of two coloured QR codes. This combination method can be enhanced

with embedding more coloured QR codes and produce a single new invention of QR code.

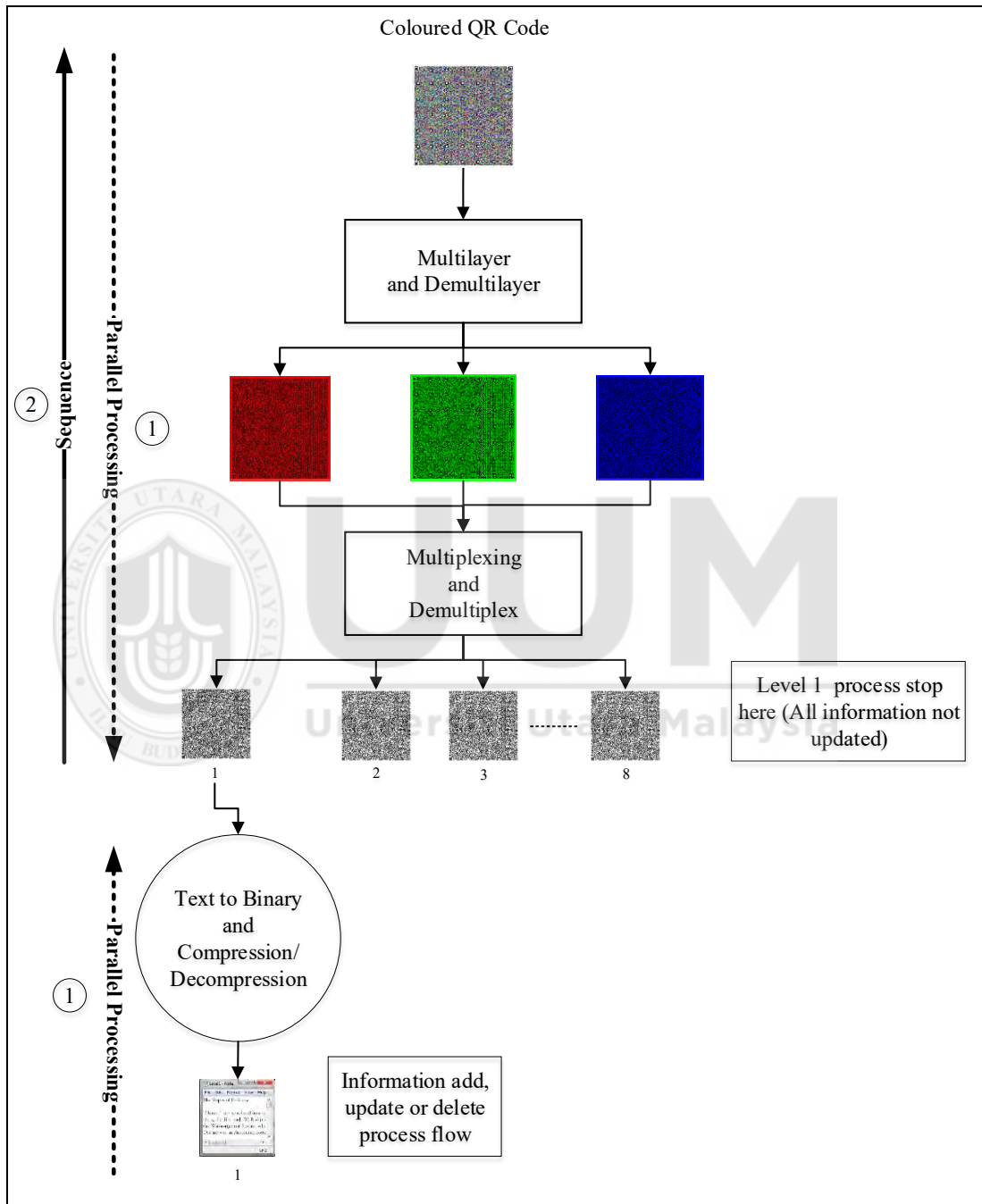


Figure 6.2. The example of method implementation of parallel processing for partial extraction level 1.

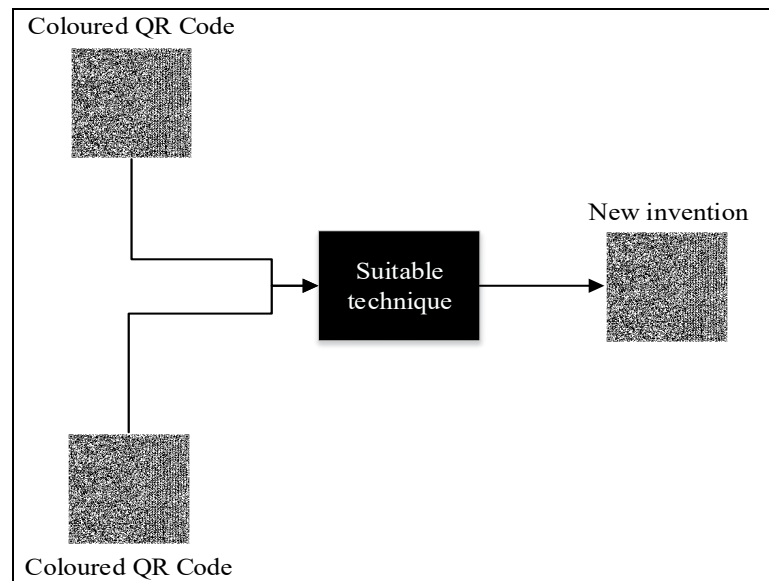


Figure 6.3. The combination of two coloured QR codes.

3. This research can be enhanced by considering the brightness and darkness aspects during the decoding process on the image of the coloured QR code. The brightness and darkness aspects of the QR code image need to be considered because the colour of the image will change due to the effect of light (Yang et al., 2016). The technique must show how to convert to the normal colours if there is light during the decoding process. The model cannot generate the original text if the colour combination does not tally with the actual colour of the QR Code during the encoding process. Figure 6.4 illustrates the effect of light during the decoding process. The research can contribute to how to solve the light effect problem during the decoding process of coloured QR code.

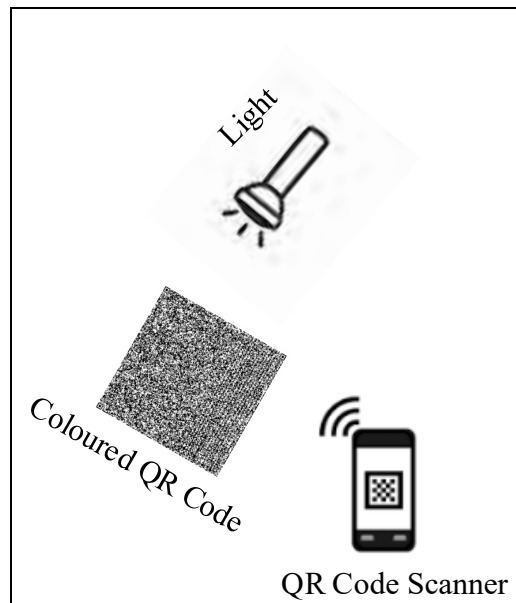


Figure 6.4. The effect of light during decoding process.

6.8 Summary

The coloured QR code model is a method to increase data storage in order to have more data embedded in a single QR code. This model requires a set of test data or input to perform a dynamic experiment on data storage in the QR code. Thus, the data helps to prove the model developed whether it can be used to increase the data in QR code. Various automated methods were used such as throughput time, total characters, and error correction level. Thus, the research in this thesis proposed a model to derive and generate a large data capacity to perform a text-based information by using compression, multiplexing, and multilayer techniques. Having to furnish the text-based data that conforms to the specifications of a solution model created, this model gives positive results of data capacity by using this criteria. The model is able to increase data storage of the QR code, in terms of using the compression, multiplexing, and multilayer techniques. This model is expected to give hope for those who are

interested in increasing the data in a single QR code. This feature benefit can reduce time constraint due to less throughput time and reduce the use of large amounts of QR codes.

Based on the findings collected from the conducted experiment as discussed earlier, it significantly proves that this model has increased the data capacity as compared to what was employed in the current research of QR code. In addition, supported by the findings from the partial extraction technique, it is shown that the content of the QR code can be customised in a faster way as compared to the traditional technique. The present study also shows that the proposed model is able to use other input such as image, other type of characters, and public/private key. It is feasible to be adopted in the process control industry, product description, offline e-book etc.

All objectives from this thesis have been fulfilled and all the research questions have been answered through this research. As this research is fully completed, this model that produces coloured QR codes is ready to be used by the QR code industry to save storage capacity in the future.

REFERENCES

- Abas, A., Yusof, Y., & Ahmad, F. K. (2017). Expanding the data capacity of QR codes using multiple compression algorithms and base64 encode/decode. *Journal of Telecommunication, Electronic and Computer Engineering*, 9(2–2).
- Ahlawat, S., & Rana, C. (2017). A Review on QR Codes : Colored and Image Embedded. *International Journal of Advanced Research in Computer Science*, 8(5), 410–413.
- Anonymous. (2013). PM-Code's world. Retrieved July 7, 2018, from <http://pmcode.co-site.jp/>
- Asare, I. T., & Asare, D. (2015). The Effective Use of Quick Response (QR) Code as a Marketing Tool. *International Journal of Education and Social Science*, 2(12), 67–73.
- Bagherinia, H., & Manduchi, R. (2012). High Information Rate and Efficient Color Barcode Decoding. In *International Workshop on Color and Photometry in Computer Vision (CPVC)* (pp. 1–10).
- Bhardwaj, N., Kumar, R., Verma, R., Jindal, A., & Bhondekar, A. P. (2016). Decoding algorithm for color QR code: A mobile scanner application. In *2016 International Conference on Recent Trends in Information Technology, ICRTIT 2016*. <https://doi.org/10.1109/ICRTIT.2016.7569561>
- Bishop, T. (2007). Software notebook: Color is key to Microsoft's next-generation bar code. Retrieved July 7, 2018, from <https://www.microsoft.com/en-us/research/project/high-capacity-color-barcodes-hccb/>

- Blasinski, H., Bulan, O., & Sharma, G. (2013). Per-Colorant-Channel Color Barcodes for Mobile Applications : An Interference Cancellation. In *IEEE Transaction on Image processing* (Vol. 22, pp. 1498–1511).
- Boob, A., Shinde, A., Rathod, D., & Gaikwad, A. (2014). Qr Code Based Mobile App and Business Process Integration. *International Journal of Multidisciplinary and Current Research*, 2(Sept and Oct 2014), 1014–1017.
- British Standards. (2009). *Information technology : automatic identification and data capture techniques, QR code 2005 bar code symbology specification. ISO Standards*.
- Bulan, O., & Sharma, G. (2011a). High Capacity Color Barcodes : Per Channel Data Encoding via Orientation Modulation in. *IEEE Transactions on Image Processing*, 20(5), 1337–1350.
- Bulan, O., & Sharma, G. (2011b). *High Capacity Data Embedding For Printed Documents*. University of Rochester.
- Bunma, D., & Vongpradhip, S. (2014). Using augment reality to increase capacity in QR code. In *4th International Conference on Digital Information and Communication Technology and Its Applications, DICTAP 2014* (pp. 440–443). <https://doi.org/10.1109/DICTAP.2014.6821727>
- Chandran, A. (2014). Review on Color Qr Codes : Decoding Challenges. *International Journal of Engineering Research & Technology (IJERT)*, 3(4), 848–851.
- Chang, J. H. (2014). An introduction to using QR codes in scholarly journals. *Science Editing*, 1(2), 113–117. <https://doi.org/10.6087/kcse.2014.1.113>

- Charoensiriwath, C., Surasvadi, N., Pongnumkul, S., & Pholprasit, T. (2015). Applying QR code and mobile application to improve service process in Thai hospital. In *Proceedings of the 2015 12th International Joint Conference on Computer Science and Software Engineering, JCSSE 2015* (pp. 114–119). <https://doi.org/10.1109/JCSSE.2015.7219781>
- Chiang, J. S., Li, H. T., Hsia, C. H., Wu, P. H., & Hsieh, C. F. (2013). High density QR code with multi-view scheme. In *Proceedings of the International Symposium on Consumer Electronics, ISCE* (Vol. 49, pp. 49–50). <https://doi.org/10.1109/ISCE.2013.6570246>
- Chuang, J.-C., Hu, Y.-C., & Ko, H.-J. (2010). A Novel Secret Sharing Technique Using QR Code. *International Journal of Image Processing (IJIP)*, 4(5), 468–475.
- Coleman, J. (2011). QR Codes: What Are They and Why Should You Care? In *Kansas Library Association College and University Libraries Section Proceedings* (Vol. 1, pp. 16–23). <https://doi.org/10.4148/culs.v1i0.1355>
- Commission, & International Organization for Standardization., I. E. (2000). *Information technology -- automatic identification and data capture techniques -- bar code symbology -- QR code*. Geneva: Geneva : ISO : IEC, 2000.
- Convert Words to Pages. (2016). Retrieved May 7, 2017, from <http://www.wordstopages.com/>
- Čović, Z., & Šimon, J. (2016). Usage of QR codes in promotion on social networks. In *Proceedings of 2016 International Conference on Smart Systems and Technologies, SST 2016* (pp. 123–127). <https://doi.org/10.1109/SST.2016.7765645>

- Dagan, I., Binyamin, G., & Eilam, A. (2016). Delivery of QR Codes to Cellular Phones through Data Embedding in Audio. In *ISCEE International Conference on the Science of Electrical Engineering* (pp. 3–6).
- Demir, S., Kaynak, R., & Demir, K. A. (2015). Usage Level and Future Intent of Use of Quick Response (QR) Codes for Mobile Marketing among College Students in Turkey. *Procedia - Social and Behavioral Sciences*, *181*, 405–413.
<https://doi.org/10.1016/j.sbspro.2015.04.903>
- Denso-Wave. (2015). QR Code Standardization. Retrieved July 7, 2018, from <http://www.qrcode.com/en/about/standards.html>
- Denso, A. (2011). *Qr code essentials*. Retrieved from <http://www.nacs.org/LinkClick.aspx>. Retrieved from <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:QR+Code+®+Essentials#0>
- Denso Wave. (2014). Denso Wave, The Inventor of QR Code. Retrieved July 7, 2018, from <https://www.denso-autoid-eu.com/en/about-us/20-years-qr-code.html>
- Dita, I., Otesteanu, M., & Quint, F. (2011). Data Matrix Code - A Reliable Optical Identification of Microelectronic Components. *2011 IEEE 17th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, *1*(1), 39–44.
- Donald D. Hearn, M. Pauline Baker, W. C. (2010). *Computer Graphics with Open GL (4th Edition)* (4th ed.). Pearson.

- Donoho, D. L., Vetterli, M., Devore, R. A., & Daubechies, I. (1998). Data Compression and Harmonic Analysis. In *IEEE Transaction on Information Theory* (Vol. 44, pp. 2435–2476).
- Falas, T., & Kashani, H. (1994). Two-Dimensional Bar-code Decoding with Camera-Equipped Mobile Phones. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference* (pp. 2–5).
- Farizshah, M., & Abd Jalil, K. (2012). The Embedding of Arabic Characters in QR Code. In *International Conference Open Systems (ICOS), 2012*.
- Feng, X., & Zheng, H. (2010a). Design and realization of 2D color barcode with high compression ratio. In *International Conference on Computer Design and Applications, ICCDA 2010* (Vol. 1, pp. 314–317).
<https://doi.org/10.1109/ICCDA.2010.5540872>
- Feng, X., & Zheng, H. (2010b). Design and realization of 2D color barcode with high compression ratio. In *International Conference on Computer Design and Applications, ICCDA 2010* (Vol. 1, p. 4).
<https://doi.org/10.1109/ICCDA.2010.5540872>
- Ferreira, R. a. S., & André, P. S. (2014). Colour multiplexing of quick-response (QR) codes. *Electronics Letters*, 50(24), 1828–1830.
<https://doi.org/10.1049/el.2014.2501>
- Fried, I. (2007, July 16). Microsoft gives bar codes a splash of color. *ZDNet*. Retrieved from <https://www.zdnet.com/article/microsoft-gives-bar-codes-a-splash-of-color/>

- Frost, C., Mammarella, M., Kohler, E., Reyes, A. D. L., Hovsepian, S., Matsuoka, A., & Zhang, L. (2007). Generalized File System Dependencies. *SOSP'07, 1*, 14.
- Fukuchi, K. (2010). Libqrencode, a c library for encoding data in a qr code symbol. Retrieved July 7, 2018, from <https://fukuchi.org/works/qrencode/>
- Galiyawala, H. J., & Pandya, K. H. (2014). To Increase Data Capacity of QR Code Using Multiplexing with Color Coding : An example of Embedding Speech Signal in QR Code. In *2014 Annual IEEE India Conference (INDICON)* (pp. 2–7).
- Galiyawala, H. J., & Pandya, K. H. (2015). To increase data capacity of QR code using multiplexing with color coding: An example of embedding speech signal in QR code. In *11th IEEE India Conference: Emerging Trends and Innovation in Technology, INDICON 2014* (pp. 2–7).
<https://doi.org/10.1109/INDICON.2014.7030441>
- Garateguy, G. J. (2014). *Optimal Embedding of QR codes into Color, Gray Scale and Binary Images*. University of Delaware. Retrieved from <http://udspace.udel.edu/handle/19716/13350>
- Garateguy, G. J., Member, S., Arce, G. R., Lau, D. L., Member, S., & Villarreal, O. P. (2014). QR Images : Optimized Image Embedding in QR Codes. *IEEE Transactions on Image Processing, 23*(7), 2842–2853.
- Gerstner, T., Decarlo, D., Alexa, M., Finkelstein, A., Gingold, Y., & Nealen, A. (2013). Pixelated image abstraction with integrated user constraints. *Computers and Graphics (Pergamon), 37*(5), 333–347.
<https://doi.org/10.1016/j.cag.2012.12.007>

- Goel, S., & Singh, A. K. (2014). Cost Minimization by QR Code Compression. *International Journal of Computer Trends and Technology (IJCTT)*, 15(4), 157–161.
- Grillo, A., Lentini, A., Querini, M., & Italiano, G. F. (2010). High capacity colored two dimensional codes. *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, 5(1), 709–716. <https://doi.org/10.1109/IMCSIT.2010.5679869>
- Gunawi, H. S., Prabhakaran, V., Krishnan, S., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2007). Improving File System Reliability with I / O Shepherding. In *21st ACM Symposium on Operating Systems Principles* (p. 14).
- Gutierrez, F., Abud, M. A., Vera, F., & Sanchez, J. A. (2013). Application of contextual QR codes to augmented reality technologies. In *23rd International Conference on Electronics, Communications and Computing, CONIELECOMP 2013* (pp. 264–269). <https://doi.org/10.1109/CONIELECOMP.2013.6525798>
- Guwalani, P., Kala, M., Chandrashekar, R., Shinde, J., & Mane, D. (2014). Image File Security using Base-64 Algorithm. *International Journal Computer Technology & Applications*, 5(6), 1892–1895.
- Hahn, H. I., & Joung, J. K. (2002). Implementation of Algorithm to Decode Two-Dimensional Barcode PDF-417. In *ICSP'02 Proceedings*.
- Hajduk, V., Broda, M., Kováč, O., & Levický, D. (2016). Image steganography with using QR code and cryptography. In *26th Conference Radioelektronika 2016* (pp. 350–353). <https://doi.org/10.1109/RADIOELEK.2016.7477370>
- Harish, N., & Gurav, S. (2014). Embedding a Large Information In QR Code Using Multiplexing Technique. *Taraksh Journal of Communications*, 1(6), 6–9.

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Ullah Khan, S. (2015). The rise of “big data” on cloud computing: Review and open research issues. *Information Systems*, 47, 98–115.
<https://doi.org/10.1016/j.is.2014.07.006>

He, D., Sun, Y., Jia, Z., Yu, X., Guo, W., He, W., ... Lu, X. (2010). A Proposal of Substitute for Base85 / 64 – Base91. In *The SUMMER 8th International Conference on Computing, Communications and Control Technologies 2010* (Vol. 85, pp. 4–7).

Husain, A., Bakhtiari, M., & Zainal, A. (2014). Printed document integrity verification using barcode. *Jurnal Teknologi (Sciences and Engineering)*, 70(1), 99–106. <https://doi.org/10.11113/jt.v70.2857>

Intermec Technologies Corporation. (2007, November). The 2D Revolution How evolving business needs and improved technology are driving explosive growth in two-dimensional bar coding, 2. Retrieved from <https://www.technologynetworks.com/tn/go/lc/view-white-paper-230951>

Jahagirdar, K. S., & Borse, S. B. (2015). QR Code with Colored Image. *International Journal of Computer Applications*, 115(16), 38–41.

Jancke, G. (2015). High Capacity Color Barcodes (HCCB). Retrieved July 28, 2018, from <https://www.microsoft.com/en-us/research/project/high-capacity-color-barcodes-hccb/>

Jennifer Farley. (2010). A Short Guide To Color Models. *SitePoint*, 1. Retrieved from <https://www.sitepoint.com/a-short-guide-to-color-models/>

- Kajaree, D., & Behera, R. . (2017). A Survey on Machine Learning: Concept, Algorithms and Applications. *International Journal of Innovative Research in Computer and Communication Engineering*, 5(2), 1302–1309.
<https://doi.org/10.15680/IJIRCCE.2017>.
- Kan, T., Teng, C.-H., & Chou, W.-S. (2009). Applying QR code in augmented reality applications. In *Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry - VRCAI '09* (Vol. 1, p. 253). <https://doi.org/10.1145/1670252.1670305>
- Kato, H., & Tan, K. T. (2005). 2D barcodes for mobile phones. In *Proceeding Mobile Technology, Applications and Systems, 2005 2nd International Conference* (p. 8).
- Kattan, A., & Poli, R. (2008). Evolutionary Lossless Compression with GP-ZIP. In *IEEE Congress on Evolutionary Computation, CEC 2008* (pp. 335–364).
- Kieseberg, P., Leithner, M., Mulazzani, M., Munroe, L., Schrittwieser, S., Sinha, M., & Weippl, E. (2010). QR code security. In *Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia - MoMM '10* (p. 430). <https://doi.org/10.1145/1971519.1971593>
- Kim, H. M., Kim, W., & Cho, D. (2004). A New Color Transforming for RGB Coding. In *ICIP '04. 2004 International Conference* (pp. 107–110).
- Kingsley G. Morse Jr. (2005). Compression Tools Compared. Retrieved May 2, 2017, from <http://www.linuxjournal.com/article/8051?page=0,0>
- Kumar, K., Sharma, P., & Singh, A. K. (2012). Configuring the System to Share Internet from Single User to Multi-user with Single Internet Dongle. *International Journal of Soft Computing and Engineering (IJSCE)*, (4), 32–35.

- Kumaraguru, S., & Prof Dr D. S. Bormane. (2012). Identification of QR Code based on Pattern Recognition with Mobile Phones. *International Journal of Modern Engineering Research (IJMER)*, 2(5), 3544–3547.
- Li, M., Cao, P., Yu, L., Liuping, F., Chen, J., & Jing, W. (2017). The research of information hiding and extraction based on QR code positioning function. In *2nd IEEE International Conference on Computer and Communications, ICC 2016 - Proceedings* (pp. 589–593).
<https://doi.org/10.1109/CompComm.2016.7924769>
- Li, Z., Chen, Z., Sudarshan, M. S., & Yuanyuan, Z. (2004). C-Miner: Mining Block Correlations in Storage Systems. In *Proceedings of the Third USENIX Conference on File and Storage Technologies* (p. 14).
- Liao Zhao-lai, Huang Ting-lei, Wang Rui, Z. X. (2010). A Method of Image Analysis for QR Code Recognition. In *2010 International Conference on Intelligent Computing and Integrated Systems* (pp. 250–253).
<https://doi.org/10.1109/ICISS.2010.5657187>
- Lin, J., & Fuh, C. (2013). 2D Barcode Image Decoding. *The Scientific World Journal*, 2013(3), 1.
- Liu, Y., Yang, J., & Liu, M. (2008). Recognition of QR Code with mobile phones. *2008 Chinese Control and Decision Conference*, 203–206.
<https://doi.org/10.1109/CCDC.2008.4597299>
- Liu, Z., Zheng, H., & Jia, H. (2009). Design and implementation of color two-dimension barcode with high compression ratio for Chinese characters. In *Proceedings - 2009 International Conference on Information Engineering and Computer Science, ICIECS 2009*.
<https://doi.org/10.1109/ICIECS.2009.5363553>

- Luo, M., Wang, S., & Lin, P. Y. (2016). QR code steganography mechanism with high capacity. In *2016 International Conference On Communication Problem-Solving (ICCP)* (pp. 1–2). <https://doi.org/10.1109/ICCPS.2016.7751131>
- Lyons, S. (2009). *Two-Dimensional Barcodes for Mobile Phones*. University of Toronto.
- Magadam, B. (2017). Data security in QR code using Steganography. *International Journal of Innovative Research in Computer and Communication Engineering*, 5(5), 10058–10063.
- Majumdar, S., Maiti, A., Bhattacharyya, B., & Nath, A. (2015). A new encrypted Data hiding algorithm inside a QR Code TM implemented for an Android Smartphone system : S _ QR algorithm Introduction : *International Journal of Innovative Research in Advanced Engineering (IJIRAE)*, 2(4), 40–46.
- Marktscheffel, T., Gottschlich, W., Popp, W., Werli, P., Fink, S. D., Bilzhause, A., & Meer, H. de. (2016). QR code based mutual authentication protocol for Internet of Things. In *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)* (pp. 1–6). <https://doi.org/10.1109/WoWMoM.2016.7523562>
- Melgar, M. E. V., & Santander, L. M. (2016). Channel capacity analysis of 2D barcodes: QR Code and CQR Code-5. In *2016 IEEE Colombian Conference on Communications and Computing, COLCOM 2016 - Conference Proceedings* (Vol. 1). <https://doi.org/10.1109/ColComCon.2016.7516376>
- Meyer, D. T., Aggarwal, G., Cully, B., Lefebvre, G., Feeley, M. J., Hutchinson, N. C., & Warfield, A. (2008). Parallax : Virtual Disks for Virtual Machines. In *EuroSys08* (p. 44).

- Nandhini. (2017). Performance Evaluation of Embedded Color QR Code on Logos. In *2017 Third International Conference On Science Technology Engineering and Management (ICONSTEM)* (pp. 1009–1014).
- Narayanan, D., Donnelly, A., & Rowstron, A. (2008). Write Off-Loading : Practical Power Management for Enterprise Storage. *ACM Transactions on Storage (TOS)*, 4(3), 15.
- Nikolaos, T., & Kiyoshi, T. (2010). QR-Code Calibration for Mobile Augmented Reality Applications. In *SIGGRAPH 2010* (p. 4503).
- Nurwono, K., & Kosala, R. (2009). Color quick response code for mobile content distribution. In *Proceedings of MoMM2009* (pp. 267–271). Retrieved from <http://dl.acm.org/citation.cfm?id=1821799>
- Okazaki, S., Navarro, a, & Campo, S. (2013). Cross-media integration of Qr code: a preliminary exploration. *Journal of Electronic Commerce ...*, 14(2), 137–148. Retrieved from <http://csulb.edu/journals/jecr/issues/20132/paper1.pdf>
- Online Reference and Tool. (2012). Retrieved July 7, 2018, from http://www.rapidtables.com/web/color/RGB_Color.htm
- Oswal, S., Singh, A., & Kumari, K. (2016). Deflate compression algorithm. *International Journal of Engineering Research and General Science* 2016, 4(1), 430–436.
- Pandya, K. H., & Galiyawala, H. J. (2014). A Survey on QR Codes : in context of Research and Application. *International Journal of Emerging Technology and Advanced Engineering*, 4(3), 258–262.

- Parikh, D., & Jancke, G. (2008). Localization and Segmentation of A 2D High Capacity Color Barcode Microsoft ' s HCCB Barcode localization. In *Applications of Computer Vision, 2008. WACV 2008. IEEE Workshop on* (Vol. 1, p. 6).
- Pascale, D. (2003). *A Review of RGB Color Spaces*. (D. Pascale, Ed.) (1st ed.). Montreal, Canada: The BabelColor Company.
- Pathak, S., Singh, S., Singh, S., Jain, M., Sharma, A., & 5. (2011). Data Compression Scheme of Dynamic Huffman Code for Different Languages. *International Conference on Information and Network Technology 2011*, 4(September 2010), 201–206.
- Pillai, P. N., & Naresh, K. (2014). Improving the Capacity of QR Code by Using Color Technique. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 3(7), 10561–10568. <https://doi.org/10.15662/ijareeie.2014.0307025>
- Prakash, R., & Singh, S. (2010). Improving RAID Performance and Reliability with Non-volatile Write Journals. *CYPRESS Perform*, (March), 1–5.
- Purcaru, E., & Roma, C. (2011). 2D Barcode for DNA Encoding. *Journal of Mobile, Embedded and Distributed Systems*, 3(3), 142–153. Retrieved from <http://jmeds.eu/index.php/jmeds/article/view/2D-Barcode-for-DNA-Encoding/pdf>
- Qianyu, J. (2014). *Exploring the Concept of QR Code and the Benefits of Using QR Code for Companies*. Lapland University of Applied Sciences.
- Rajasekar, S., Philominathan, P., & Chinnathambi, V. (2013). *Research methodology. Research Methodology*.

- Ramya, M., & Jayasheela, M. (2014). Improved Color QR Codes for Real Time Applications with High Embedding Capacity. *International Journal of Computer Applications*, 91(8), 8–12.
- Rathod Rinkalkumar, M. (2014). A Review on 1D & 2D Barcode with QR Code Basic Structure and Characteristics. *International Journal of Futuristic Trends in Engineering and Technology Vol. 4 (01), 2014, 4(1)*, 4–7.
- Rawat, D., Sahu, R., & Puthran, Y. (2015). Optimizing the Capacity of QR Code To Store Encrypted Image. *International Journal of Emerging Trends in Engineering Research (IJETER)*, 3(1), 1–4.
- Richard L. Villars, Olofson, C. W., & Eastwood, M. (2011). *Big Data: What it is and Why you should care*. IDC. MA, USA.
- Rouse, M. (2015). Multiplexing. Retrieved July 28, 2018, from <https://searchnetworking.techtarget.com/definition/multiplexing>
- Rungraungsilp, S., Ketcham, M., Wiputtikul, T., & Phonphak, K. (2012). Data Hiding Method for QR Code Based on Watermark by comparing DFT with DWT Domain. *International Conference on Computer and Communication Technologies (ICCCT'2012)*, 1(1), 154–158.
- Ryu, J. (2015). *The Modernization of the QR Code through Color and Brightness Level*. *John J. Ryu's Science Research*. Indiana. Retrieved from <https://joonuryuscience.com/2015/06/05/the-modernization-of-the-qr-code-through-color-and-brightness-levels/>

- Sangkwon, H., Hyung, J. B., Junhoi, K., Sunghwan, S., Sunghoon, K., & Wook. (2012). Drug Authentication Using High Capacity and Error Correctable Encoded Microtaggants. In *The 16th International Conference on Miniaturized Systems for Chemistry and Life Sciences* (pp. 1429–1431).
- Sarkar, S., Pu, L., Wu, H. C., Huang, S. C. H., & Wu, Y. (2017). New multimedia archiving technique using multiple quick-response codes. In *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB* (pp. 4–9). <https://doi.org/10.1109/BMSB.2017.7986236>
- Shahbahrami, A., Bahrampour, R., Rostami, M. S., & Mostafa Ayoubi. (2011). Evaluation of Huffman and Arithmetic Algorithms for Multimedia Compression Standards. *International Journal of Computer Science, Engineering and Applications*, 1(4), 34–47. <https://doi.org/10.5121/ijcsea.2011.1404>
- Sharma, M. (2010). Compression Using Huffman Coding. *IJCSNS International Journal of Computer Science and Network Security*, 10(5, (May), 133–141.
- Sharma, S., & Sejwar, V. (2016). QR code steganography for multiple image and text hiding using improved RSA-3DWT algorithm. *International Journal of Security and Its Applications*, 10(7), 393–406. <https://doi.org/10.14257/ijisia.2016.10.7.35>
- Shen, S., Lu, X., Qi, H., & Jiang, X. (2013). A Robust QR Code Extraction Algorithm. In Z. Wen & T. Li (Eds.), *Eighth International Conference on Intelligent Systems and Knowledge Engineering* (Vol. 2013, pp. 475–484). Springer Heidelberg New York Dordrecht London.
- Shen, S., Lu, X., Qi, H., & Jiang, X. (2014). A Robust QR Code Extraction Algorithm. *Advances in Intelligent Systems and Computing*, 1, 475–484. <https://doi.org/10.1007/978-3-642-54924-3>

- Shiang-yen, T. A. N., Foo, L. Y., & Idrus, R. (2010). Application of Quick Response (QR) Codes in Mobile Tagging System for Retrieving Information about Genetically Modified Food. *Proceedings of the 9th WSEAS International Conference on Data Networks, Communications, Computers, DNCOCO*, (June 2015), 114–118. <https://doi.org/978-960-474-245>
- Singh, A., Verma, V., & Raj, G. (2017). A Novel Approach for Encoding and Decoding of High Storage Capacity Color QR Code. In *2017 7th International Conference on Cloud Computing, Data Science & Engineering* (pp. 425–430).
- Skawattananon, C., & Vongpradhip, S. (2013). An improved method to embed larger image in QR code. In *Proceedings of the 2013 10th International Joint Conference on Computer Science and Software Engineering, JCSSE 2013* (pp. 64–69). <https://doi.org/10.1109/JCSSE.2013.6567321>
- Stinner, V. (2017). *Programming with Unicode Documentation Release 2011* (1st ed.). Sphinx. Retrieved from <https://media.readthedocs.org/pdf/unicodebook/latest/unicodebook.pdf>
- Subpratatsavee, P., & Kuacharoen, P. (2012). An Implementation of a High Capacity 2D Barcode. *Advances in Information Technology Communications in Computer and Information Science*, 344, 159–169.
- Sun, M., Fang, Z., Fu, L., & Zhao, F. (2009). Identification of QR Codes Based on Pattern Recognition. In *Computer and Computing Technologies in Agriculture-II--Proceedings of the Third IFIP International Conference on Computer and Computing Technologies in Agriculture(CCTA 2009)* (pp. 397–401).
- Sundaram, V., Wood, T., & Prashant, S. (2006). Efficient Data Migration in Self-managing Storage Systems. In *Autonomic Computing, 2006. ICAC '06. IEEE International Conference* (p. 4).

- Süsstrunk, S., Buckley, R., & Swen, S. (1999). Standard RGB Color Spaces. In *IS&T/SID 7th Color Imaging Conference* (pp. 1–8).
- Sutheebanjard, P., & Premchaiswadi, W. (2010). QR Code Generator. In *Eighth International Conference on ICT and Knowledge Engineering* (pp. 89–92).
- Szövetség, M. A., & Várallyai, L. (2012). From barcode to QR code applications. *Szám Journal of Agricultural Informatics*, 3(2), 9–17. Retrieved from <http://www.magisz.org/journal>
- Tank, A. H., Unde, M. M., Patel, B. J., & Raskar, P. (2016). Storage and transmission of information using grey level QR (quick-response) code structure. In *Conference on Advances in Signal Processing, CASP 2016* (pp. 402–405). <https://doi.org/10.1109/CASP.2016.7746204>
- Taveerad, N., & Vongpradhip, S. (2016). Development of Color QR Code for Increasing Capacity. *Proceedings - 11th International Conference on Signal-Image Technology and Internet-Based Systems, SITIS 2015*, 645–648. <https://doi.org/10.1109/SITIS.2015.42>
- Thomas, A., & Paul, R. (2013). An Effective Method for Removing Scratches and Restoring Low -Quality QR Code Images. *International Journal for Advance Research in Engineering and Technology Wind to Your Thoughts*, 1(V), 5–9.
- Tiwari, S. (2017). An introduction to QR code technology. In *15th International Conference on Information Technology, ICIT 2016* (Vol. 1, pp. 39–44). <https://doi.org/10.1109/ICIT.2016.38>
- Toh, S. R., Goh, W., & Yeo, C. K. (2016). Data exchange via multiplexed color QR codes on mobile devices. In *Wireless Telecommunications Symposium* (Vol. 2016–May). <https://doi.org/10.1109/WTS.2016.7482035>

- Trochim, W. M. K. (2015). Research Methods: Knowledge base, *I*(1), 1–369.
Retrieved from
https://www.researchgate.net/publication/243783609_The_Research_Methods_Knowledge_Base
- Umaria, M. M., & Jethava, G. B. (2015). Enhancing Data Storage Capacity in Quick Response Code Using Multiplexing and Data Compression Technique. *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*, (1), 1091–1093. <https://doi.org/10.1109/CICN.2015.214>
- Victor, N. (2012). Enhancing the Data Capacity of QR Codes by Compressing the Data before Generation. *International Journal of Computer Applications (0975–8887)*, *60*(2), 17–21.
- Vizcarra Melgar, M. E., Zaghetto, A., Macchiavello, B., & Nascimento, A. C. A. (2012). CQR codes: Colored quick-response codes. In *IEEE International Conference on Consumer Electronics - Berlin, ICCE-Berlin* (Vol. 2401, pp. 321–325). <https://doi.org/10.1109/ICCE-Berlin.2012.6336526>
- Vongpradhip, S. (2013). Use Multiplexing to Increase Information in QR Code. In *The 8th International Conference on Computer Science & Education (ICCSE 2013)* (pp. 361–364).
- Wakahara, T., Yamamoto, N., & Ochi, H. (2010). Image processing of dotted picture in the QR code of cellular phone. In *Proceedings - International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2010* (pp. 454–458). <https://doi.org/10.1109/3PGCIC.2010.77>

- Wang, S., Yang, T., Li, J., Yao, B., & Zhang, Y. (2015). Does a QR code must be black and white? In *Proceedings of 2015 International Conference on Orange Technologies, ICOT 2015* (pp. 161–164).
<https://doi.org/10.1109/ICOT.2015.7498513>
- Warasart, M., & Kuacharoen, P. (2012). Paper-based Document Authentication using Digital Signature and QR Code. In *4th International Conference on Computer Engineering and Technology (ICCET 2012)*.
- Weeks, H. R. M., Arthur, R., & Sartor, L. J. (1999). Histogram specification of 24-bit color images in the color difference (C-Y) color space. *Journal of Electronic Imaging*, 8(3), 290–300.
- Winter, M. (2011). *Scan Me - Everybody's Guide to the Magical World of Qr Codes* (1st ed.). California: Westsong Publishing. Retrieved from <https://books.google.com/books?id=s5ZxqwwYKk8C&pgis=1>
- Yadav, S. H., & Dawande, P. N. A. (2016). A Survey on Image Embedding In QR Code. *International Journal of Advanced Research in Computer and Communication Engineering*, 5(4), 339–341.
<https://doi.org/10.17148/IJARCCE.2016.5486>
- Yang, Z., Cheng, Z., Loy, C. C., Lau, W. C., Li, C. M., & Li, G. (2016). Towards Robust Color Recovery For High Capacity Color QR Codes. In *2016 IEEE International Conference on Image Processing (ICIP)* (pp. 2866–2870).
- Yang, Z., Xu, H., Deng, J., Loy, C. C., & Lau, W. C. (2017). Robust and Fast Decoding of High-Capacity Color QR Codes for Mobile Applications. *IEEE Transactions on Image Processing*, PP(c), 1.
<https://doi.org/10.1109/TIP.2018.2855419>

- Yeap, K. H., Cheong, Y. K., Nisar, H., & Teh, P. C. (2014). A simple data storage system using QR code. In *2014 5th International Conference on Intelligent and Advanced Systems (ICIAS)* (pp. 1–5).
<https://doi.org/10.1109/ICIAS.2014.6869536>
- Zhang, M., Yao, D., & Zhou, Q. (2012). The Application and Design of QR Code in Scenic Spot's eTicketing System -A Case Study of Shenzhen Happy Valley. *International Journal of Science and Technology*, 2(12). Retrieved from <http://www.ejournalofsciences.org>
- Zhang, W., & Yang, T. (2015). An Improved Algorithm for QR Code Image Binarization. In *2014 International Conference on Virtual Reality and Visualization And Visualization* (pp. 154–159).
<https://doi.org/10.1109/ICVRV.2014.51>
- Zhang, Y., Gao, T., Li, D., & Lin, H. (2012). An Improved Binarization Algorithm of QR Code Image, (1), 2376–2379.
- Zhu, B., Li, K., & Patterson, H. (2008). Avoiding the Disk Bottleneck in the Data Domain Deduplication File System Challenges and Observations. In *FAST '08: 6th USENIX Conference on File and Storage Technologies* (pp. 269–282).

Appendix A : Result of Maximum Characters

Result of maximum total characters stored in QR code from 20 times tested at error correction level H.

No. of Test	Normal	Zip	GZip	LZW	Huffmann Coding	Huffman+GZip
1	1271	474	635	434	113	471
2	1271	471	638	434	112	466
3	1271	476	637	433	111	477
4	1271	472	636	436	114	474
5	1271	475	637	433	112	470
6	1271	473	635	438	112	473
7	1271	475	635	433	111	474
8	1271	473	641	438	111	472
9	1271	474	636	438	114	468
10	1271	474	638	439	113	470
11	1271	473	634	433	113	468
12	1271	473	637	438	111	474
13	1271	471	633	441	111	477
14	1271	471	634	433	111	472
15	1271	473	635	437	113	471
16	1271	469	636	440	111	471
17	1271	470	636	438	111	479
18	1271	469	633	437	113	471
19	1271	477	636	436	112	467
20	1271	478	632	433	109	467

Appendix B : Encode Level L

The algorithm of encoding process for error correction level L.

```
/* Algorithm Module Index E1 – Initialisation */  
/*-----*/
```

```
/* Creating the package to be used */  
package qrdecmm;
```

```
/* Initialisation java library */  
call import java.io.FileInputStream;  
call import java.io.FileOutputStream;  
call import it.sauronsoftware.base64.Base64;  
call import java.io.IOException;
```

```
/* Creating main class */  
public class QRCodeCMM {
```

```
/* Initialisation variables to be used */  
initialize static String plainTextFile = "D:/QR Code/QRCode/Journal3/levelL.txt";  
initialize static String gZipTextFile = "D:/QR Code/QRCode/Journal3/gZipTextFile.zip";  
initialize static String base91TextFile = "D:/QR  
Code/QRCode/Journal3/base91TextFile.b91";  
initialize static String filePath = "D:/QR Code/QRCode/Journal3/";  
initialize static String fileName = "fileNumber";  
initialize static String fileType = ".txt";  
initialize static String fileTypePNG = "png";  
initialize static String fileRGB = filePath + "fileRGB." + fileTypePNG;  
initialize static final int size = 551;
```

```
/* Creating main programme */  
public static void main(String[] args) {
```

```
/* Counting the total characters including line feed and carriage return */  
initialize object CounterLetters count = new CounterLetters();  
execute count.CountLetter(plainTextFile);
```

```
/* Algorithm Module Index E2 – Compression utility (GZip) */  
/*-----*/
```

```
initialize object GZip gZip = new GZip();  
execute gZip.gZipFile(plainTextFile, gZipTextFile);
```

```
/* Algorithm Module Index E3 – Encoder for Base64*/  
/*-----*/
```

```

initialize object base91cli base91 = new base91cli();
try {
initialize object FileInputStream ifs = new FileInputStream(gZipTextFile);
initialize object FileOutputStream ofs = new FileOutputStream(base91TextFile);
execute base91.encode(ifs, ofs);
} catch (Exception e) {
display error by using System.err.println(e);
}

```

```

/* Algorithm Module Index E4 – Compatibility QR Code ANSI to UTF */
/*-----*/

```

```

initialize object ansiToUTF8 utf8 = new ansiToUTF8();
execute utf8.convert(base91TextFile);

```

```

/* Algorithm Module Index E5 – Creating blank file */
/*-----*/

```

```

initialize object CreateQRCode1 create1 = new CreateQRCode1(size);
initialize object CreateQRCode9 create9 = new CreateQRCode9(size);
initialize object CreateQRCode17 create17 = new CreateQRCode17(size);

```

```

/* Counting characters */
initialize object CountChar countChar = new CountChar();
execute int countCharacter = countChar.count(base91TextFile);

```

```

/* Divide characters with related value and fit each of file */
initialize object DivideCharacters divide = new DivideCharacters();

```

```

/* Create blank files */
execute create1.createBlank40Files(filePath, fileName, fileType);

```

```

/* Algorithm Module Index E6 – Embedded text characters to each file */
/*-----*/

```

```

/* Embedded text characters to each blank file created */
execute int totalFiles = divide.divideCharacter(base91TextFile, countCharacter, filePath,
fileName);

```

```

/* Top up with blank files if not enough */
    if (totalFiles < 25) {
        totalFiles = 24;
    }

```

```

/* Specify three group contains 8 files each */
initialize int eight = 8;
initialize String[] multiColourLayerFail = {"QRRed", "QRGreen", "QRBlue"};
initialize int colourCombineRGB[][][] = new int[size][size][3];
initialize MultiLayerQRCode multiLayerQRCode = new MultiLayerQRCode();

```

```

/* Algorithm Module Index E7 and E8– Create black and white QR Code and
monocoloured QR Code. */
/*-----
-*/

/* Start with first group (red) */
if (totalFiles >= 0) {
    initialize int total8 = 8;

    /* Create black and white QR Code from 1 to 8 (Module Index E7)
execute create1.generateQRCodeVersion40(filePath, fileName, fileType,
fileTypePNG, total8);

    /* Create monocoloured QR Code group 1 (Module Index E8)
try {
    execute resultFinal = create1.readImage(filePath, fileName, fileTypePNG, total8);
} catch (IOException ex) {
}
execute int[][] plotResultBlackWhite =
create1.generateMultiplexQRCode(resultFinal, total8);

    /* Combine pixels among 8 black and white QR Codes */
for (int x = 0; x < size; x++) {
    for (int y = 0; y < size; y++) {
        execute colourCombineRGB[x][y][0] = plotResultBlackWhite[x][y];
    }
}

    /* Generate first monocoloured QR Code */
execute create1.generateQRCodeVersion40MonoColour(filePath, fileTypePNG,
plotResultBlackWhite, multiColourLayerFail[0]);
}

/* Start with second group (green) */
if (totalFiles >= 8) {
    initialize int total16 = 16;

    /* Create black and white QR Code from 9 to 16 (Module Index E7)
create9.generateQRCodeVersion40(filePath, fileName, fileType, fileTypePNG,
total16);
initialize int[][][] resultFinal = new int[eight][size][size];

    /* Create monocoloured QR Code group 2 (Module Index E8)
try {
    execute resultFinal = create9.readImage(filePath, fileName, fileTypePNG, total16,
eight);
} catch (IOException ex) {
}
}

```

```

    execute int[][] plotResultBlackWhite =
create9.generateMultiplexQRCode(resultFinal, total16);

    /* Combine pixels among 8 black and white QR Codes */
    for (int x = 0; x < size; x++) {
        for (int y = 0; y < size; y++) {
            execute colourCombineRGB[x][y][1] = plotResultBlackWhite[x][y];
        }
    }

    /* Generate second monocoloured QR Code */
    execute create9.generateQRCodeVersion40MonoColour(filePath, fileTypePNG,
plotResultBlackWhite, multiColourLayerFail[1]);
    }

    /* Start with third group (blue) */
    if (totalFiles >= 16) {
        initialize int total24 = 24;

        /* Create black and white QR Code from 17 to 24 (Module Index E7)
        execute create17.generateQRCodeVersion40(filePath, fileName, fileType,
fileTypePNG, total24);
        initialize int[][][] resultFinal = new int[eight][size][size];

        /* Create monocoloured QR Code group 3 (Module Index E8)
        try {
            execute resultFinal = create17.readImage(filePath, fileName, fileTypePNG,
total24, eight);
        } catch (IOException ex) {
        }

        execute int[][] plotResultBlackWhite =
create17.generateMultiplexQRCode(resultFinal, total24);

        /* Combine pixels among 8 black and white QR Codes */
        for (int x = 0; x < size; x++) {
            for (int y = 0; y < size; y++) {
                execute colourCombineRGB[x][y][2] = plotResultBlackWhite[x][y];
            }
        }

        /* Generate third monocoloured QR Code */
        execute create17.generateQRCodeVersion40MonoColour(filePath, fileTypePNG,
plotResultBlackWhite, multiColourLayerFail[2]);
    }

    /* Algorithm Module Index E9– Create coloured QR Code. */
    /*-----
    */

```

```
initialize CombineRGB combineRGBColour = new CombineRGB();  
execute combineRGBColour.combineRGB(colourCombineRGB, fileRGB);  
}  
}
```



Appendix C : Decode Level L

The algorithm of decoding process for error correction level L.

```
/* Algorithm Module Index D1 – Initialisation */
/*-----*/

/* Creating the package to be used */
package qrdecmmn;

/* Initialisation variables to be used */

call import com.google.zxing.NotFoundException;
call import com.google.zxing.WriterException;
call import java.io.FileInputStream;
call import java.io.FileOutputStream;
call import it.sauronsoftware.base64.Base64;
call import java.awt.Colour;
call import java.io.BufferedWriter;
call import java.io.File;
call import java.io.FileWriter;
call import java.io.IOException;

/* Creating main class */
public class QRCodeCMN {

/* Initialisation variables to be used */
initialize static String plainTextFile = "D:/QR Code/QRCode/Journal3/Decode/fileText.txt";
initialize static String gZipTextFile = "D:/QR
Code/QRCode/Journal3/Decode/gZipTextFile.gzip";
initialize static String base91TextFile = "D:/QR
Code/QRCode/Journal3/Decode/base91TextFile.b91";
initialize static String filePath = "D:/QR Code/QRCode/Journal3/Decode/";
initialize static String filePathBefore = "D:/QR Code/QRCode/Journal3/";
initialize static String fileName = "fileNumberMerged";
initialize static String fileType = ".txt";
initialize static String fileTypePNG = "png";
initialize static String fileRGB = filePath + "fileRGB." + fileTypePNG;
initialize static String fileRGBBefore = filePathBefore + "fileRGB." + fileTypePNG;
initialize static File fileRGBDecode = new File(fileRGBBefore);
initialize static String[] multiColourLayerFile = {"QRRedDecode", "QRGreenDecode",
"QRBlueDecode"};
initialize static String fileRed = filePath + multiColourLayerFile[0] + "." + fileTypePNG;
initialize static String fileGreen = filePath + multiColourLayerFile[1] + "." + fileTypePNG;
initialize static String fileBlue = filePath + multiColourLayerFile[2] + "." + fileTypePNG;
initialize static File fileRedDecode = new File(fileRed);
initialize static File fileGreenDecode = new File(fileGreen);
```

```

initialize static File fileBlueDecode = new File(fileBlue);
initialize static String fileQRCodeBlackWhite[] = {"QRCodeBlackWhiteRed",
"QRCodeBlackWhiteGreen", "QRCodeBlackWhiteBlue"};
initialize static String fileOutputTextDecode = filePath + fileName + fileType;
initialize static String plainTextFileDecode = "D:/QR
Code/QRCode/Journal3/Decode/plainTextFile.txt";
initialize static String gZipTextFileDecode = "D:/QR
Code/QRCode/Journal3/Decode/gZipTextFile.gzip";
initialize static String base91TextFileDecode = "D:/QR
Code/QRCode/Journal3/Decode/fileNumberMerged.txt";
initialize static long startdecodeMultilayer, stopdecodeMultilayer;
initialize static long startdecodeRedQRCode, startdecodeGreenQRCode,
startdecodeBlueQRCode;
initialize static long stopdecodeRedQRCode, stopdecodeGreenQRCode,
stopdecodeBlueQRCode;
initialize static long startdecodeRedBlackQRCode, startdecodeGreenBlackQRCode,
startdecodeBlueBlackQRCode;
initialize static long stopdecodeRedBlackQRCode, stopdecodeGreenBlackQRCode,
stopdecodeBlueBlackQRCode;
initialize static long startdecodeBlackQRCode, stopdecodeBlackQRCode;
initialize static long startdecodebase91, stopdecodebase91;
initialize static long startdecodeGUnzip, stopdecodeGUnzip;
initialize static long startdecodeAll, stopdecodeAll;

/* Creating main class */
public static void main(String[] args) {

initialize object DecodeColourQR decode = new DecodeColourQR();
initialize object DecodeQRCode QRCodeText = new DecodeQRCode();
initialize int size = 551;
initialize int files = 8;

/* Algorithm Module Index D2 – Demultilayer*
/*-----*/

/* Decode From Coloured To Red, Green, and Blue Monocoloured */
try {
    execute Colour[][] resultcolourQRCodeDecode =
decode.readImage(fileRGBDecode);
    execute decode.decodeMultiLayerQRCodeRGB1(resultcolourQRCodeDecode,
filePath, multiColourLayerFile);

/* Algorithm Module Index D3 – Demultiplexing*
/*-----*/

/* Initialize the information of Black and White QR Code */
initialize QRCodeBlackWhite = new int[files][size][size];

/* Demultiplexing Red QR Code */

```

```

execute Colour[][] resultcolourRedQRCodeDecode = decode.readImage(fileRedDecode);
execute QRCodeBlackWhite =
decode.decodeQRCodeBlackWhite(resultcolourRedQRCodeDecode, 0);
execute decode.decodeQRCodeBlackWhite1(QRCodeBlackWhite, filePath,
fileQRCodeBlackWhite[0]);

/* Demultiplexing Green QR Code */
execute Colour[][] resultcolourGreenQRCodeDecode =
decode.readImage(fileGreenDecode);
execute QRCodeBlackWhite =
decode.decodeQRCodeBlackWhite(resultcolourGreenQRCodeDecode, 1);
execute decode.decodeQRCodeBlackWhite1(QRCodeBlackWhite, filePath,
fileQRCodeBlackWhite[1]);

/* Demultiplexing Blue QR Code */
execute Colour[][] resultcolourBlueQRCodeDecode = decode.readImage(fileBlueDecode);
execute QRCodeBlackWhite =
decode.decodeQRCodeBlackWhite(resultcolourBlueQRCodeDecode, 2);
execute decode.decodeQRCodeBlackWhite1(QRCodeBlackWhite, filePath,
fileQRCodeBlackWhite[2]);
} catch (IOException ex) {
}

/* Algorithm Module Index D4 – Decode Black and White QR Code */
/*-----*/

/* Counting the total characters including line feed and carriage return */
initialize object CounterLetters count = new CounterLetters();
try {
initialize object BufferedWriter writer = null;
initialize and excute writer = new BufferedWriter(new FileWriter(fileOutputTextDecode));

/* Creating Naming Conversion for Black and White QR Code Image Name */
for (int i = 0; i < 24; i++) {
    initialize String filePathDecodeFinal = null;
    if ((i >= 0) && (i < 8)) {
        initialize filePathDecodeFinal = fileQRCodeBlackWhite[0] + i + "." +
fileTypePNG;

    } else if ((i >= 8) && (i < 16)) {
        initialize filePathDecodeFinal = fileQRCodeBlackWhite[1] + (i - 8) + "." +
fileTypePNG;

    } else if ((i >= 16) && (i < 24)) {
        initialize filePathDecodeFinal = fileQRCodeBlackWhite[2] + (i - 16) + "." +
fileTypePNG;
    }
}

initialize String filePathDecodeComplete = filePath + filePathDecodeFinal;

```



```

/* Decode Black and White QR Code */
initialize and excute String valueText =
QRCodeText.decodeTextQRCode(filePathDecodeComplete);

/* Write to Text File */
execute writer.write(valueText);

}
    writer.close();
} catch (WriterException ex) {
    ex.printStackTrace();
} catch (IOException ey) {
    ey.printStackTrace();
} catch (NotFoundException ez) {
    ez.printStackTrace();
}

/* Algorithm Module Index D5 – Decode Encoder/Decoder Text (Base64) to
Compression file */
/*-----*/

initialize Base64cli base64 = new base64cli();

try {
    initialize object FileInputStream ifs = new FileInputStream(base91TextFileDecode);
    initialize object FileOutputStream ofs = new FileOutputStream(gZipTextFileDecode);

    execute Base64.decode(ifs, ofs);
} catch (Exception e) {
System.err.println(e);
}

/* Algorithm Module Index D6 –Extraction Compression File to Text File */
/*-----*/

initialize object GZip gZip = new GZip();
execute gZip.gunzipIt(gZipTextFileDecode, plainTextFileDecode);
}
}

```

Appendix D : Partial Extraction (Decode) Level 1

The algorithm of partial extraction level 1 (decode) process for error correction level L.

```
/* Algorithm Module Index P1 – Initialisation */  
/*-----*/
```

```
/* Creating the package to be used */  
package qrdecmn;
```

```
/* Initialisation java library */  
call import com.google.zxing.NotFoundException;  
call import com.google.zxing.WriterException;  
call import java.io.FileInputStream;  
call import java.io.FileOutputStream;  
call import it.sauronsoftware.base64.Base64;  
call import java.awt.Colour;  
call import java.io.BufferedWriter;  
call import java.io.File;  
call import java.io.FileWriter;  
call import java.io.IOException;
```

```
/* Creating main class */  
public class QRCodeCMN {
```

```
initialize static String plainTextFile = "D:/QR Code/QRCode/Journal3/Decode/fileText.txt";  
initialize static String gZipTextFile = "D:/QR  
Code/QRCode/Journal3/Decode/gZipTextFile.gzip";  
initialize static String base91TextFile = "D:/QR  
Code/QRCode/Journal3/Decode/base91TextFile.b91";  
initialize static String filePath = "D:/QR Code/QRCode/Journal3/Decode/";  
initialize static String filePathBefore = "D:/QR Code/QRCode/Journal3/";  
initialize static String fileName = "fileNumberMerged";  
initialize static String fileType = ".txt";  
initialize static String fileTypePNG = "png";  
initialize static String fileRGB = filePath + "fileRGB." + fileTypePNG;  
initialize static String fileRGBBefore = filePathBefore + "fileRGB." + fileTypePNG;  
initialize static File fileRGBDecode = new File(fileRGBBefore);  
initialize static String[] multiColourLayerFile = {"QRRedDecode", "QRGreenDecode",  
"QRBlueDecode"};  
initialize static String fileRed = filePath + multiColourLayerFile[0] + "." + fileTypePNG;  
initialize static String fileGreen = filePath + multiColourLayerFile[1] + "." + fileTypePNG;  
initialize static String fileBlue = filePath + multiColourLayerFile[2] + "." + fileTypePNG;  
initialize static File fileRedDecode = new File(fileRed);  
initialize static File fileGreenDecode = new File(fileGreen);  
initialize static File fileBlueDecode = new File(fileBlue);
```

```

initialize static String fileQRCodeBlackWhite[] = {"QRCodeBlackWhiteRed",
"QRCodeBlackWhiteGreen", "QRCodeBlackWhiteBlue"};
initialize static String fileOutputTextDecode = filePath + fileName + fileType;
initialize static String plainTextFileDecode = "D:/QR
Code/QRCode/Journal3/Decode/plainTextFile.txt";
initialize static String gZipTextFileDecode = "D:/QR
Code/QRCode/Journal3/Decode/gZipTextFile.zip";
initialize static String base91TextFileDecode = "D:/QR
Code/QRCode/Journal3/Decode/fileNameMerged.txt";
initialize static int blackWhiteQRCode = 0;

```

```

/* Creating main programme */

```

```

public static void main(String[] args) {

```

```

initialize object DecodeColourQR decode = new DecodeColourQR();
initialize object DecodeQRCode QRCodeText = new DecodeQRCode();
initialize int size = 551;
initialize int files = 8;

```

```

/* Algorithm Module Index P2 – Extracting coloured QR Code (Demultilayer) */
/*-----*/

```

```

try {

```

```

execute Colour[][] resultcolourQRCodeDecode = decode.readImage(fileRGBDecode);
execute decode.decodeMultiLayerQRCodeRGB1(resultcolourQRCodeDecode, filePath,
multiColourLayerFile);

```

```

/* Algorithm Module Index D3 – Demultiplexing*/
/*-----*/

```

```

/* Initialize the information of Black and White QR Code */
initialize QRCodeBlackWhite = new int[files][size][size];

```

```

/* Demultiplexing Red QR Code */
execute Colour[][] resultcolourRedQRCodeDecode = decode.readImage(fileRedDecode);
execute QRCodeBlackWhite =
decode.decodeQRCodeBlackWhite(resultcolourRedQRCodeDecode, 0);
execute decode.decodeQRCodeBlackWhite1(QRCodeBlackWhite, filePath,
fileQRCodeBlackWhite[0]);
} catch (IOException ex) {
}

```

```

/* Algorithm Module Index P4 – Decode Black and White QR Code */
/*-----*/

```

```

/* Counting the total characters including line feed and carriage return */
initialize object CounterLetters count = new CounterLetters();

```

```

try {

```

```

initialize object BufferedWriter writer = null;
initialize and excute writer = new BufferedWriter(new FileWriter(fileOutputTextDecode));

/* Creating Single Naming Conversion for a Black and White QR Code Image Name */

initialize String filePathDecodeFinal = null;
execute int i = blackWhiteQRCode;
initialize filePathDecodeFinal = fileQRCodeBlackWhite[0] + i + "." + fileTypePNG;

initialize String filePathDecodeComplete = filePath + filePathDecodeFinal;

/* Decode Single Black and White QR Code */
initialize and execute String valueText =
QRCodeText.decodeTextQRCode(filePathDecodeComplete);

/* Write to Text File */
execute writer.write(valueText);

/* Close to Text File */
execute writer.close();

} catch (WriterException ex) {
    ex.printStackTrace();
} catch (IOException ey) {
    ey.printStackTrace();
} catch (NotFoundException ez) {
    ez.printStackTrace();
}

/* Algorithm Module Index P5 – Decode Encoder/Decoder Text (Base64) to
Compression file */
/*-----*/

initialize Base64cli base64 = new base64cli();

try {
    initialize object FileInputStream ifs = new FileInputStream(base91TextFileDecode);
    initialize object FileOutputStream ofs = new FileOutputStream(gZipTextFileDecode);

    execute Base64.decode(ifs, ofs);
} catch (Exception e) {
System.err.println(e);
}

/* Algorithm Module Index P6 –Extraction Compression File to Text File */
/*-----*/
initialize object GZip gZip = new GZip();
execute gZip.gunzipIt(gZipTextFileDecode, plainTextFileDecode);

/* Algorithm Module Index P7 – Open Text Application and Ready to Manipulate */
/*-----*/

```

execute "C:\windows\system32\notepad.exe"

}}}



Appendix E : Partial Extraction (Re-encode) Level 1

The algorithm of partial extraction level 1 (re-encode) process for error correction level L.

```
/* Algorithm Module Index P8 – Initialisation */
/*-----*/

/* Creating the package to be used */
package qrdecmm;

/* Initialisation java library */
call import java.io.FileInputStream;
call import java.io.FileOutputStream;
call import it.sauronsoftware.base64.Base64;
call import java.io.IOException;

/* Creating main class */
public class QRCodeCMM {

/* Initialisation variables to be used */
initialize static String plainTextFile = "D:/QR
Code/QRCode/Journal3/singleTextLevel1L.txt";
initialize static String gZipTextFile = "D:/QR Code/QRCode/Journal3/gZipTextFile.zip";
initialize static String base64TextFile = "D:/QR
Code/QRCode/Journal3/base64TextFile.b64";
initialize static String filePath = "D:/QR Code/QRCode/Journal3/";
initialize static String fileName = "fileNumber";
initialize static String fileType = ".txt";
initialize static String fileTypePNG = "png";
initialize static String fileRGB = filePath + "fileRGB." + fileTypePNG;
initialize static final int size = 551;

/* Creating main programme */
public static void main(String[] args) {

/* Counting the total characters including line feed and carriage return */
initialize object CounterLetters count = new CounterLetters();
execute count.CountLetter(plainTextFile);

/* Algorithm Module Index P9 – Compression utility (GZip) */
/*-----*/

initialize object GZip gZip = new GZip();
execute gZip.gZipFile(plainTextFile, gZipTextFile);

/* Algorithm Module Index P10 – Encoder for Base64*/
/*-----*/
```

```

initialize object base91cli base91 = new base91cli();
try {
initialize object FileInputStream ifs = new FileInputStream(gZipTextFile);
initialize object FileOutputStream ofs = new FileOutputStream(base64TextFile);
execute base91.encode(ifs, ofs);
} catch (Exception e) {
display error System.err.println(e);
}

/* Algorithm Module Index P11 – Compatibility QR Code ANSI to UTF */
/*-----*/

initialize object ansiToUTF8 utf8 = new ansiToUTF8();
execute utf8.convert(base64TextFile);

/* Algorithm Module Index P12 – Creating Black and White QR Code */
/*-----*/

initialize object CreateQRCode1 create1 = new CreateQRCode1(size);
initialize object CreateQRCode9 create9 = new CreateQRCode9(size);
initialize object CreateQRCode17 create17 = new CreateQRCode17(size);

/* Counting characters */
initialize object CountChar countChar = new CountChar();
execute int countCharacter = countChar.count(base64TextFile);

/* Divide characters with related value and fit each of file */
initialize object DivideCharacters divide = new DivideCharacters();

/* Create blank files */
execute create1.createBlank40Files(filePath, fileName, fileType);

/* Embedded text characters to each blank file created*/
execute int totalFiles = divide.divideCharacter(base64TextFile, countCharacter, filePath,
fileName);

/* Checking if the file exceed more than 1 */
    if (totalFiles >= 2) {
        System.exit();
    }

/* Specify three group that contains 8 files each */
initialize int eight = 8;
initialize String[] multiColourLayerFail = {"QRRed", "QRGreen", "QRBlue"};
initialize int colourCombineRGB[][][] = new int[size][size][3];
initialize MultiLayerQRCode multiLayerQRCode = new MultiLayerQRCode();

/* Algorithm Module Index P12 and P13– Create black and white QR Code and place it
in a group. */

```

```

/*-----
-*/

/* Start with first group (red) */
if (totalFiles >= 0) {
    initialize int total8 = 8;

    /* Create an updated single black and white QR Code (Module Index P12)
    execute create1.generateQRCodeVersion40(filePath, fileName, fileType,
fileTypePNG, total8);

    /* get information from QR Code group 1 (Module Index P13)
    try {
        execute resultFinal = create1.readImage(filePath, fileName, fileTypePNG, total8);
    } catch (IOException ex) {
    }
    execute int[][] plotResultBlackWhite =
create1.generateMultiplexQRCode(resultFinal, total8);

    /* Combine pixels among 8 black and white QR Codes */
    for (int x = 0; x < size; x++) {
        for (int y = 0; y < size; y++) {
            execute colourCombineRGB[x][y][0] = plotResultBlackWhite[x][y];
        }
    }

/* Algorithm Module Index P14 – Create monocoloured QR Code. */
/*-----
-*/
    execute create1.generateQRCodeVersion40MonoColour(filePath, fileTypePNG,
plotResultBlackWhite, multiColourLayerFail[0]);
    }
}

/* Algorithm Module Index P15– Create coloured QR Code. */
/*-----
-*/

    initialize CombineRGB combineRGBColour = new CombineRGB();
    execute combineRGBColour.combineRGB(colourCombineRGB, fileRGB);
}
}

```


Appendix F : Partial Extraction (Decode) Level 2

The algorithm of partial extraction level 2 (decode) process for error correction level L.

```
/* Algorithm Module Index P16 – Initialisation */
/*-----*/

/* Creating the package to be used */
package qrdecmmn;

/* Initialisation variables to be used */

call import com.google.zxing.NotFoundException;
call import com.google.zxing.WriterException;
call import java.io.FileInputStream;
call import java.io.FileOutputStream;
call import it.sauronsoftware.base64.Base64;
call import java.awt.Colour;
call import java.io.BufferedWriter;
call import java.io.File;
call import java.io.FileWriter;
call import java.io.IOException;

/* Creating main class */
public class QRCodeCMN {

/* Initialisation variables to be used */
initialize static String plainTextFile = "D:/QR Code/QRCode/Journal3/Decode/
singleTextLevel2L.txt ";
initialize static String gZipTextFile = "D:/QR
Code/QRCode/Journal3/Decode/gZipTextFile.gzip";
initialize static String base91TextFile = "D:/QR
Code/QRCode/Journal3/Decode/base91TextFile.b91";
initialize static String filePath = "D:/QR Code/QRCode/Journal3/Decode/";
initialize static String filePathBefore = "D:/QR Code/QRCode/Journal3/";
initialize static String fileName = "fileNumberMerged";
initialize static String fileType = ".txt";
initialize static String fileTypePNG = "png";
initialize static String fileRGB = filePath + "fileRGB." + fileTypePNG;
initialize static String fileRGBBefore = filePathBefore + "fileRGB." + fileTypePNG;
initialize static File fileRGBDecode = new File(fileRGBBefore);
initialize static String[] multiColourLayerFile = {"QRRedDecode", "QRGreenDecode",
"QRBlueDecode"};
initialize static String fileRed = filePath + multiColourLayerFile[0] + "." + fileTypePNG;
initialize static String fileGreen = filePath + multiColourLayerFile[1] + "." + fileTypePNG;
initialize static String fileBlue = filePath + multiColourLayerFile[2] + "." + fileTypePNG;
initialize static File fileRedDecode = new File(fileRed);
```

```

initialize static File fileGreenDecode = new File(fileGreen);
initialize static File fileBlueDecode = new File(fileBlue);
initialize static String fileQRCodeBlackWhite[] = {"QRCodeBlackWhiteRed",
"QRCodeBlackWhiteGreen", "QRCodeBlackWhiteBlue"};
initialize static String fileOutputTextDecode = filePath + fileName + fileType;
initialize static String plainTextFileDecode = "D:/QR
Code/QRCode/Journal3/Decode/plainTextFile.txt";
initialize static String gZipTextFileDecode = "D:/QR
Code/QRCode/Journal3/Decode/gZipTextFile.zip";
initialize static String base91TextFileDecode = "D:/QR
Code/QRCode/Journal3/Decode/fileNumberMerged.txt";

/* Creating main class */
public static void main(String[] args) {

initialize object DecodeColourQR decode = new DecodeColourQR();
initialize object DecodeQRCode QRCodeText = new DecodeQRCode();
initialize int size = 551;
initialize int files = 8;

/* Algorithm Module Index P17 – Demultilayer*/
/*-----*/

/* Decode From Coloured To Red, Green, and Blue Monocoloured */
try {
    execute Colour[][] resultcolourQRCodeDecode =
decode.readImage(fileRGBDecode);
    execute decode.decodeMultiLayerQRCodeRGB1(resultcolourQRCodeDecode,
filePath, multiColourLayerFile);

/* Algorithm Module Index P18 – Demultiplexing*/
/*-----*/

/* Initialize the information of Black and White QR Code */
initialize QRCodeBlackWhite = new int[files][size][size];

/* Demultiplexing Red QR Code */
execute Colour[][] resultcolourRedQRCodeDecode = decode.readImage(fileRedDecode);
execute QRCodeBlackWhite =
decode.decodeQRCodeBlackWhite(resultcolourRedQRCodeDecode, 0);
execute decode.decodeQRCodeBlackWhite1(QRCodeBlackWhite, filePath,
fileQRCodeBlackWhite[0]);

/* Algorithm Module Index P19 – Decode Selected Group of Black and White QR Code
*/
/*-----*/

/* Counting the total characters including line feed and carriage return */
initialize object CounterLetters count = new CounterLetters();
try {

```

```

initialize object BufferedWriter writer = null;
initialize and excute writer = new BufferedWriter(new FileWriter(fileOutputTextDecode));

/* Identify Index Location and Creating Naming Conversion of a Group Black and White QR Code Image Names */
for (int i = 0; i < 8; i++) {
    initialize String filePathDecodeFinal = null;
    if ((i >= 0) && (i < 8)) {
        initialize filePathDecodeFinal = fileQRCodeBlackWhite[0] + i + "." +
fileTypePNG;
    }

initialize String filePathDecodeComplete = filePath + filePathDecodeFinal;

/* Decode Black and White QR Code */
initialize and excute String valueText =
QRCodeText.decodeTextQRCode(filePathDecodeComplete);

/* Write to Text File */
execute writer.write(valueText);

}
    writer.close();
} catch (WriterException ex) {
    ex.printStackTrace();
} catch (IOException ey) {
    ey.printStackTrace();
} catch (NotFoundException ez) {
    ez.printStackTrace();
}

/* Algorithm Module Index P20 – Decode Encoder/Decoder Text (Base64) to Compression file */
/*-----*/

initialize Base64cli base64 = new base64cli();

try {
    initialize object FileInputStream ifs = new FileInputStream(base91TextFileDecode);
    initialize object FileOutputStream ofs = new FileOutputStream(gZipTextFileDecode);

    execute Base64.decode(ifs, ofs);
} catch (Exception e) {
System.err.println(e);
}

/* Algorithm Module Index P21 –Extraction Compression File to Text File */
/*-----*/
initialize object GZip gZip = new GZip();
execute gZip.gunzipIt(gZipTextFileDecode, plainTextFileDecode);

```

```
/* Algorithm Module Index P22 – Open Text Application and Ready to Manipulate*/  
/*-----*/  
execute "C:\windows\system32\notepad.exe"  
  
}  
}
```



Appendix G : Partial Extraction (Re-encode) Level 2

The algorithm of partial extraction level 2 (re-encode) process for error correction level L.

```
/* Algorithm Module Index P23 – Initialisation */  
/*-----*/  
  
/* Creating the package to be used */  
package qrcodecmm;  
  
/* Initialisation java library */  
call import java.io.FileInputStream;  
call import java.io.FileOutputStream;  
call import it.sauronsoftware.base64.Base64;  
call import java.io.IOException;  
  
/* Creating main class */  
public class QRCodeCMM {  
  
/* Initialisation variables to be used */  
initialize static String plainTextFile = "D:/QR  
Code/QRCode/Journal3/singleTextLevel1L.txt";  
initialize static String gZipTextFile = "D:/QR  
Code/QRCode/Journal3/gZipTextFile.gzip";  
initialize static String base64TextFile = "D:/QR  
Code/QRCode/Journal3/base64TextFile.b64";  
initialize static String filePath = "D:/QR Code/QRCode/Journal3/";  
initialize static String fileName = "fileNumber";  
initialize static String fileType = ".txt";  
initialize static String fileTypePNG = "png";  
initialize static String fileRGB = filePath + "fileRGB." + fileTypePNG;  
initialize static final int size = 551;  
  
/* Creating main programme */  
public static void main(String[] args) {  
  
/* Counting the total characters including line feed and carriage return */  
initialize object CounterLetters count = new CounterLetters();  
execute count.CountLetter(plainTextFile);  
  
/* Algorithm Module Index P24 – Compression utility (GZip) */  
/*-----*/  
  
initialize object GZip gZip = new GZip();
```

```

execute gZip.gZipFile(plainTextFile, gZipTextFile);

/* Algorithm Module Index P25 – Encoder for Base64*/
/*-----*/

initialize object base91cli base91 = new base91cli();
try {
initialize object FileInputStream ifs = new FileInputStream(gZipTextFile);
initialize object FileOutputStream ofs = new FileOutputStream(base64TextFile);
execute base91.encode(ifs, ofs);
} catch (Exception e) {
display error System.err.println(e);
}

/* Algorithm Module Index P26 – Compatibility QR Code ANSI to UTF */
/*-----*/

initialize object ansiToUTF8 utf8 = new ansiToUTF8();
execute utf8.convert(base64TextFile);

/* Algorithm Module Index P27 – Characters distribution to files */
/*-----*/

initialize object CreateQRCode1 create1 = new CreateQRCode1(size);
initialize object CreateQRCode9 create9 = new CreateQRCode9(size);
initialize object CreateQRCode17 create17 = new CreateQRCode17(size);

/* Counting characters */
initialize object CountChar countChar = new CountChar();
execute int countCharacter = countChar.count(base64TextFile);

/* Divide characters with related value and fit each of file */
initialize object DivideCharacters divide = new DivideCharacters();

/* Create blank files */
execute create1.createBlank40Files(filePath, fileName, fileType);

/* Embedded text characters to each blank file created*/
execute int totalFiles = divide.divideCharacter(base64TextFile, countCharacter,
filePath, fileName);

/* Checking if the file exceed more than 8 */
if (totalFiles >= 8) {
System.exit();
}

```

```

/* Specify three group that contains 8 files each */
initialize int eight = 8;
initialize String[] multiColourLayerFail = {"QRRed", "QRGreen", "QRBlue"};
initialize int colourCombineRGB[][][] = new int[size][size][3];
initialize MultiLayerQRCode multiLayerQRCode = new MultiLayerQRCode();

/* Algorithm Module Index P28– Create black and white QR Code and combine
it in a group. */
/*-----*/
-----*/

/* Start with first group (red) */
if (totalFiles >= 0) {
    initialize int total8 = 8;

    /* Create an updated single black and white QR Code (Module Index
P12)
    execute create1.generateQRCodeVersion40(filePath, fileName, fileType,
fileTypePNG, total8);

    /* get information from QR Code group 1 (Module Index P13)
    try {
        execute resultFinal = create1.readImage(filePath, fileName, fileTypePNG,
total8);
    } catch (IOException ex) {
    }
    execute int[][] plotResultBlackWhite =
create1.generateMultiplexQRCode(resultFinal, total8);

    /* Combine pixels among 8 black and white QR Codes */
    for (int x = 0; x < size; x++) {
        for (int y = 0; y < size; y++) {
            execute colourCombineRGB[x][y][0] = plotResultBlackWhite[x][y];
        }
    }

/* Algorithm Module Index P29 – Create monocoloured QR Code. */
/*-----*/
-----*/
    execute create1.generateQRCodeVersion40MonoColour(filePath,
fileTypePNG, plotResultBlackWhite, multiColourLayerFail[0]);
}
}

/* Algorithm Module Index P30– Create coloured QR Code. */
/*-----*/
-----*/

```

```
initialize CombineRGB combineRGBColour = new CombineRGB();  
execute combineRGBColour.combineRGB(colourCombineRGB, fileRGB);  
}  
}
```



Appendix H : Processing Time Module

The module of processing time for the current experiments of encoding, decoding and partial extraction.

```
private static String toString(long nanoSecs) {
    int minutes = (int) (nanoSecs / 60000000000.0);
    int seconds = (int) (nanoSecs / 1000000000.0) - (minutes * 60);
    int millisecs = (int) (((nanoSecs / 1000000000.0) - (seconds + minutes * 60)) *
1000);

    if (minutes == 0 && seconds == 0) {
        return millisecs + "ms";
    } else if (minutes == 0 && millisecs == 0) {
        return seconds + "s";
    } else if (seconds == 0 && millisecs == 0) {
        return minutes + "min";
    } else if (minutes == 0) {
        return seconds + "s " + millisecs + "ms";
    } else if (seconds == 0) {
        return minutes + "min " + millisecs + "ms";
    } else if (millisecs == 0) {
        return minutes + "min " + seconds + "s";
    }
    return minutes + "min " + seconds + "s " + millisecs + "ms";
}
```